HP 3000 Computer Systems

HEWLETT PACKARD

APPLICATION DESIGN
Student Workbook

CUSTOMER TRAINING

Course No. 22808A
Part No. 22808-93001

NOV 1980

HP 3000 Computer Systems
Training Course

Application Design
Student Workbook

HEWLETT PACKARD

19447 PRUNERIDGE AVE., CUPERTINO, CALIFORNIA 95014

Part No. 22808-93001

Printed in U.S.A. 11/80

PREFACE

This student workbook was written to assist the student in taking notes while attending the HP 3000 Application Design Training Course. Each of the pages is a copy of an overhead projection slide that the instructor will use in presenting the course material. You will find that by making generous notes on these pages, this workbook will be more useful as a reference after leaving the classroom.

The course material recommended for each student is given below:

Student Workbook 22808-93001
General Information Manual 30000-90008
V/3000 Reference Manual 32209-90001
KSAM Reference Manual 30000-90079
IMAGE Reference Manual 32215-90003
QUERY Reference Manual 30000-90042
Reference Training Manual 30000-90143

CONTENTS

HP3000:

APPLICATION
DESIGN

HEWLETT PACKARD

notes:

references:

MODULES:

I INTRODUCTION

II MPE OPERATING ENVIRONMENT

III TRANSACTION PROCESSING OPTIONS

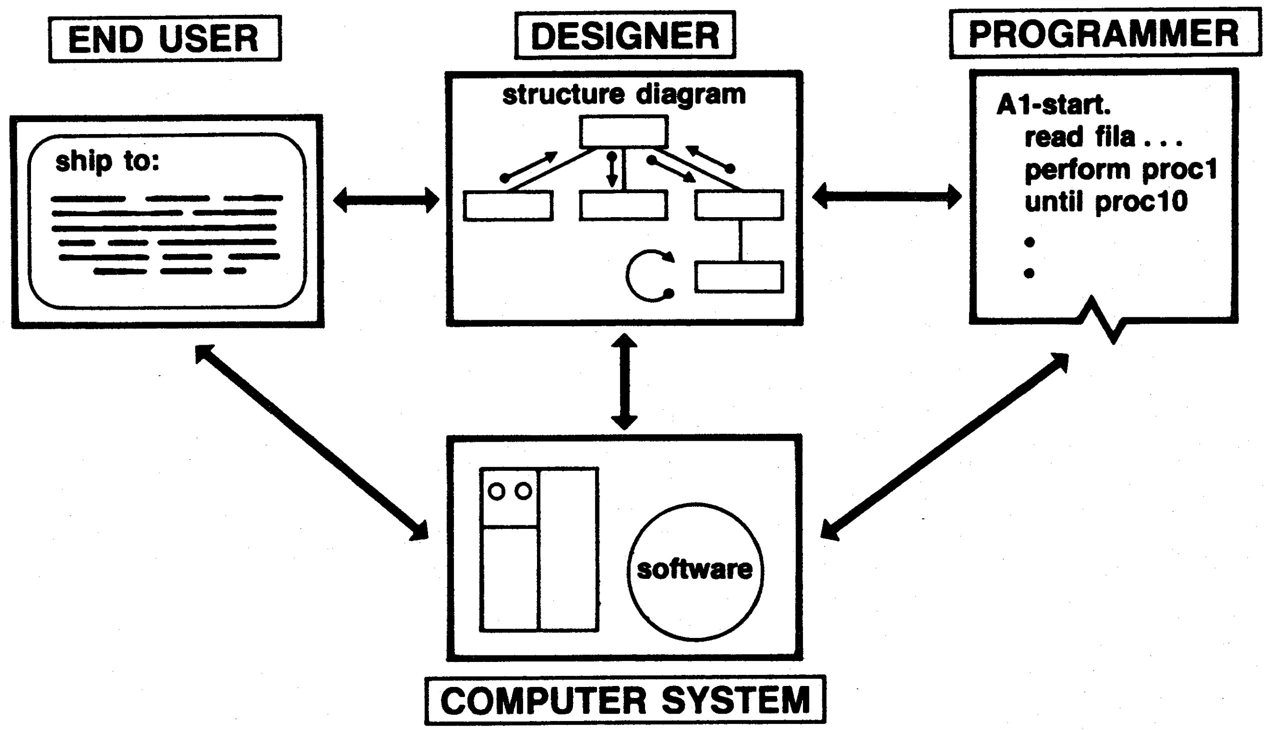
IV DATA MANAGEMENT OPTIONS

V SUMMARY

notes:

references:

FOUR VIEWPOINTS



notes:

references:

the $\boxed{\text{END USER}}$ wants:

- conversational interface to application
- fast response to interactive requests
- ability to generate 1-time reports without programming
- separation from specific details of computer software

anything else?

notes:

references:

the $\boxed{\text{APPLICATION DESIGNER}}$ wants:

- to understand and satisfy end user needs
- to understand the capabilities of the computer system in order to satisfy these needs
- to provide a design that insures
  - fast response
  - rapid access to information
  - high rate of transactions

what else?

HEWLETT PACKARD

notes:

references:

the $\boxed{\text{APPLICATION PROGRAMMER}}$ wants:

- to be able to translate the designer's specifications into a working program
- to understand how to use the computer system's software
- programming tools to help code, debug, and optimize application programs

anything else?

notes:

references:

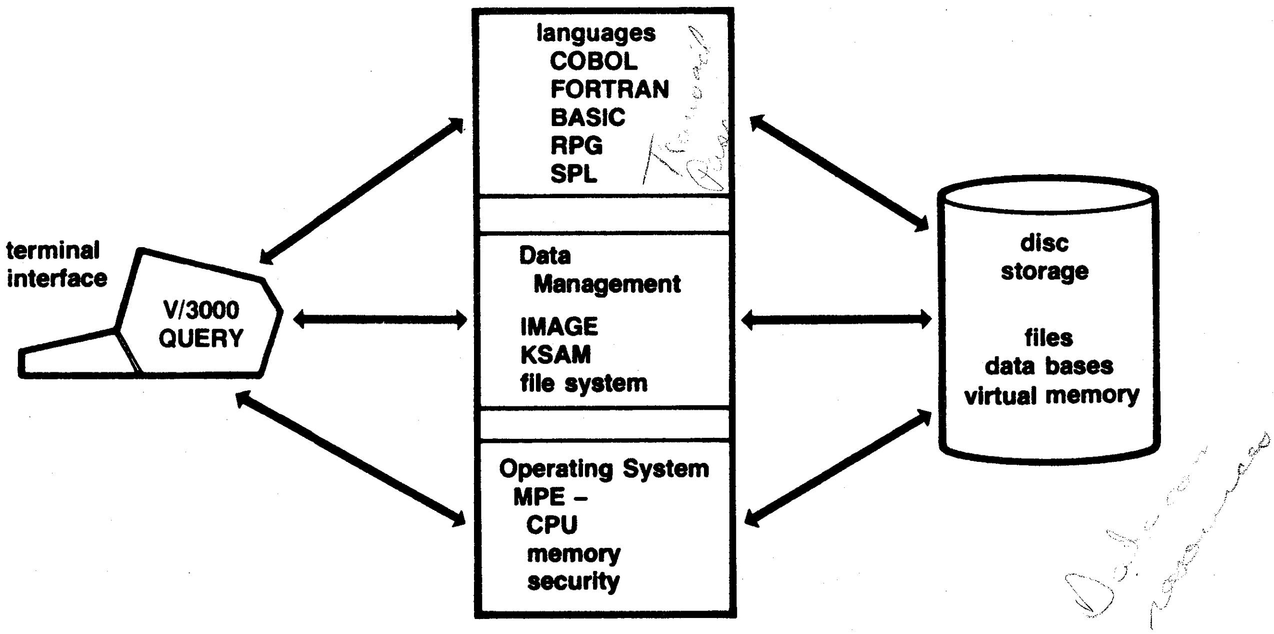
the $\boxed{\text{COMPUTER SYSTEM}}$ should provide:

- **hardware and software that are capable of supporting the given application**
- **support and training for both the designer and programmer to help implement a successful application**

notes:

references:

SOME HP3000 RESOURCES

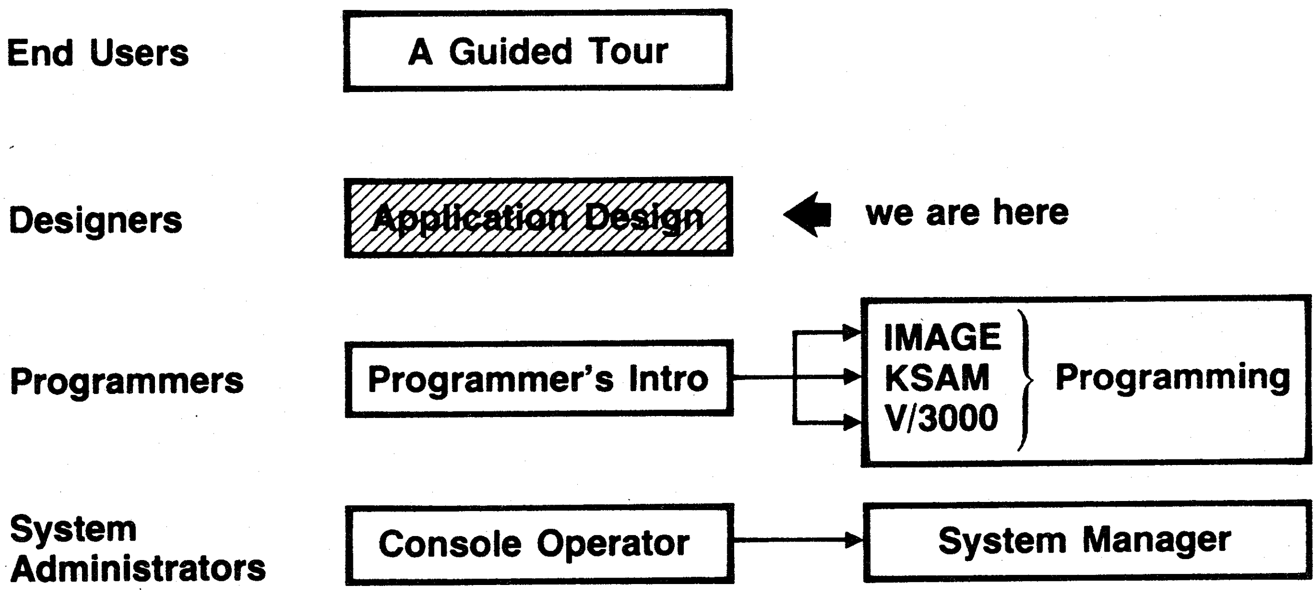


SUPPORT, TRAINING, & DOCUMENTATION

notes:

references:

TRAINING for different users



notes:

references:

DESIGN CONSIDERATIONS

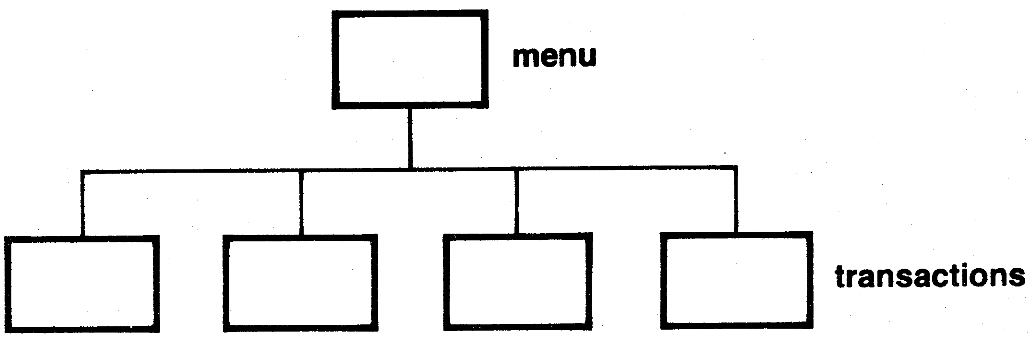
- **STRUCTURE**
  - **what it does** — function chart
  - **how it does it** — structure chart
- **IMPLEMENTATION**
  - **who uses it and how**
  - **what restrictions are needed** — Response time, Disc space, development time

HEWLETT PACKARD

notes:

references:

TYPICAL TERMINAL APPLICATION



How does this structure work on an HP3000?

notes:

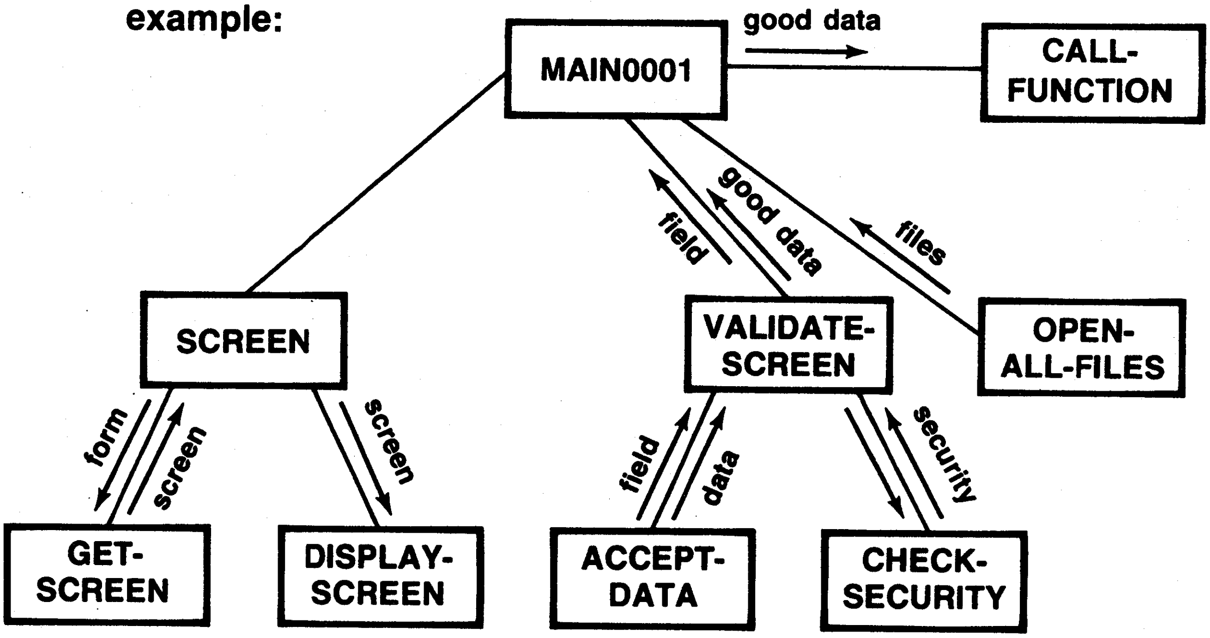
- Think of task in terms of "functions".
- Chart these functions into a set of menu-driven transactions.

references:

Structure Chart

- outlines HOW application does it

example:



HEWLETT PACKARD

notes:

- "Decompose" data as well as the code.

references:

DATA DICTIONARY

- based on data flow in structure chart
  - list of each piece of data for each function (use structure chart as checklist)
  - note characteristics of data items
    - duplicated?
    - sorted?
    - used by more than one function?

notes:

- Prepare for data management decisions from the start.

references:

Design for Maintenance

- who will maintain your programs?

  usually someone else, so make it easy to read and simple to follow

- a program that is easy to maintain is usually easy to use

notes:

references:

understanding the END USER

questions to ask:

- who are they?
- what do they want?
- where do they want it? remote or local
- in what form?
- when?
- and how fast?

notes:

- If you can explain your design to the end-user in terms he/she understands, the design has a good change of working, and of being easy to maintain.

references:

WORKSESSION I-1

notes:

references:

Worksession I-1

The purpose of this worksession is to characterize your application. There is no correct answer, but the more thorough you can be at this stage, the more useful the course will be.

1. Have you settled on a programming language or languages in which to code your application?

   If yes, which? Cobol

2. Characterize the structure of the application:

   A. Summarize, in one sentence if possible, the purpose of your application.

   To produce paychecks & maintain payroll information

   B. Briefly list the main functions of your application—for example: maintain bill of materials, maintain vendor file, etc.

   maintain tax info / deductions
   maintain personnel data
   maintain history of hours worked
   print weekly paycheck
   main

   C. Connect the functions you listed above into a menu tree (as in slide I-10).

   D. Take one of the functions from the menu tree (or choose a subfunction) and determine the flow of data within this function. Show this in any format—very roughly with circles and arrows or, more formally, as a structure chart.

3. Characterize your end-user:

   A. Who will enter data? clerks

   Where? payroll office

   When? Monday / Tuesday

   How fast must response time be? 3 seconds

   Is the flow even? no If not, what are the peak times?

   Monday afternoon, Tuesday morning

Worksession I-1 (cont.)

B. Who will modify data? clerks

Where? payroll office

When? Tuesday afternoon, Wednesday morning

How frequently? weekly

How up-to-date must modifications be? within minutes? hourly? daily? weekly?

minutes

C. Who will retrieve the data?

In what form?

How often? weekly

What kinds of reports are needed?

Do they want unscheduled reports?

4. List your security, accounting, and recovery needs:

A. What functions are restricted?

To whom?

B. List any sensitive items:

Who can see them?

Who can modify them?

C. Do you need audits?

On what transactions?

Worksession I-1 (cont.)

D. How important is recovery? ______________________________

____

What transactions must be recovered in case of a crash?

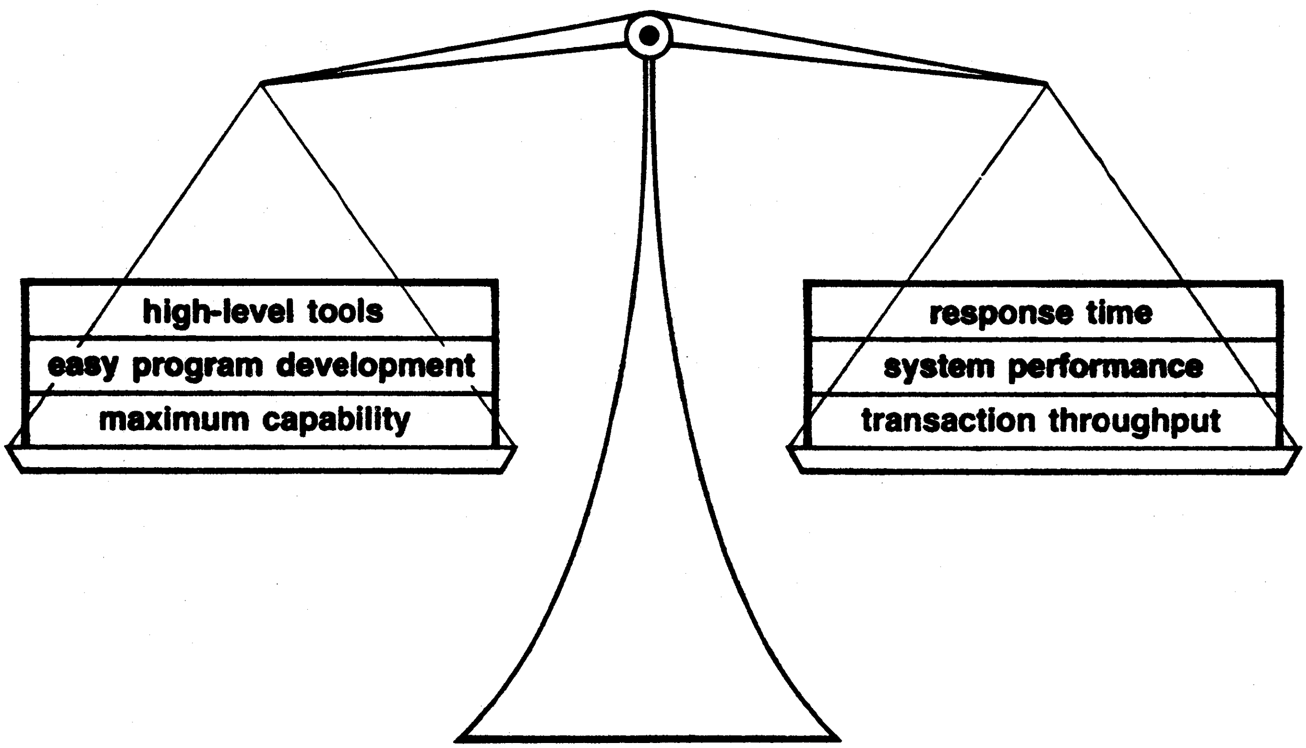
____

____

How long can sys be down during recovery?

Hardware requirements

DESIGN TRADEOFFS



HEWLETT PACKARD

notes:

- Generally, the easier to use, the harder to implement.

references:

APPLICATION DESIGN

ART

not

SCIENCE

HEWLETT PACKARD

notes:

- If designing applications were a science, we could write a program to do it.

references:

MPE OPERATING ENVIRONMENT

- Architecture Overview
- The Process
  - code segments
  - data segments
- Process Generation
- Multiprogramming

notes:

references: General Information Manual
System Reference Manual
MPE Commands Reference Manual
MPE Intrinsics Reference Manual
MPE Pocket Guide

ARCHITECTURE OVERVIEW

HARDWARE **SOFTWARE**

- **Stack Architecture**
- **Separation Of Code & Data**
- **Virtual Memory**
- **The Process**

HEWLETT PACKARD

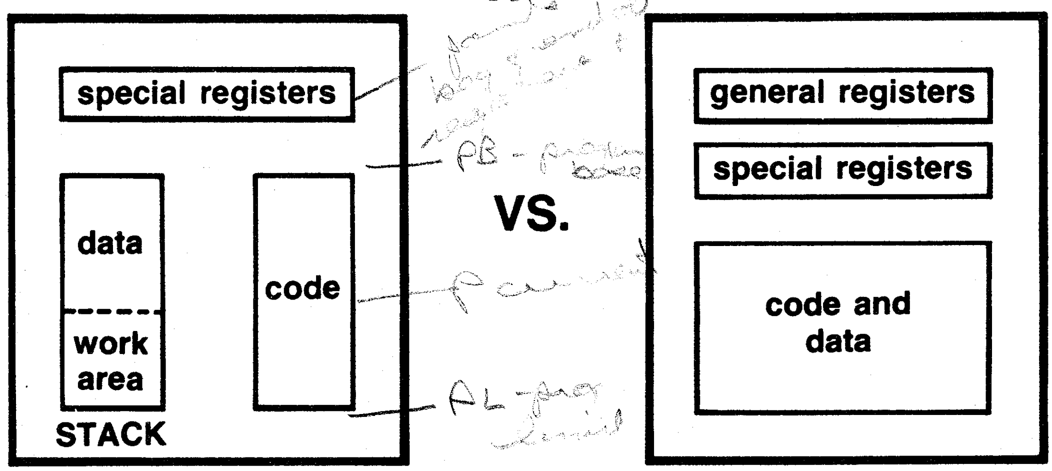
notes:

- HP3000 architecture combines hardware and software.
- Hardware controls the transfers between data stack and central processor.
- Micro-coded instructions in the central processor (firmware), reduce software needs, are super fast.

references:

architecture overview

HP3000 STACK ARCHITECTURE



STACK MACHINE | **REGISTER MACHINE**

- processing in stack
- code and data always separate

- processing in general registers
- may combine code with data

notes:

- No general registers are needed in an HP3000.
- Code and data are separate.
- Both are features that save memory space.

Advantages of stack:

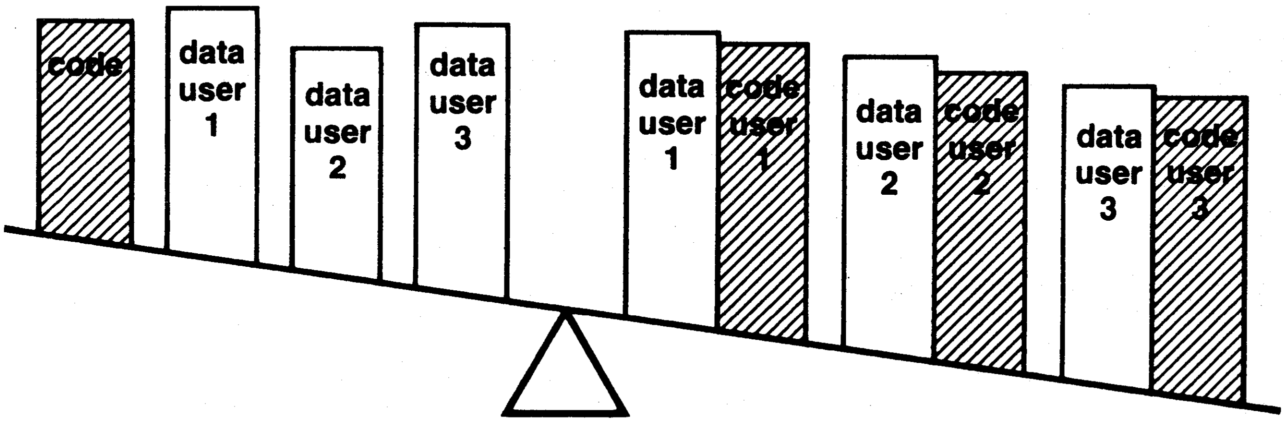
① ... is done within the stack

② hardware protected execution pointers (registers) keep track of where data is

③ dynamic allocation of space w/in stack for calculations

references:

SHARED CODE SAVES MEMORY SPACE



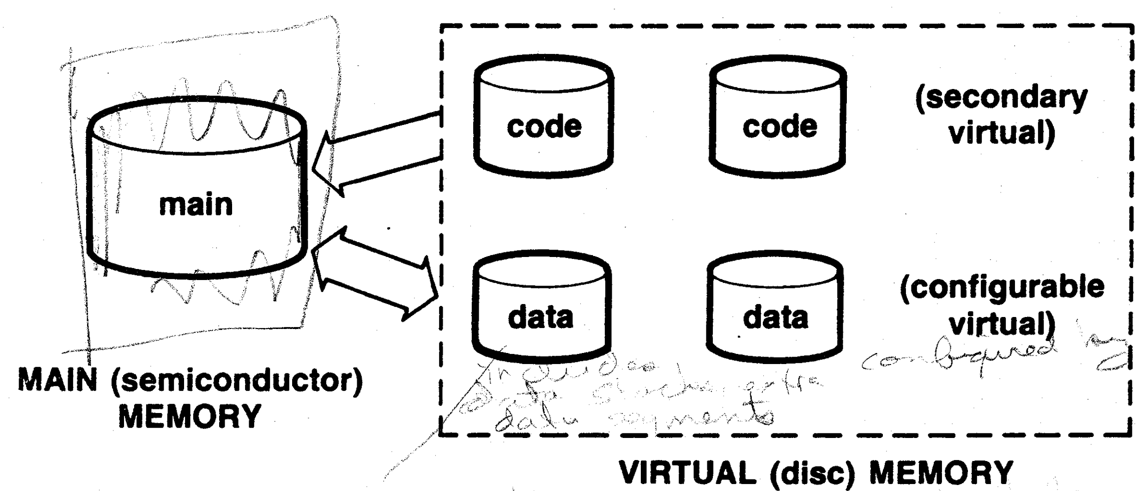
code/data separate VS code/data together

notes:

- When code and data are inseparable, copies of code waste space.

references:

VIRTUAL (DISC) MEMORY FOR STORAGE



10 Kilo sectors/disc

virtual memory can be a bound

notes:

- To the user, there is no difference between main and virtual memory.

- To the system, disc I/O is needed to transfer code from virtual to main memory, and transfer data to and from main memory.

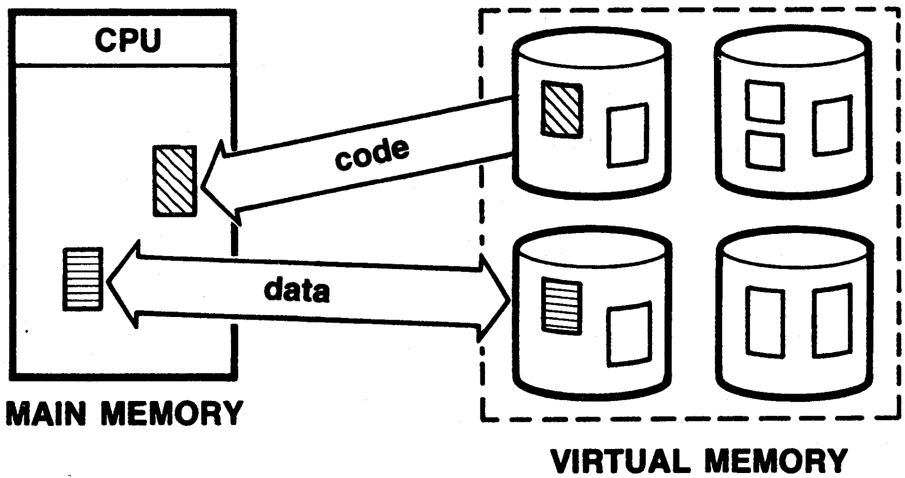
- Note: only the virtual memory used for data is "configured" into the system. The virtual memory for code is simply the program file that results from preparing code for execution.

To be executing must be in main.

To be recognized as a process must be in virtual (recognized by a CST table

references:

MAIN MEMORY FOR EXECUTION



HEWLETT PACKARD

notes:

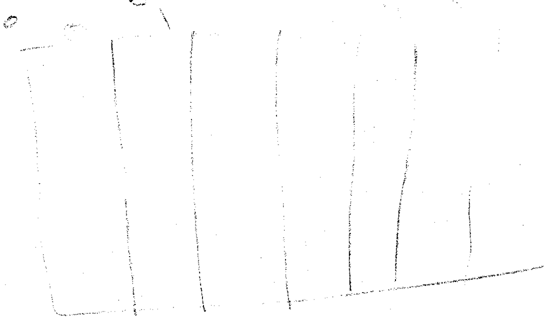
Totally rebuilt when system is brought up.

- In order to execute, a program's code and data must both be in main memory. This is managed by MPE.

- Note: only the code and data actually required for immediate execution must be in main memory; some code segments may remain on disc.

1 bank = 64k words

CST
DST
PCB
others



references:

WORKSESSION II-1

notes:

references:

Worksession II-1 (architecture)

1. The HP 3000 is a stack machine. True or false? T ____________

2. One characteristic of a stack machine is that code and data are separate.
   True or false? ____________

3. Describe one advantage of separate code and data?

   ____

   ____

   ____

4. Give at least one difference between main memory and virtual memory.

   ____

   ____

5. In order for a program to execute, all of its code and data must be in main memory.
   True or false? F ____________

THE PROCESS

- General Characteristics
- Components
- Code Segments — Overview

notes:

references:

PROCESS DEFINITION

"A process is

the unique execution of a program

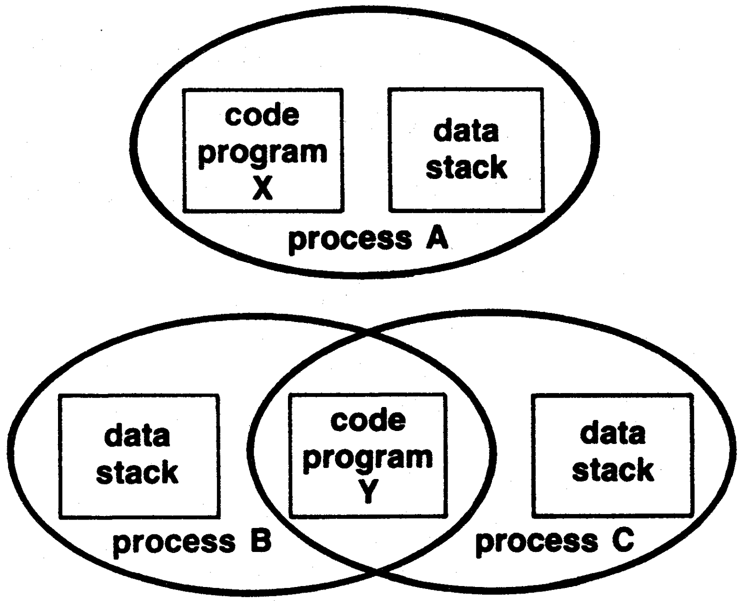
by a particular user

at a particular time."

notes:

references:

PROCESS CHARACTERISTICS



code: **non-modifiable**

shared

re-entrant

data: **absolutely private**

separate from code

modifiable

notes:

- Code cannot change during process execution (non-modifiable), is returned intact after interruption (re-entrant), and can be shared by many users.

stack

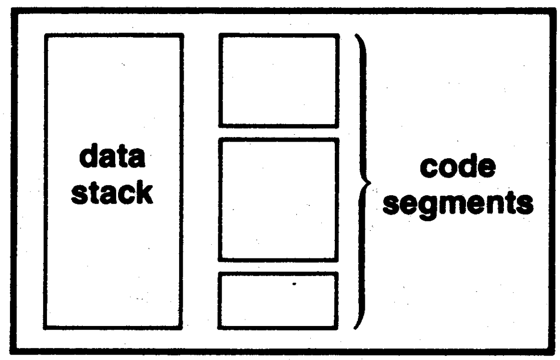
- Data cannot be seen or changed by other users executing the same code (absolutely private), is stored separately from code, and can not be modified by any users sharing the code.

references:

PROCESS COMPONENTS

An executing process consists of:

- **1 or more code segments**
- **1 data segment (the "stack")**



PROCESS

notes:

- Note: in addition to code segments and data stack, the executing process may also need "extra data segments". If so, these will be in main memory too. (Extra data segments are discussed later in this module.)

references:

the process

WORKSESSION II-2

notes:

Mode
disable
enable Interrupts
Traps - arithmetic overflow
Right or Left - which instruction (of byte)
is left to complete

Overflow results
Carry bit
CC condition code
that corresponds to entry
in code segment table
(ST points to location
of that code segment (in
either main or virtual
memory)

references:

Worksession II-2 (the process)

1. If you execute the same program twice, does this result in one process or two processes?

   two

2. If you and another user each execute the same program, does this result in one or two processes?

   two

3. Shared code can be modified. True or false? false
   Explain your answer.

   sharable but
   not modifiable

4. Private data can be modified. True or false? true
   Explain your answer.

   by owner

5. What are the two required ingredients of an executing process?

   1 code segment
   1 data ~~[illegible]~~ stack

CHARACTERISTICS OF CODE SEGMENTS

- variable lengths
- managed by system
- naturally relocatable
- defined by user

notes:

Come from program file - prg create
or SL's (system library, account library, group library)

- Code segment definition: Any group of instructions that the programmer decides should be kept together as a unit.

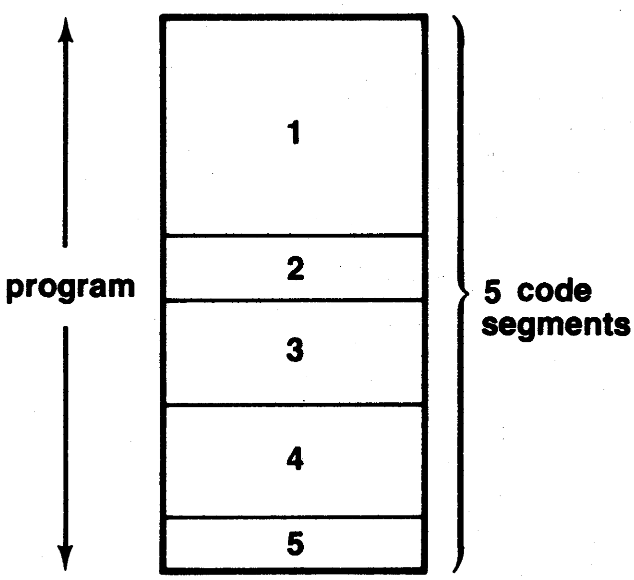
Smallest unit of code used by the system at run time

Code segments are made up of Relocatable binary modules - RBM's

references:

the process

Code segments are variable length



- optimal length depends on:
  - amount of memory
  - frequency of use
  - number of segments

notes:

- Maximum length = 16K words,
- Most MPE segments are 5K or less; try for user code segments in the same size range. 4-8K

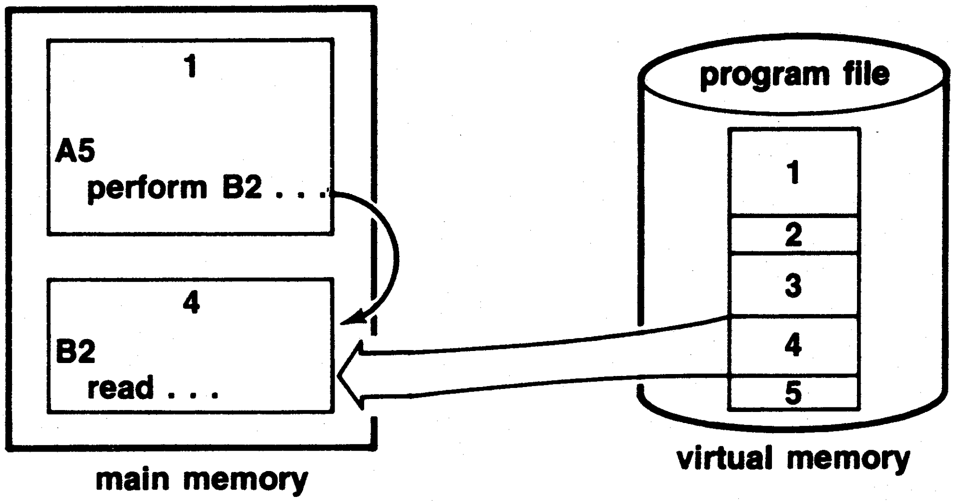
If lots of memory to spare - code segments can be large.

If high freq of use - large?
low - take out of segment ... in smaller segment

references:

of segments. 63 max segments per program

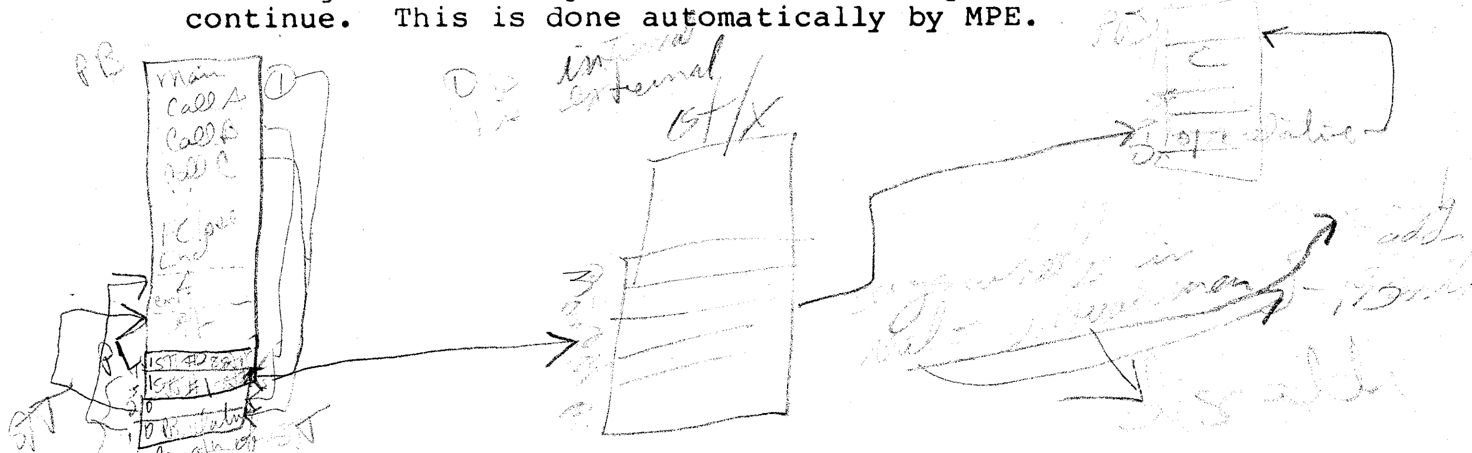
Code segments are managed by the system



HEWLETT PACKARD

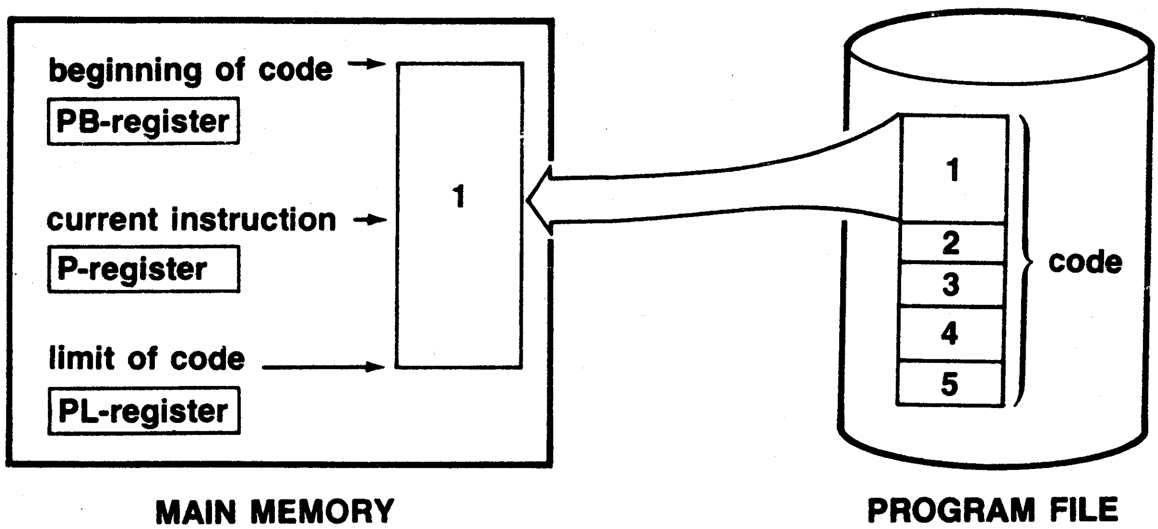
notes:

- In this example, code segment 1, which is currently executing, references code in segment 4 (not in memory). At that point, code segment 4 is copied into main memory so execution can continue. This is done automatically by MPE.



references:

Code segments are naturally relocatable



notes:

- Program registers in the CPU keep track of the location of any executing code segment. (These registers are PB, P and PL.)
- Code can be placed anywhere in memory simply by updating registers.
- All addresses in code are PB-relative.

references:

the process

Code segments are defined by user

segment:	
①	program-ID sample-program
②	start section 10
③	init section 20 term section 20
④	main section 30
⑤	err-proc section 40

Code defines segment boundaries* therefore, programmer controls:

- **the number of code segments**
- **the size of each segment**
- **which code goes into which segment**

*** except RPG and APL**

HEWLETT PACKARD

notes:

- Maximum number of code segments per program is 63.

- Note: code segments defined in program can be changed using a "segmenter" program.

References:

Check language reference manuals (COBOL, COBOL II, RPG, SPL, BASIC Compiler) for details on segmenting code.

Get size of code segment from PMAP

PMAP

references:

WORKSESSION II-3

notes:

references:

Worksession II-3 (code segments)

1. What is one advantage of variable-length code segments?

   customize size

2. A. Is there a maximum code segment size? Yes If so, what is it? ~~16~~ 64

   B. Is there a maximum number of code segments per program? Yes
   If so, what is it? 16

3. Suppose code in one segment causes a transfer to code in another segment; what must your program do to manage this transfer?

   nothing

4. Does your program need to know the address in main memory of the currently executing code segment? Explain your answer.

   No registers keep track of this.

5. What characteristics of code segmentation can be controlled by the programmer?

   size, how many, contents

CODE SEGMENT DESIGN

How should code be segmented?

- **Concept of Working Set**
- **Concept of Locality**
- **Size Considerations**

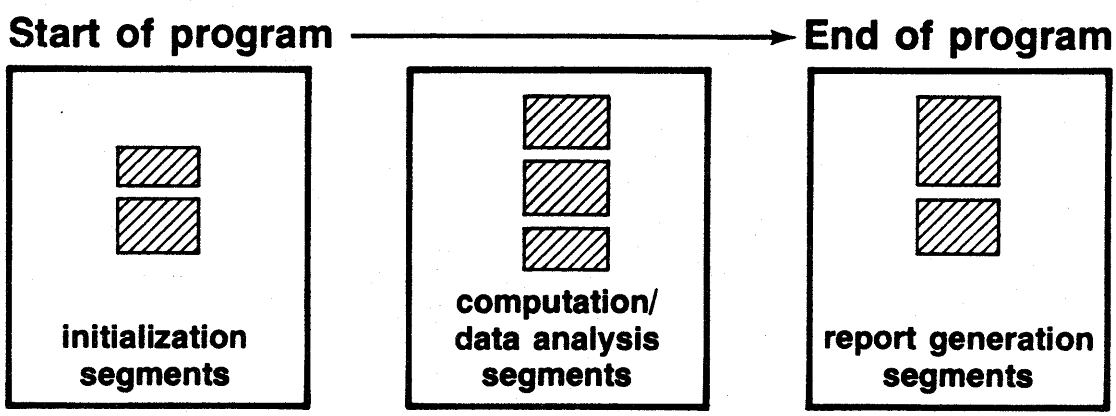
notes:

Constraints:
Physical size of main memory.
Time involved in swapping segments
(# of interrupts)

references:

WORKING SET —

- **The smallest set of segments that must be in main memory for a program to work efficiently.**



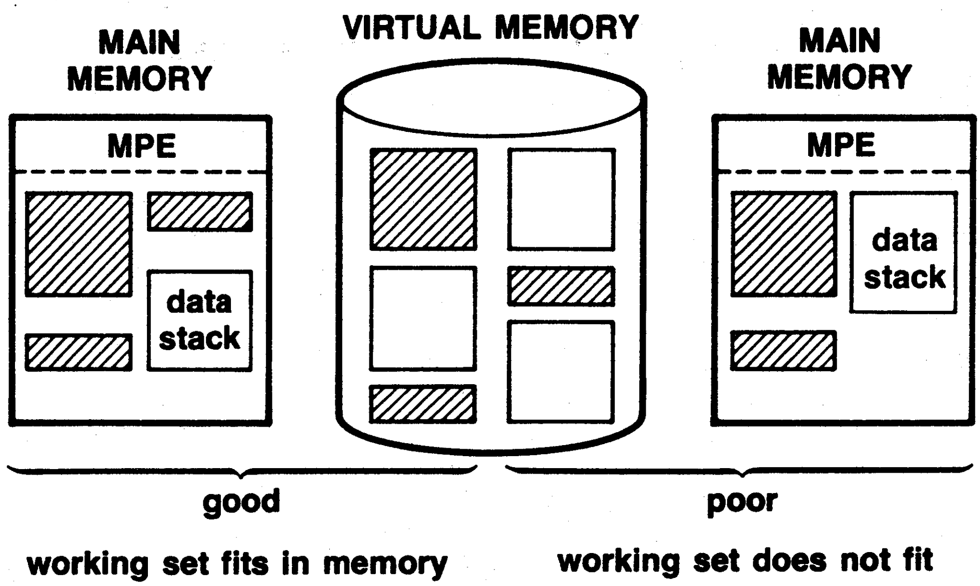
notes:

- The working set for any executing program is "dynamic". It changes continually throughout the life of the program.

references:

WORKING SET (2)

- **The entire working set should fit in main memory for efficient processing**



notes:

- Note: the data stack is NOT part of the working set; the stack must always be in memory when a program executes.

- In this example, the three code segments currently in the working set (shaded boxes) all fit in main memory on the left - only two segments fit on the right.

references:

WORKING SET (3)

How can you make sure the working set fits in main memory?

A) You can add more memory

OR

B) You can run fewer programs at a time

OR

C) You can structure your program to achieve better code locality — and a smaller working set

notes:

Stay in + stay out of code segments for as long as possible

- Which of these solutions makes sense for your application?
  A can be expensive
  B limits the application
  C should be attempted

references:

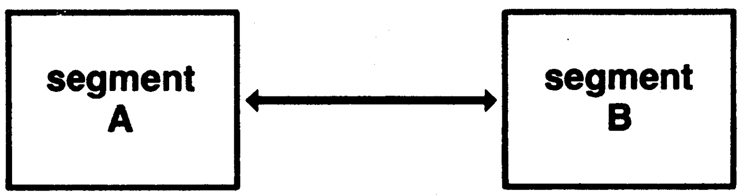
CODE LOCALITY

- **Good locality on an HP3000 means:**

  Control stays in one segment for as long as possible — when it leaves a segment, it stays out as long as possible.

- **Poor locality means:**

  Control branches between code segments frequently — puts more code in working set.

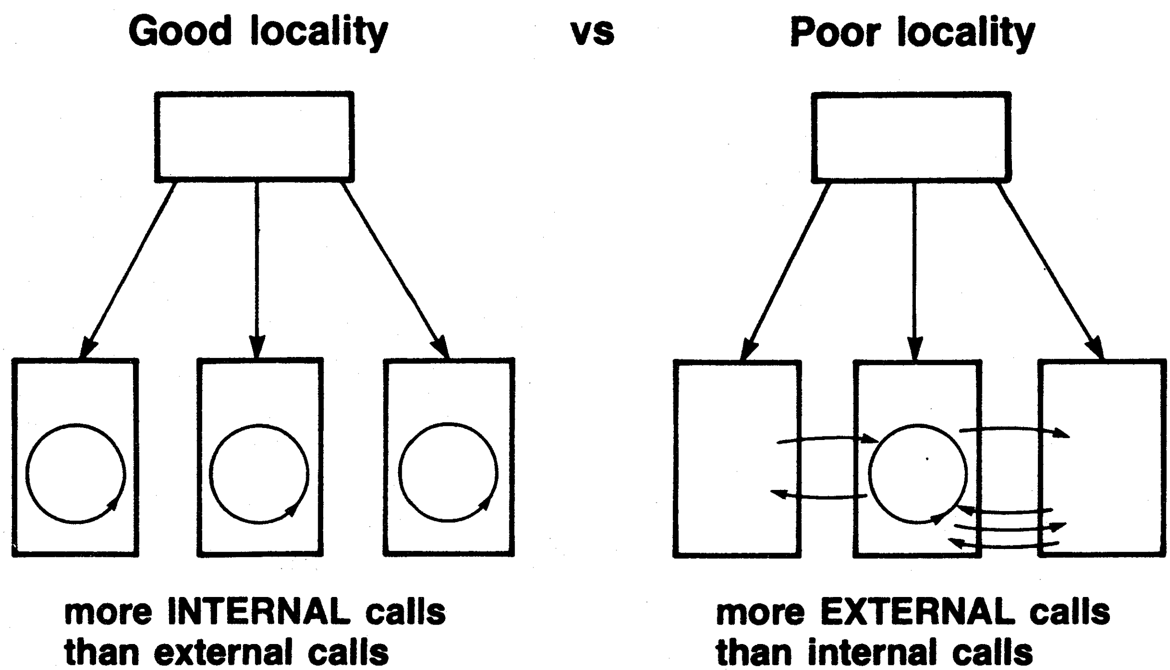


- **If transfers between A and B are frequent, put that code in the same segment.**

notes:

Segment transfer: Takes [illegible] as long as intrasegment calls.

references:



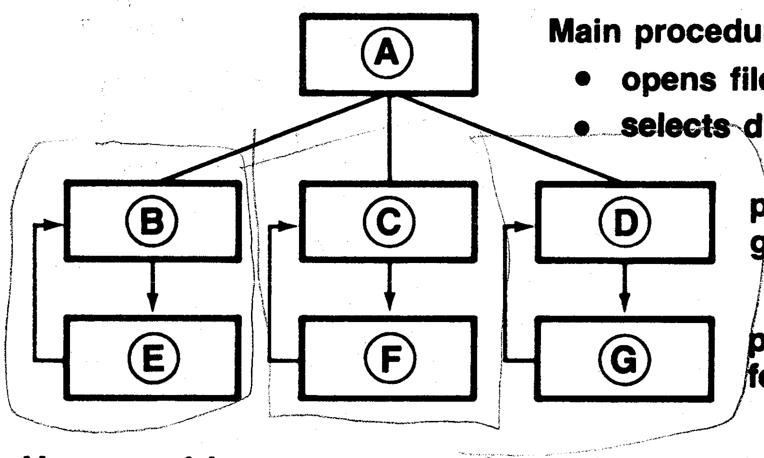
notes:

- This is simply another way of looking at locality.

references:

segment design

EXAMPLE: suppose a program generates displays using 3 display formats



Main procedure
- **opens file**
- **selects display format**

procedures B, C, D get records

procedures E, F, G format display screen

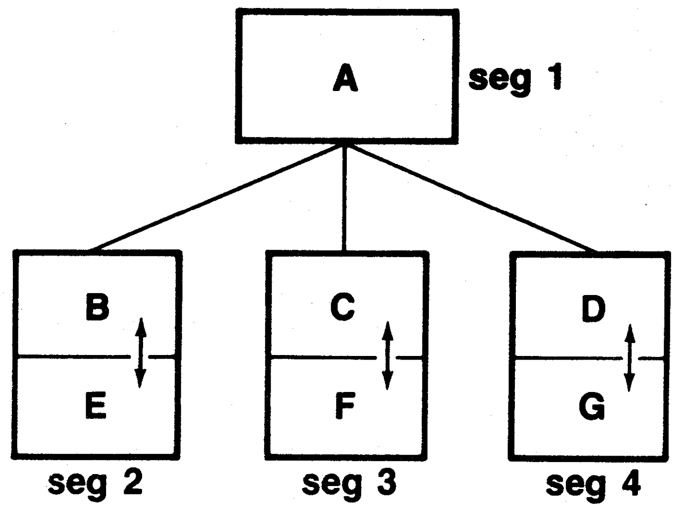
How would you segment these program blocks?

notes:

references:

segment design

SOLUTION:



HEWLETT PACKARD

notes:

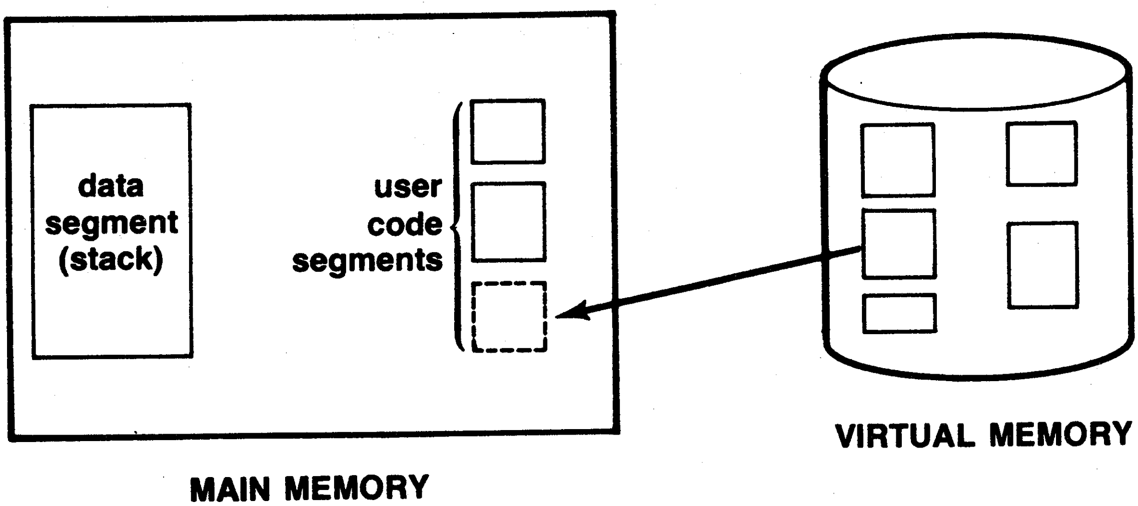
- This solution keeps code from crossing segment boundaries each time a record is read and formatted.

- Segments are not too large; there is no redundant code.

references:

FIRST rule for segmenting code:

1. **Stay in segment as long as possible, and stay out as long as possible**



notes:

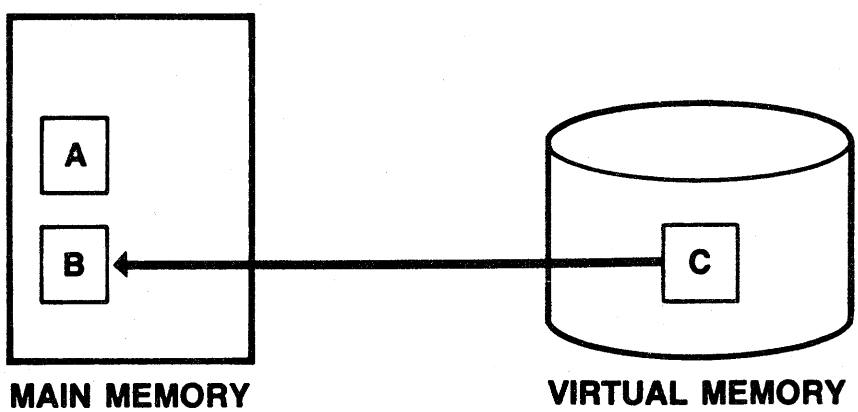
- Every time a program crosses a segment boundary, it increases the chance that code must be transferred from disc.
- When a referenced segment must be transferred from disc, the program suspends.

references:

SECOND rule for segmenting code:

2. **Make segments the same size – easier for MPE to find space**

example:



- **segment C can overlay segment B**

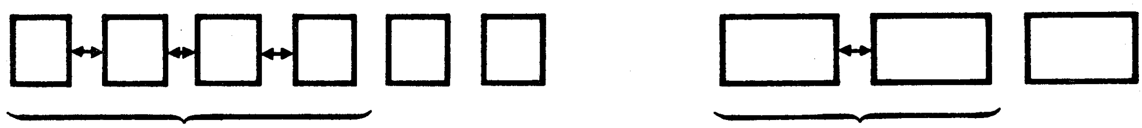
notes:

references:

THIRD rule for segmenting code:

3. **Keep segments small, but not too small**
   - **remember the first rule**

many small segments **vs** **fewer large segments with good locality**



working set **working set**

- **small segments MAY cause excessive inter-segment transfers**
- **larger segments MAY reduce inter-segment transfers**

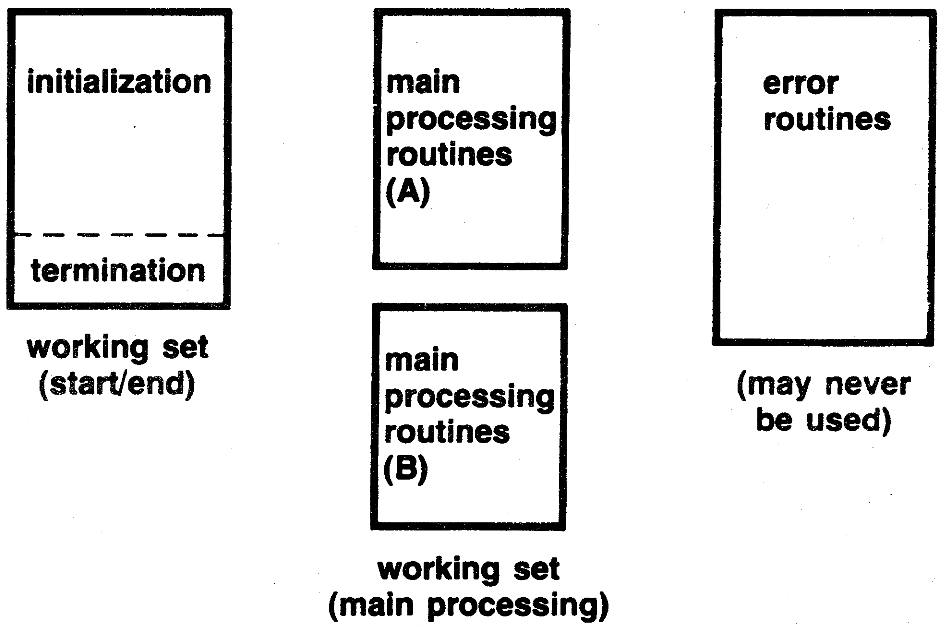
notes:

- Remember: the maximum number of segments per program is 63.
- To help find space in memory, segments should be 5k words or less.

references:

FOURTH rule for segmenting code:

4. **Separate infrequently used code from code that is executed most often**



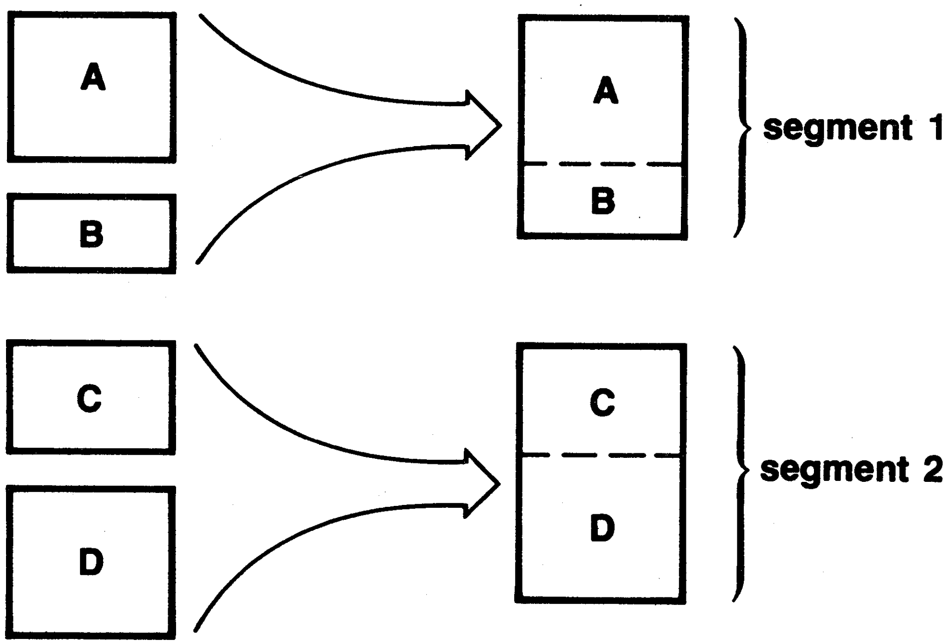
notes:

- A routine can be large if it is seldom (or never) executed.
- Such code should always be separated from code that is executed frequently.

references:

FINAL rule for segmenting

5. Code in segmentable program units



notes:

- Code is prepared into segmentable blocks, called RBMs (Relocatable Binary Modules).
- Programmer controls which code is placed into which RBM.

references:

COMPILERS AND SEGMENTATION

compiler		defaults
COBOL '68	—	2 segments (1 for Initialization) (1 for Main Program)
COBOL II	—	1 segment (Initialization + Main Program)
FORTRAN	—	1 segment (Main Program)
BASIC	—	1 segment (Program is smallest unit)
RPG	—	compiler divides program into 4K segments (user can specify 1K, 2K, or 3K)
SPL	—	1 segment (Main Program)
APL	—	no segmentation (compilation generates data only)

user control:

COBOL — section-name priority-number

FORTRAN, SPL, BASIC — $CONTROL SEGMENT = segment-name

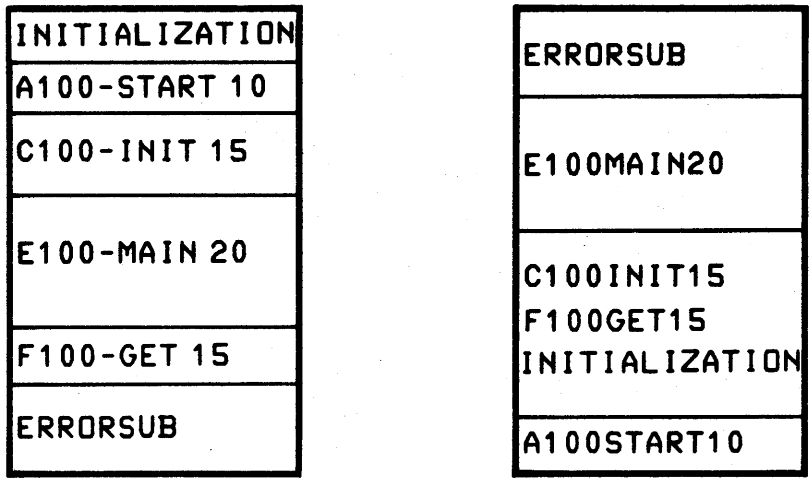
notes:

- Note differences between compilers.
- Check manuals for specific details.

references:

How COBOL II determines RBMs and Segments

Compiler **Segmenter**



6 RBMs **4 Segments**

notes:

references:

WORKSESSION II-4

notes:

references:

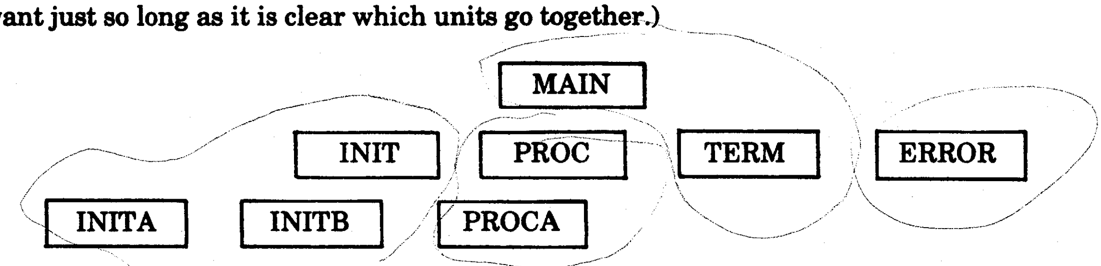
Worksession II-4 (segment design)

1. Define a working set.

2. How does code locality affect performance on an HP 3000?

3. Give at least three rules for effective segmentation.

4. Segment the following sample program by marking the program units that you would put in the same segments. (Each box represents a program unit; you can mark them any way you want just so long as it is clear which units go together.)



MAIN calls INIT, PROC, TERM once each.

INIT calls INITA, INITB once each; each is small and executes quickly.

PROC and PROCA work together to do most of the processing.

TERM performs termination procedures; it is small and executes quickly.

ERROR may be called by any other procedure in case of error.

DATA SEGMENTS

Stack

- **Layout**
- **Management**

Extra Data Segments

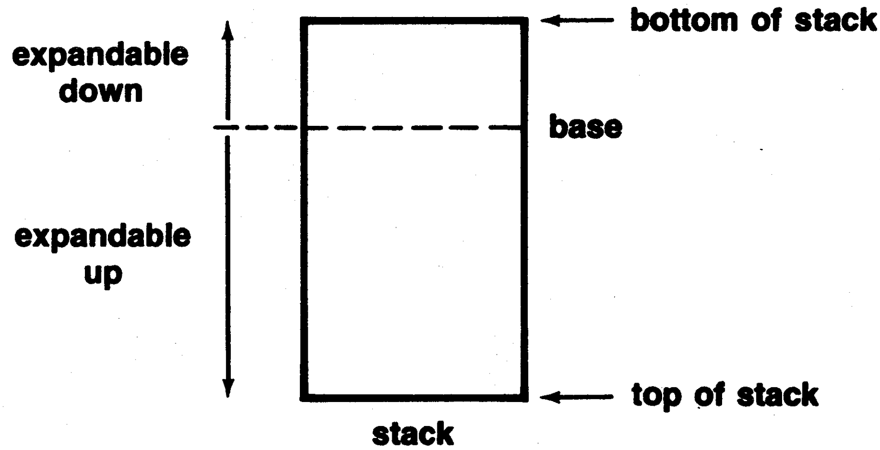
HEWLETT PACKARD

notes:

references:

WHAT IS A STACK?

- 1 word of data stacked on top of another
- LIFO - last item added to top is the first item removed



notes:

- Note: HP3000 data stack is always shown "top down".

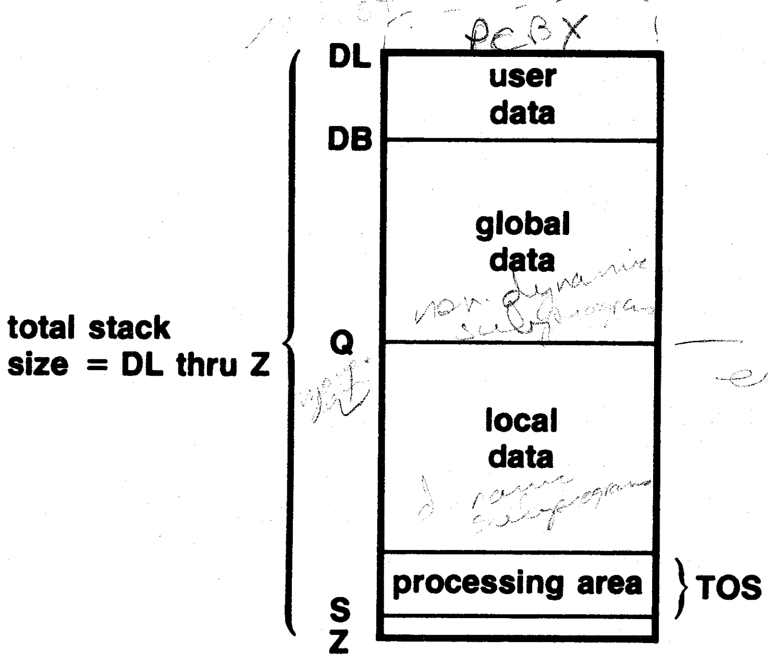
references:

data stack

HP3000 DATA STACK

- 1 per process
- variable length

(except V plus)



- **user managed data area DB-relative (negative)**
- **global data DB-relative (positive)**

— points to a currently executing program

- **local data for subroutine Q-relative (positive/negative)**
- **top-of-stack (TOS) S-relative (negative)**

uses of the stack:

PCBX 1. Preserve environment (open files, extra data segments) system managed area

DB → 2. Main program: Holds global variables that need to stay around for entire life of program; working storage, NO code. User managed. STATIC. Never changes. Defined at prep.

notes:

- DL = data limit, the lowest address available to programmer.
- DB = data base, the base for all stack expansion whether up or down.
- Q = dynamic base for the current subroutine(s).
- S = stack top (top of stack), the area where code is executed; as new values are moved onto stack, S moves toward Z
- Z = stack limit, the very top of the stack (except for a small overflow area managed by system).

Q → 3. Subprograms: local variables, working storage. Dynamic: like dynamic subprograms

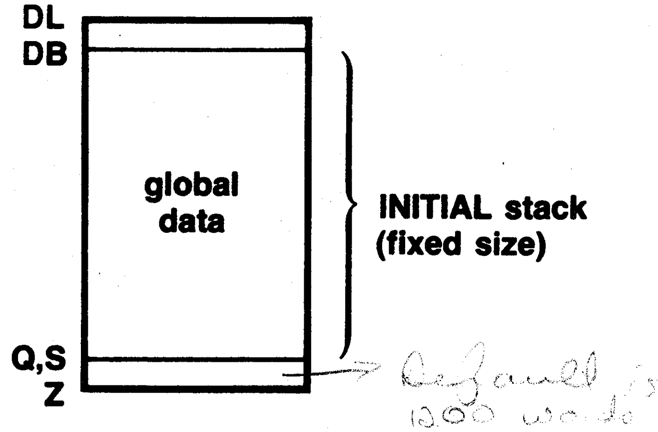
S → 4. Temporary variables: a scratch pad for calculations. Extremely dynamic

references:

DL → 5. Subsystem information; used by compilers

6. Overflow area: 128 words. Used by

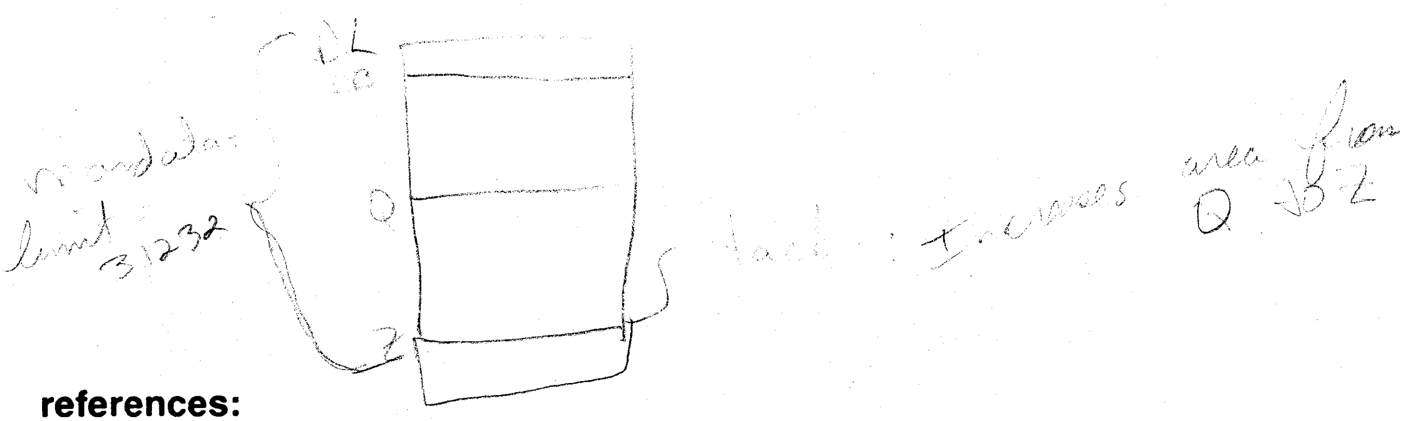
DATA STACK - ANOTHER VIEW



- **the INITIAL stack must be as large as your global data**

notes:

- Before any execution, Q and S are at the same location.
- As code is executed, the stack expands dynamically up to the limit set by user.



references:

STACK MANAGEMENT

What can a program do to manage stack size?

- Increase stack limits (MAXDATA or STACK)
- Shrink the stack dynamically (ZSIZE or DLSIZE)
- Design to keep global area small Put constants in code or in a dynamic subprogram.
- Segment code to reduce stack size

use NOCAT intrinsic

Default maxdata = DB + (DL) Q + stacksize (from config [illegible] 1/3 - min 1000)

A stack is a data segment - can only be 32 K words

32K = 32767

notes:

maxdata limit = 31232

1535

128

overflow 1407 = max size of PCBX if stack size is at max.

If [illegible] dynamic open

Runqueue; ABCB This gives the PCBX an extra data segment.

[illegible] much [illegible] For stack overflow: Try to minimize the size of global. Use as much dynamic as possible. use maxdata.

references:

STACK LIMITS

- stack size (DL-Z) estimated by system
- user can increase this estimated size:
  - STACK = increase stack size all at once
  - MAXDATA = increases stack size in 1K increments

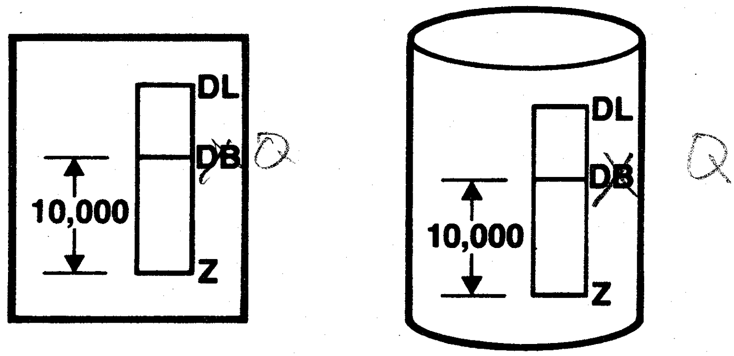
notes:

references:

data stack

example:

STACK = 10000 words



MAIN VIRTUAL

- **DB to Z allocated at once in both MAIN and VIRTUAL MEMORY**

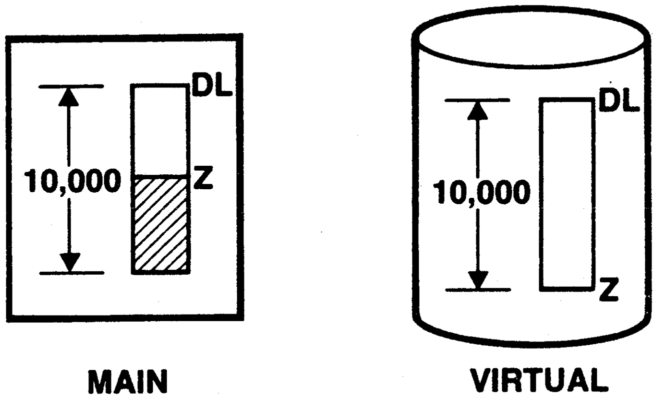
notes:

- Note: the STACK command does not expand DL-DB area.
- Can be wasteful of memory since main memory space allocated from start.

references:

example:

MAXDATA = 10000



- **maximum stack (DL to Z) allocated in VIRTUAL MEMORY**
- **expanded in MAIN MEMORY as needed (1K increments)**

notes:

- MAXDATA saves main memory.
- Costs in disc I/O needed for incremental expansion.

references:

Use MAXDATA –

- if you need to expand DL → DB area
- if you run out of stack space during execution

NOTE: Neither MAXDATA nor STACK will shrink stack automatically

notes:

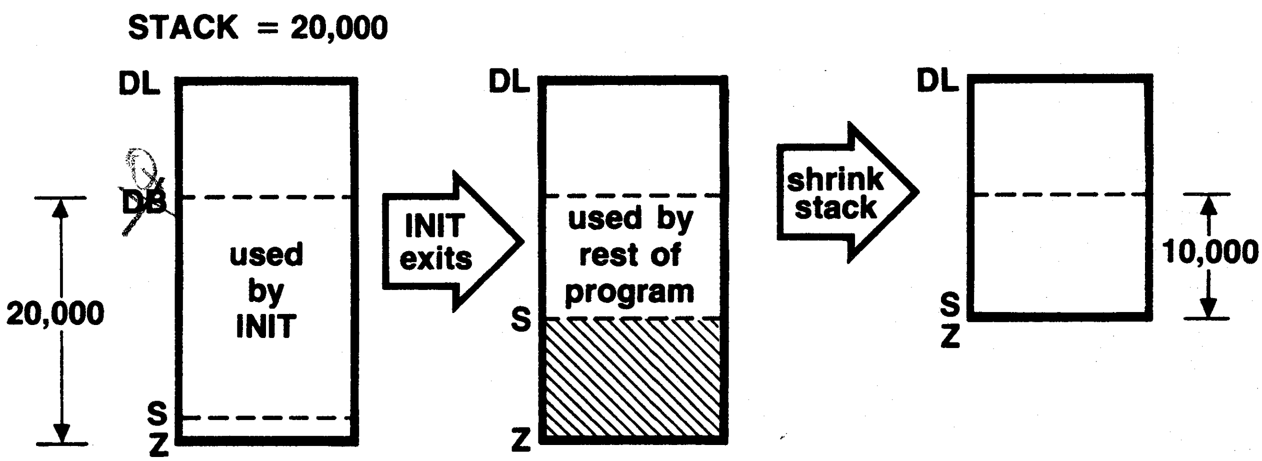
- Use MAXDATA if you get a "stack overflow" message when you execute program.

references:

USE STACK = if you need a large stack immediately

example:

- INIT segment requires 20,000 words
- rest of program requires only 10,000 words



notes:

- STACK saves disc I/O required to expand stack with MAXDATA - use it if you know you will need the extra stack space immediately.

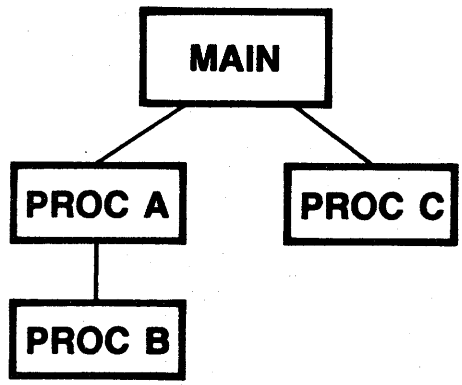
- You can shrink stack when space no longer needed.

references:

EXAMPLE OF STACK GROWTH (1)

assume:

program modules shown below:



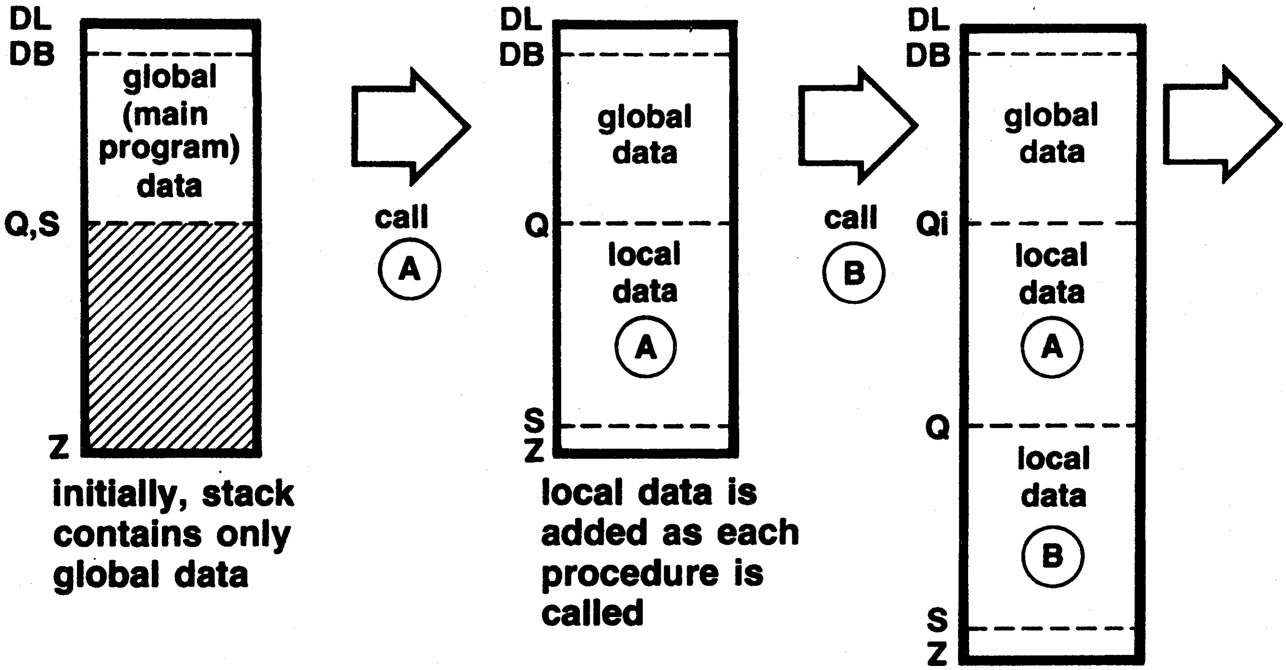
HEWLETT PACKARD

notes:

- In this example, assume MAXDATA is specified to allow stack to expand past size estimated by system.
- Each increase means swap to and from disc.

references:

EXAMPLE OF STACK GROWTH (2)



initially, stack contains only global data

local data is added as each procedure is called

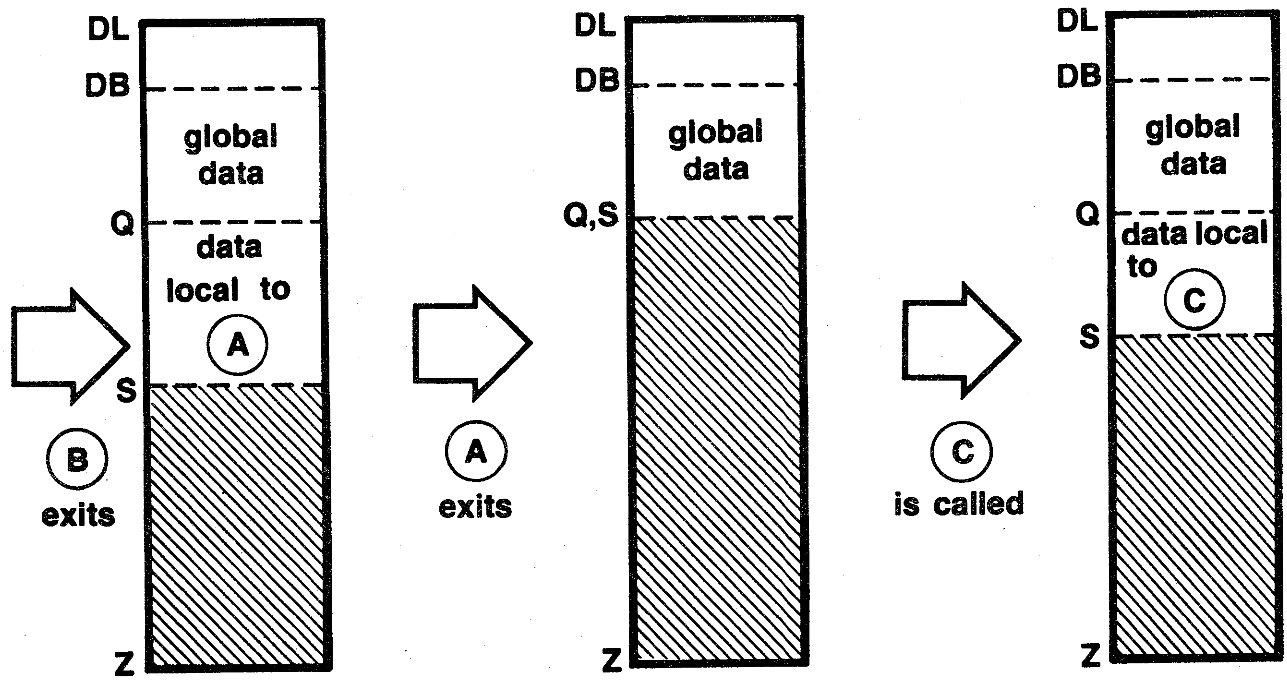
stack expands past default estimate

notes:

- The first location of Z is the system-determined stack limit.
- After procedure B is called, Z is moved past this limit in 1K increments. Z can expand up to the MAXDATA limit, but no further.

references:

EXAMPLE OF STACK GROWTH (3)



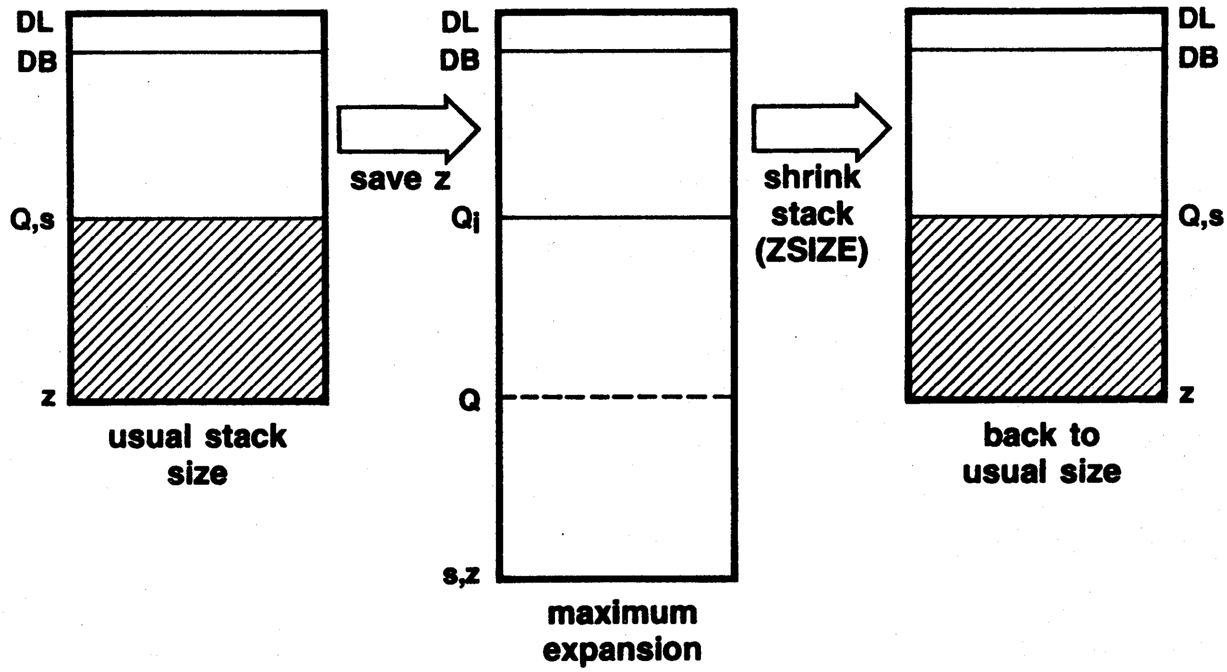
notes:

- Shaded areas of stack are unused except when A calls B (previous slide).
- Stack does not shrink automatically.

references:

USE ZSIZE TO SHRINK STACK

example:



notes:

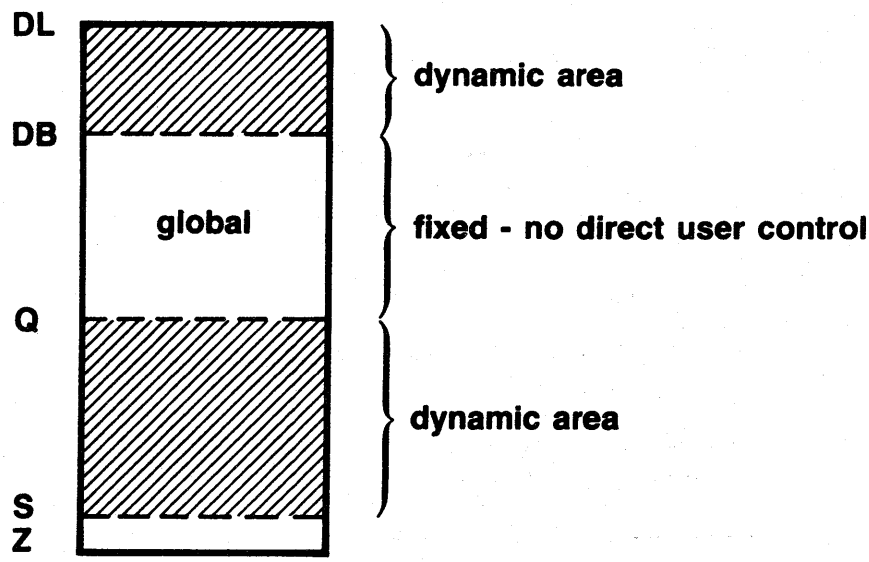
- It is good practice to shrink the stack after the program has finished with procedures that expand the stack past its normal size.

- See appendix A for sample procedures to determine relative location of Z (usual stack size) and then shrink the stack back to this size.

- These procedures can also be used to expand stack programmatically. Similar procedures can manage DL-DB area.

references:

DESIGN TO KEEP GLOBAL AREA SMALL



notes:

- Use global area for:
  main program data
  data common to more than one procedure
  data maintained by a procedure between calls

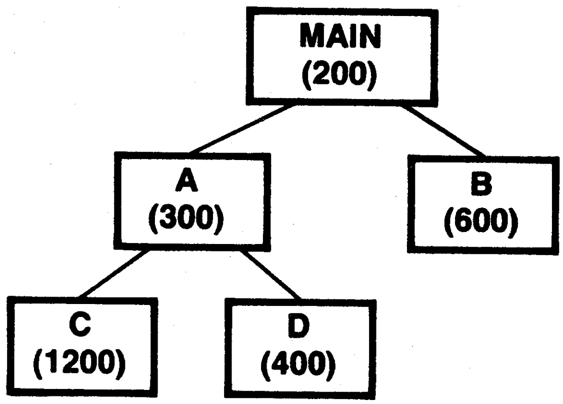
- Place constant data (such as error messages, screen displays) in code segment.

- For COBOL programs, global area contains Working Storage for main program plus some other general purpose data. It also contains data for subprograms unless they are compiled with DYNAMIC option (more on dynamic subprograms in the language unit of Module III).

references:

SEGMENT CODE TO REDUCE STACK SIZE

problem: restructure this program to reduce stack size



- largest stack requirement when A calls C

HEWLETT PACKARD

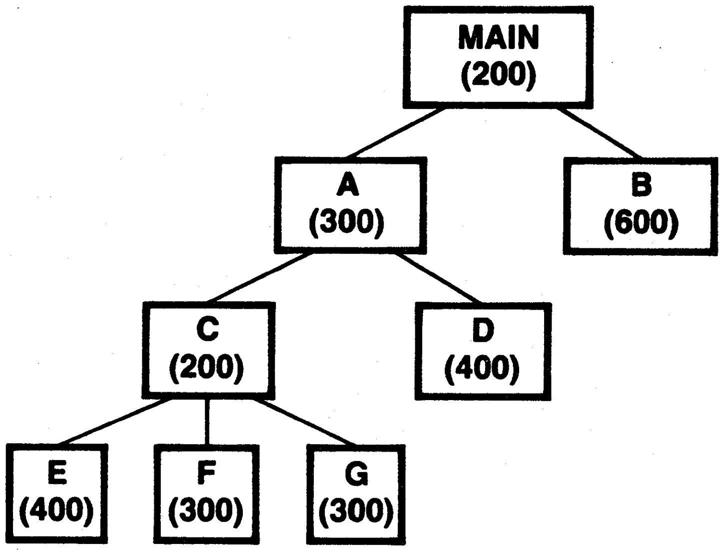
notes:

- Numbers in parentheses are stack requirements in words.
- When MAIN calls A and A calls C, the total words needed=1700.

references:

data stack

solution: break largest procedure into subprocedures that are not in direct line



HEWLETT PACKARD

notes:

- In this solution, the largest stack requirement is 1100 words.
- But, keep the other factors in mind when segmenting - don't reduce the stack size only to cause more transfers between code segments.

references:

WORKSESSION II-5

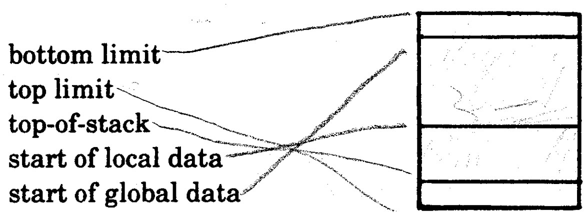
notes:

references:

Worksession II-5 (data stack)

1. The data stack can be shared by more than one process. True or false? False

2. Assume a program with one called procedure that is currently executing. Label the stack diagram to show:



3. Using your stack drawing, shade (or otherwise mark) the areas whose size can be managed by the user.

4. Describe briefly three methods for the programmer to manage stack size.

using subprograms
STACK=
segment code
use Z-size

5. A. Suppose your application calls procedure "X" that doubles the usual size of the stack. "X" is called once only, and the call is neither at the beginning nor end of execution. Is this a situation where you could use ZSIZE effectively? Explain your answer.

Yes

B. Suppose this single very large procedure "X" is the first procedure called by your program, and the default stack size is not sufficient. Would you use STACK or MAXDATA to expand your stack limit? Are there any drawbacks to your choice?

Use stack Stack takes main mem space

Worksession II-5 (cont.)

C. Suppose "X" uses the DL-DB area of the stack; would you use STACK or MAXDATA to expand your stack size? Are there any drawbacks to this choice?

Maxdata

6. How does putting error messages in a code segment help keep your stack small?

Error messages are called infrequently and the code is only in the stack when the error message routine is called

EXTRA DATA SEGMENTS

- What are they?
- Why use them?
- Limitations

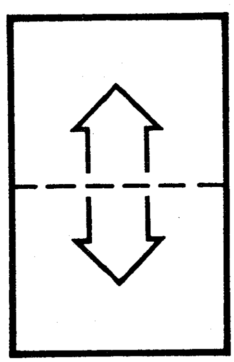
[illegible] memory

notes:

references:

WHAT IS AN EXTRA DATA SEGMENT?

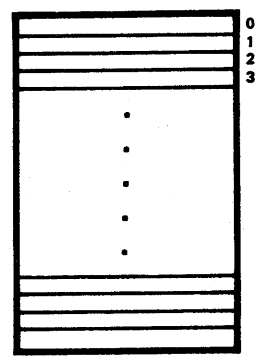
■ a block of unstructured, uninitialized memory



data stack

- structured
- private
- 1 per process

VS



extra data segment

- linear
- private or sharable
- up to 255 per process

notes:

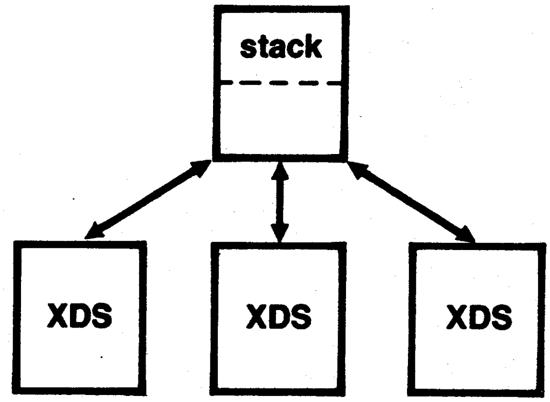
- Extra data segments must be managed by the application.

Message files allow
to pass messages
between processes

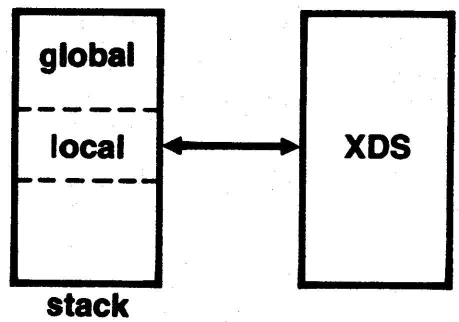
references:

WHY USE THEM?

- to provide LINEAR unstructured storage
  - large arrays
  - table look-up



- to provide GLOBAL storage for procedures
  - saves data after procedure exits
  - available for other procedures



HEWLETT PACKARD

notes:

```
- Some other uses:
     to decrease stack size
     to share data between related processes (processes in same
     family)

- Note: the file system uses extra data segments extensively
  for data buffers.
```

Can be used in JCL's

references:

WHAT ARE THEIR LIMITATIONS?

- require special capability and user management
- user program must:
  - create and delete any extra data segments
  - move data from XDS to stack, and from stack to XDS
- XDS must be in main memory when accessed (together with code and user stack)
- use resources — disc I/O, memory, CPU time

HEWLETT PACKARD

notes:

created by program
needs ds cap. before
list the remain

references:

WORKSESSION II-6

notes:

references:

Worksession II-6 (extra data segments)

1. Give at least 2 differences between the data stack and an extra data segment:

2. Which of the following data storage needs can be solved by using extra data segments?

   A. A program needs storage for an array that is too large for the data stack.

   B. A program needs an area to hold local data from a procedure after the procedure has exited.

   C. A program needs a storage area for data to be passed to another program in the same process tree.

PROCESS GENERATION

- Life Cycle of Process
- Code Libraries

notes:

references:

"COMPILE-LOAD-GO"

What does HP3000 do?

standard terms:	compile	load	go
HP3000 terms:	compile	prep	run

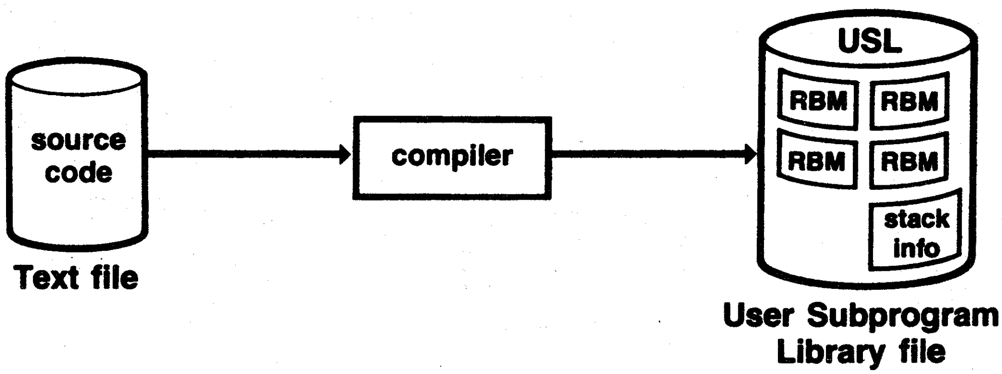
HEWLETT PACKARD

notes:

- Compile stage common to most systems; it produces object code from source code.
- Loading in HP3000 has two stages:
  PREP resolves some externals, links code segments
  RUN (in first phase) resolves remaining externals, sets up stack
- RUN (in second phase) executes program.

references:

COMPILE



- segments planned – but not final

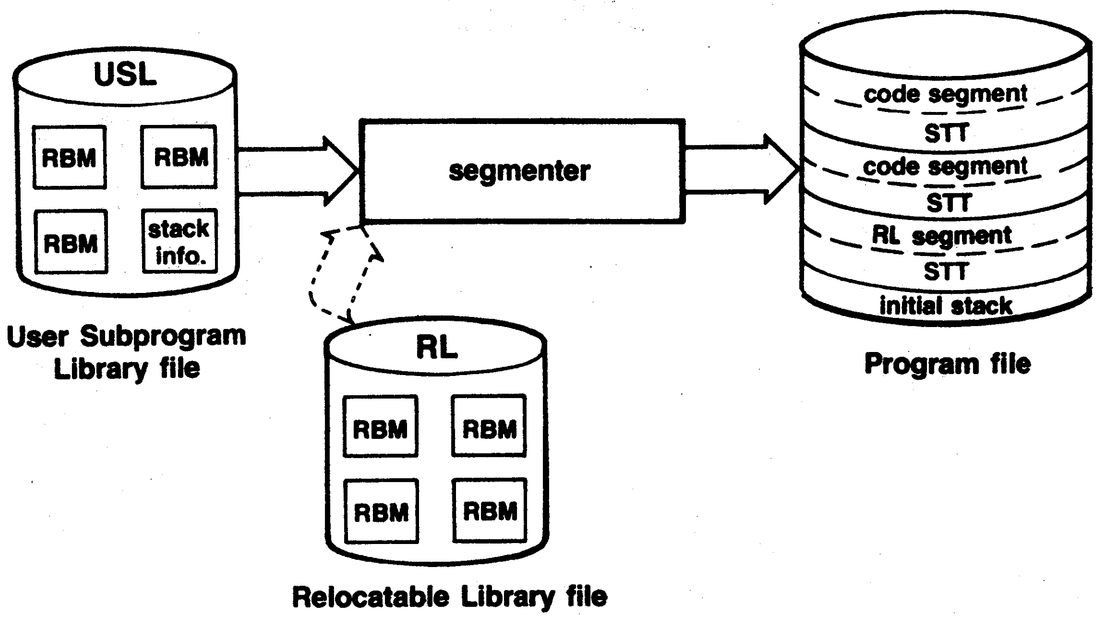
HEWLETT PACKARD

notes:

- This stage uses a compiler program (different compilers for each language).
- Source code in text file compiled into "Relocatable Binary Modules" in USL file.
- Stack information kept in USL file with RBMs.
- RBM is basic building block; one or more may be combined into code segment, but RBM cannot be split into two or more segments.

references:

PREPARE



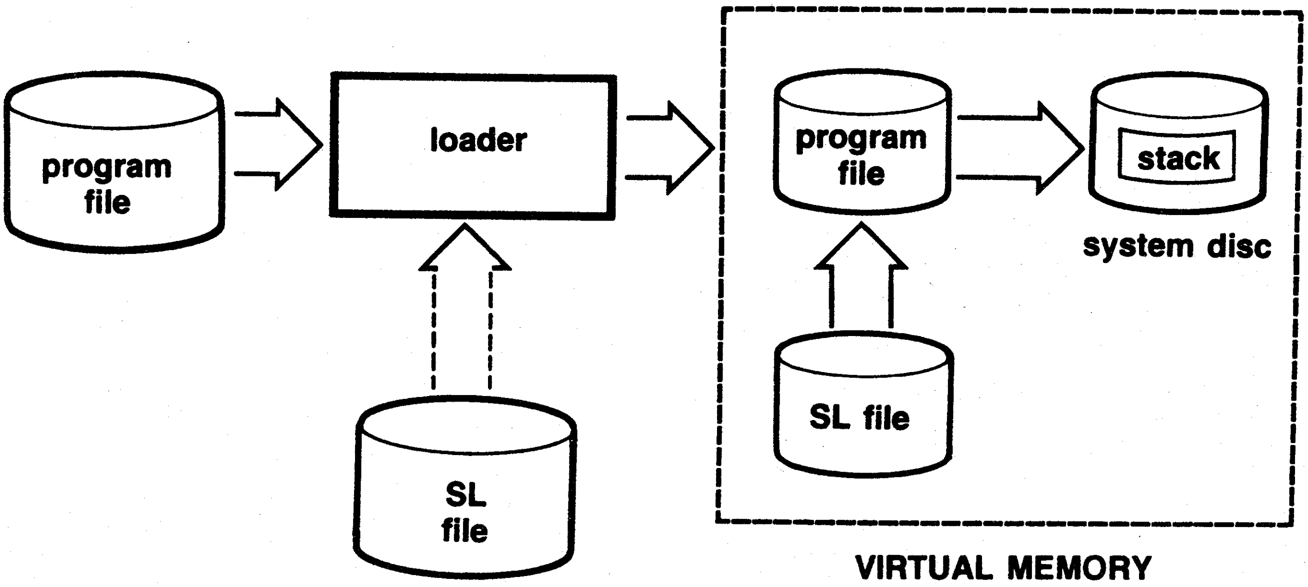
■ **initial stack, code segments linked in Program file**

notes:

- This stage uses the Segmenter program.
- Sets up final code segments in program file. The segments are linked through an STT (Segment Transfer Table) associated with each code segment in program file.
- Sets up initial stack (global stack data) in program file.
- Resolves externals from "Relocatable Library" and builds an RL segment in program file.

references:

RUN (1)



- first phase links program units
- not yet a process

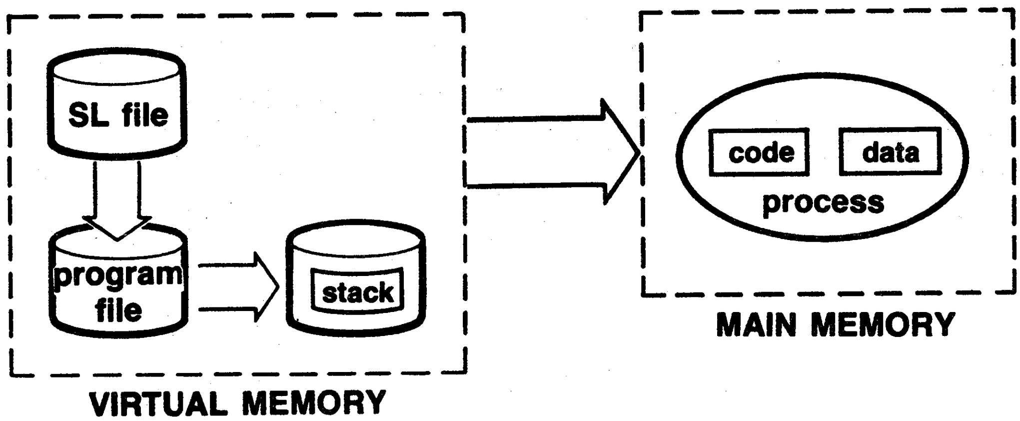
HEWLETT PACKARD

notes:

- RUN, in this stage, uses the Loader program.
- Allocates space for stack on system disc.
- Completes linkage for code segments in system tables.

references:

RUN (2)



■ **creates and executes process**

notes:

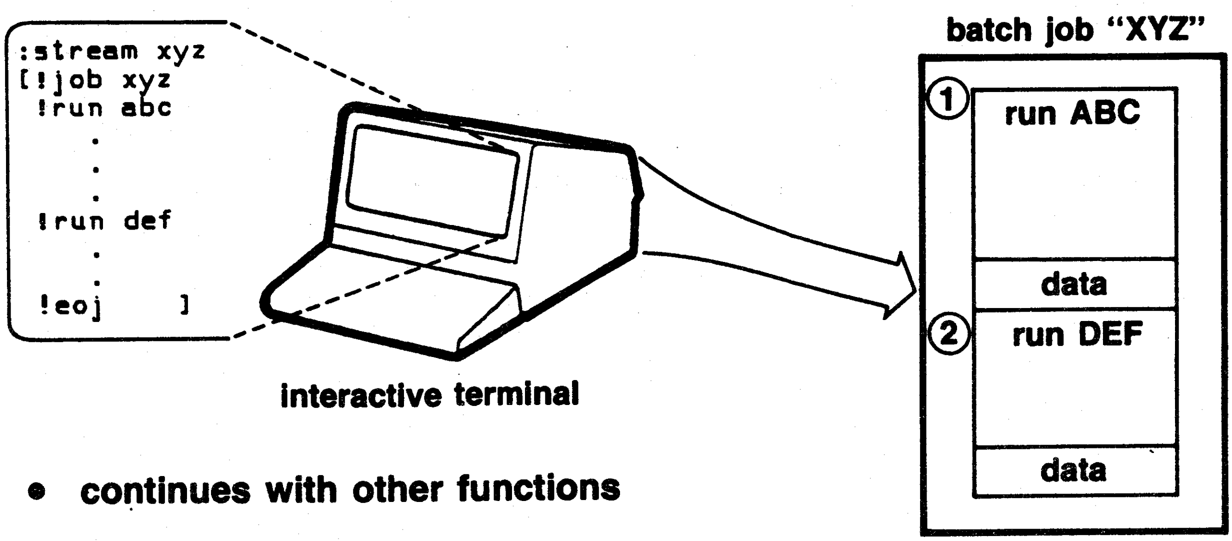
- Finally, RUN creates the process, making an entry for the process in system tables.
- Finds space in main memory for code and data.
- Executes the process.

PCB
Process Control Block

references:

STREAM capability

■ **control batch job execution from terminal**



- **continues with other functions**

- **execution sequence set in stream file**
- **no operator intervention**

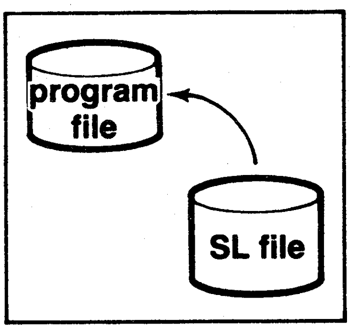
notes:

- Compiling can take time - so use STREAM for long compilations.
- Also, use STREAM to run sequences of programs.
- Allows you to perform other functions while long jobs execute in batch.
- Resolves externals from Segmented Library.

references:

ALLOCATE PROGRAM

■ when one program used frequently



allocated program

- all externals resolved
- ready to run
- only needs:
  memory space
  disc space for stack

notes:

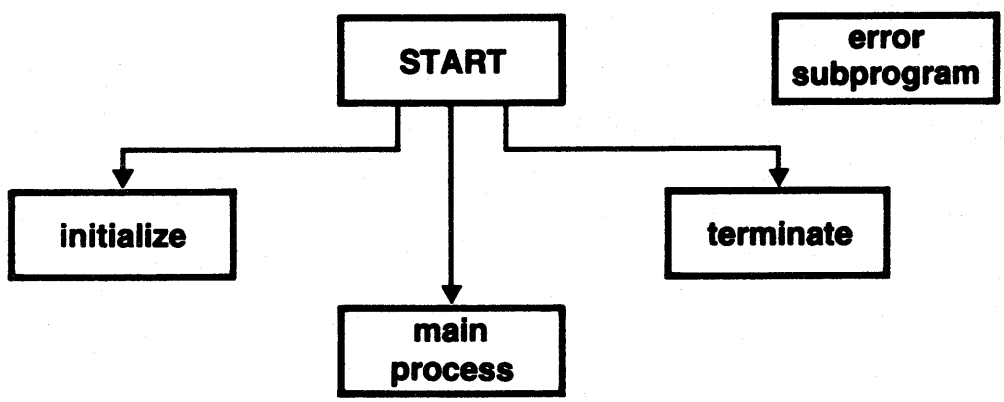
- Allocate uses resources, so don't allocate many programs.
- This is particularly useful, to save some RUN overhead, when one program is run frequently.

"Out of DST on this — solution: deallocate the program

references:

SAMPLE PROGRAM

problem: program to retrieve order information by order number

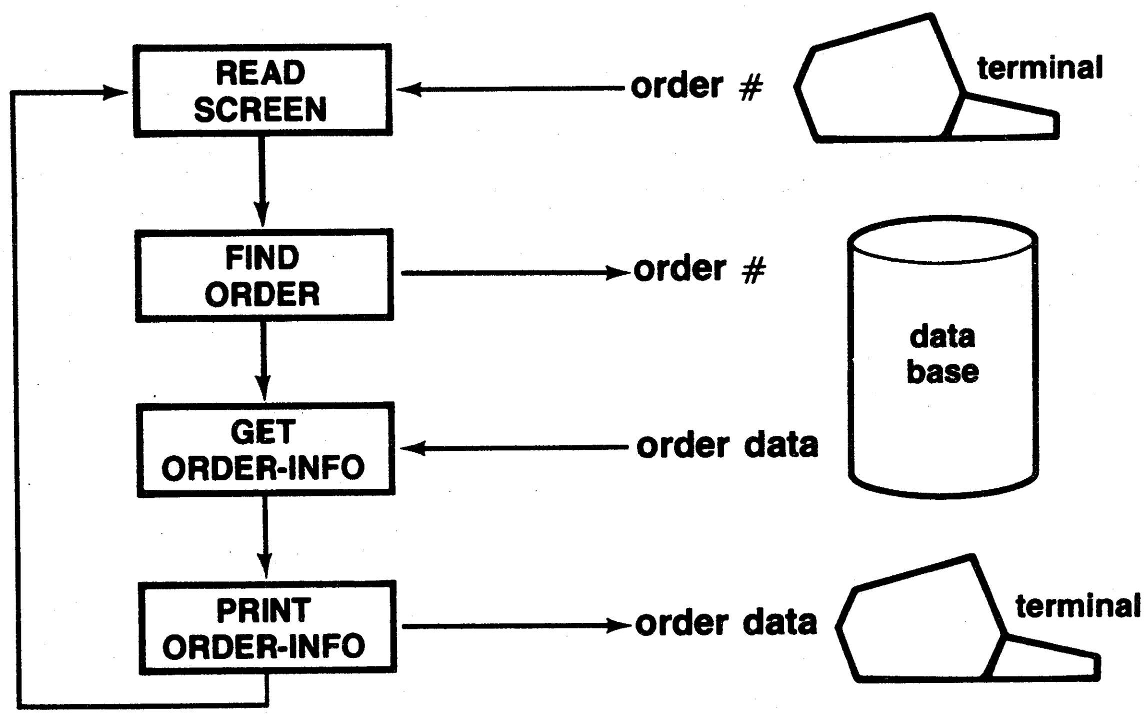


notes:

- See appendix A for source code listing of this sample program.

references:

Flow of MAIN process (simplified)



notes:

references:

How Program is Segmented

A100-START	initialize perform MAIN until done terminate	1st segment
C100-INIT	open data base forms file terminal	2nd segment
D100-TERM	close data base forms file terminal	
B100-EXIT	exit	
E100-MAIN	read screen (order #) find order in data base get order information print information	3rd segment
error subprogram	error messages	4th segment

notes:

- Look for RBM boundaries in code.
- Are these the same as the segment boundaries?
- Look at PMAP produced by segmenter (PREP stage) for final segment boundaries.

references:

DYNAMIC ERROR SUBPROGRAM

- puts error messages in local area of stack
- reduces global stack size

notes:

- A separate, dynamic code segment contains all error messages. This saves permanent (global) stack space.

references:

sample program

STREAM FILE

- allows concurrent processing during compile and prep

```
!JOB MGR.DESIGN
!PURGE PDEMO1P
!PURGE UDEMO1U
!FILE COBTEXT=SDEMO1S        <-- source file
!FILE COBUSL=UDEMO1U         <-- USL file
!FILE COBLIST=$NULL
!RUN COBOLII.PUB.SYS;PARM=5
!FILE COBTEXT=ERRORSUB
!RUN COBOLII.PUB.SYS;PARM=5
!PREP UDEMO1U, PDEMO1P;MAXDATA=11000;PMAP
!SAVE PDEMO1P                <-- program file
!EOJ
```


notes:

- Identify the source file, USL file, program file.

references: MPE Commands Reference Manual

DEMONSTRATION

notes:

1. Log on
2. Run "PDEMO1P" (the program file)
3. Enter one of the following 8-digit order numbers:

   12340010
   12340015
   12340020
   12340025
   12340030
   12340035
   12340040
   12340045
   12340050
   12340055

references:

LIBRARIES

- using code libraries

  RL — Relocatable Library

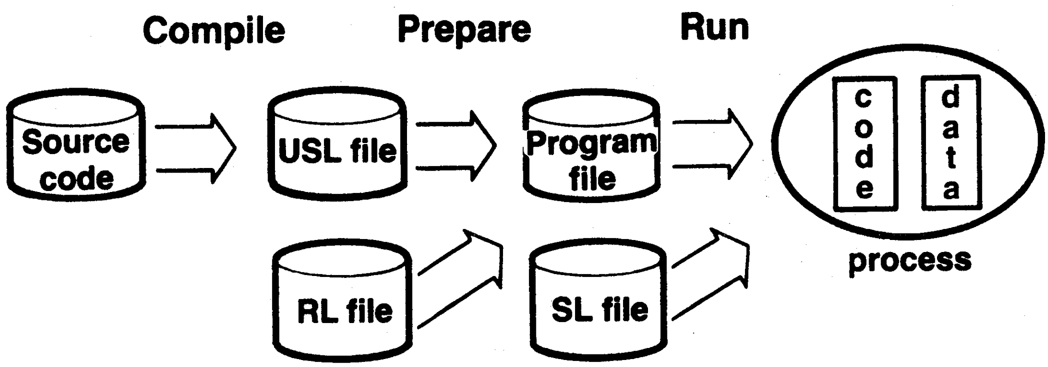
  SL — Segmented Library

- all libraries are created and managed by the segmenter

notes:

references:

AN OVERVIEW



- RLs part of program file
- SLs part of process

notes:

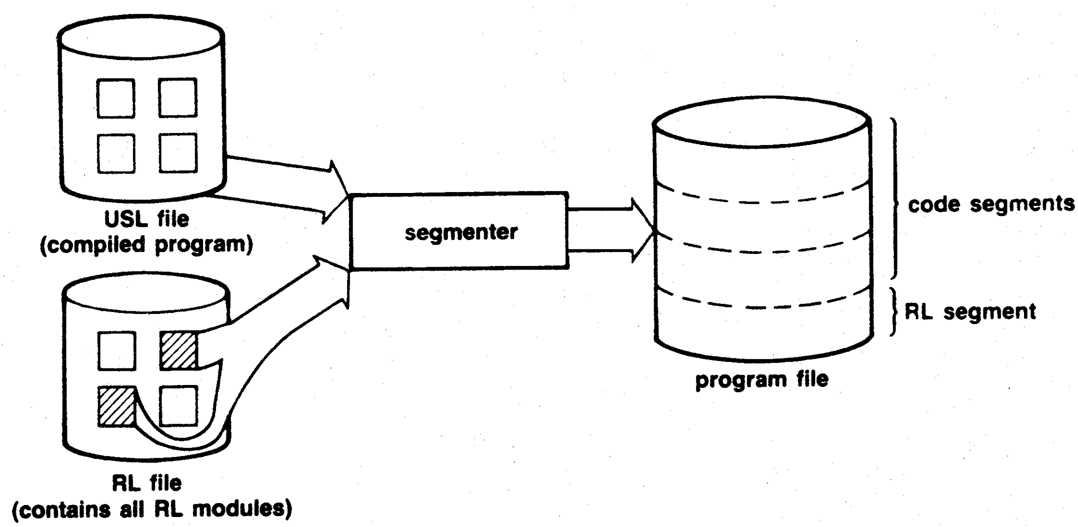
- Each RL is physically part of the program file that references it.

- An SL is simply "linked" to the executing process that references it; that is, it is brought into memory and linkages established to it.

references:

RELOCATABLE LIBRARY

- linked at PREP time



notes:

- A Relocatable Library file is made up of compiled units (RBMs) just like the USL file. These units are not yet segmented.

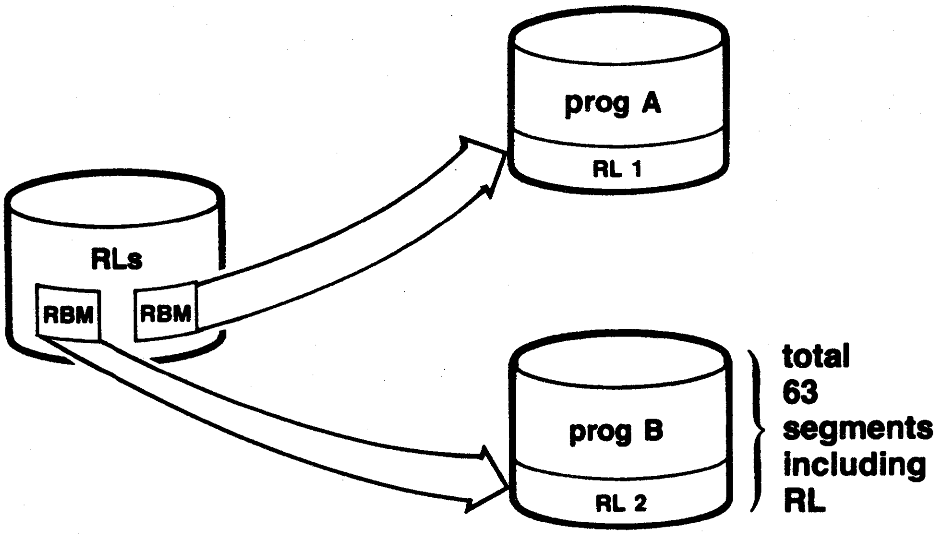
- The PREP command calls the segmenter to join RL units to program file.

- Only the RL units referenced by the program are copied to the program file. All the RL units are placed in 1 segment.

references:

use RLs

- for routines private to different programs
- for small routines
- for routines that seldom change



notes:

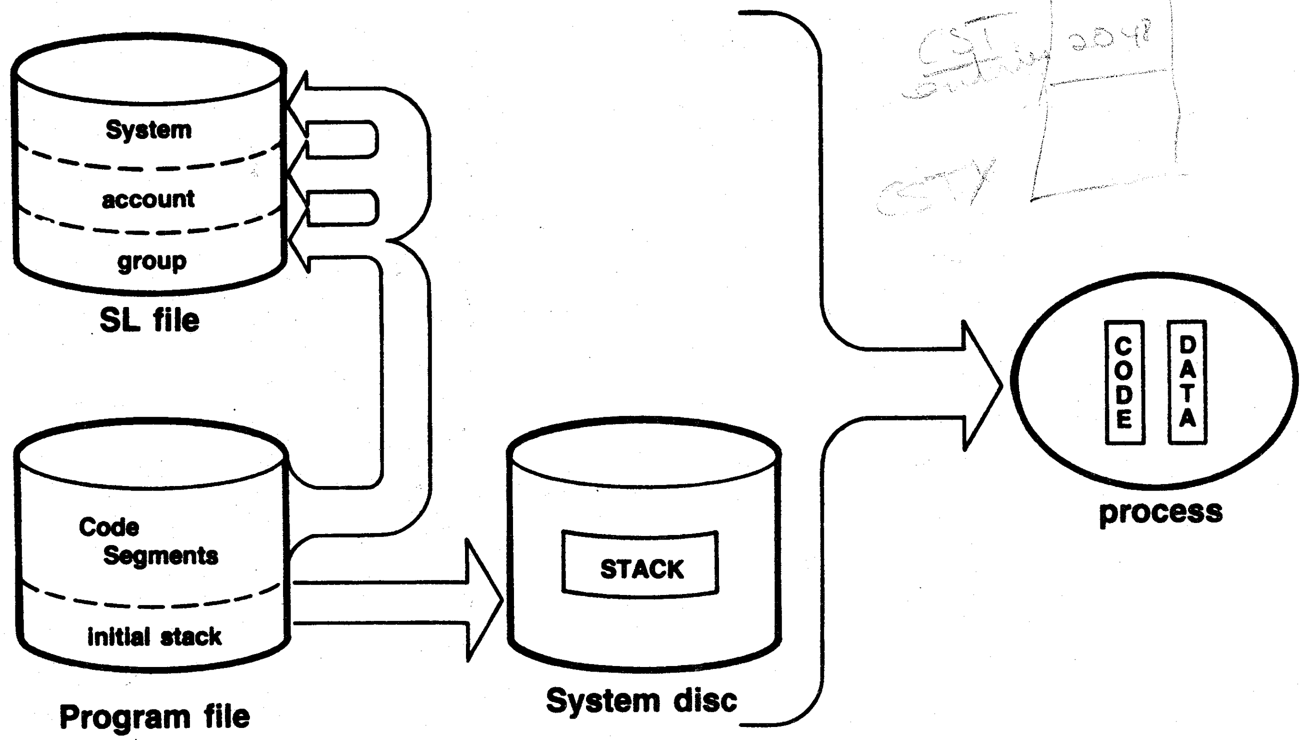
- There is a copy of the RL in every program that references that RL.
- When an RL is changed, the program file must be re-prepared.
- RLs are very useful during program development to keep different versions of code. PREP whichever RL you want into the program file for testing.

references:

using libraries

SEGMENTED LIBRARY

- **linked at RUN time**



notes:

occupy local area of stack

- SLs, unlike RLs, are already segments.

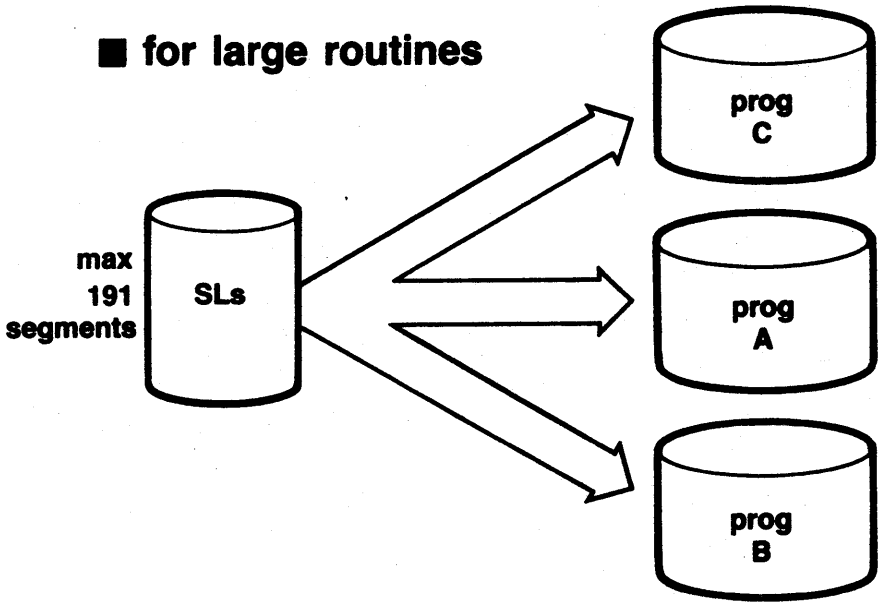
- There are three parts to the file of Segmented Library routines. The system SL is checked automatically for any referenced routines at run time; you must specifically request RUN to look for account and group SL's. (A lot of system code used by applications is kept in the system SL.)

- SLs cannot modify data in the initial global stack because that part of the stack is already established in each program file when the SLs are linked to the program.

references:

use SLs

- for routines common to many programs
- for routines that may change
- for large routines



notes:

- SLs can be modified without affecting the program file.
- Only one copy of each SL is needed, however many programs reference it.
- Each SL requires an entry in the CST (code segment table) which can have a maximum of 191 entries.

references:

WORKSESSION II-7

notes:

references:

Worksession II-7 (using libraries)

1. Given the following routines, decide whether you would put them in an RL or an SL.

   A. A routine to perform a large, complex mathematical function, such as random number generation, that is used by several programs in your application.

   RL ____________ or SL ___X___

   Why? large size

   B. Two small routines that determine the current location of the stack limit (Z) and then shrink (or expand) the stack to that limit.

   RL ___X___ or SL ____________

   Why? ____________

   C. A routine to reformat some data used by your main program (not by a procedure or dynamic subprogram).

   RL ___X___ or SL ____________

   Why? ____________

2. If your program has 62 code segments, would you add an RL?

   Yes ____________ or No ___X___

   Explain: limit is 63

3. If the segmented library contains 190 SL segments, would you put any code into a new SL segment?

   Yes ___X___ or No ____________

   Explain: limit is 191

Worksession II-7 (cont.)

4. If your program already has a large number of ~~RL~~ RBM segments, would you add more?

   Yes ______ or No __X__

   Explain: [illegible] 16 K limitation

   ____

   ____

5. If many programs will share a library routine, would you put the routine in an RL or SL?

   RL ______ or SL __X__

   Explain: ______________________________

   ____

   ____

6. If the routine is subject to frequent modification, would you put it in an RL or SL?

   RL ______ or SL __X__

   Explain: ______________________________

   ____

   ____

MULTIPROGRAMMING

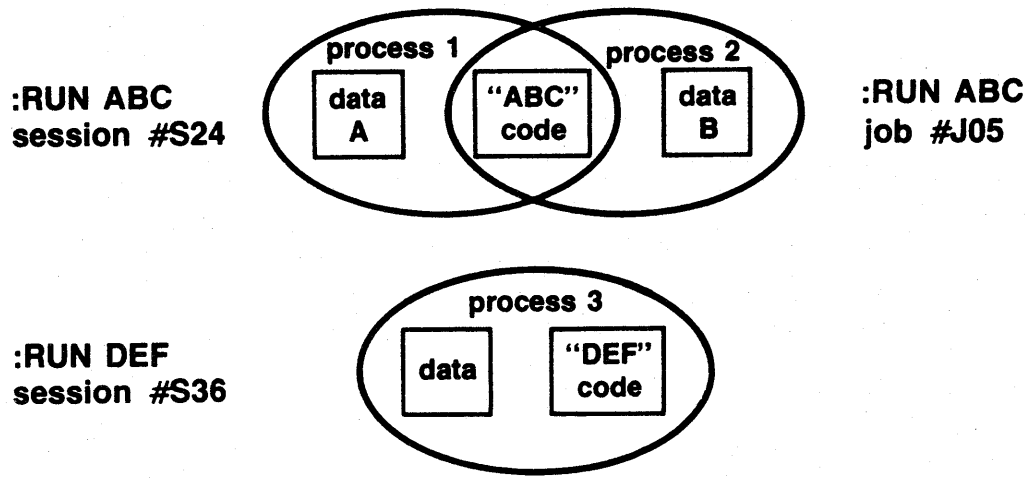
- General Considerations
- MPE Process Management

notes:

references:

MULTIPROGRAMMING

- multiple programs executing at the same time
- multiple processes (same program) executing at the same time



- how are they all managed?

notes:

references:

PROCESS EXECUTION

suppose 3 processes all start at the same time:

1. :RUN ABC . . .
2. :RUN ABC . .
3. :RUN XYZ . . .

execution SEEMS simultaneous

execution ACTUALLY sequential

on the HP3000, only 1 process executes at a time

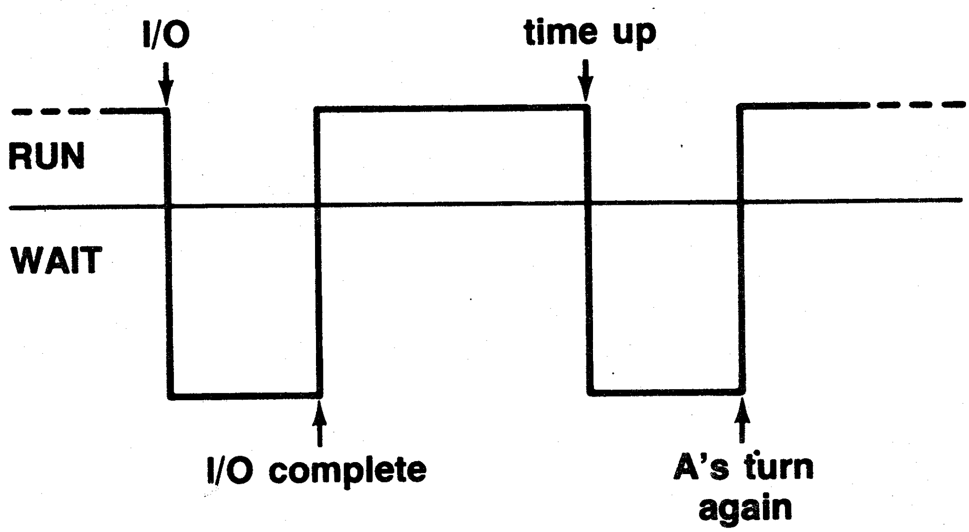
notes:

- While one process executes, other processes that seem to be executing are actually waiting.

references:

EXAMPLE: EXECUTING PROCESS

- while process A executes, other processes WAIT
- process waits for I/O or for time-out, etc.

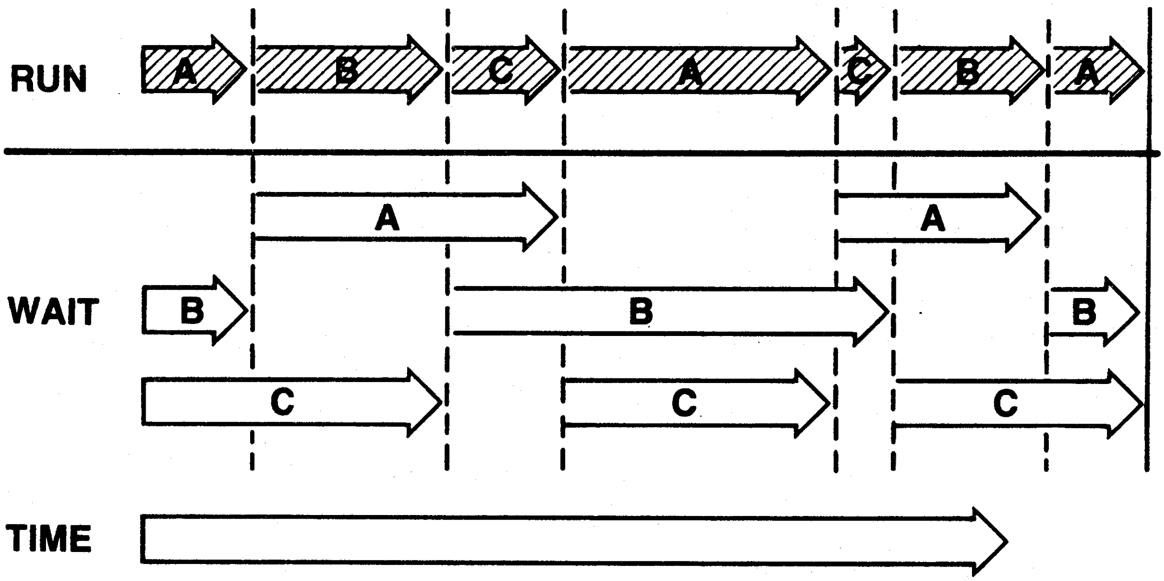


notes:

- Processes wait in suspended state either because they are waiting for I/O to complete or because their allocated "time slice" is up.

references:

EXAMPLE: MULTIPLE PROCESSES



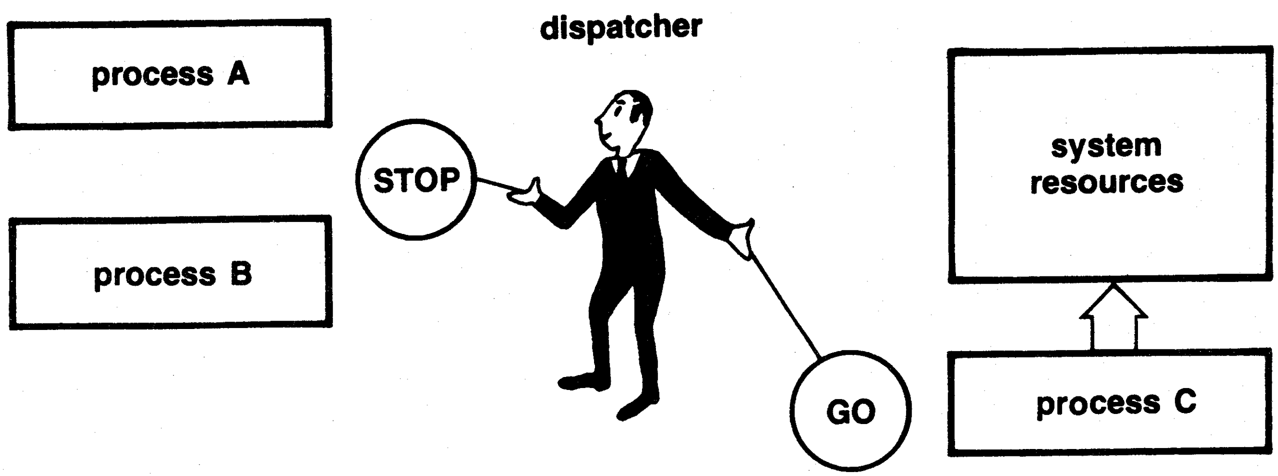
- what determines which process executes?

notes:

references:

Scheduling and Memory Management

- parallel functions to determine which process executes next



HEWLETT PACKARD

notes:

- The Dispatcher is a program permanently in main memory.

references:

the Next Process

selected for execution has:

- **highest priority** – in dynamic queue where priority changes as processes execute

and

- **is ready** – has all resources (except memory)
  – is not waiting for I/O

notes:

references:

Dynamic Scheduling Queues

3 Queues: **high priority**

C-Queue –
for terminal transactions

D-Queue
for batch transactions

E-Queue
for overnight transactions

low priority

HEWLETT PACKARD

notes:

- In the C subqueue, the system constantly recalculates the average time to execute a process, then raises or lowers the process' priority accordingly.

- In the D or E subqueues, average can be specified by system manager.

- The aim of priority management is to favor short transactions.

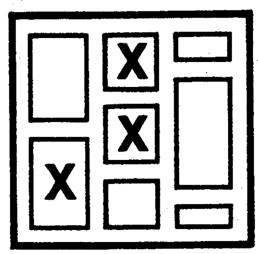
references:

Finding Memory for Next Process

MPE:

- looks for existing free space in memory (including segments marked for overlay)
- if not enough, begins marking segments for overlay

  - mark segment for overlay
  - rearrange free space
  - is there enough space now?
  - if not



notes:

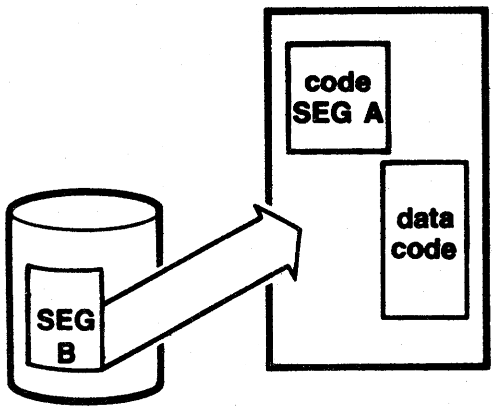
- Data stacks are selected for overlay before code segments because they are usually larger; data must be written back to virtual memory before it can be overlayed in main memory.

- System code, as well as user code, may be overlayed; everything (except the Dispatcher) is fair game.

- What happens if there are no more segments that can be overlayed, and the current code and data still won't fit? The process can't execute!

references:

Finding Memory for Next Segment



- code in SEG A calls procedure in SEG B
- if SEG B is in memory, all is fine
- if not, find space for SEG B
- same procedure as finding initial space for process

notes:

- When an executing process needs MORE memory, MPE goes through the same procedure it used to find the initial memory for the process.
- In this example, if segment "SEG A" must be overlayed to find room for "SEG B", and "SEG B" returns quickly to "SEG A", the resulting disc transfers can significantly affect performance.

references:

WORKSESSION II-8

notes:

references:

Worksession II-8 (multiprogramming)

1. A. When two users run the same program at the same time, is program execution simultaneous? Explain.

   B. If the two users run different programs, do the two programs execute simultaneously? Explain.

2. Consider the following list of program functions, and decide which queue (C, D, or E) to put them in:

   A. A program run weekly at night to print paychecks on the line printer.

   B. A program to process user requests at a terminal for confirmation of airline reservations.

   C. A program run as a batch job to update a data base from a transaction file.

   D. A program that accepts data from a terminal and writes it to a file.

3. Is there anything you can do as a program designer to help the memory manager find space in main memory for your program?

SUMMARY

- the process is the basic operating unit
- for efficient processing:
  - segment code efficiently
  - keep data stack small
  - consider the other processes

notes:

references:

TRANSACTION PROCESSING

- Definition
- Accounting and Security
- Transaction Processing Options
  - — Process Handling
- Language Considerations
- Data Entry Techniques
  - — user control
  - — V/3000 control/design

notes:

references: System Manager's Reference Manual
MPE Intrinsics Refernce Manual
V/3000 Reference Manual
Individual language reference manuals

DEFINITIONS

- **TRANSACTION PROCESSING** – any interaction between a computer system and its users

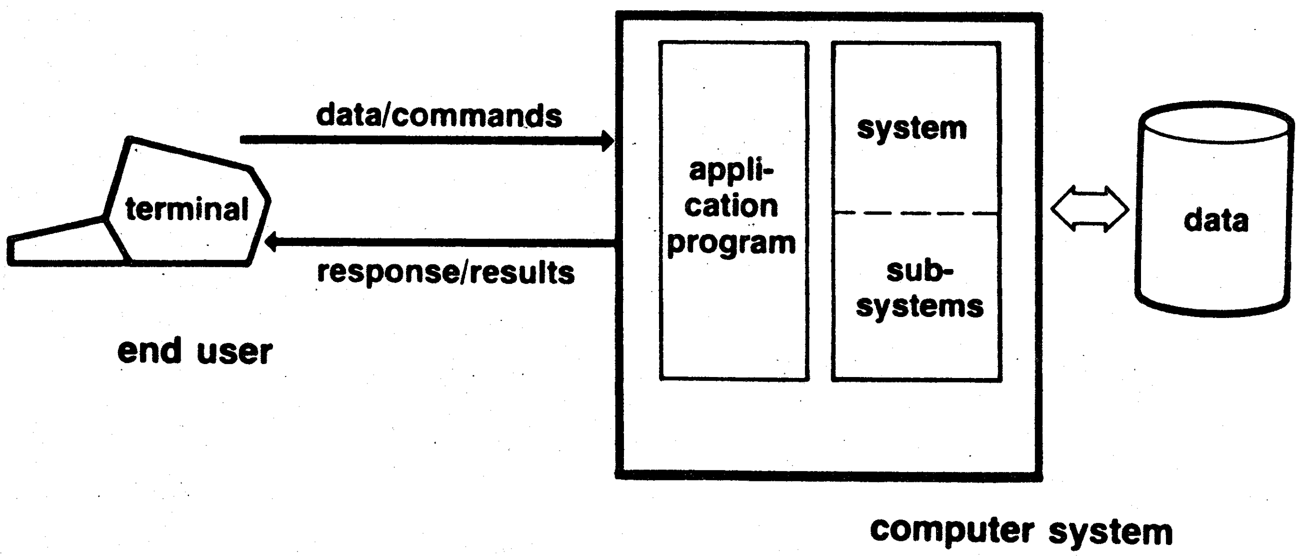
- **TRANSACTION** – the smallest useful unit of work, performed by the computer, and defined by the user

notes:

references:

Interactive Transaction Processing System

- provides terminal users, connected directly with computer system, with access to information stored in computer's data base and files.



notes:

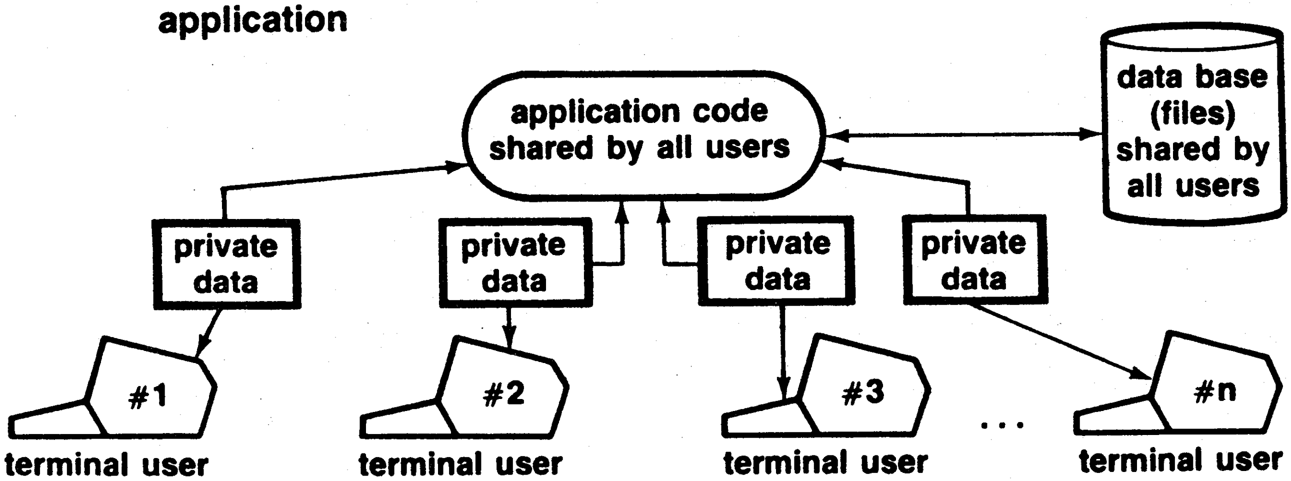
- The end user is directly responsible for the transaction; he/she is not a data processing professional.

- The terminal provides the interface between this user and the computer.

references:

An Interactive Transaction Processing system should provide:

- communication through terminals with computer system by relatively large number of users
- ability to handle uneven processing load with heavy terminal and disc I/O demands
- sharable code and separate data for all users of a particular application



notes:

references:

Interactive Transaction Processing System

Advantages include:

- interaction where decisions are made — people most familiar with data, enter it, interact with it, receive reports on it

- speeds up business cycle — data is entered, corrected, and retrieved where it is used — no more waiting for the computer center

- users see it as their system — more chance for success

notes:

references:

WORKSESSION III-1

notes:

references:

Worksession III-1 (transaction processing)

1. Define a "transaction".

2. Describe at least one advantage of an interactive transaction processing system.

   OR describe one disadvantage of a batch system.

ACCOUNTING and SECURITY

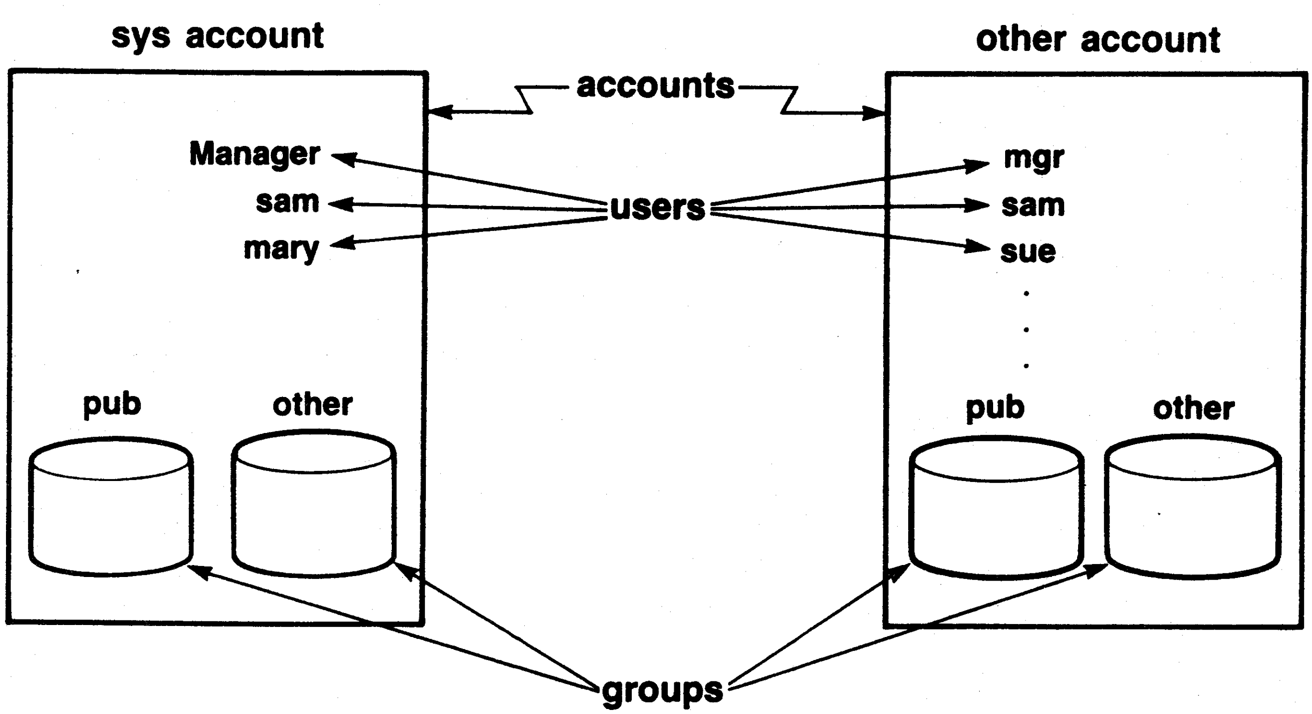
- Accounting Structure
- File Security

notes:

references:

accounting structure

MPE Account Structure



HEWLETT PACKARD

notes:

- One system-wide account available to all (SYS)
- One public group in each account for all account users (PUB)

references:

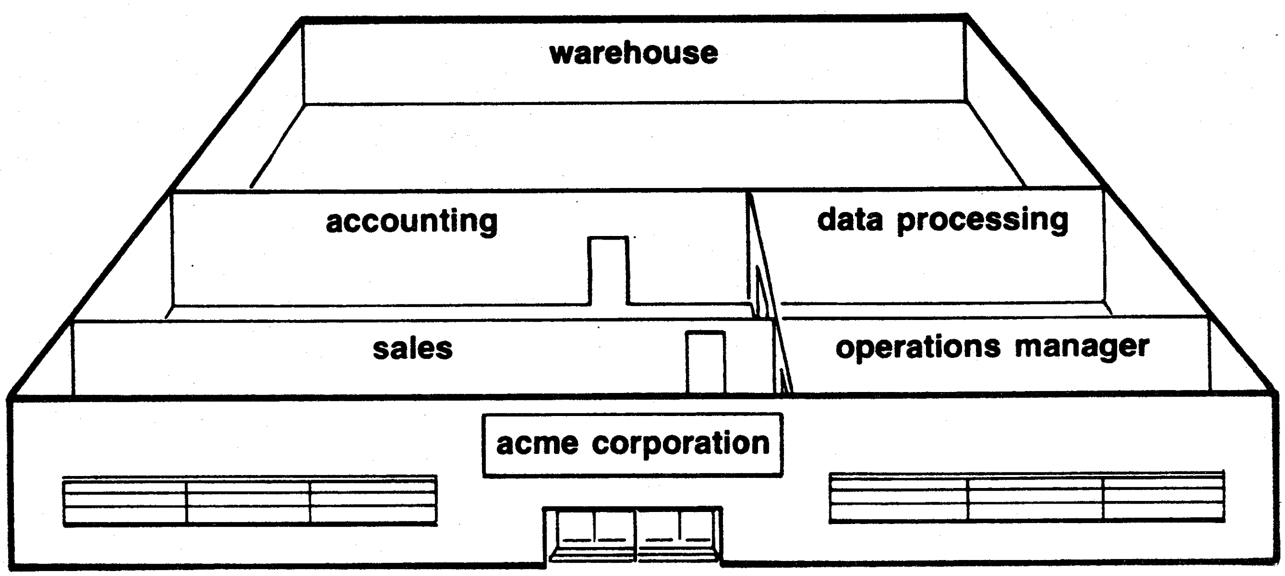
PURPOSE

- protect application and data from unauthorized access
- allow user to do only what he or she needs to do

notes:

references:

EXAMPLE: ACME CORPORATION



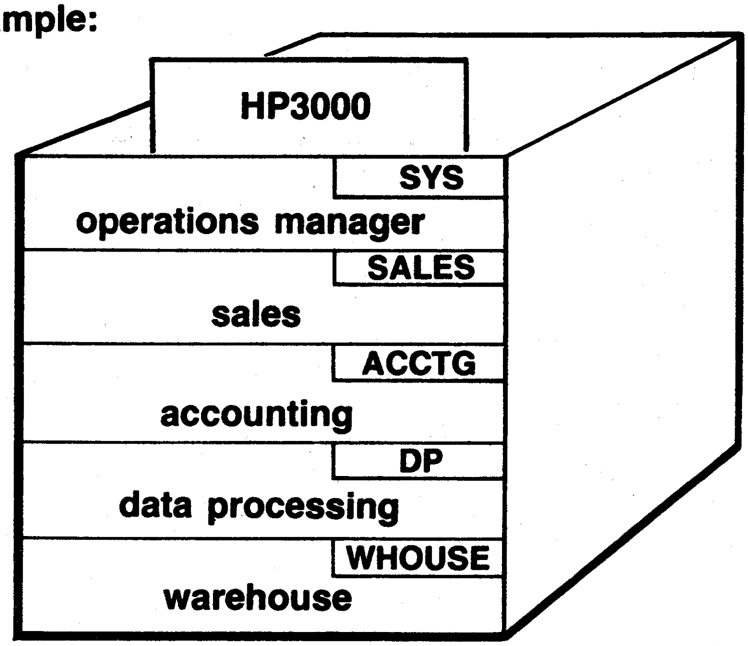
notes:

references:

ACCOUNTS

- should reflect corporate structure

example:



notes:

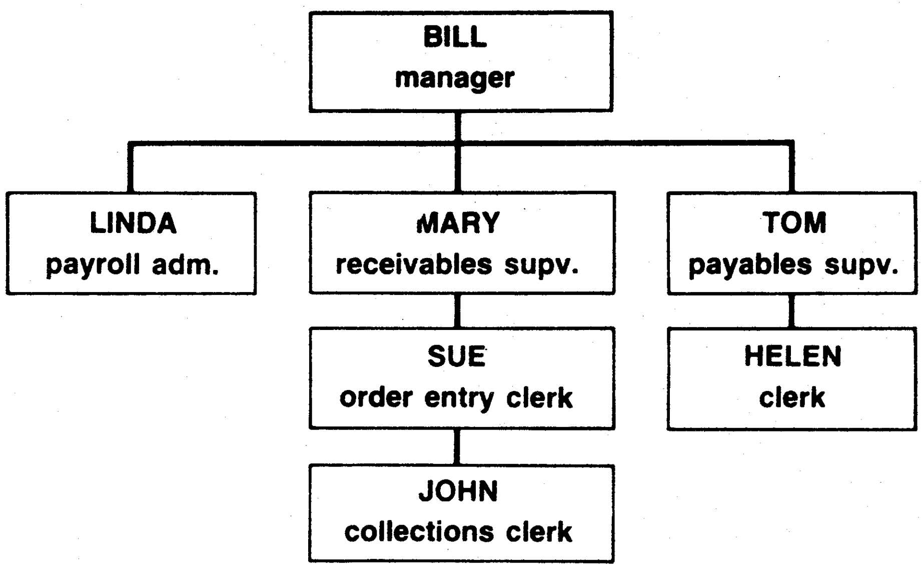
references:

USERS and GROUPS

- consider organization of department

example:

**Acme Corp.
Accounting Dept.**



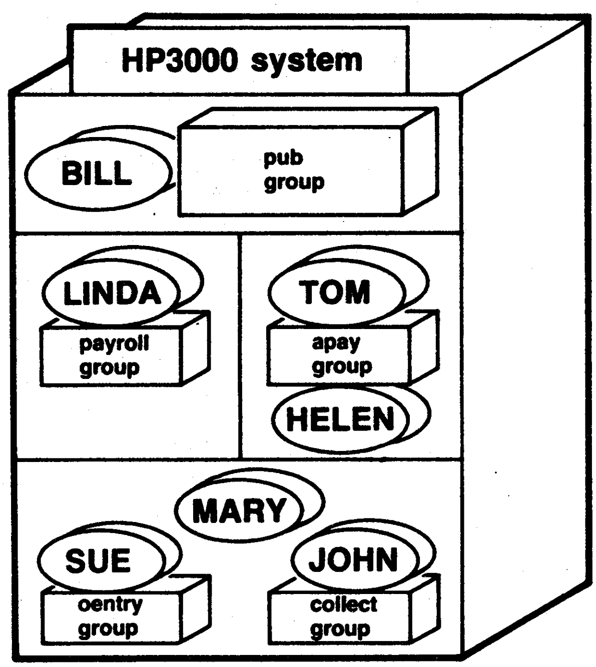
notes:

references:

USERS/GROUPS

- users and groups reflect organization/needs

example:



notes:

references:

CAPABILITIES and RESOURCES

- capabilities associated with USERS
- resources associated with GROUPS

USERS

Capabilities

- apply security
- manage files
- process handling
- privileged mode

GROUPS

Resources

- files
- disc usage
- cpu time

notes:

references:

security

level 1	
account	— password
user	— password
group	— password
file	— lockword

level 2	
account	— capability limits
user	— capability limits
group	— resource limits
file	— access protection

notes:

- First level provides absolute privacy; it is applied to each level of accounting structure, plus files.

- Second level controls and monitors system use.

references:

FIRST LEVEL OF SECURITY

```
:hello mary.acctg,collect         :run myfile.collect.acctg
 account password? ________        file lockword? ________
 user password? ________
 group password? ________
```

MPE command interpreter

account = acctg

group = collect

user = mary

file = myfile

HEWLETT PACKARD

notes:

- Command interpreter tests for both log-on security and file security. It prevents unauthorized users from logging on, using files.

references:

SECOND LEVEL OF SECURITY

ACCOUNT = ACCTG

GROUP = COLLECT

FILE = CUSTOMER
- lockword = secret
- creator = Bill

ACCESS;
- read – any user
- write – creator only (Bill)

FILE = INVOICES
- lockword = stayout
- creator = John

ACCESS;
- read – group user
- write – creator, group librarian

USER = MARY

CAPABILITIES:
- group librarian
- interactive

notes:

- After passing first level tests, access to files and types of system use is controlled at second level of security.

references:

TIPS

- avoid too much during development mode
  - can be a nuisance

- add passwords, lockwords after program developed and tested

- use full security during production to protect data, control access

notes:

references:

WORKSESSION III-2

notes:

references:

Worksession III-2 (accounting structure)

Given the ACCTG account structure shown in the preceding slides, answer the following questions:

1. A. What must the user MARY do in order to read the file INVOICES in the group COLLECT of account ACCTG? Explain.

   B. Does MARY need to do anything more in order to modify the file INVOICES? Explain.

2. A. Can a user in the account SALES read the CUSTOMER file in the COLLECT group of ACCTG? Explain your answer.

   B. Can this same user in account SALES modify the file CUSTOMER? Explain your answer.

Worksession III-2 (cont.)

3. Can the user in account SALES read the file INVOICES in the COLLECT group of account ACCTG? Explain.

4. Given: a program file CUSTINV in group COLLECT of ACCTG that allows execution access to group users. Can a user in group OENTRY of ACCTG run this program? Explain your answer.

TRANSACTION PROCESSING OPTIONS

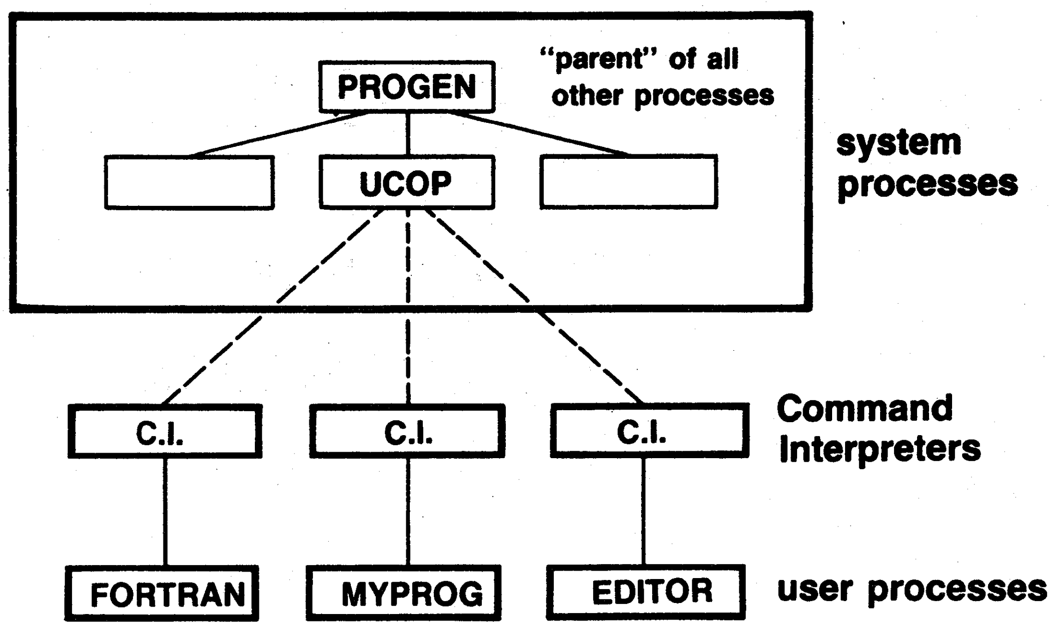
- SESSION MODE
- PROCESS HANDLING
- OTHER OPTIONS

notes:

references:

options

PROCESS TREE



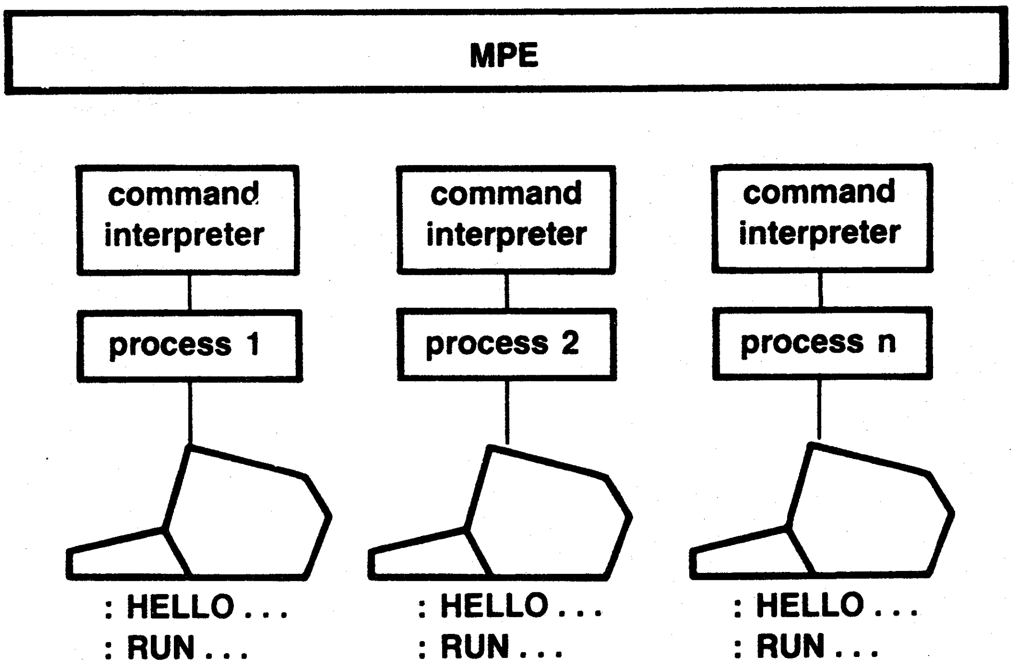
notes:

- Each user process is part of a process tree originating with the first process "PROGEN", and with a separate command interpreter as its parent.

- User processes can themselves be parents of other user processes. (More on this soon.)

references:

1 PROCESS PER TERMINAL — Session Mode



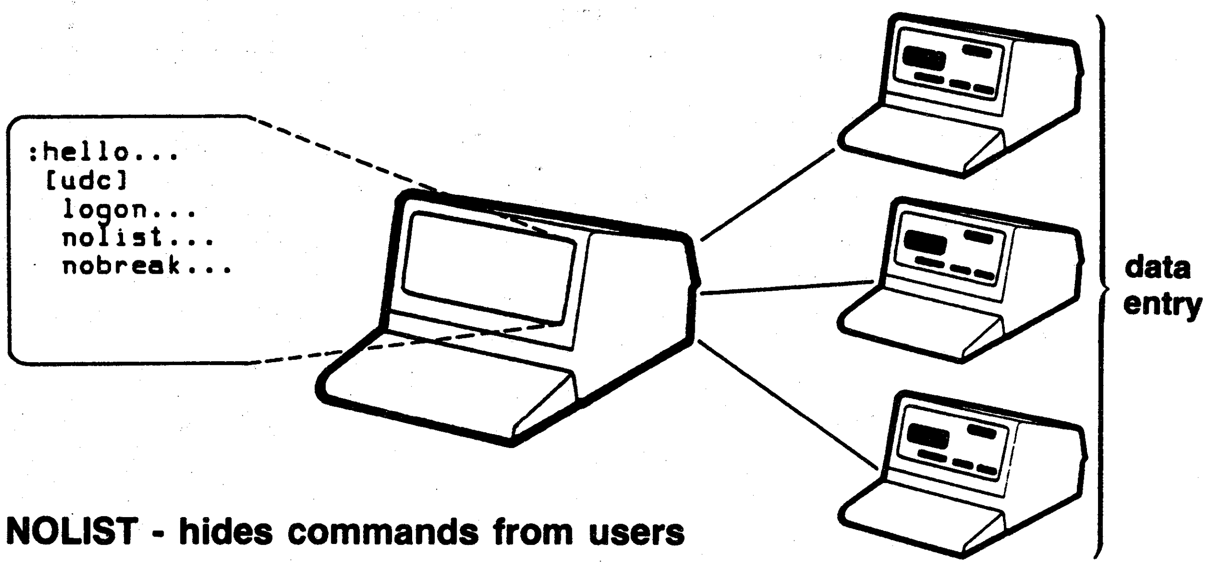
notes:

- This is the mode of operation for which MPE is designed. As a result, it is the simplest mode to develop and test.

references:

Single Process Control

- with logon UDCs, end users don't run programs



- NOLIST - hides commands from users
- NOBREAK - exit only under program control

notes:

- Such a UDC (User Defined Command) can be used to separate the end user from most contact with the operating system.
- Note the user still has to log on; but logging off can be included in the UDC.

references:

SESSION MODE

ADVANTAGES

- simple development and testing
- no special capabilities needed
- simple local terminal logic

DISADVANTAGES

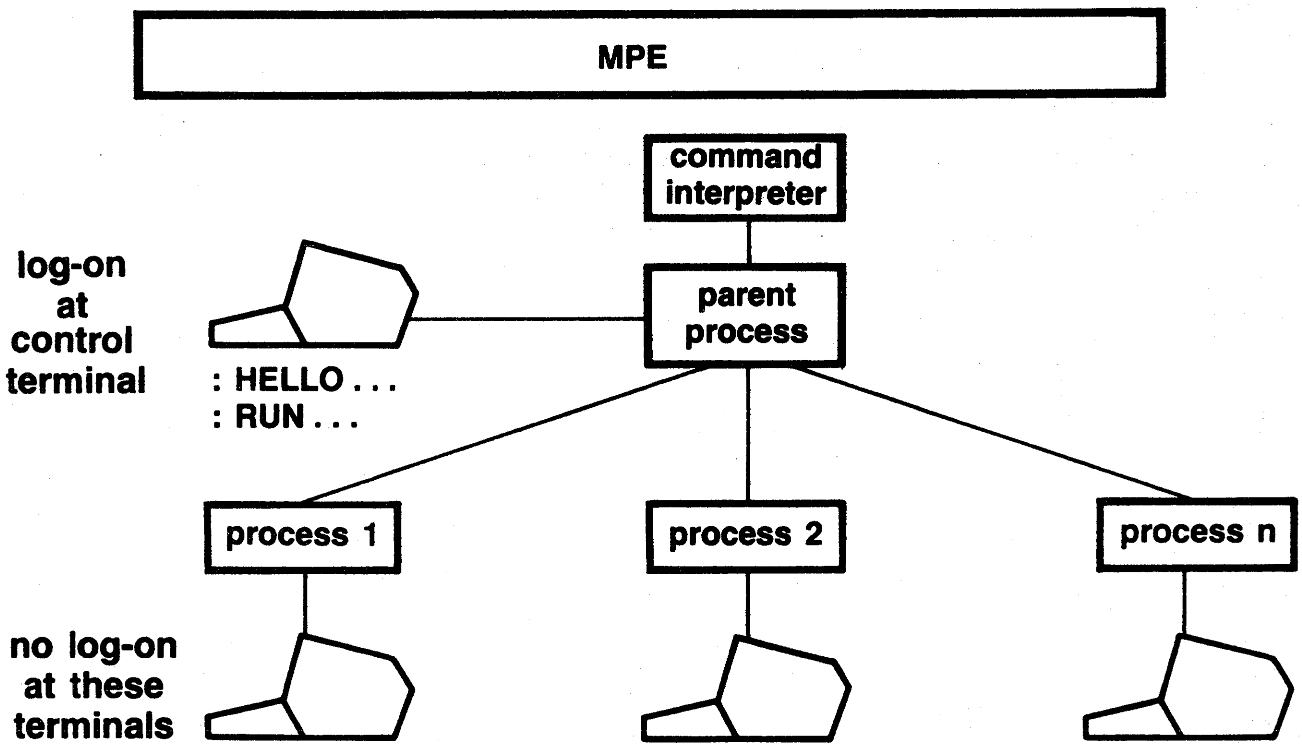
- extra overhead — each log-on, log-off requires I/O
- limited global terminal logic
- extra responsibility for end user

notes:

references:

options

1 PROCESS PER TERMINAL (Process Handling)



notes:

- With process handling, the end user can be completely separated from contact with the operating system.
- Note: There is still only one process associated with each terminal.

references:

PROCESS HANDLING

ADVANTAGES

- END-USER is isolated from MPE commands
- stack sizes are smaller; code units smaller
- session overhead reduced

DISADVANTAGES

- program testing more complex
- extra overhead for process creation
- BASIC and COBOL '68 must use SPL routines
- requires special capability, careful management

notes:

references:

SOME Other Options

- **Specialized Single Program**
  multiple applications

- **Central Terminal Control**
  multiple applications

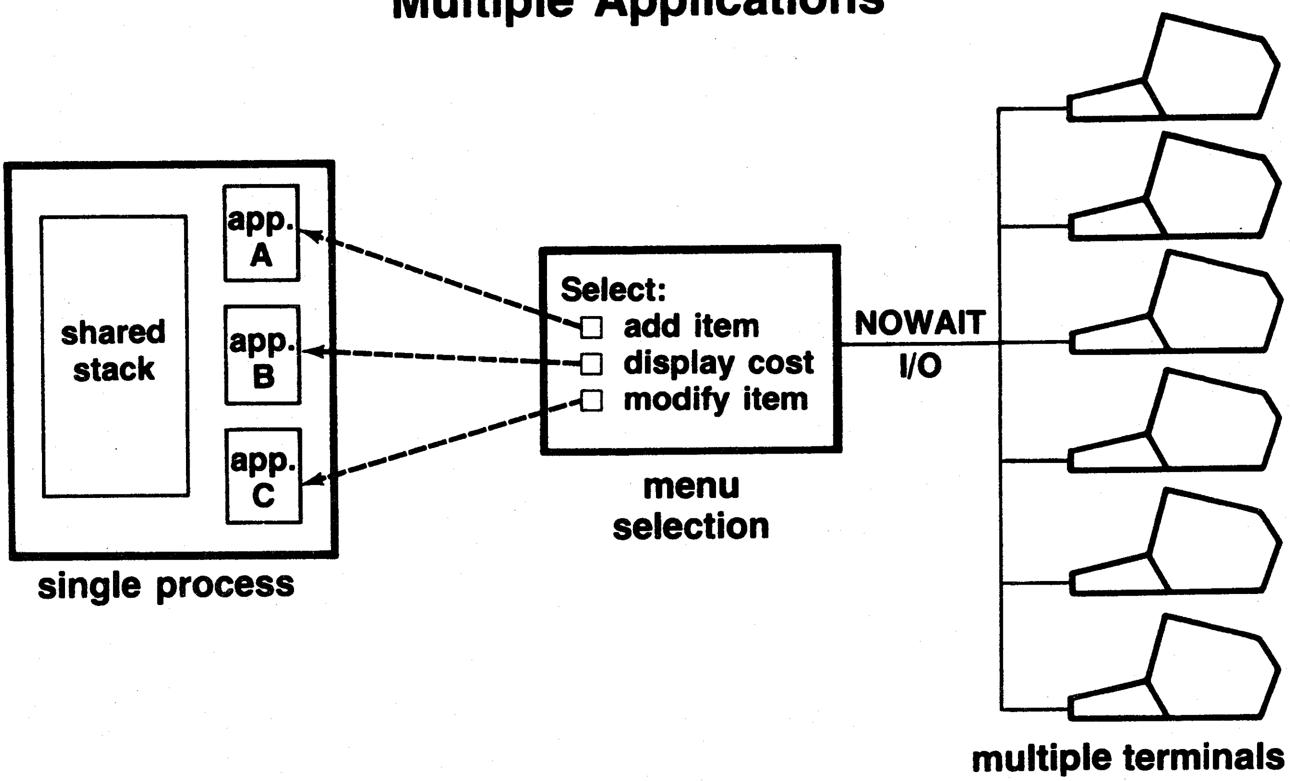
notes:

- These other options allow multiple terminals to be controlled by a single process.

references:

options

SPECIALIZED SINGLE PROGRAM
Multiple Applications



notes:

- Most interactive processes are menu-driven. The difference here is that multiple terminals select different functions "simultaneously".

- The multiple terminals are the reason NOWAIT I/O is used, but only to control the terminal I/O.

references:

ADVANTAGES

- simple inter-task communication
- shared stack
- fast transfers with NOWAIT I/O

DISADVANTAGES

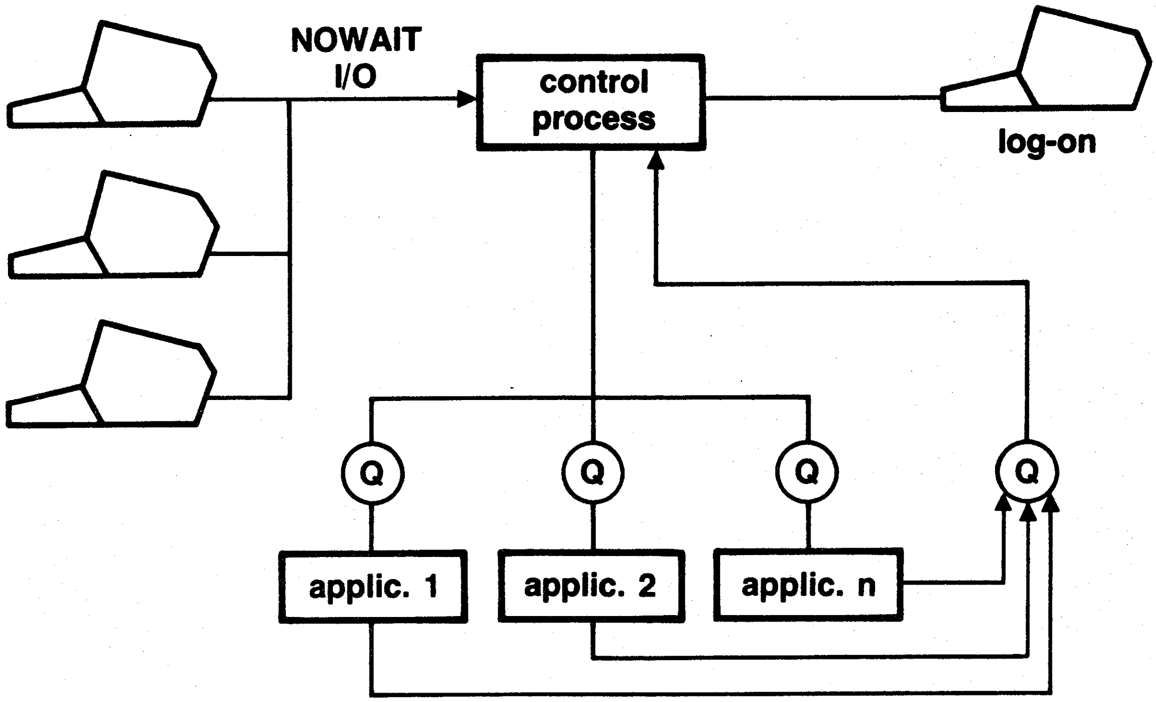
- complex task handling
- stack can be very large
- program can be very large
- NOWAIT I/O requires privileged mode

HEWLETT PACKARD

notes:

references:

CENTRAL TERMINAL CONTROL
Child Applications



notes:

- The control process communicates with the "child" applications through "Queuing" files that allow the processes to pass messages and be sure the messages are received.

- Again, multiple terminals connected to the control process require NOWAIT I/O.

references:

ADVANTAGES

- fast multi-terminal handling (NOWAIT I/O)
- central control over transactions
- individual processes allow small stacks, small segments

DISADVANTAGES

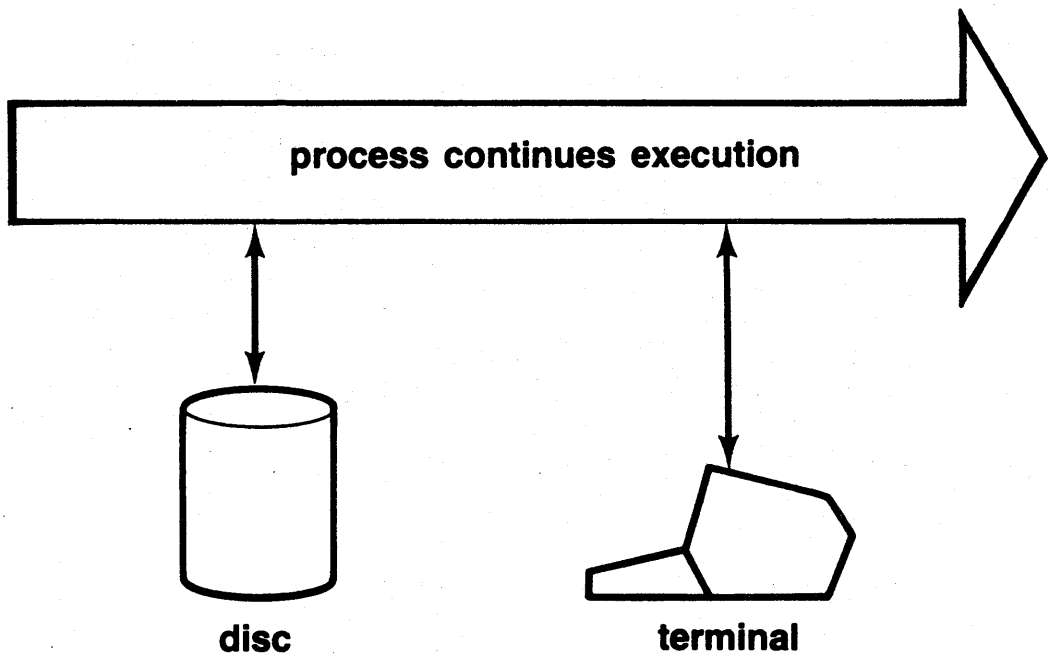
- privileged mode required for NOWAIT I/O
- more complicated programming required

notes:

references:

options

NOWAIT I/O



notes:

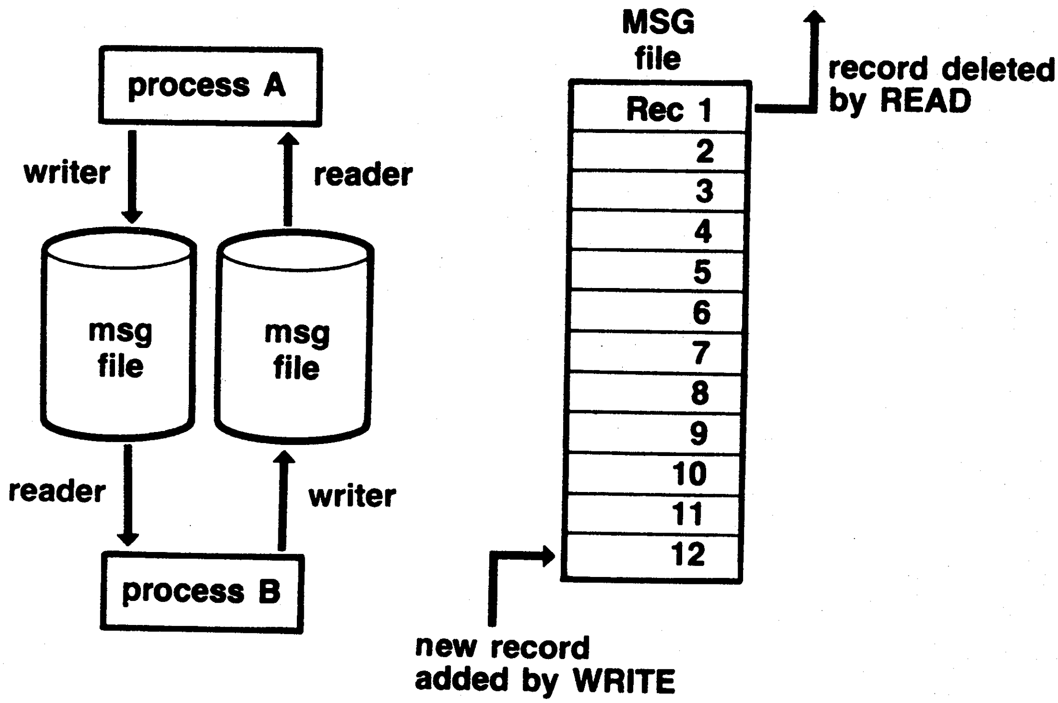
- Requires Privileged Mode to use. Because MPE also operates in Privileged Mode, using this mode allows a user to damage the system itself. So it must be used with great care, if used at all.

- NOTE: HP does not support applications that use Priviliged Mode.

references:

options

QUEUING



notes:

- Only available with MPE IV Inter Process Communication subsystem.
- Deletion of record after it is read allows writer to be sure message has been received.

references:

WORKSESSION III-3

notes:

references:

Worksession III-3 (options)

1. The "standard" MPE processing option runs one process per terminal in session mode. Give at least one advantage and one disadvantage of this option.

   Advantage(s): ______________________________

   Disadvantage(s): ______________________________

2. A. Describe one of the other options we discussed.

   B. Give one advantage, one disadvantage, of the option you described.

PROCESS HANDLING

How to do it

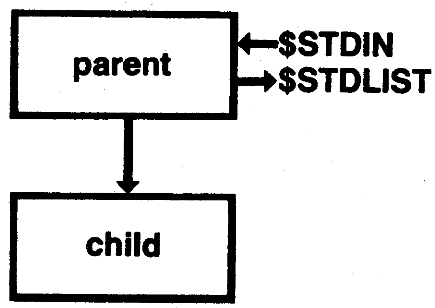
- **CREATE**
- **ACTIVE/SUSPEND**
- **TERMINATE**

Example

notes:

references:

CREATE



■ **loads program file (child) and links child to parent**

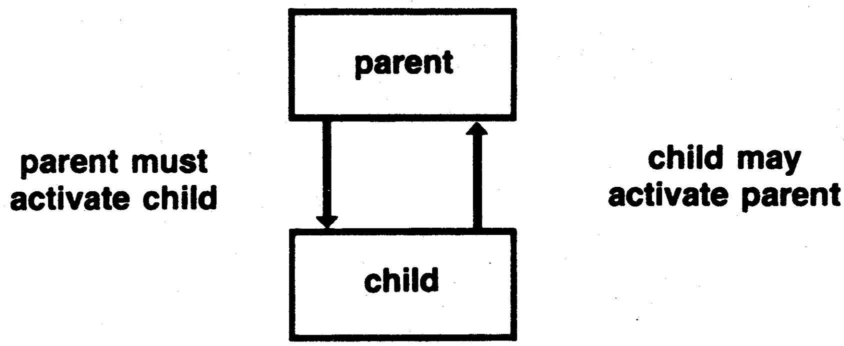
notes:

- The parent process must have Process Handling capability; the child process only needs this capability if it uses process handling procedures.

- Parent can request at create time to be reactivated when child terminates.

references:

ACTIVATE



- makes process ready to execute — process either newly created or suspended

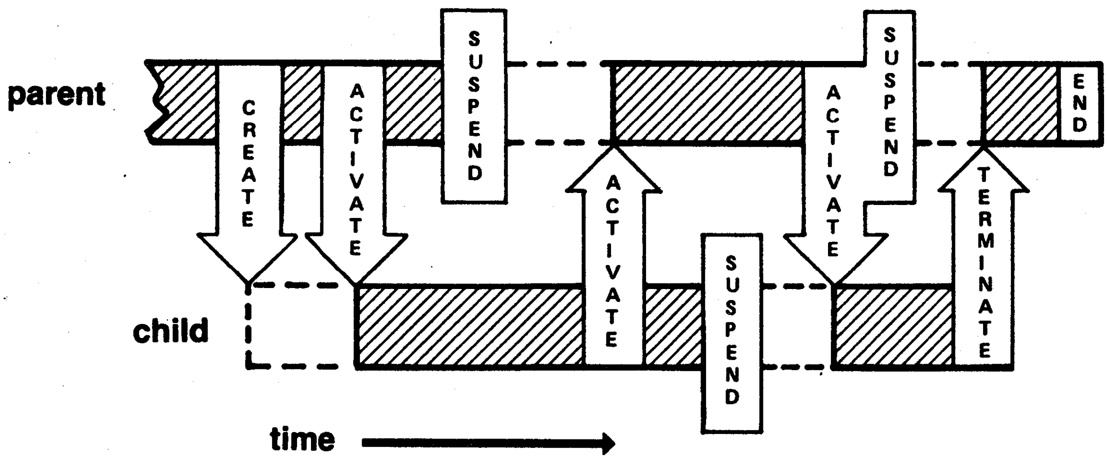
notes:

- Either parent or child can activate related process.
- Calling process may choose to suspend when activated process starts up. If it does this, the calling process must specify who will reactivate the suspended process.
- Always use checks to determine whether process is suspended or already active before activating.

references:

ACTIVATE/SUSPEND

- **PARALLEL PROCESSING — parent and child process both run**
- **SYNCHRONIZED PROCESSING — parent suspends when child active, and vice versa**



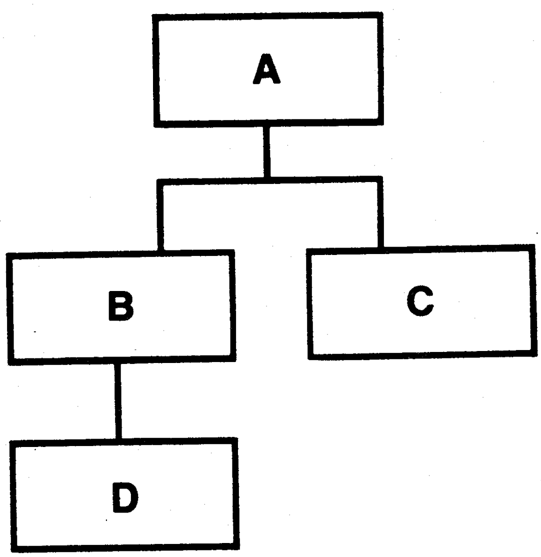
notes:

- Parallel processing is particularly dangerous since an active process cannot recognize that it is being activated. So make checks before activation.

- When child terminates, the termination reactivates the parent (now suspended).

references:

TERMINATE/KILL



- if A terminates, B, C, & D are terminated
- if B terminates, D is also terminated
- if C, or D terminates, other processes remain
- A may kill B or C
- B may kill D

notes:

- A process is said to terminate however it stops; normal program termination, abnormal termination (abort), or because it is the child of a terminated process.

references:

DEADLOCKS —

MUTUAL SUSPENDS – parent and child each suspend, wait to be reactivated by other

MISSING ACTIVATION – parent activates child and suspends, child terminates without activating parent

UNSEEN TERMINATION – parent does not see child's termination since parent was active

TO AVOID —

- check before activating
- check before terminating
- check before suspending

notes:

references:

SAMPLE PROGRAM

CONTROL PROGRAM (parent):
- **creates and activates child processes**

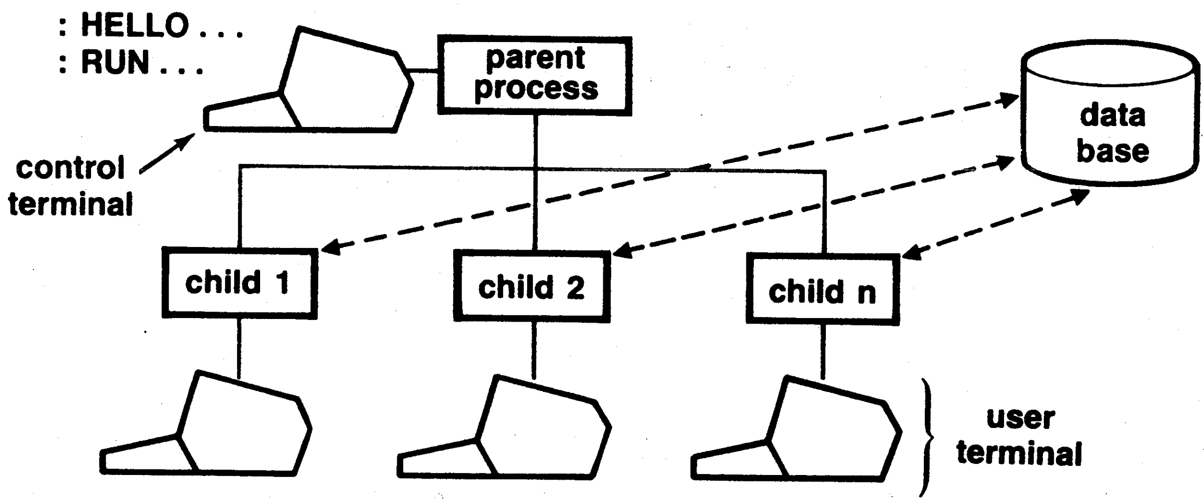
ORDER RETRIEVAL PROGRAM (child):
- **prompts user at terminal for order**
- **finds order in data base**
- **displays order at terminal**
- **prompts for next order**

notes:

- The child program is the same program used as a demonstration in Module II, with minor changes that allow it to reactivate parent and suspend.

references:

sample program



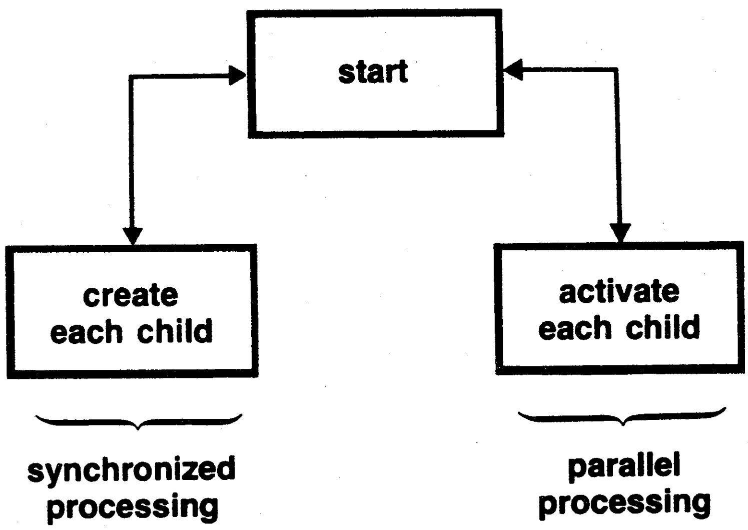
HEWLETT PACKARD

notes:

- Only the parent process must log on, run the program.
- The parent process, controlled from the control terminal, controls each child process.
- User terminals see only screens from executing child processes.

references:

PARENT PROCESS



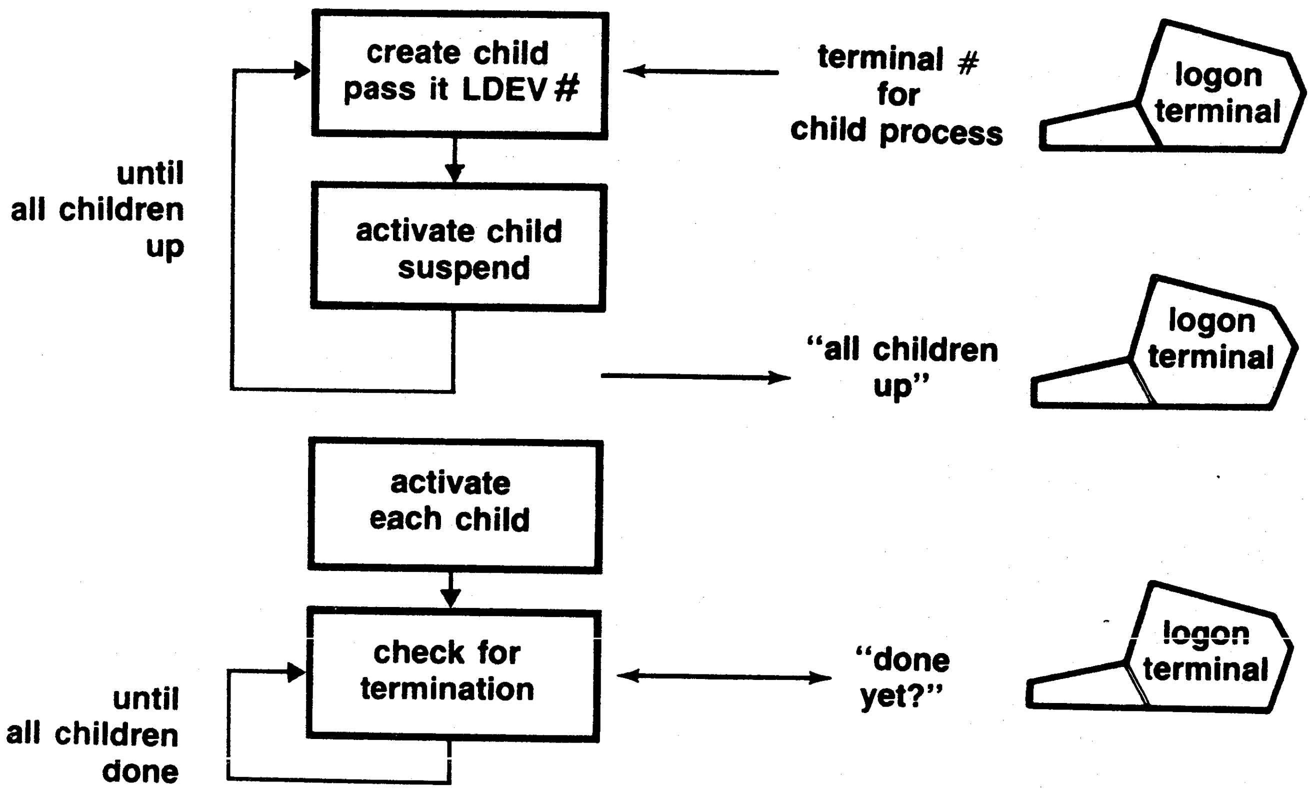
notes:

- The sample program uses both synchronized and parallel processing in order to illustrate both methods.

references:

sample program

PROCESS FLOW



notes:

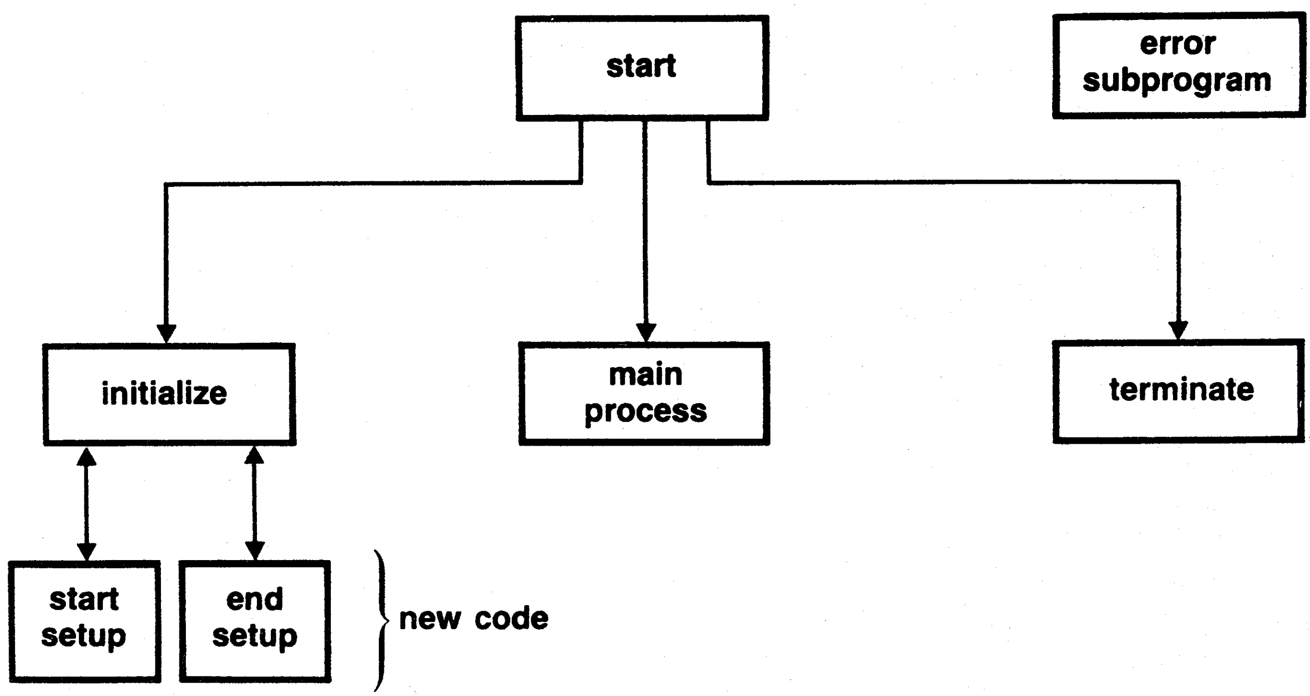
- This is the flow of the parent process.

- Note the two stages: the first stage creates each child in turn and suspends. The child then opens a user terminal, reactivates its parent and suspends. In the second stage, Parent and child processes execute in parallel until the child processes complete.

references:

sample program

CHILD PROCESS



notes:

- This is the flow of the child processes.
- Note that it is identical to the sample program in Module II except for the "Start Setup" and "End Setup" procedures.
- Look at the source code, PMAP, for this sample program in appendix A.

references:

DEMONSTRATION

notes:

- Log on
- Run the parent process, PDADP

references:

LANGUAGE CHARACTERISTICS

- COBOL
- FORTRAN
- BASIC
- RPG
- SPL
- APL

HEWLETT PACKARD

notes:

references:

COBOL

good for business data processing — better for I/O than for computation

Advantages

- widely known and used
- simple record structuring
- good data editing and formatting
- interface to system intrinsics (COBOL II)

DISADVANTAGES

- long-winded
- inefficient computation

HEWLETT PACKARD

notes:

references:

COBOL – tips on using

- compare and move equal length fields
- use signed numeric items rather than unsigned
- use COMP or COMP-3 to avoid conversion
  - 1-9 digits — use COMP (PIC S9(9) COMP)
  - 10 or more digits — use COMP-3 PIC S9(16) COMP-3)
- begin COMP items on a word boundary
- use indexing rather than subscripting
- avoid the COMPUTE statement
- keep structure out of the LINKAGE SECTION

notes:

- The HP3000 is a word-oriented system. Byte (character) boundaries are not supported by the hardware and require special handling.
- In general, these hints are all due to special ways the system works. For instance, COMP items of 9 digits or less use a fast hardware-support binary arithmetic.

references:

more COBOL tips — Dynamic vs. Static Subprograms

Dynamic —
- data is placed in local (shared) area of the stack; keeps stack small
- extra overhead because data must be re-initialized on each call
- since global area of stack is not used, can be put into SL

Static —
- data is placed in global area of stack — increases minimum stack size
- data only initialized once, on first call — less overhead
- any MPE files opened by static subprogram are available to entire program
- cannot be placed in an SL

notes:

references:

FORTRAN

good for computational applications

Advantages

- widely known and used
- efficient computations
- modular — easy to segment
- easy interface to MPE intrinsics

DISADVANTAGES

- no data structuring
- limited control structures

notes:

references:

FORTRAN TIPS

- Avoid formatted reads and writes – causes external calls
- Assign equal length fields for character manipulation
- Avoid multiple entry points to subroutine
- Don't mix data types within expression
- Avoid double integers as loop variables
- Avoid exponentiation of double precision and complex data – causes external calls
- Use EQUIVALENCE statements to redefine character data

HEWLETT PACKARD

notes:

- Again, these tips are due to the way the HP3000 works. Whenever there is hardware support in the form of firmware, execution is faster than if a compiler must make external calls to special software procedures.

references:

BASIC

good for engineering and scientific applications, and for applications that manage character strings

ADVANTAGES

- fast development through interpreter
- good string handling
- good matrix manipulation
- compile after developing and testing

DISADVANTAGES

- variable names limited to 2 characters
- awkward segmentation
- computation less efficient than SPL or FORTRAN

notes:

references:

RPG

good for business data processing in batch mode, and for report generation

ADVANTAGES

- easy conversion from other machines
- quick development

DISADVANTAGES

- inflexible program control
- inflexible file management
- minimum control over segmentation
- no subroutine capability

notes:

references:

SPL

good for computational applications, and systems programming

ADVANTAGES

- designed for use on HP3000 —
- most efficient execution
- flexible and highly modular

DISADVANTAGES

- limited data editing and formatting
- no data structuring capability

notes:

references:

SPL — Tips on using

- move words rather than bytes, whenever possible
- pass word address, not byte address, if word is called for
- when array size varies, create array dynamically

notes:

- The first two tips are another example of how to use a word-oriented machine.
- The last tip can reduce stack size - you don't want the compiler to allocate a stack based on the largest possible array size when the size is variable.

references:

APL

good for engineering and scientific applications

ADVANTAGES

- excellent array handling
- powerful operators
- quick development
- modular

DISADVANTAGES

- heavy use of system resources
- no segmentation
- cryptic

notes:

references:

WORKSESSION III-4

notes:

references:

Worksession III-4 (languages)

1. In all languages, it is important to keep word boundaries in mind when programming for the HP 3000. True or False?

   ____

   ____

2. All languages give you the capability to segment code into variable length segments. True or False?

   ____

   ____

3. Consider the following application needs:

   a) Generate a formatted report.
   b) Execute machine instructions on the HP 3000.
   c) Manipulate character strings.

   Indicate which language (or languages) you would select to perform each of these tasks. Choose from one of the following:

   COBOL FORTRAN BASIC RPG SPL APL

   a. __________ why? ______________________________

   ____

   b. __________ why? ______________________________

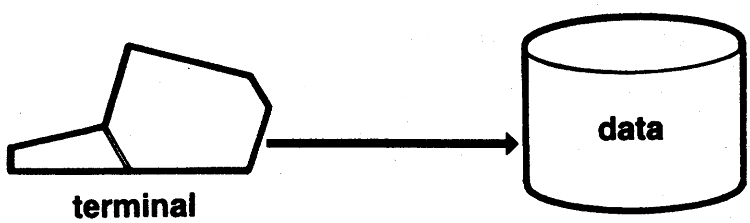
   ____

   c. __________ why? ______________________________

   ____

DATA ENTRY TECHNIQUES

- **User Controlled**
- **V/3000 Controlled**



notes:

references:

USER CONTROL

Program manages terminal interface directly

- **good for simple interactions**
- **forms control is complex**

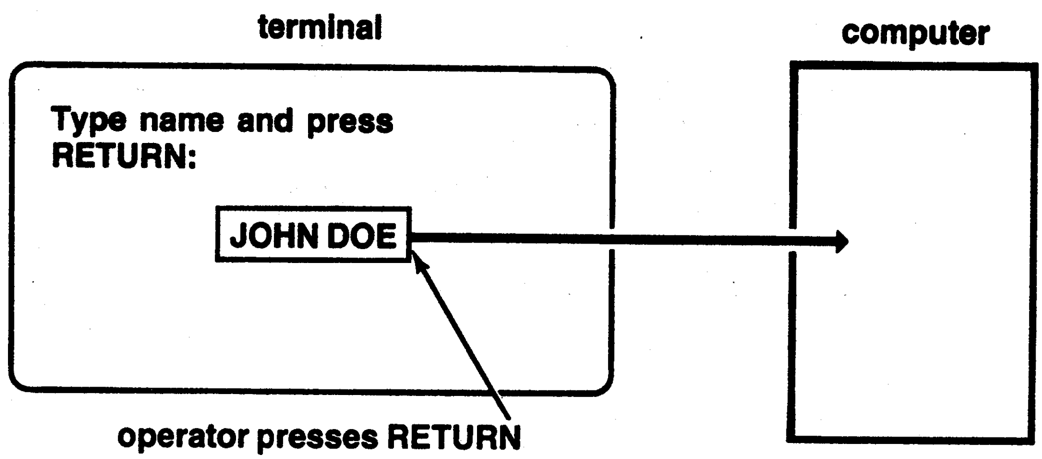
notes:

- If most terminal interaction is "conversational", user-control should be adequate.

- If large complex forms are needed in a data entry type application, control of these forms may be very difficult to manage.

references:

CHARACTER MODE TRANSFERS



- simple to use
- no special coding — just read or write

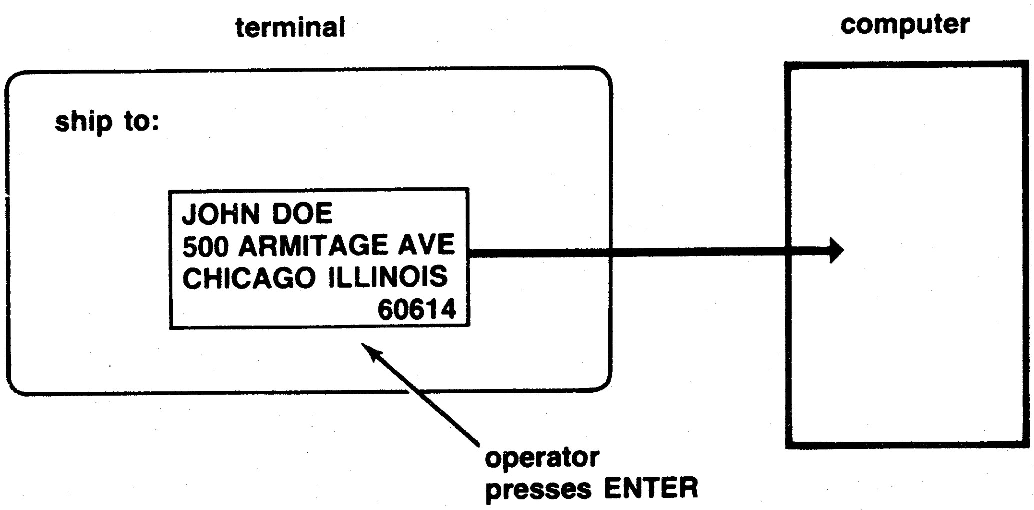
notes:

- This is the method for which the terminal I/O system was designed.

- It is simple to use and works very well for short conversational transfers.

references:

BLOCK MODE TRANSFERS



- requires special coding
- complex forms can create large stack

notes:

- This type of transfer is best for complex data entry applications.
- But, it is not at all simple to code; can get all but the most experienced user into trouble.

references:

BLOCK MODE — Programmatic control of:

- cursor positioning
- scroll display
- video enhancements
- alternate character sets
- ask for and acknowledge data transfers

uses "ESC" sequences:

display "shipto:"

display

$$\underbrace{\text{“ESC\&a10r5c ESC[ESC\&dJ ESC\&a25C ESC] ESC\&d@ ESCW”}}$$

Sets up 1 unprotected field, with half-bright inverse video, and turns on format mode

notes:

- Note that you only need to code the ESC sequences once, and they can even be saved in a file to conserve stack space.

references:

SUMMARY

Character Mode	vs	Block Mode
• easy to code		• hard to code
• each character echoed from computer		• characters echoed within terminal
• use for interactive applications		• use for data entry applications

■ Both interrupt CPU for each character transferred from terminal

notes:

references:

TERMINAL CAPABILITIES

- **262X — Interactive Terminals**

2621		asynchronous only (no multipoint)
2622, 2624, 2626	V/3000	

- **264X — Display Stations**

2640A		synchronous or asynchronous (multipoint on some)
2640B, 2641, 2644, 2645, 2647, 2648	V/3000	

- **307X — Data Capture Terminals**

3075, 3076	V/3000	asynchronous only (hard-wired multipoint)
3077		

notes:

- Note those terminals that allow V/3000, those that do not.
- Standard transfers are asynchronous - depend on a start-bit and a stop bit to delineate characters.
- Synchronous transfers (where allowed) are non-standard, use lots of memory and special multipoint software, but are fast and accurate.

references:

WORKSESSION III-5

notes:

references:

Worksession III-5 (data entry)

1. Given the following application tasks, indicate whether you would use character mode or block mode:

   A. The program prompts the user for a "YES" or "NO" response; if YES, it displays information on the screen; if NO, it issues another prompt.

      Character or Block mode? ______________________________

      Explain: ______________________________

      ____

      ____

   B. The program displays a form into which the user enters a complete set of order information.

      Character or Block mode? ______________________________

      Explain: ______________________________

      ____

      ____

2. Suppose you decide you want to use block mode transfers or V/3000. Are these capabilities available on any HP 3000 terminal? Explain your answer.

   ____

   ____

   ____

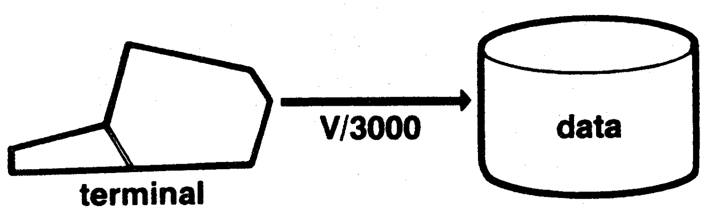
   ____

V/3000

- FORMS DESIGN with intelligent edits
- STAND-ALONE data entry system

OR

- FRONT-END to transaction processing system

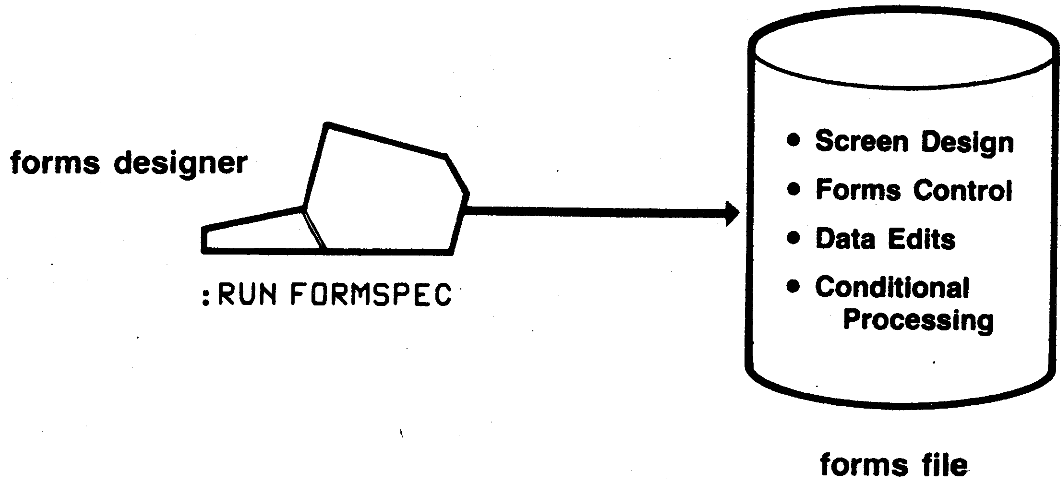


notes:

- Consider V/3000 for applications that need block mode terminal transfers.
- Does not require any programming effort to design forms; has special procedures that make forms control, data transfers, etc. very simple.

references:

FORMS DESIGN



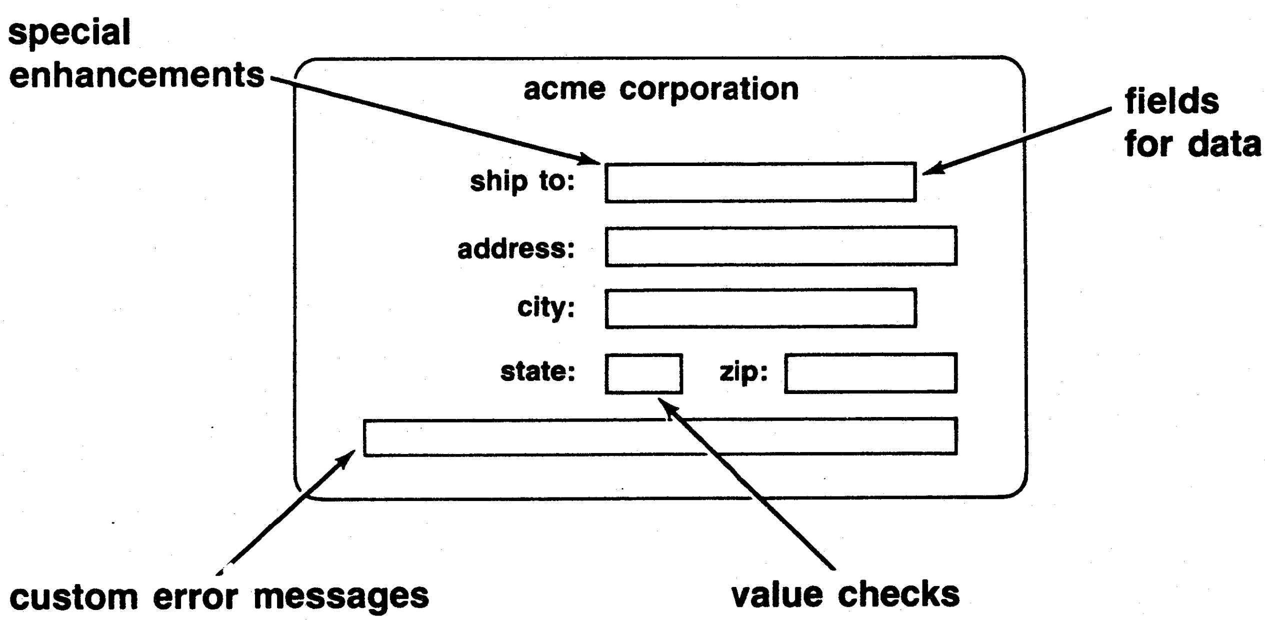
HEWLETT PACKARD

notes:

- No programming is needed to create the forms file which may include edits and processing specifications.

references:

SCREEN DESIGN



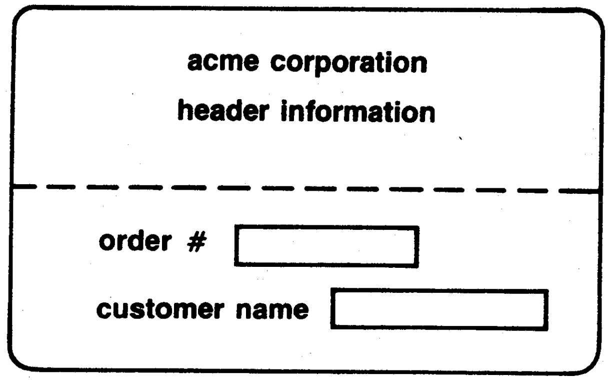
notes:

- Screen design is so simple that it is easy to develop very elaborate screens.
- The more elaborate the screen, the more stack space is needed. EVERY character and special enhancement adds to size of form, hence to stack. Everything on the screen (including such cosmetic features as lines of asterisks) adds to the stack size.

references:

FORMS CONTROL

- current form may be **FROZEN**
- next form may be **APPENDED**



- either may be **REPEATED**
- forms family allows field edits to vary, while screen remains the same

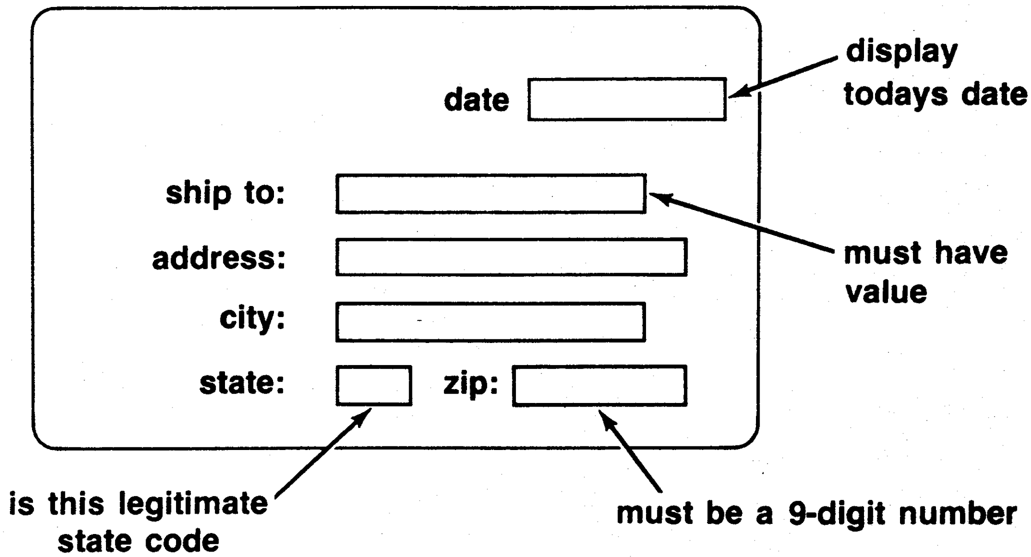
HEWLETT PACKARD

notes:

- Forms control is as easy as screen design and is generally a good way to save stack space and disc I/O
- Appended and frozen forms help keep forms a uniform size (saves stack).
- Repeating forms and forms in same family need not be reprinted on screen (saves disc I/O).

references:

FIELD EDITS

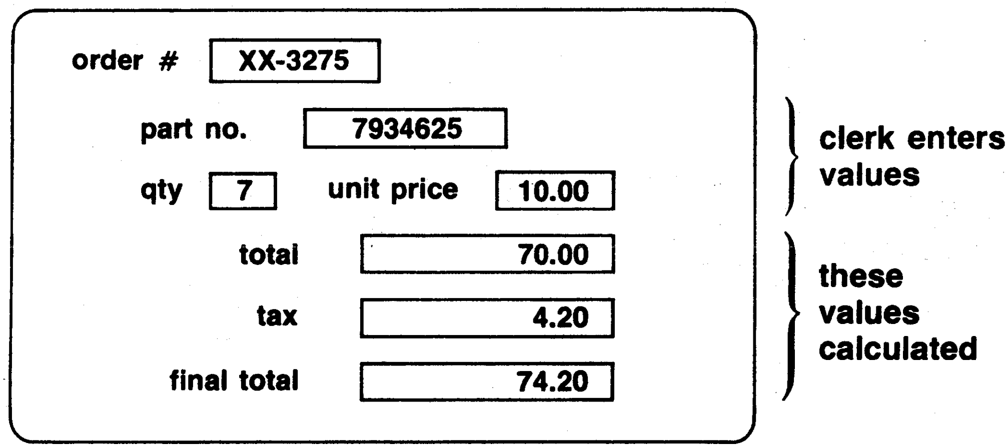


notes:

- Edits provide simple way to check on entered data. But, keep edits simple to save stack space.

references:

CALCULATIONS



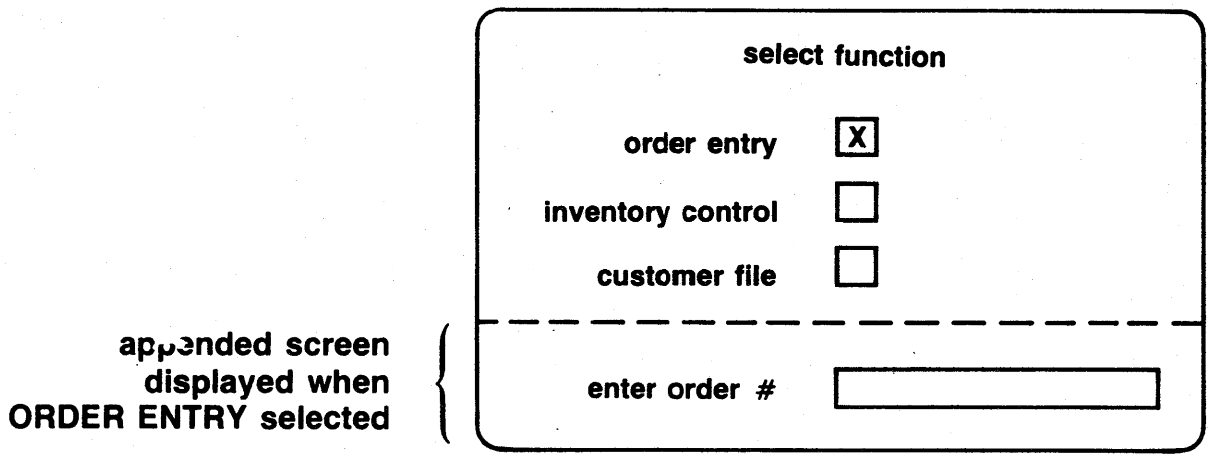
- any field can have calculated value

notes:

- This provides a good way to help prevent operator error, speed up entry, and keep data accurate. But, the operator has less control over entered data.

references:

CONDITIONAL PROCESSING



- **next function depends on value entered**

notes:

- This type of processing lets you avoid a lot of programming effort. But, it also adds a lot of data to the stack.

references:

Tips on Field Edits

- use most concise edit
  - EQ, IN, GQ use less code than IF . . . ELSE
  - avoid long tables
  - omit field name if possible
- use system constants - $EMPTY, $TODAY, etc. rather than literal values
- keep custom messages short

IN SHORT - KEEP ALL EDITS SHORT

notes:

references:

WORKSESSION III-6

notes:

references:

Worksession III-6

1. Suppose you have a form with 10 lines of header information, including 2 data fields, followed by 8 detail lines with 9 fields into which data can be entered. Thus, the entire form has 18 lines and 11 data fields. The other forms in the file each have between 8 and 10 lines, each with between 7 and 10 fields for data.

   Why is this poor forms design? ______________________________

   ____

   ____

   What can you do to improve it? ______________________________

   ____

   ____

   ____

2. Assume an application that accepts data through V/3000 forms. The accuracy of the data can be checked through edits stored in the forms file, but these edits tend to be quite long and must be applied to each field. Under what circumstances would you choose to put these edits in your application program rather than in the forms file? Explain.

   ____

   ____

   ____

   ____

3. Suppose you decide to perform all your edits through FORMSPEC rather than coded into your application. What can you do to make the edits more efficient?

   ____

   ____

   ____

   ____

Worksession III-6 (cont.)

4. You have an order entry application in which totals must be calculated from quantity, unit price, tax, shipping weight. What are the advantages of letting FORMSPEC perform the calculations instead of the data entry operator?

   ____

   ____

   ____

   ____

   What are the disadvantages? ______________________________

   ____

   ____

   ____

V/3000 FORMS FILE

- Code Records
- Managing Forms Files

notes:

references:

CODE RECORDS

- variable length depends on form and field data
- contain everything to display, edit, use form
- many types:

  K - global records

  L - form records

  O - custom messages

  plus others

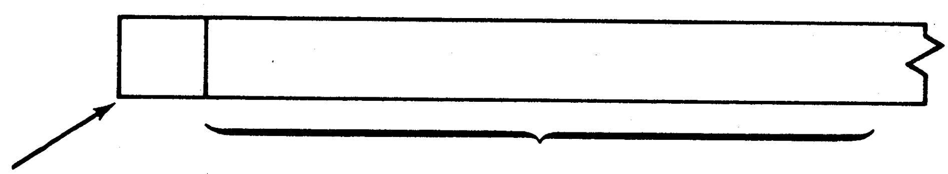
notes:

- Code records and the V/3000 data buffers are what a V/3000 formsfile consists of.

references:

FORM RECORDS

- 1 for every form in file



HEWLETT PACKARD

notes:

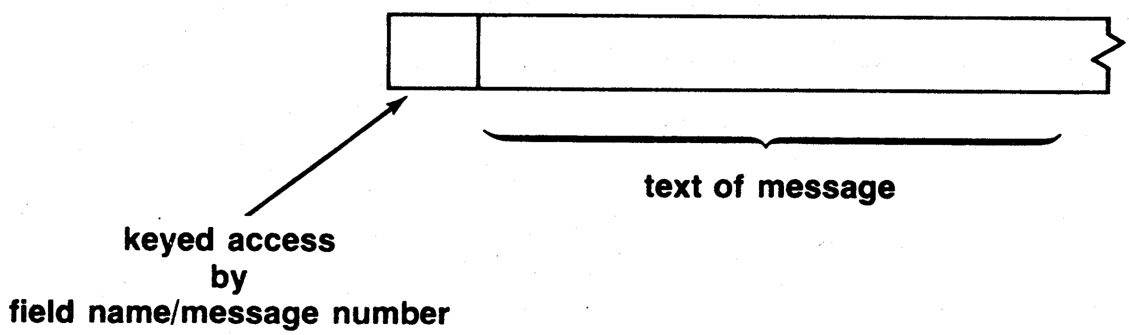
- All the information associated with a form is kept in the form record for that form.

- A form with complex edits, many special enhancements, etc., can generate a VERY big form record.

references:

MESSAGE RECORDS

■ 1 for every message for every field

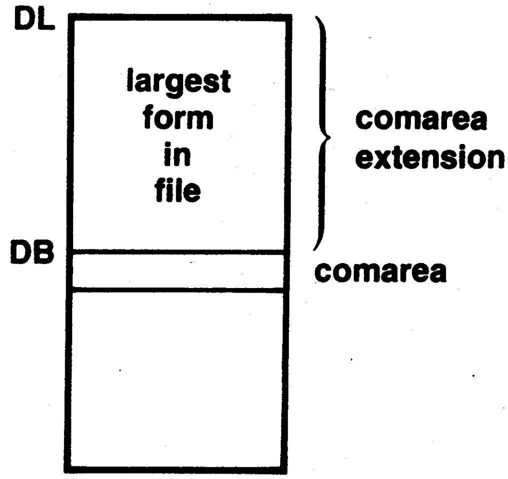


notes:

- Custom error messages are invaluable for helping operators, since they make error correction much simpler. But, take care in their design. Each message adds to the stack.

references:

FORMS FILES AND THE STACK



- MAXDATA always required (6K minimum)
- Stack must hold LARGEST form in EACH open forms file

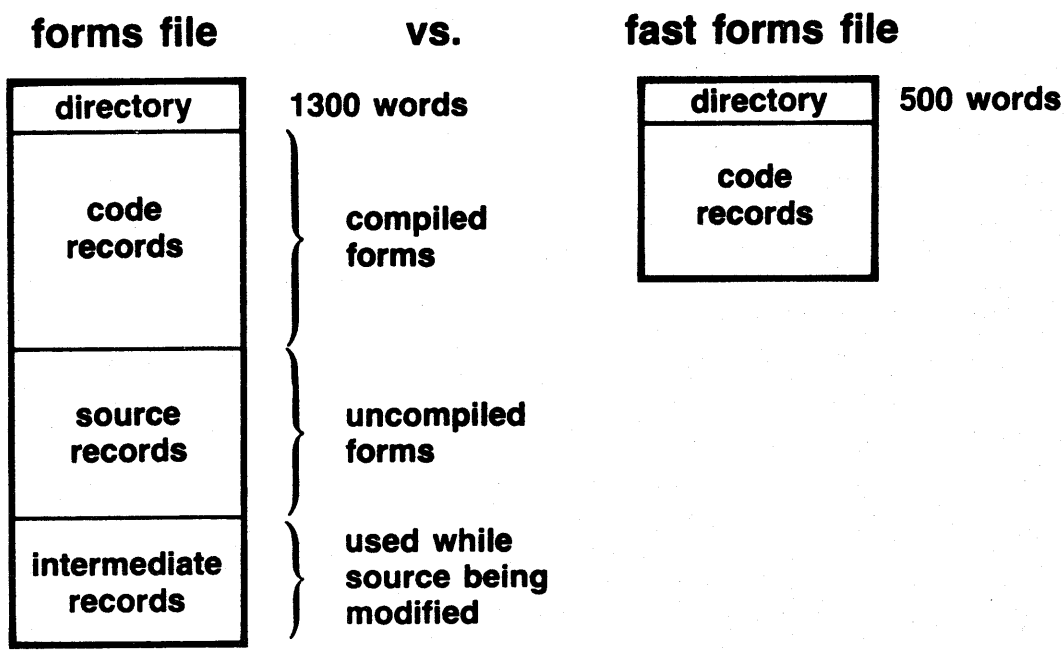
notes:

- The Comarea extension must hold not only the largest form in the file, but also all message records, two sets of data buffers (one for data to be edited, the other for data as it appears on the screen), plus a global record for all information that applies to the entire form.

references:

V/3000 forms file

FORMS FILE SIZE



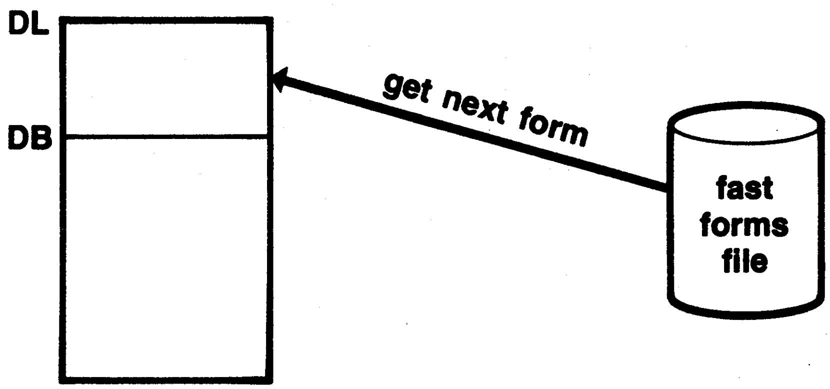
HEWLETT PACKARD

notes:

- The directory for either type is kept in the stack, the other records are brought in as needed.

references:

FORMS and DISC I/O



- each new form means disc access to bring code record into stack
- only repeated forms do NOT require disc I/O

notes:

- Forms in the same family also can save on response time; they do not need to be repainted on the screen.
- The sample program (see appendix A) uses a single form. This form is never repainted on screen, nor brought from disc. The procedure that "gets the next form" is smart enough to realize the "next" form is the same form.

references:

Tips on Form Design

- avoid one long form, many short forms
  - stack size based on longest form
- use repeating forms where possible
  - not "re-painted"
  - saves disc I/O
  - faster response for new form
- avoid fancy touches in protected areas of screen:
  - alternate character sets
  - display enhancements
  - lines of dashes, asterisks, etc.

notes:

references:

WORKSESSION III-7

notes:

references:

Worksession III-7 (V/3000 structure)

1. V/3000 must run with MAXDATA set to at least 6K. Explain why you think this is necessary.

____

____

____

____

____

   Would the STACK= parameter be an acceptable substitute for MAXDATA?
   Explain your answer.

____

____

____

____

2. Indicate by a YES or NO after each of the following statements whether it increases the size of the code record associated with each form.

   A. Text that is displayed on the form but is not transferred as data.

      Yes ____________ or No ____________

   B. Special enhancements that are part of the text but do not enhance the data fields.

      Yes ____________ or No ____________

   C. The size of the unprotected fields into which data is entered.

      Yes ____________ or No ____________

   D. The number of unprotected fields into which data is entered.

      Yes ____________ or No ____________

   E. The length of the field edits associated with each field.

      Yes ____________ or No ____________

   F. The number of fields for which edits are specified.

      Yes ____________ or No ____________

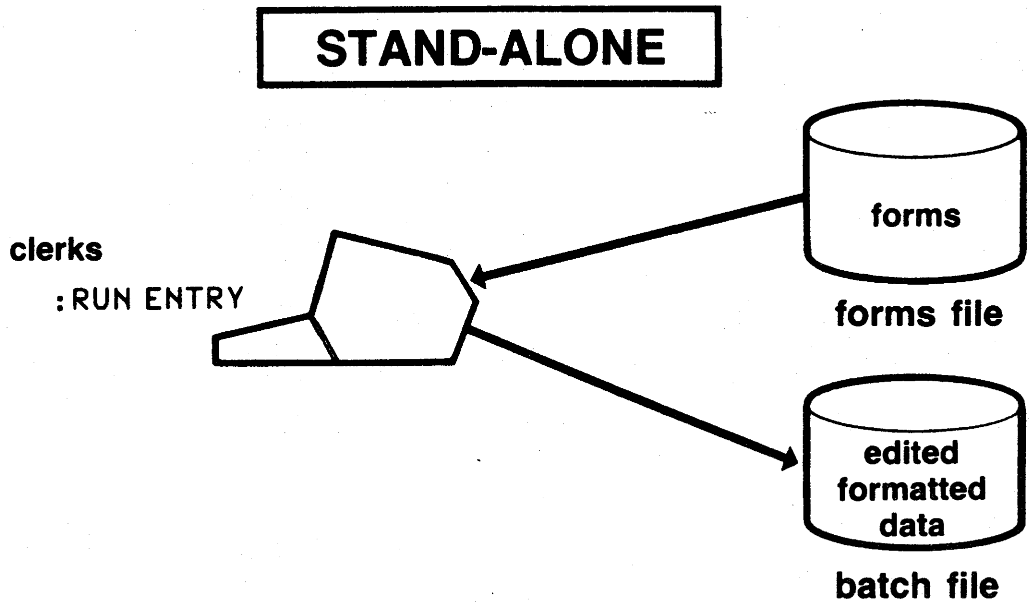
   G. The total number of fields in the form.

      Yes ____________ or No ____________

Worksession III-7 (cont.)

3. Explain why repeating forms are faster than other forms.

V/3000 data entry



- ENTRY — a general-purpose data entry program
- REFORMAT — a general-purpose reformatting facility

notes:

- These applications, provided with V/3000, allow immediate data entry without programming.
- ENTRY is useful during forms design in order to test the forms, but is too general purpose to be a highly efficient data entry application, and it does not transfer data to or from IMAGE data bases or KSAM files.

references:

ENTRY PROGRAM

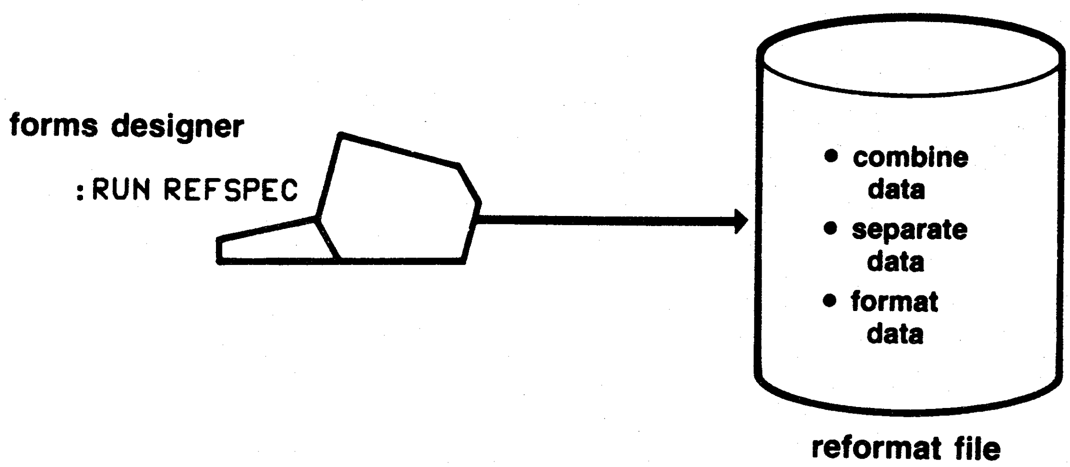
- available in all languages (except APL)
- browse and modify entered data
- no direct transfer to data base
- excellent tool to test forms design
- easy to modify to suit application needs

notes:

references:

REFORMAT CAPABILITY

- use to specify new combinations of entered data



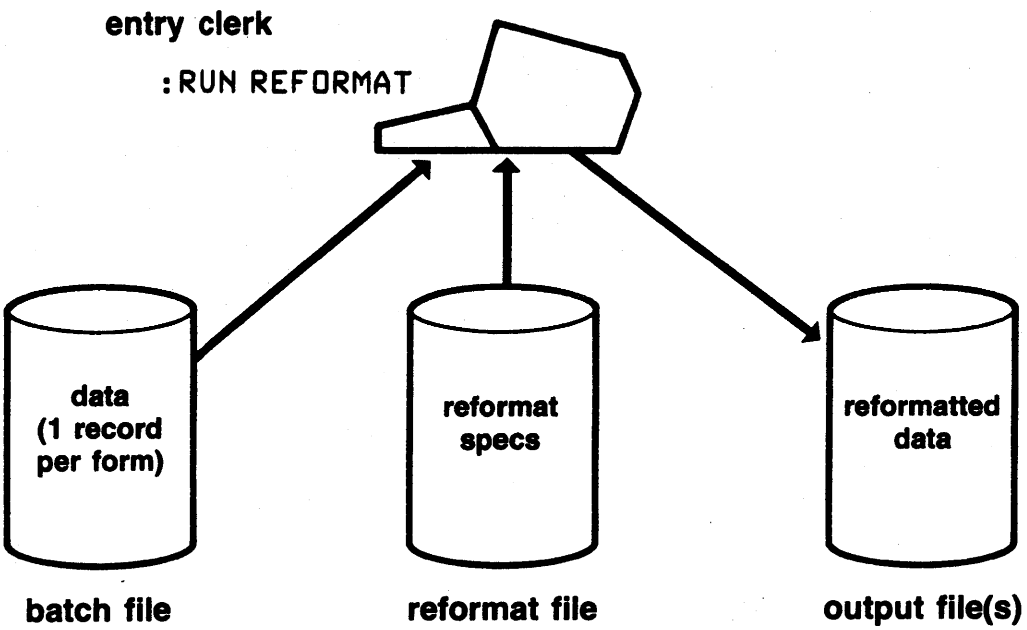
- REFSPEC allows you to combine records generated by ENTRY into a single redord, or to break ENTRY records into multiple records.

- Also, provides means to reformat individual data fields, omit fields, or add literal data.

- It cannot repeat a record as part of other records. Once an ENTRY record is reformatted, it cannot be reformatted again in the same file.

references:

REFORMAT



notes:

- This capability is most useful during conversions. The data entered in new V/3000 forms can be reformatted to suit an existing application's needs. Thus, the data is made available before the application is rewritten.

references:

REFORMAT TIPS

- Use REFORMAT to separate or combine data records

- REFORMAT cannot repeat the same header record preceding multiple details

- Use as interim method until existing application changed to process V/3000 data

notes:

references:

WORKSESSION III-8

notes:

references:

Worksession III-8 (V/3000 data entry)

1. The data entered into a set of V/3000 forms must be written to an IMAGE data base. Can this be done using ENTRY? Or must you write a special program to transfer the data?

   Explain your answer. ______________________________

   ____

   ____

2. Suppose you have an application that expects a separate record for each part number entered on an order form. You plan to use a new V/3000 form to enter orders in which you allow up to 10 part numbers to be entered on one form. Is there any way you can use your new form with your existing application?

   Yes or No? ____________ Explain your answer. ______________________________

   ____

   ____

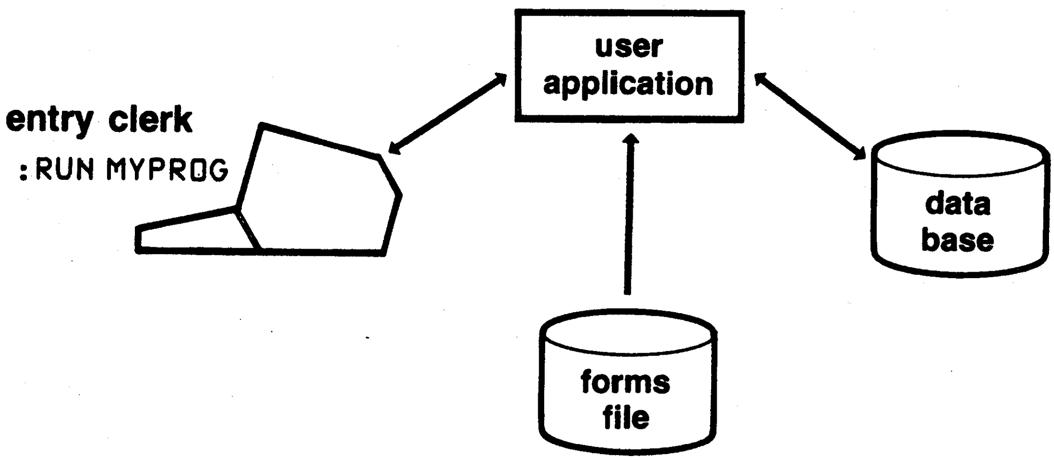
   ____

   ____

   ____

FRONT-END

- V/3000 procedures to manage forms files, field edits, data entry and reporting



HEWLETT PACKARD

notes:

- The application uses V/3000 prodecures to handle the interface with the terminal, the forms file, the program buffers, or an MPE batch file. It can use other procedures to transfer data between the program buffers and a data base.

references:

IN APPLICATION PROGRAM

- application uses existing Forms File
  - all edits can be in forms
  - much processing in forms
- V/3000 provides form and data control procedures
- can direct entered data to IMAGE or KSAM
- can retrieve and display data from IMAGE or KSAM
- unlike ENTRY, tailored to user needs

notes:

- Consider V/3000 for other uses besides data entry.
- Good for anything that requires form-handling, or block mode terminal I/O.

references:

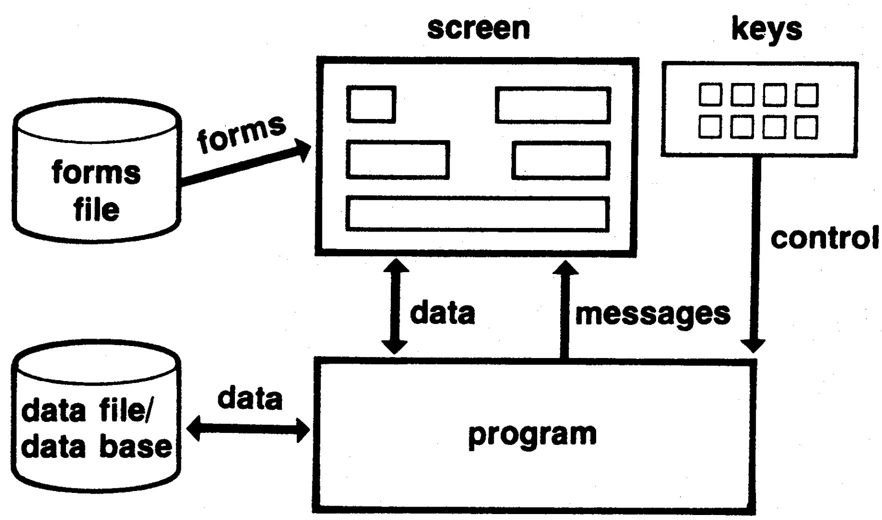
FORM AND DATA CONTROL

VGETNEXTFORM
VSHOWFORM

VREADFIELDS
VFIELDEDITS
VGETFIELD

VSETERROR
VPUTWINDOW

VPUTFIELD



notes:

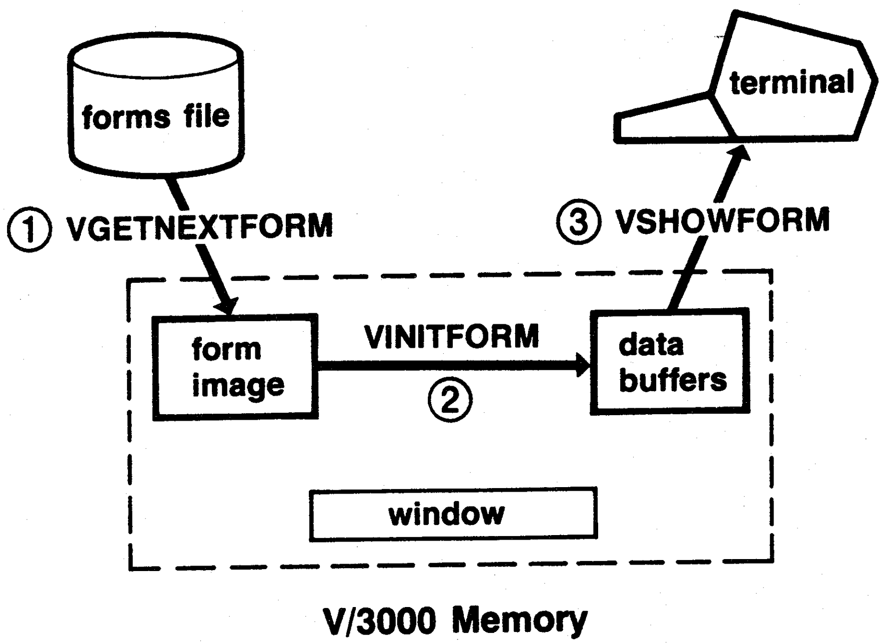
- Additional procedures transfer data between the program and files or a data base.

- Look at the sample program in the appendix; it uses these procedures (and some others) to manage the terminal interface.

references:

Form Control

- display and initialize form

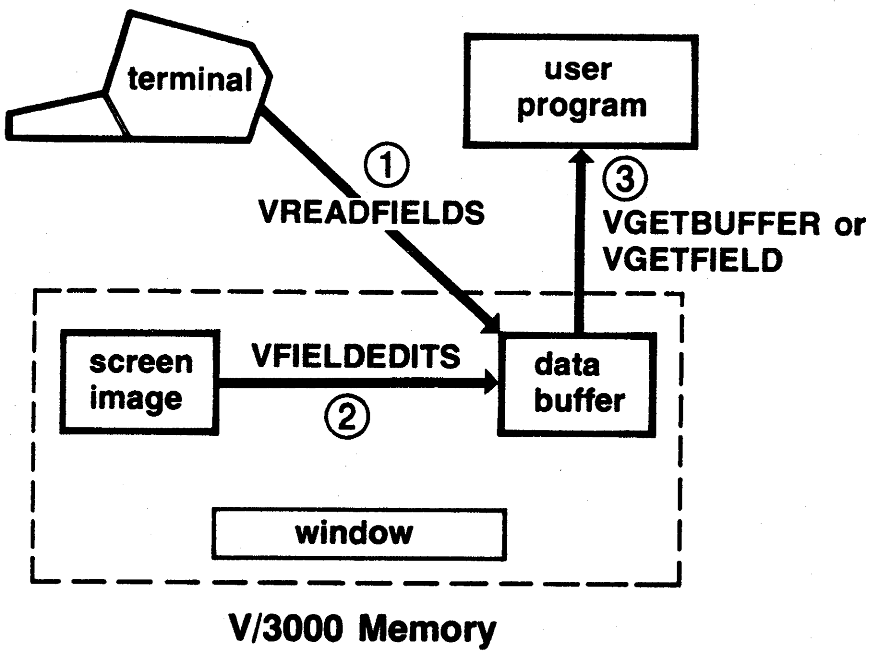


HEWLETT PACKARD

notes.

references:

Collect and Edit Data

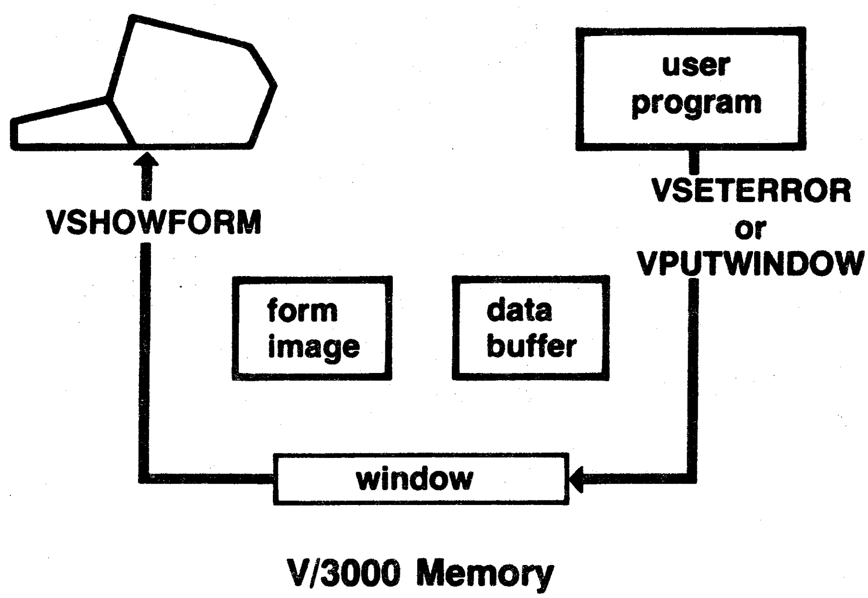


notes:

- In step 3, VGETFIELD provides greater independence from the form. Each field is referenced independently which means the form can change without causing the program to be re-written.

references:

Process Errors



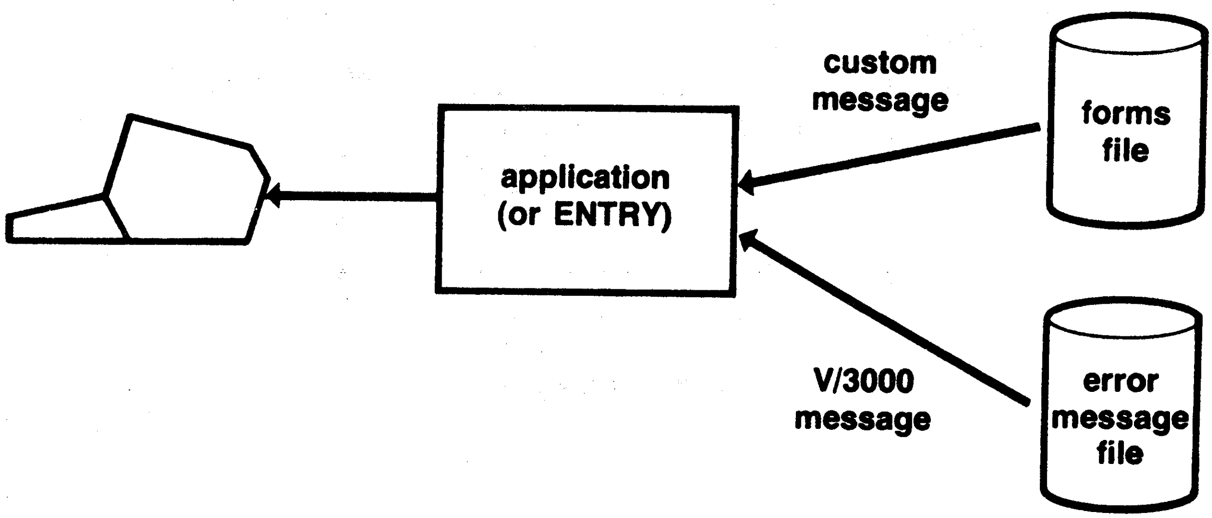
notes:

- Error processing differs depending on whether edits are done in the program or by V/3000.
- In either case, messages are passed through the "window" area of the V/3000 memory to a "window" line on the screen.

references:

ERROR HANDLING

- all errors require disc I/O to retrieve error message



notes:

references:

Tips for Programming with V/3000

- put all forms into 1 forms file
- open only 1 forms file at a time
- always use fast forms file for production

notes:

- Consider making provisions in your code to "time-out" when doing terminal I/O. This is a safeguard against operators leaving the terminal hanging.

references:

Should You Use V/3000?

■ does V/3000 do everything you want?	YES
■ can you afford the extra stack space?	YES
■ do you need intelligent front-end edits?	YES

THEN

V/3000 should work well for you

notes:

references:

BUT

- if you need a keypunch replacement
- if you are short of stack space
- if you have SIMPLE forms you can code easily

THEN

V/3000 may be more than you need.

notes:

references:

WORKSESSION III-9

notes:

references:

Worksession III-9 (V/3000 programming)

1. Suppose you have an application that needs to display small amounts of constant data on the terminal screen in a format that uses an elaborate format with field enhancements. This can be done a) with ENTRY or b) in your program with V/3000 procedures. Which would you choose? Explain your answer.

2. Suppose you have two independent functions in your application, one of which must be selected by user input at the terminal. You can ask the user to make the selection on a V/3000 menu form, or you can issue a prompt to be answered by Yes or No.

   Would you use V/3000 here? Explain your answer, including any factors that might affect your choice.

   Would you change your answer if there were 3 or more functions to select? Explain.

SUMMARY

Select:

- **a processing method**
- **an accounting structure**
- **a programming language**
- **a data entry method**

that suits your application, and helps your end user

Be critical over the terminal that is running
Harder to code + test.

security

block mode
[illegible]
[illegible]

HEWLETT PACKARD

notes:

Look at
hardware requirements
good on the system

references:

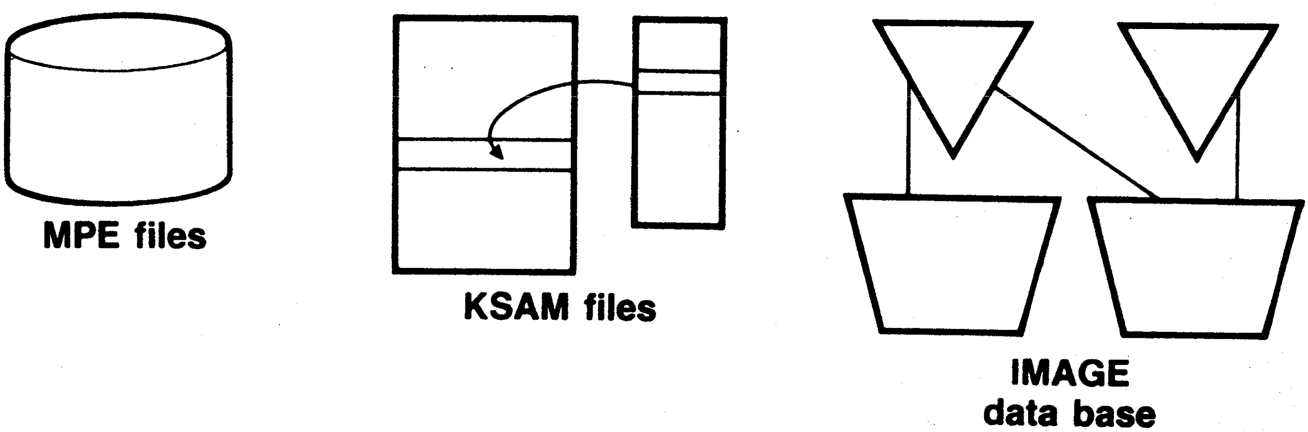
DATA MANAGEMENT

- Options
- MPE files
- KSAM files
- IMAGE/QUERY - good for files that change a lot - banking
- Choosing the right method

notes: How will the data be accessed? Structured files are good for quick retrieval - KSAM, IMAGE.

references: MPE Intrinsics Reference Manual
KSAM Reference Manual
IMAGE Reference Manual
QUERY Reference Manual

OPTIONS



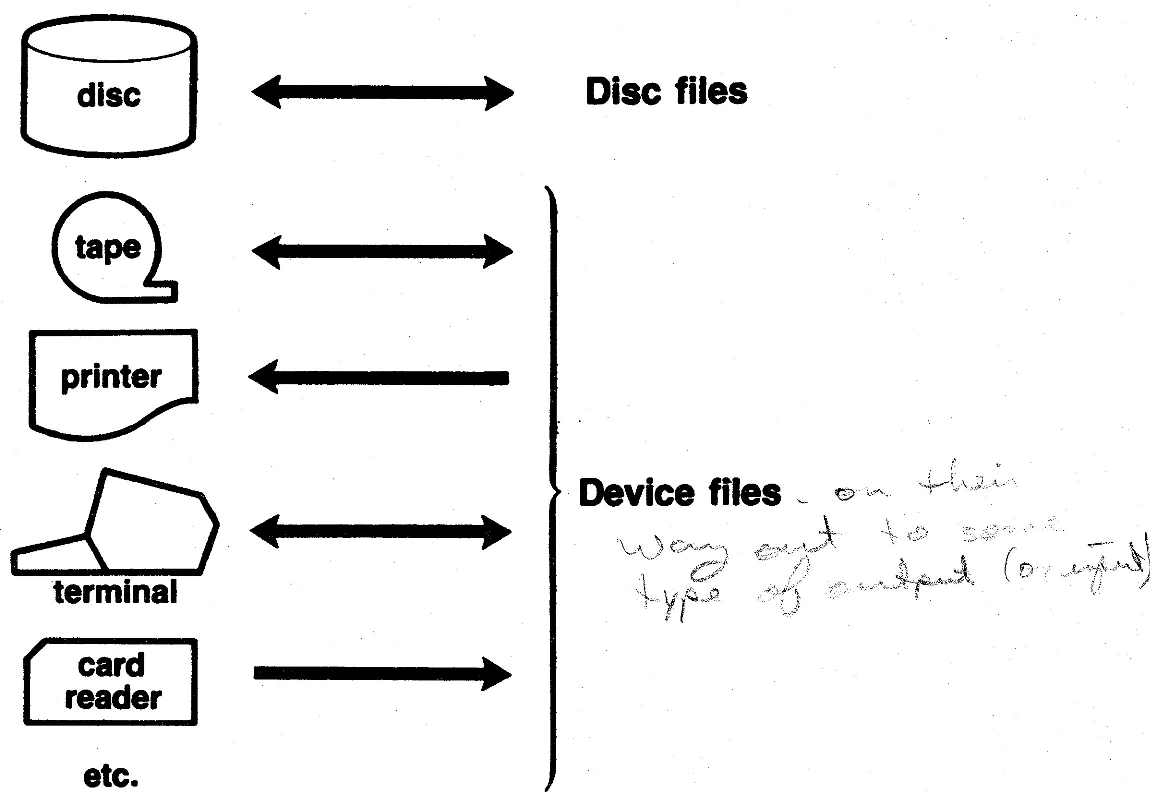
HEWLETT PACKARD

notes:

references:

options

MPE FILES



notes:

- Disc files provide both sequential access and random access by record number.

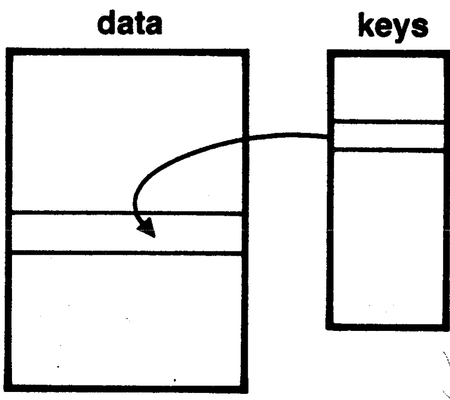
- Device files allow sequential access only.

- Because device files are slower than disc, "spooling" provides a buffer between devices and a program. Spooling is managed by the system.

- All files are managed alike by the MPE file system whether they are disc or device files.

references:

KSAM files



many keys

- keyed sequential access (like ISAM)
- keys and data maintained on separate disc files
- many access options
  - sequential
  - chronological
  - keyed

notes:

- Keyed access has many options:
  multiple keys, duplicate keys, partial keys, approximate keys.

up to 16 keys

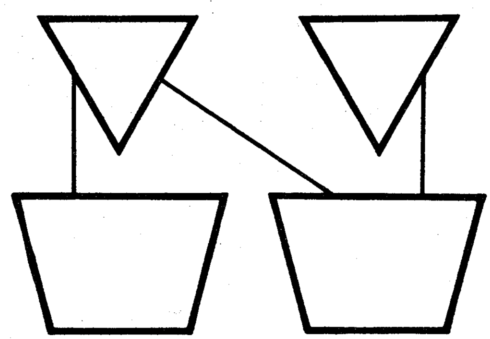
Can do generic searches

references:

IMAGE data base

master - details

- 2-level network structure
- data structure independent from program
- reduces ~~eliminates~~ data redundancy
- access at data item level
- special security and locking
- QUERY for rapid data retrieval



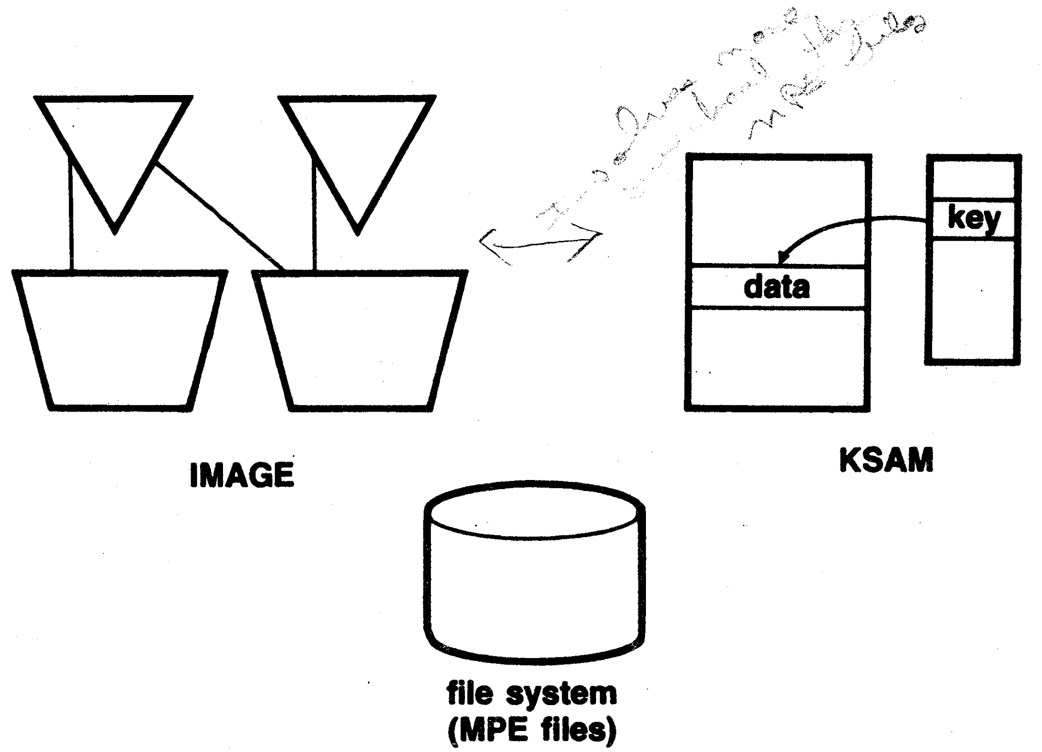
notes:

- IMAGE has many access methods:
  serial access,
  directed access (by record number)
  chained access (items with same value),
  calculated access (find item by its value).

- IMAGE has a special security system that goes beyond the standard file system security. It also provides simple logging and recovery procedures.

references:

File System underlies both IMAGE & KSAM



notes:

- KSAM is built directly on the file system; the user has the same controls.

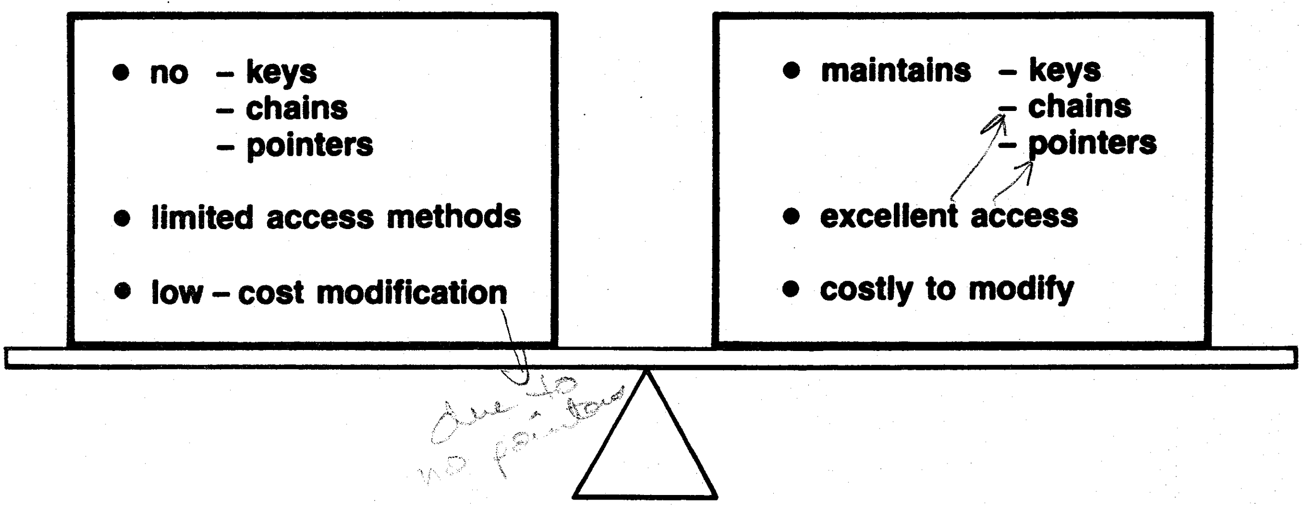
- IMAGE only indirectly uses the file system; the user has no direct control over IMAGE files.

references:

options

UNSTRUCTURED vs. STRUCTURED

MPE files — **IMAGE or KSAM**



notes:

IV-7

- It requires less overhead to modify an unstructured file, but it is not necessarily easier. Your application must locate the record to be modified. This is done for you in the structured systems.

references:

WORKSESSION IV - 1

notes:

references:

Worksession IV-1 (structure)

1. Suppose your application has many on-line inquiries, but all updates are batch. In this case, would you store your data in structured or unstructured forms? Explain your answer.

   Structured fast retrieval

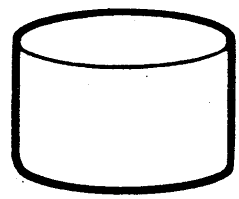
2. Suppose a lot of new data must be added on-line, and inquiries are infrequent. Would you store the data in structured IMAGE or KSAM files or in unstructured MPE files? Explain your answer.

   unstructured

Image has MPE + Image security

MPE FILES

- Using MPE Files
- Sharing Files



HEWLETT PACKARD

notes:

Can specify up to 16 buffers

(80x3 = 240).

Default is two buffers

size of each is recl/block x blocksize

NR = 80, 3 ASCII 3 x 80 = 240 bytes

Block

Each extent

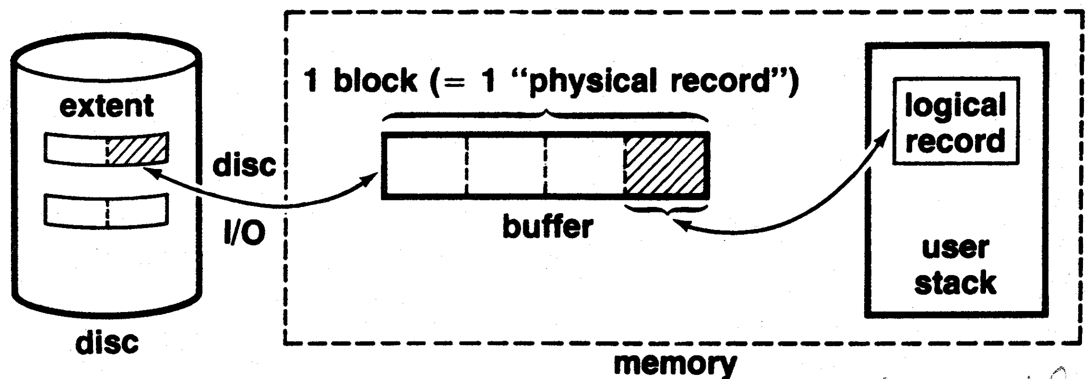
references:

File label always takes up 1 block

Rec = 80, 16, F, ASCII

Blocks, Buffers, and Extents

data is transferred one block at a time



1 buffer = 1 block

The buffer is in an extra data segment

When reading: disc to buffer to user stack

notes:

32k - max size of buffer

Sector = 256 bytes

- A block is a physical record, the smallest unit transferred to/from disc.
- A buffer holds 1 block of data in memory, is the buffer between the disc and the program.
- A logical record is the smallest unit of data processed by a program.
- An extent is a contiguous piece of a file on disc; most files are broken into extents.
- A sector (not shown on slide) is the smallest addressable unit on a file (only the file system knows sector addresses); every block, every extent, must start on a sector boundary. Each sector=128 words or 256 characters.

references:

Extents - Contiguous areas (sectors) on disc next to each other

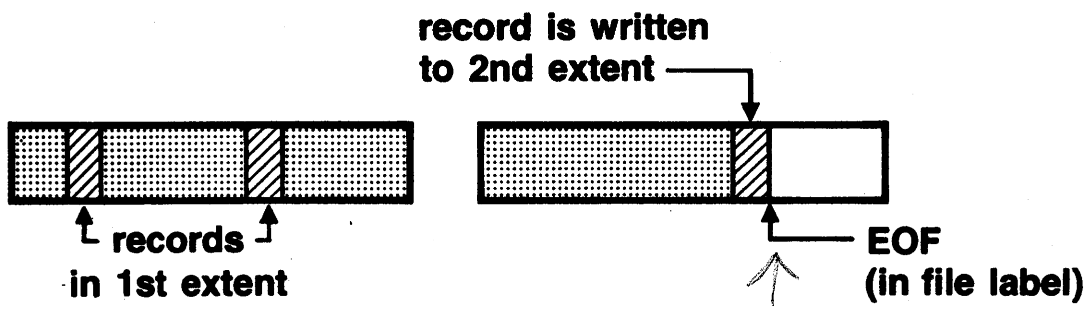
... boundaries

Extent Management (1)

- **allocation and initialization of extents is automatic**

example:

- initial allocation = 1 (default)
- access is random



HEWLETT PACKARD

notes:

- When system allocates a new extent, it only initializes extents up to the EOF; extent past EOF may contain garbage.

- Allocation and initialization take time.

Extents are initialized to 3?? File labels are updated

FCONTROL & FCLOSE update EOF in extent data

references:

Extent Management (2)

- user can decide:
  - how many extents – file in small or large "pieces"
  - how many to allocate initially – contiguous or discrete "pieces"
- user can force initialization of all extents

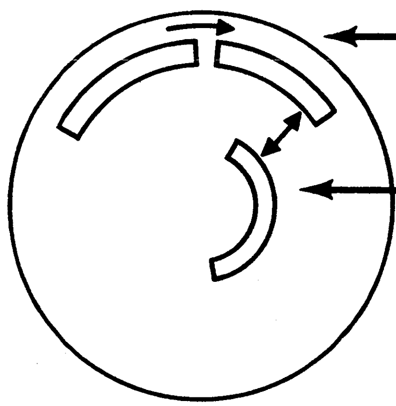
notes:

- By default, all files are broken into 8 extents; user can specify as few as 1 extent, as many as 32.
- 1 extent allocated initially - this can be changed easily.

references:

Initial Allocation of Extents

- initial allocation can reduce
  - on-line allocation time
  - seek time



- disc seek time depends on placement of file on the disc

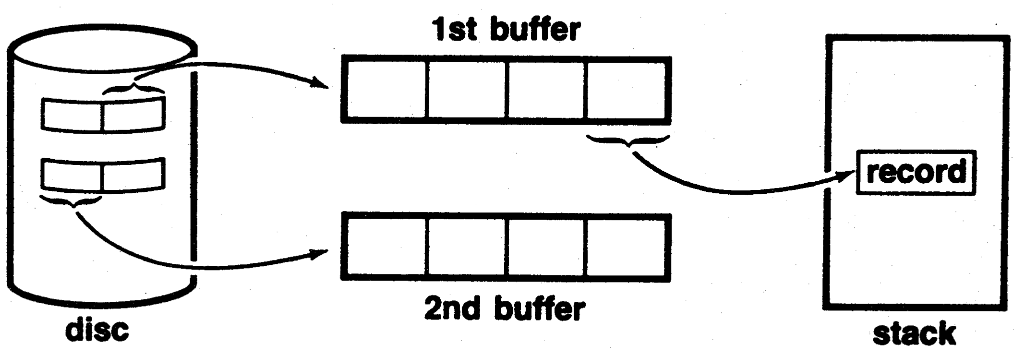
notes:

- Generally, initial allocation of extents not necessary.
- Note that system attempts to make all initially allocated extents contiguous. This MAY reduce seek time.
- Allocate more than one extent if you know allocation will occur during peak hours of on-line processing and slow response time.

references:

Choosing the Number of Buffers

- 2 buffers – default assigned by MPE
- allows pre-reading of sequential files



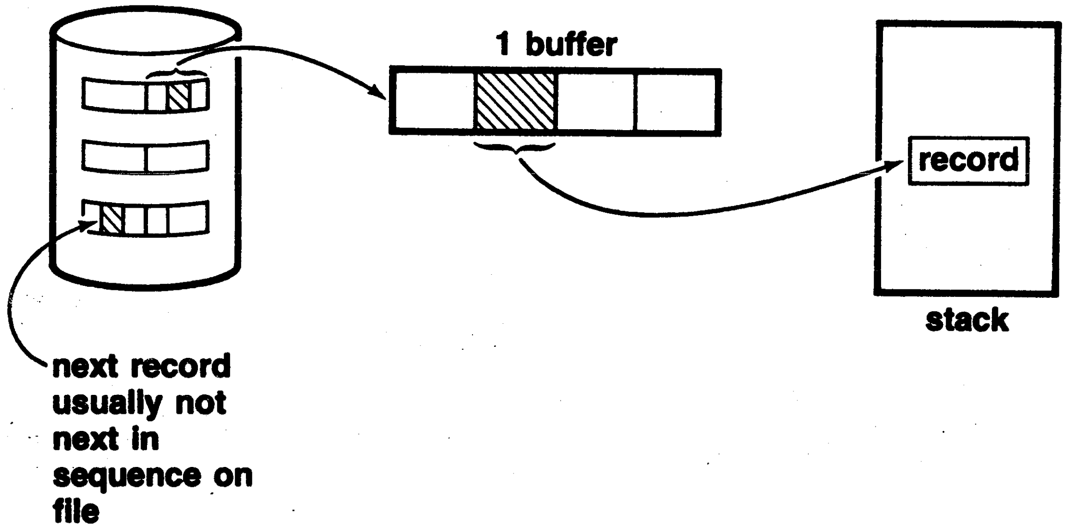
notes:

- Pre-reading means that while records in one buffer are being processed other records are being read into the second buffer. This only works for reading sequential files, since the next block to be read is predictable.

Good to use Defaults

references:

Use 1 Buffer for Random Access



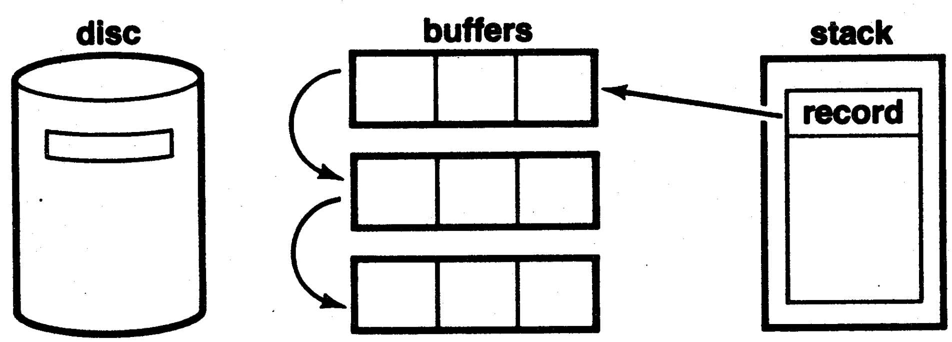
notes:

- No pre-reading advantage for random files, so no advantage from having more than 1 buffer.
- The single buffer saves memory space.

references:

Use more than 2 buffers, only

- when loading data into sequential files
- when no other users on system



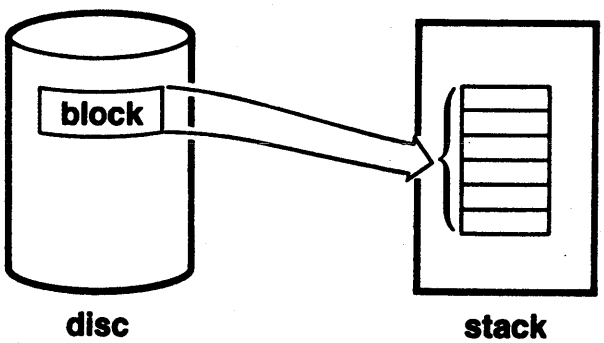
notes:

- Many buffers use lots of memory and offer NO advantage except in the exceptional situation shown above.
- Particularly, avoid more than two buffers in a multiprogramming environment.
- The number of buffers can be changed each time the file is opened, so you can experiment.

references:

Consider NOBUF – (0 buffers)

■ **transfers block directly to user stack**



- fast
- allows multi-rec

HEWLETT PACKARD

notes:

- Program must "deblock" logical records from the block transferred to/from the program.
- Stack must be large enough to hold the entire block.
- Stack must be "frozen" in memory making it hard for MPE to find memory space for executing processes in a multiprogramming environment.

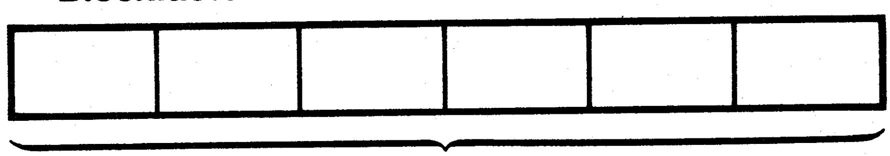
references:

Choosing a Blocksize

- **Blocksize – a function of record size and the blocking factor (number of records in a block)**

example:

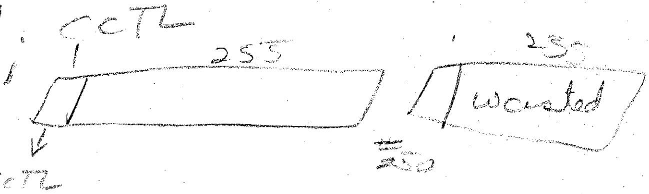
- **Recordsize = 80 characters (40 words)**
- **Blockfactor = 6**



1 block = 480 characters (240 words)

notes:

Rec= -256,1,,CCTL



- Blocksize is a permanent file characteristic; it is not easy to change.
- Note: The slide only illustrates fixed length records. Undefined records are always 1 record per block; Variable-length records need extra space in each record and each block for a record count.

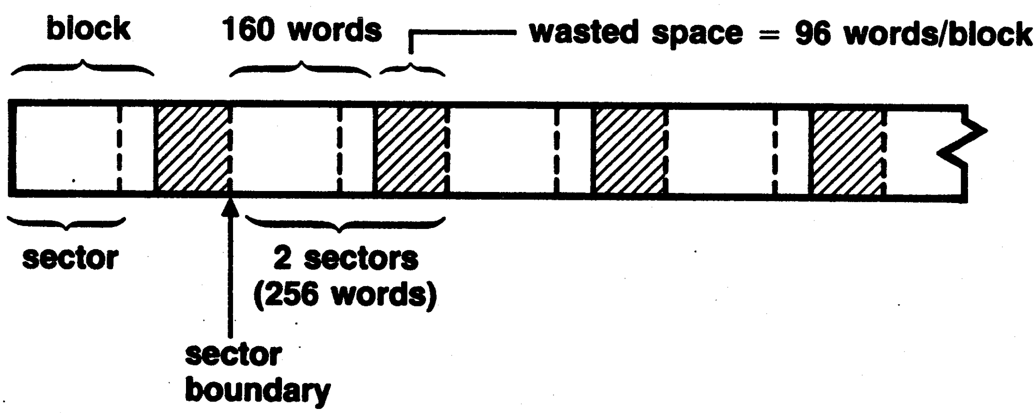
references:

Blocksize and Disc Space

- Blocks always start on sector boundaries

example:

- assume block = 160 words



notes:

- In this example, over half of every other sector is wasted.
- Blocksize should always equal, or be slightly less than, a multiple of sector size (128 words).
- The first block of every file is set aside for the 128-word file label; if the block is much bigger than the label, this too wastes space.

references:

Blocksize and Access Mode

small block vs. LARGE BLOCK

- means small buffer – less memory space needed
- good for RANDOM ACCESS

- transfers more data at a time – uses more memory
- good for SEQUENTIAL ACCESS

notes:

- Because blocksize is not easy to change, plan for the most used case.

references:

OPENING and CLOSING FILES

Use system resources heavily –

To minimize impact:

- **open file once at start of program**
- **leave file open**
- **close file once at end of program**

Consider putting all OPENS and CLOSES in separate code segment

HEWLETT PACKARD

notes:

- When opening a new file, open the file, close it at once to save it as a permanent file, then open it again. This insures file is not lost. (New files are session temporary until closed). Or, build the file with a command and then save it as a permanent file.

- Opening a file is a major operation; it involves writing from the file label to a control block, setting up the EOF, setting up record pointers, and establishing the access path to the file. Closing a file reverses these steps.

~~When you close a file it will be [illegible]~~

references:

WORKSESSION IV - 2

HEWLETT PACKARD

notes:

7978
6250 tape drive

start stop
streaming

To use to full advantage, have a blocking factor of 4K bytes (1 block = to 4K bytes),
(to be writing out 4K at a time.

references:

Worksession IV-2 (using MPE files)

1. Suppose you plan to read an entire file from beginning to end in sequential order, and you are one of many system users:

   A. Would you specify 0, 1, 2, or more buffers? Explain.

   2 Takes up memory

   B. Would you specify a block factor that gives many records per block or few records per block (records are fixed-length). Explain your choice.

   many

2. Suppose you plan to add new records in random order, and you do not want to "deblock" each record.

   A. Would you use 0, 1, 2, or more buffers? Explain.

   1

   B. Would you specify a large or small block size? Explain.

   small

SHARING FILES

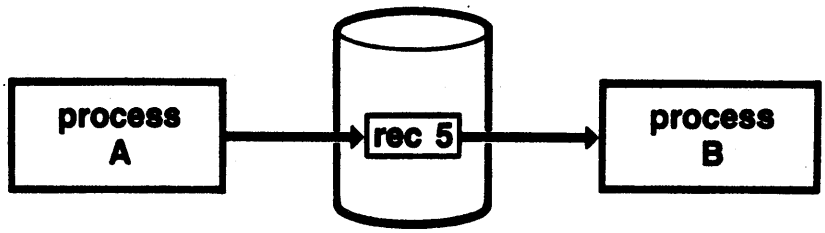
- Locking Strategies
- Multi-Access

HEWLETT PACKARD

notes:

references:

A Gentleman's Agreement



A "locks" file
A modifies record 5
A "unlocks" file

B attempts to "lock" file
B waits until A "unlocks" file
B "locks" file
B reads record 5
B "unlocks" file

HEWLETT PACKARD

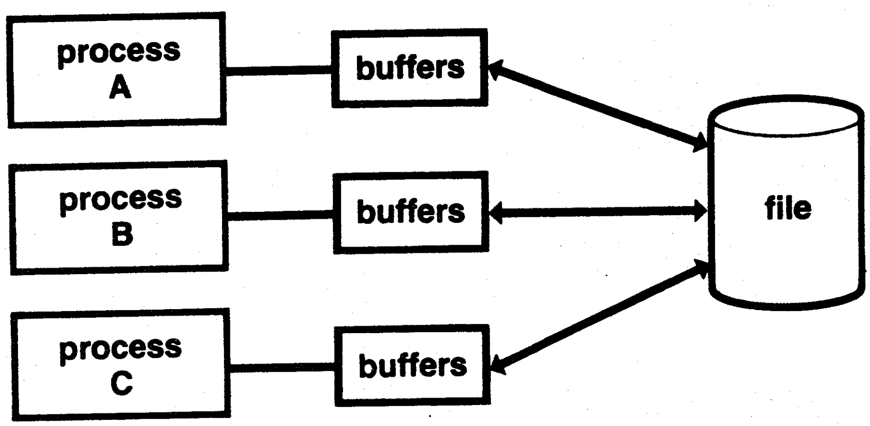
notes:

- The locking mechanism depends on all sharing processes testing and respecting locking signals.

A logical lock
not physical

references:

File is shared—Buffers are not



- **buffers contain data, current record pointer**

notes:

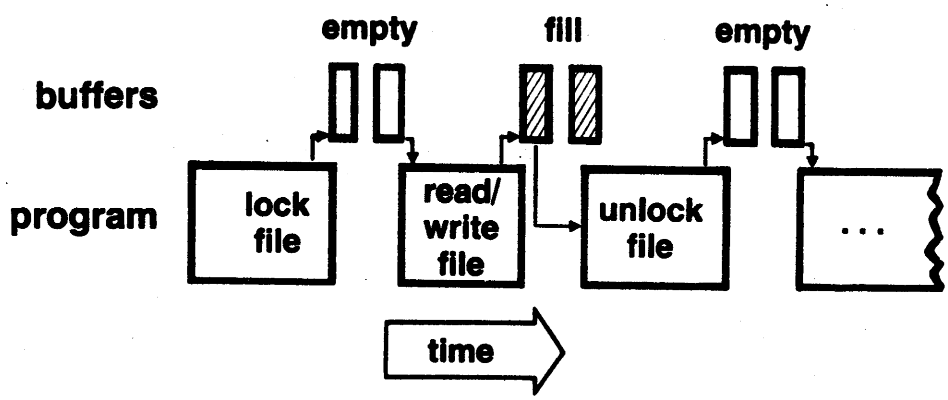
- Each buffer can contain different versions of the same record.
- Each buffer can have a different record pointer to the current record.

Each user has own buffers & pointers

references:

LOCK/UNLOCK

makes sure data is in only ONE program's buffers at a time



LOCK: starts with empty buffers

UNLOCK: ends with empty buffers

notes:

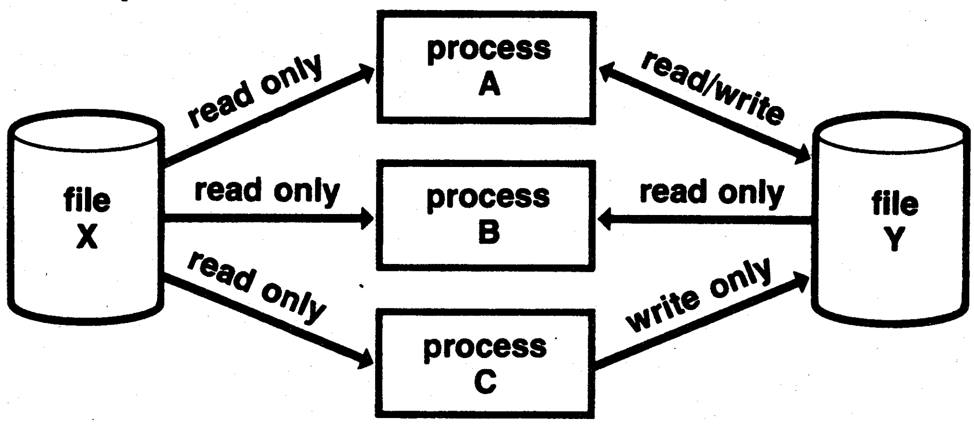
- Locks ensure that data and pointers are only in one set of buffers at a time. This keeps the file orderly with only the latest data, however many processes are concurrently accessing the file.

references:

When to Lock

■ when a file is shared, locking insures data "integrity"

example:



- which processes need to lock file X?
- which processes need to lock file Y?

HEWLETT PACKARD

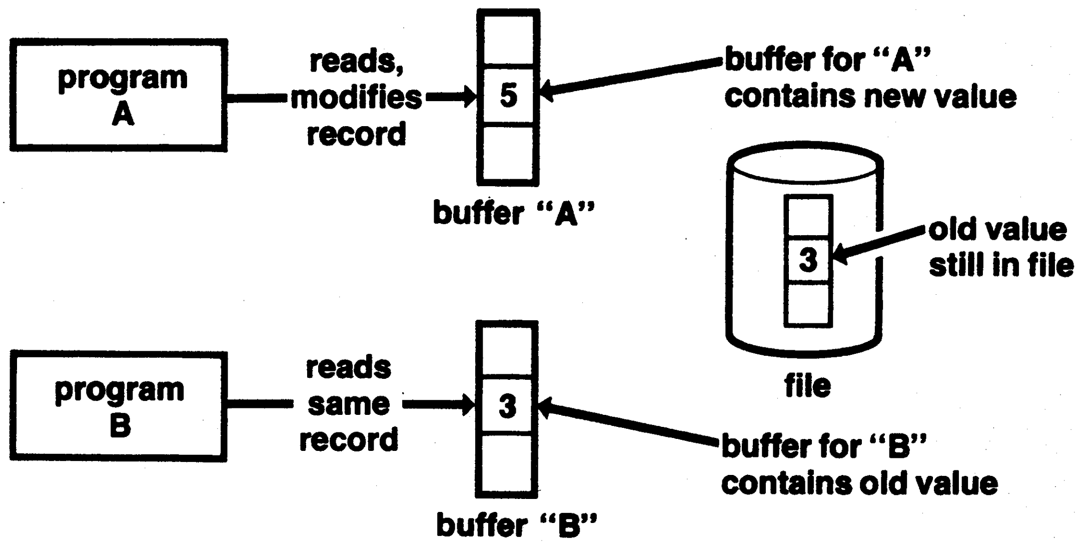
notes:

- No processes need to lock file X; it is not being modified.

- All processes need to lock file Y, even process B, that only reads the file, if it wants to read the latest data.

references:

- **what if sharing programs DO NOT LOCK/UNLOCK?**



notes:

references:

- **lock around logical transactions**

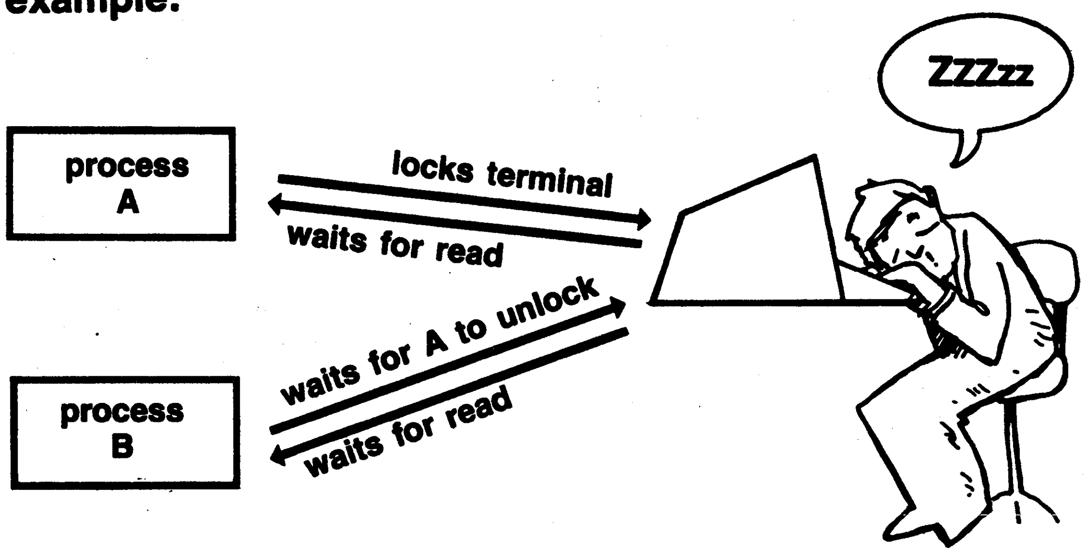
good:	**LOCK** **READ** **UPDATE** **UNLOCK**	**no changes can occur between READ and UPDATE**
poor:	**READ** **LOCK** **UPDATE** **UNLOCK**	**another user can change data (or move pointer) between READ and LOCK**

notes:

references:

- **beware of locking around a terminal read**

  example:



- **all processes wait for operator to wake up!**

notes:

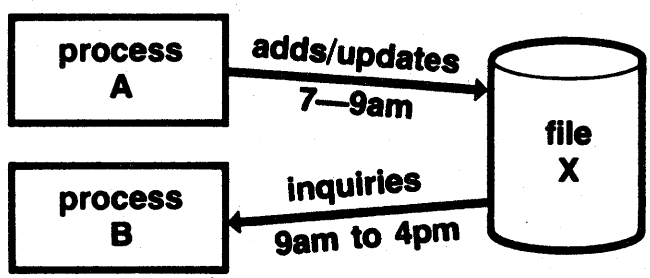
- Devices as well as disc files can be locked.
- A time-out procedure can be used to make sure terminal is not left hanging.

references:

LOCKING Uses Resources

- use locks with care
- consider designs that avoid locking

example:



HEWLETT PACKARD

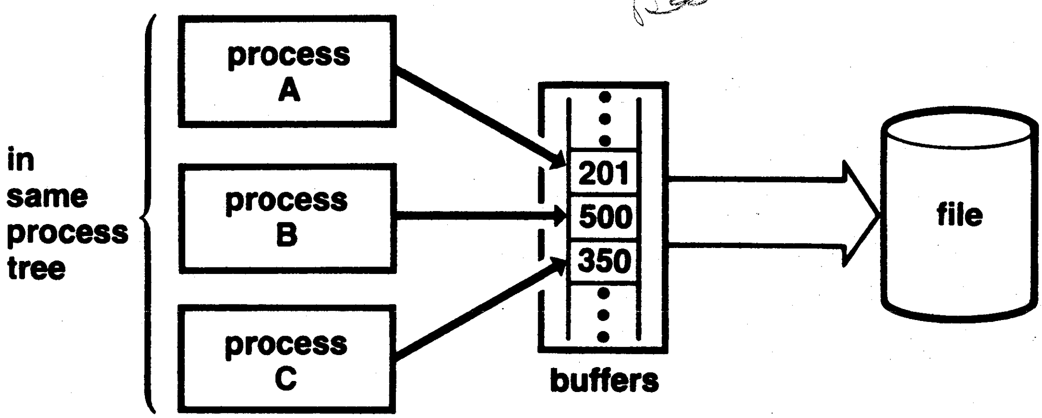
notes:

- Locking overhead is caused mainly by the number of disc transfers needed to post the buffers at each lock and unlock.

references:

MULTI-ACCESS

- shares access path (buffers, pointers)
- restricted to father/son processes
- no LOCK/UNLOCK - RIN's Resource ID #



notes:

can't use NOBUF option. needs buffer

references:

MULTI-ACCESS - Advantages

- saves memory - single rather than multiple buffers
- reduces number of locks
- reduces number of opens/closes

notes:

- Multi access provides a way to pass the file number of the shared file between processes - this cuts down on the number of opens and closes.
- Also, it allows chronological writes by many users without locking around each write.

references:

MULTI-ACCESS - Disadvantages

- only useful for process-handling applications
- requires cooperation between sharing processes
- may require locking of global resources
- buffers are required

notes:

- Only one file close takes effect, so users must cooperate to insure correct disposition of file as determined by close.
- Global resources need to be locked only if the file label is directly modified.
- NOBUF transfers are not allowed with multi-access.

references:

WORKSESSION IV - 3

notes:

references:

Worksession IV-3

1. Assume two programs share a file; program "A" updates employee records, program "B" retrieves current employee data.

   A. Which program must lock the file? Explain.

   A

   B. If program "B" can use data that is one day old, write a scenario that avoids locking altogether.

   A updates [illegible] the [illegible]
   B [illegible] to [illegible] day

Worksession IV-3 (cont.)

2. Assume two programs that share an inventory control file; program "A" adds new parts to the file; program "B" checks the latest on-hand quantity of any part in the file. If locking is used, both programs must lock the file.

   For program "A", which locking strategy would you choose? Give your reasons.

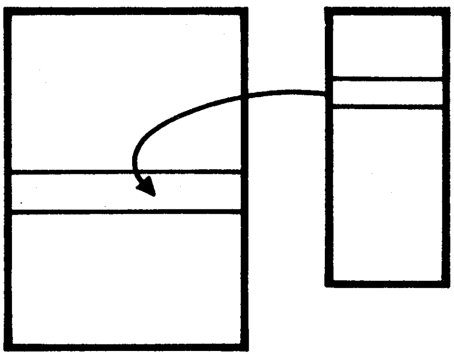
   A. LOCK file
      READ record
      UPDATE on-hand quantity
      loop back to READ next record
   UNLOCK file

   B. LOCK file
      READ record
      UPDATE on-hand quantity
   UNLOCK file
   loop back to LOCK file

3. Assuming the programs in question 2 are child processes in the same process tree, describe how they can both access the file at the same time, and allow "B" to get the latest data without locking the file.

   Use multi-access

KSAM



- Overview
- Key Selection
- Using KSAM Files

notes:

references:

OVERVIEW

- **What is a KSAM file?** Keyed Seq Acc
- **Guidelines for selecting keys**

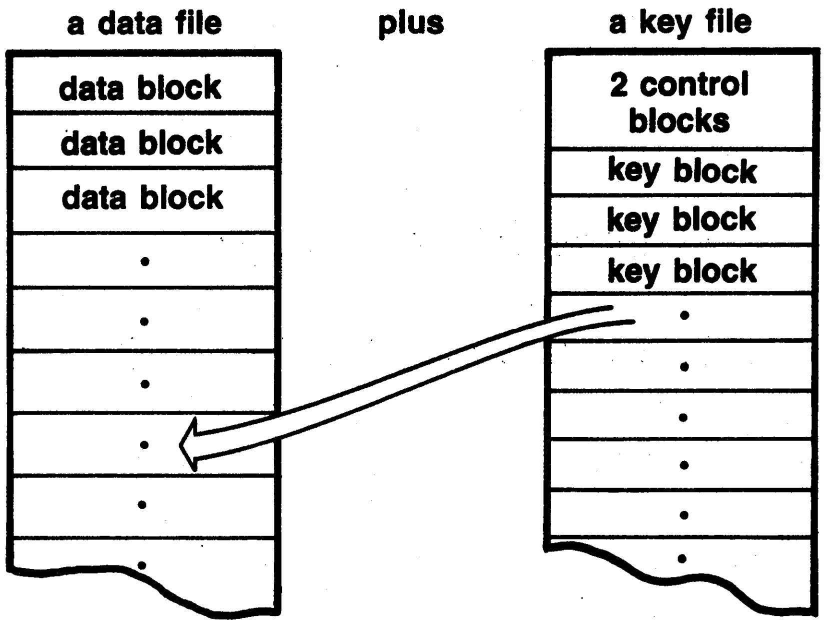
up to 16
keep ≤ 1000

notes:

references:

KSAM file

■ a KSAM file = 2 MPE files



Default is [illegible]

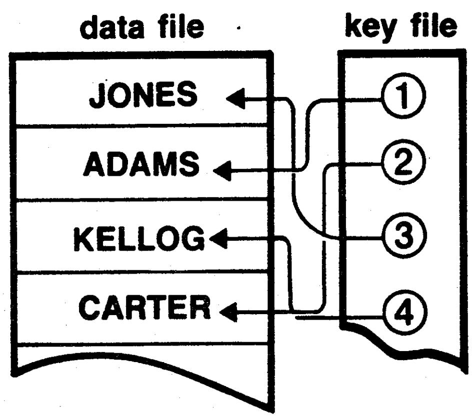
notes:

- KSAM files are accessible in all languages except APL.
- KSAM interface built into RPG and COBOL II, must make procedure calls from FORTRAN, SPL, BASIC, or COBOL '68.

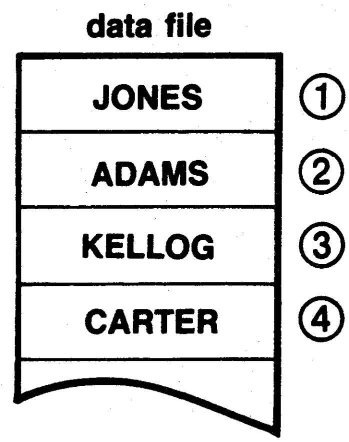
references:

KSAM file

SEQUENTIAL ACCESS



■ by key value



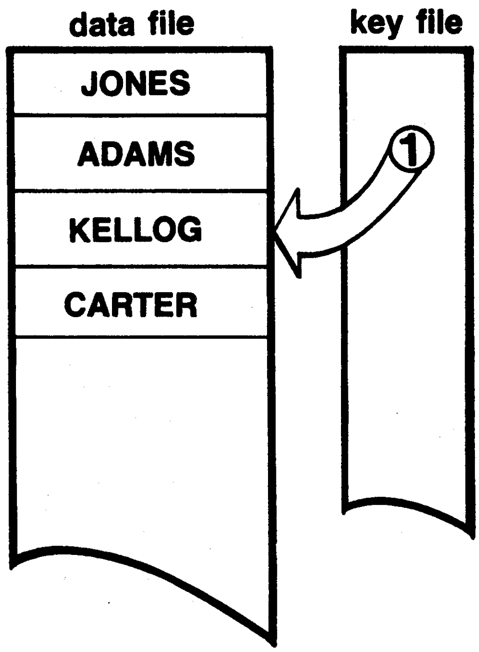
■ in chronological order

notes:

- Key access may use primary or alternate key; keys determined when file is created. Sequence in ascending order by key value.
- Data may be written to file in primary key order, or it may be written in chronological order.
- Access in chronological order is not available in BASIC or COBOL '68. Chronological access is like sequential MPE access; key file is not used.

references:

RANDOM ACCESS



■ access particular record by key value

- exact key – e.g. key = "KELLOG"
- partial key – e.g. key = "KE"
- approximate key – e.g. key ≥ "K"

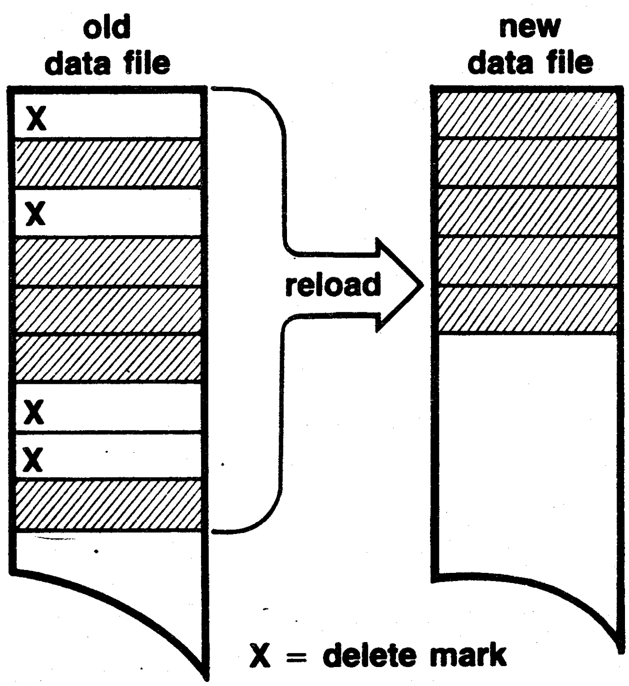
HEWLETT PACKARD

notes:

- Direct access by data record number is possible in any language except BASIC and COBOL '68; as with chronological access, the key file is not used.

references:

DELETING DATA RECORDS



- record not physically deleted
- "deleted" record in data file marked in 1st word
- reload removes records marked for deletion from data file

■ reload frequently if file has many deletions

 HEWLETT PACKARD

notes:

Use FCOPY w/ NOKSAM option to remove all deleted records.

- "Deleted" data record remains in data file with a delete flag in first word.
- Key entries for "deleted" records are physically removed from the key file; key file pointers are updated.
- Leave first word (2 characters) of each data record blank, so delete flag does not overwrite data.

references:

KSAM Utilities

use FCOPY

- for fast, unprogrammed inquiry
- for loading data into KSAM file
- for re-loading data to compress file

use KSAMUTIL

- to create file Build
- to clear data from file erase
- for recovery from system failure
- to retrieve file statistics

notes:

- It is possible to create a KSAM file programmatically in an SPL, FORTRAN, or COBOL II program, but not in BASIC or COBOL '68.

- There is no built-in KSAM logging capability and no built-in back up capability; but can use MPE facilities.

references:

KSAM Tips

- OPEN file using data file name – not key file name
- LOAD file in sequential order by primary key
- PLACE data file and key file on separate discs

HEWLETT PACKARD

notes:

- If you open the file using the key file name, the key file is opened, the data file name retrieved, and the key file closed, then the data file is opened, the key file name retrieved and the key file opened. The first open and close of the key file is not done when you open the file using the data file name.
- Loading data in primary key sequence takes a little longer than loading in chronological order (and makes a larger key file), but significantly speeds up access by primary key. It produces a tidy data and key file.
- Separate discs save seek time. ~~Don't use if files are remote.~~

references:

WORKSESSION IV - 4

notes:

references:

Worksession IV-4 (KSAM files)

1. Suppose your application keeps its employee records in a KSAM file. Once a week, it retrieves all the records in the file in sequence by employee last name; once a month, it accesses all records by department code. Also, it must occasionally locate all employees whose names start with a particular letter.

   A. Which item would you choose as the primary key, employee name or department code? Why?

   Name - it is the most frequently used

   B. Taking into consideration the item you chose as the primary key, is there any advantage to forcing new records to be added in primary key sequence? Any disadvantage? Explain.

   D - Finding space to put it is
   A - Faster retrieval

   C. Do you need to know the record in order to find the first employee whose name begins with "K"? Explain your answer.

   No - generic [illegible]

2. Over a period of time, many records are deleted from the employee file, and you notice that accessing the file is slower.

   A. Explain why this happens.

   reading deleted records

   B. What can you do to improve the access time in this situation?

   unload

GUIDELINES for KEY SELECTION

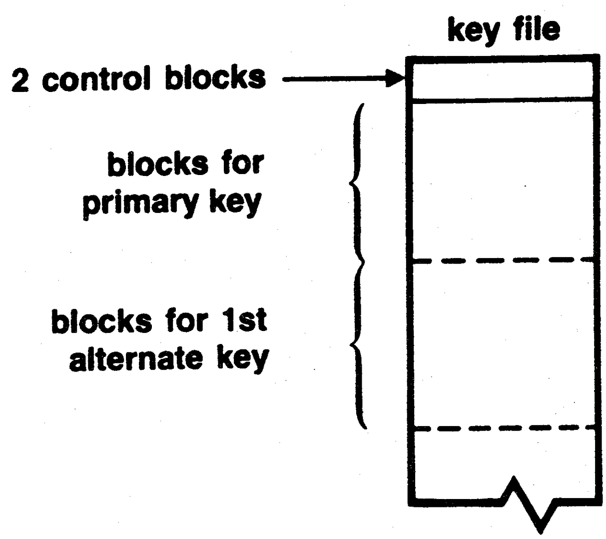
- Multiple Keys
- Changing Key Values
- Duplicate Keys

notes:

references:

Avoid Multiple Keys

- use as few keys as possible
- each key increases size of key file

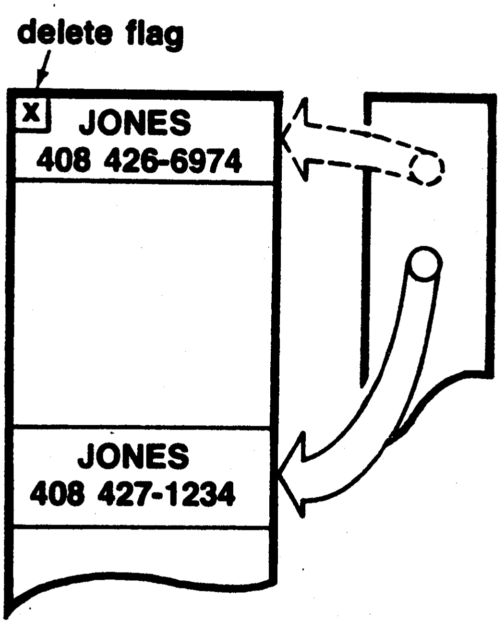


notes:

- Up to 16 different keys are allowed! But, use that many only if response time is not a consideration.

references:

Select Static Values for Keys



- update of key value forces record to be:
  - — deleted from data file
  - — added as new value
- adds unnecessary data to data file
- key file must be restructured

notes:

- In this example, assume the phone number is defined as a key. When the phone number changes, the entire data record is marked for deletion, the old key entry is deleted and the new key entry added to the key file.

references:

Key File uses B-tree Structure

- each key uses separate "tree" of key blocks
- key blocks linked through pointers
- key block structure changed when:
  - new value added
  - old value deleted
  - key value modified

HEWLETT PACKARD

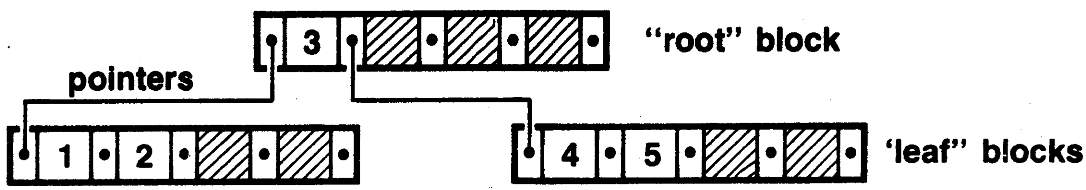
notes:

- To understand how keys are located (and managed), it helps to know something about the key file structure.

references:

Example of B-tree Structure

2-level tree:



- binary search fast for unique keys
- adding new values expensive

notes:

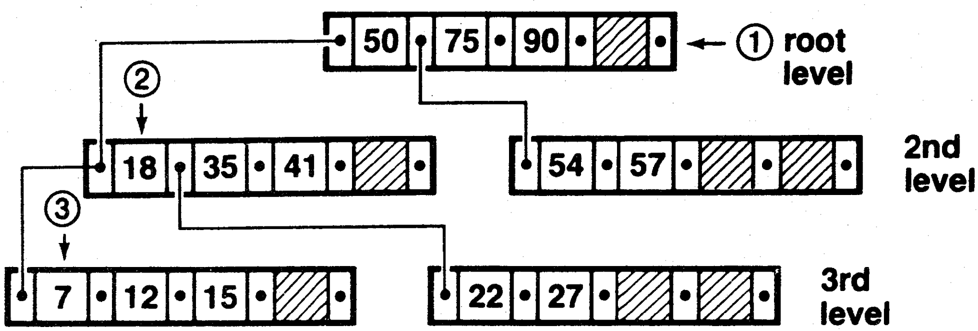
- Key values are in sequence within blocks.
- The root block always has central values - so that values are balanced, as many greater as there are lower values than the root values.
- Each block is at least 50% full, but empty entries are kept for expansion, to minimize chance of block splits.

references:

Number of Levels and Disc I/O

- binary search efficient for unique keys

  example: find key = "15"



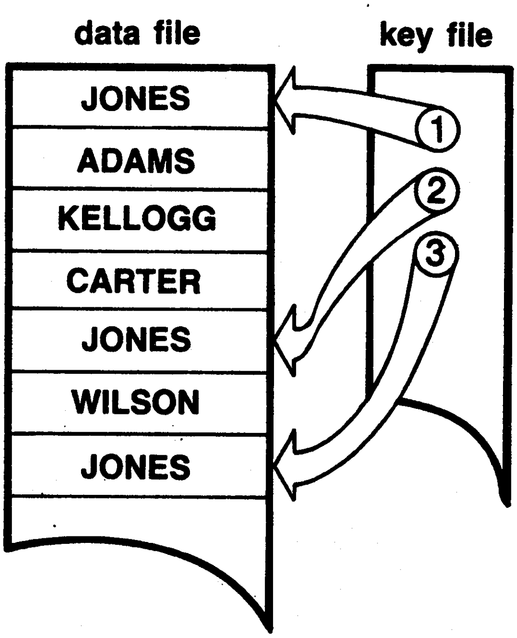
- three levels in tree – up to three disc accesses

notes:

- Because of binary search, only one disc access is needed for each level. The root level block may already be in the buffer from previous accesses, so the disc I/O to bring in the root block is often unnecessary.

references:

Creating Duplicate Keys



key file built with DUP
(key order chronological)

- order of keys depends on how file created

DUP
- chronological order
- updates show

RDUP
- keys in random order
- updates as fast as any other key

notes:

- Adding new keys is always time consuming; adding duplicate keys is even more so.
- Trying to maintain chronological order adds to the overhead of adding new duplicate keys. So, use RDUP unless chronological order is essential.

references:

Accessing Duplicate Keys

- KSAM maintains "chain" of duplicate keys in key file
- read by key gets first key in chain
- long duplicate keys slow to access
  - lose advantage of B-tree structure
  - binary search inefficient for duplicate chains

HEWLETT PACKARD

notes:

- The binary search technique is designed for unique keys. It is extremely inefficient for accessing duplicate keys and can double the disc I/O needed to find the start of a long duplicate key chain. So, keep duplicate chains short! Don't choose "male/female" as a key.

references:

WORKSESSION IV - 5

notes:

references:

Worksession IV-5 (selecting keys)

Assume a file with the following items in each record:

Customer Name (in format: First Last Initial)
Street Address
City
State
Zip Code
Phone Number
Purchase Order Number

Suppose the application needs to

a) Update customer information given only the customer name
b) Add new customers
c) Mail literature by zip code
d) Retrieve the purchase order number for a particular customer
e) List in alphabetical order all customers with the same last name

1. How many keys do you need?

____

   Which items did you select as keys? ______________________________

____

____

2. Does any key item need to be modified? If so, which?

____

____

____

3. Are any keys duplicates? If so, which?

____

____

____

4. If you need a duplicate key, would you make it a key added in chronological order (DUP) or in random order (RDUP)? Why?

____

____

____

____

Worksession IV-5 (cont.)

5. Which function (a, b, c, d, or e) do you think produces the most overhead? (Assume the customer name does not change.) Explain.

6. Which function (a, b, c, d, or e) produces the least overhead? Explain your answer.

7. Which function (a, b, c, d, or e) is especially suitable for KSAM? Explain your answer.

8. Can you think of any ways to reduce the number of keys or to make your keys more effective?

USING KSAM FILES

- Key Blocks and Buffers
- Shared Files

HEWLETT PACKARD

notes:

references:

Key Blocksize and B-tree Levels

- KSAM default good for most files, but you can change key blocksize
- in general - large blocks tend to reduce number of levels
- experiment to determine best blocksize that does not increase number of levels
- remember, the more levels, the more disc I/O for access

notes:

references:

Choosing a Key Blocksize

- **consider method of access**
  - larger blocks for sequential access
  - smaller blocks for random access

- **consider number of levels in B-tree**

  problem:

  A) blocksize is large, levels in B-tree = 2

  B) blocksize is smaller, levels in B-tree = 2

  C) blocksize is very small, levels in B-tree = 3

  which blocksize (A, B, or C) would you choose
  - for random access?
  - for sequential access?

notes:

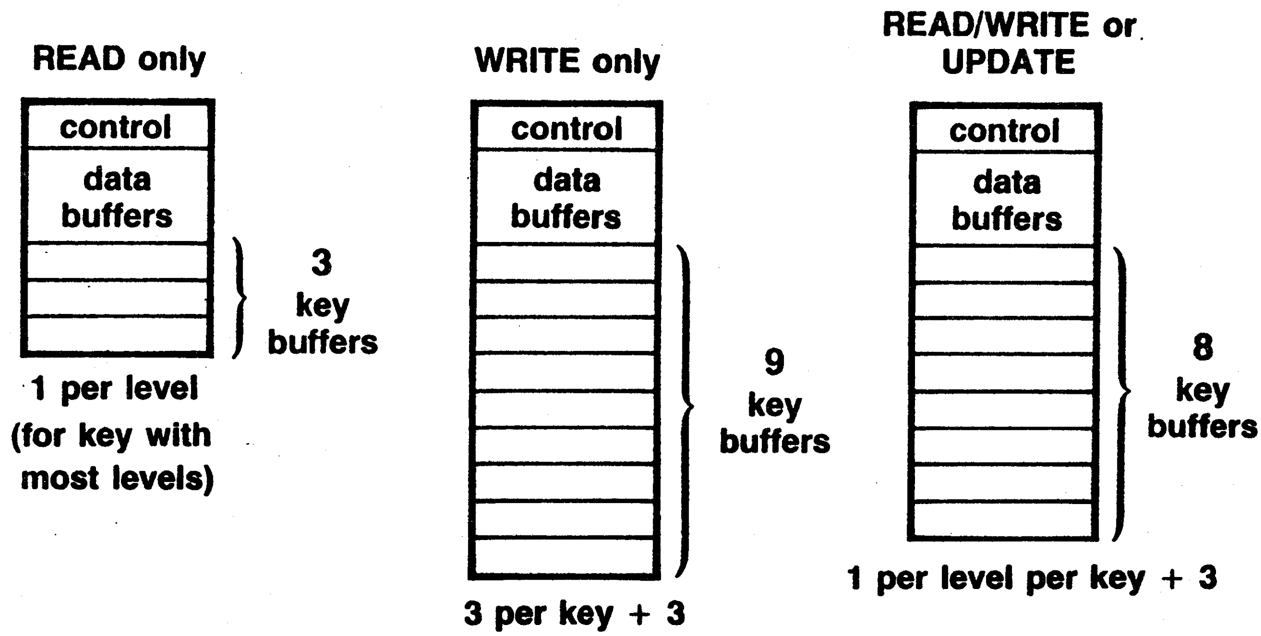
Blocking is a permanent characteristic of file

- Choose B) for random access because you want the smallest block size that does not increase the number of levels.
- Choose A) for sequential access since it gives you both a large block and few levels.
- There is no good reason to choose C) since it increases the number of levels with consequent increase in disc I/O.

references:

Default Number of Key Buffers

assume: file has 2 keys; primary key has 2 levels, and alternate key has 3 levels:



notes:

- The data file uses a single buffer - this cannot be changed.
- The number of key file buffers can be increased or reduced if the default is not working well.
- The default is based on a combination of access mode, number of keys, and number of levels in the B-tree.

IS usually the most efficient

references:

EXPERIMENT with Number of Buffers

If default not satisfactory —

- **you can change number of buffers any time file is opened**
- **consider more buffers**
  - **for loading data into file**
  - **when there are few other users on system**

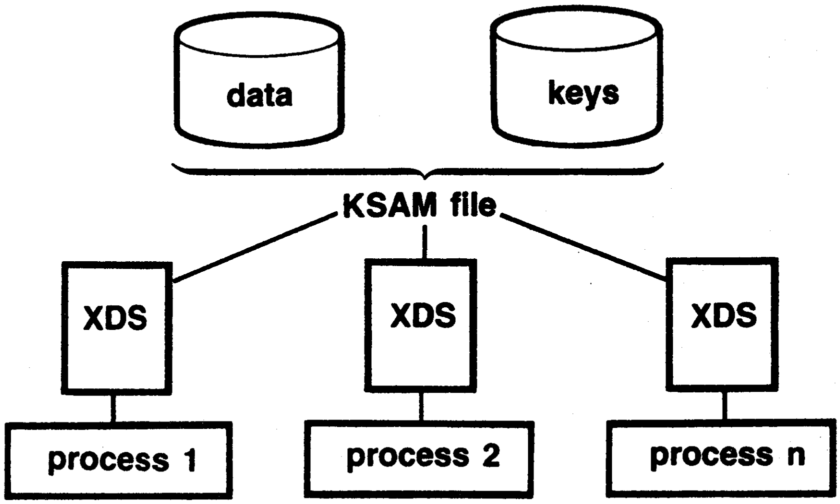
notes:

- Generally, the default works well. Still, there are situations when the default can be improved on.
- Each additional buffer increases the size of the extra data segment that holds the buffers, (an extra data segment is maintained for each open KSAM file).

references:

Extra Data Segments

- 1 per open KSAM file
- contain data buffers, key buffers, control blocks



notes:

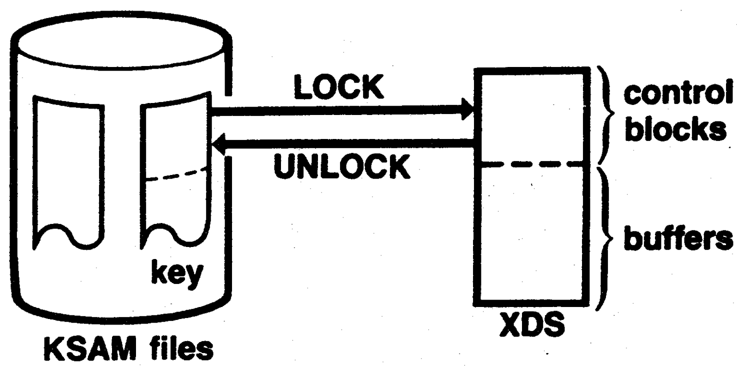
- The separate extra data segments and private control blocks add to the overhead of using KSAM files.
- Private control blocks mean the record pointers and the current EOFs are not shared.

references:

USE LOCKING!

LOCK – writes CONTROL BLOCKS from file to XDS

UNLOCK – writes CONTROL BLOCKS back from XDS to file



- causes extra disc I/O

notes:

- Locking insures that the latest record pointers, and the EOFs for both the key and data files, are posted to the file before the file is accessed by any other process.

references:

Lock around Transactions

- lock before moving pointer when access is pointer dependent
- example:

```
LOCK
   READBYKEY  ← positions pointer
   REWRITE    ← uses pointer
UNLOCK
```


notes:

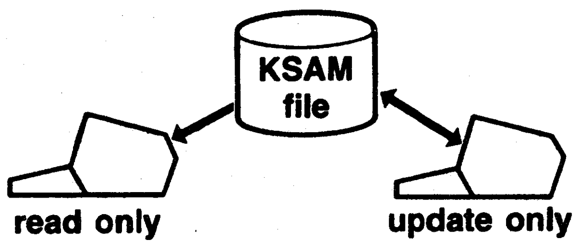
- Some procdures (such as rewrite or delete) depend on the current pointer being positioned correctly.
- Others (such as a key read) position the pointer at a particular record.
- Still others (such as sequential read) advance the pointer or leave it where it is depending on the preceding procedure.

references:

Structure Transactions (1)

to reduce overhead when files are shared:

- **separate reads from updates**
  - ~~use different terminals~~
  - if possible, at different times

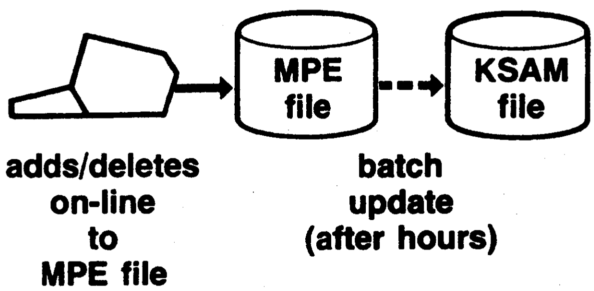


notes:

references:

Structure Transactions (2)

- add or delete records in "batch"
  1. Enter new records, deletes on-line to MPE transaction file
  2. Update KSAM file from transaction file as batch job



notes:

references:

WORKSESSION IV - 6

notes:

references:

Worksession (IV-6) (using KSAM files)

1. Suppose you specify a small key block size in an attempt to reduce your key buffer size for a random access operation. You find that access to the file is slower than it was before you reduced the key block size. You then run KSAMUTIL and find that there is a 4-level key in the file whereas there used to be at most 3 levels.

   A. Explain why reducing the block size made access slower.

   More disc I/O

   B. What would you do in this case to make the disc access faster?

2. Which is easier to do: change key block size or the number of key buffers? Explain.

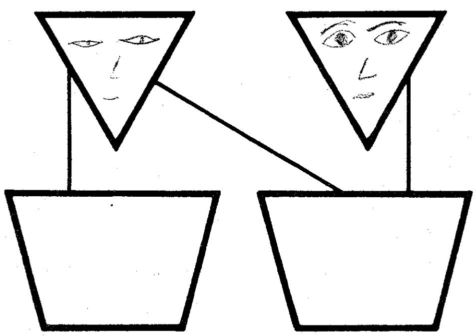
3. Two programs both access the same KSAM file. One makes on-line updates to the file; the other produces daily reports based on a sequential retrieval of all the records in the file.

   A. Explain why these programs must both lock the file when accessing it simultaneously.

   So that updating doesn't occur when reading & reading doesn't occur when updating

   B. Is there any way both these programs can execute without locking the KSAM file? Explain.

IMAGE/QUERY



- Data Base Definition
- using IMAGE
- IMAGE Structure
- QUERY

notes:

references:

DATA BASE DEFINITION

An Overview of

- **IMAGE concepts** - Masters and to us manual
- **Passwords and Security**
- **Multiple Data Bases**

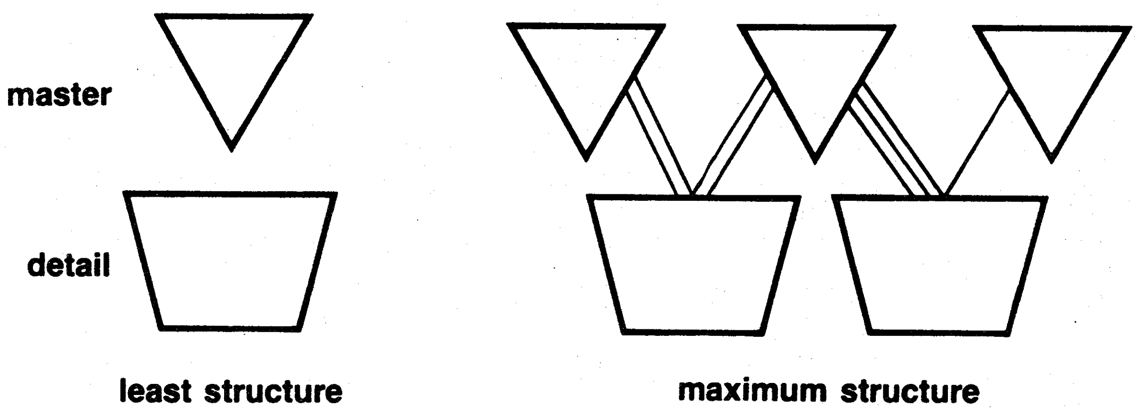
HEWLETT PACKARD

notes:

references:

What is IMAGE?

- a structured collection of data sets



stand alone — no pointers, like flat files but with [illegible]

notes:

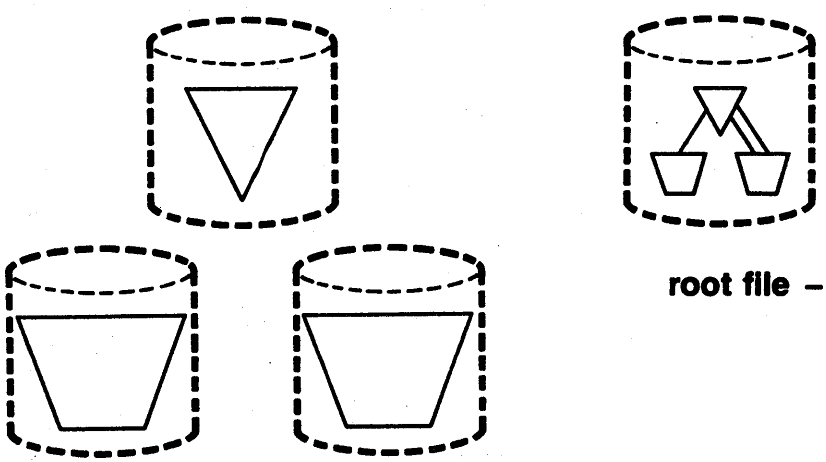
- Each data set is an MPE file. Data sets can be stand-alone (not connected to another data set) or many data sets can be connected through multiple "paths".

- This wide range of structure allow data bases to be tailored to the application.

references:

IMAGE Structure

- each data set is a file
- linked through "root file" containing data base definition



root file – contains "IMAGE" of data base

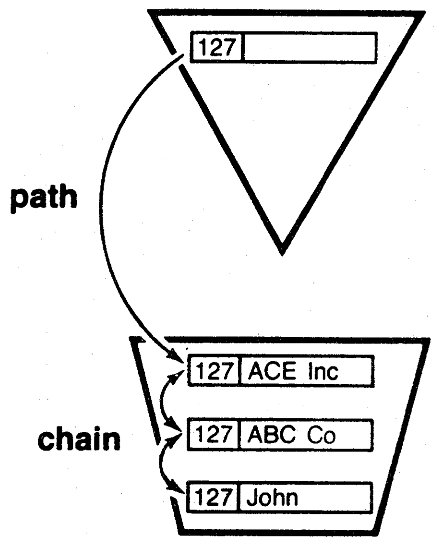
data base files

notes:

- The root file contains a full description of the data base, including all paths, chains, item definitions, passwords, etc.
- The root file is shared by all users of the same data base.

references:

Paths and Chains



Paths link master to detail data sets

- use as few as possible
- select search items that change infrequently
- choose the most-used search item as the head of a primary path

Chains link items in detail with the same search item values

- length of chain not significant
- access in either direction
- search items must not be sort items

notes: Quick retrieval means fewer disc I/O

references:

Entries and Items

- an IMAGE entry corresponds to a record
- an IMAGE item corresponds to a field within a record

item within entry

acct	name	date	code
12345	JONES JOHN H	051180	2 5
98765	MARTIN MARY X	061180	03

entry

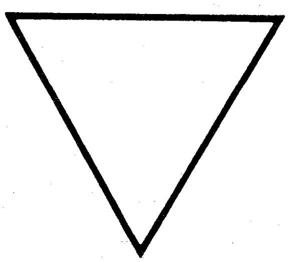
HEWLETT PACKARD

notes:

- Special item types are search items and sort items.
- Search items define the "paths"; the same search time must be in a master and its associated detail. They also define the "chains" which are simply search itmes with the same value in a detail data set.
- Sort items (which must not be search items) are items on which a chain can be sorted.

references:

Master Data Sets



- **MANUAL**
  - search item (key) PLUS data
  - may be stand-alone
  - values must be added by program
  - provide direct control over data

- **AUTOMATIC**
  - search items only
  - values added automatically
  - must be linked to a detail
  - good when search items are numerous or have many unique values
  - saves coding effort

notes:

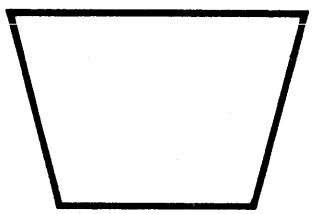
- Use master data sets for search items (keys), and for one-of-a-kind data.

- Basically, masters provide the key to the bulk of the data, which is stored in details and often has duplicate key values.

references:

Detail Data Sets

details data sets – linked to masters by paths



- use for duplicate values
  - duplicate items are linked in "chains"
- use for values linked to more than one master
- use for any items that must be sorted

notes:

- Good for such items as: sales records, purchase orders, shipments, that can be associated with several masters and that are repeated items linked through duplicate search items (key) values.
- Sort items can only be in detail sets.

references:

Choosing a Structure

- multiple paths
  - for stable data that seldom changes
  - for inquiry-type applications
  - to avoid redundant data
- stand-alone data sets
  - to provide IMAGE security and logging
  - for shared buffers in shared environment
  - for QUERY access

the structure of the data base should reflect the structure of the organization

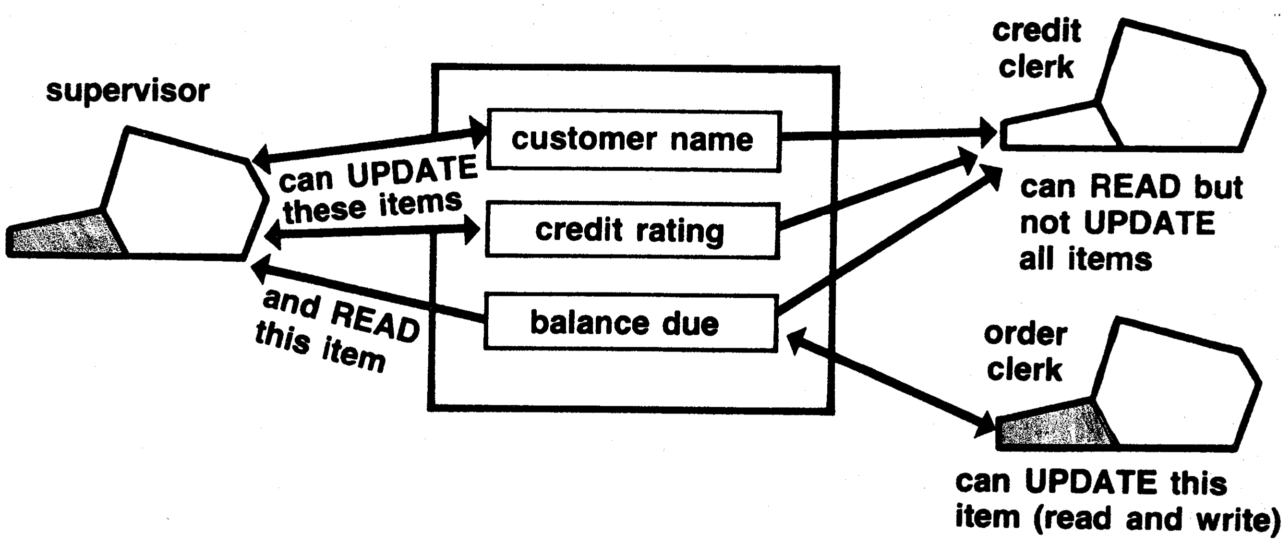
notes:

- Only define paths that are absolutely necessary.
- Each additional path increases the complexity of the data base; this, in turn, adds to the overhead particularly for modifying the data.

references:

Passwords and User Classes

■ provides access on "need-to-know" basis



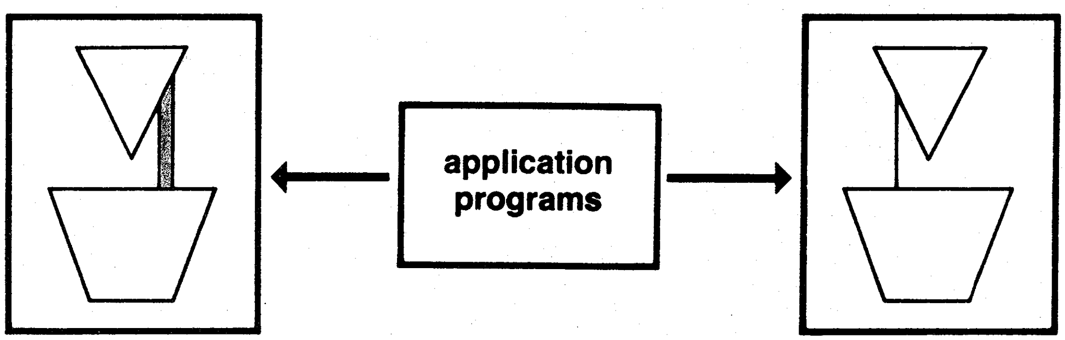
notes:

- Passwords are associated with classes of users.
- Each user class is allowed to perform specific tasks on specific items.
- There can be up to 63 different user classes. Thus, the access restrictions can be extremely precise.

references:

Multiple Data Bases

- consider more than one data base
  - for complex applications with many items



parts description inventory control

HEWLETT PACKARD

notes:

- Multiple data bases may be a good way to reflect application needs.

- They are also a solution if there are too many items for a single data base (more than 255). But, this is not the only solution. There are usually other ways to reduce the number of items; for example by combining several into one item.

references:

Multiple Data Bases

ADVANTAGES
- backup and recovery faster per data base
- simplify individual data base structure
- more appropriate for some applications

DISADVANTAGES
- more overhead
- some redundant data
- logging more complex

notes:

- If you use multiple data bases, try to open only one at a time, particularly if the data bases are accessed by the same process.

references:

Some Guidelines

- use master sets for unique items
  - automatic masters for many unique search items
  - manual masters for data plus search items
- use detail sets for:
  - repeated (duplicate) items
  - items linked to more than 1 master
- use only those paths that are really needed
  - make most-used path the "primary" path

HEWLETT PACKARD

notes:

logging - see [illegible]

references:

WORKSESSION IV - 7

notes:

references:

Worksession IV-7 (data base definition)

Assume you want to put the following items in a data base:

Customer name
Street address
City
State
Zip code
Phone number
Part description
Order number
Price (of part)
Part number
Purchase order
Quantity (of part ordered)
Unit measure (by which part is ordered)

And suppcse your application wants to perform the following functions:

a) Locate an order by its number
b) Retrieve the part ordered, the quantity ordered, the unit price, and the total price.
c) Bill the customer referring to his purchase order and the particular order number.

Answer the following questions:

1. Which item(s) would you put in a master data set(s). Explain.

Name
addr
phone #

2. Which item(s) would you put in detail data set(s)? Explain.

part descript. purchase order
order # Quant
price Unit measure
part #

Worksession IV-7 (cont.)

3. Identify any search item(s).

part order
name

4. How many paths (total) does your data base have?

3

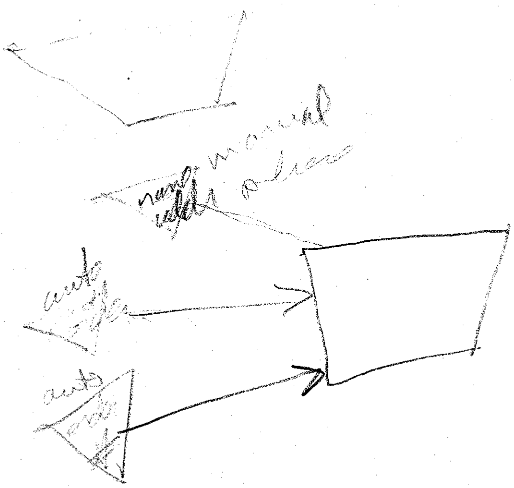
5. Are any sort items needed? If so, which?

6. Suppose you wanted another item to contain the order date, and you wanted to list all orders according to this date. How would you redesign your data base?

Auto master containing order #

Worksession IV-7(cont.)

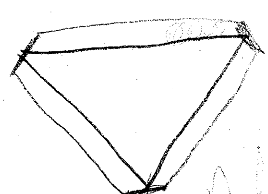
7. Draw a diagram of the data base you have designed (optionally including the order date from question 6). Show where items go and which head paths, etc.



Master

Key: order #

customer name + addr



Detail

Key: order #

description

USING IMAGE

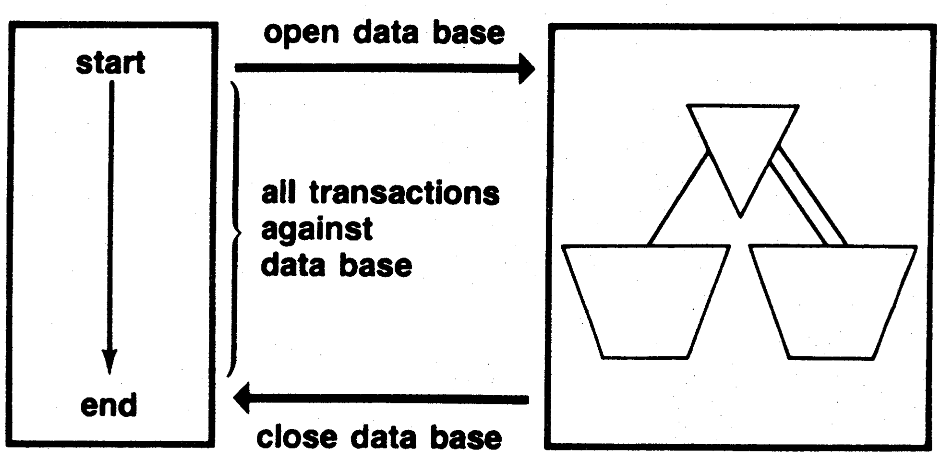
- Opens and Closes
- Types of Access
- Locking Strategies
- Maintenance

notes:

references:

Opens and Closes

- use even more overhead than MPE or KSAM files
- limit to once per process



HEWLETT PACKARD

notes:

references:

How to Open

- variety of modes
  - combine type of access (read, update, modify) with type of use (exclusive, shared)
  - coordinate all data base usage
- select the mode that
  - uses the least capability to do the job
  - allows concurrent users sufficient power to do their tasks

notes:

- Modes with low capability (such as read only) mean the least overhead.
- When other users plan to add or delete entries, use a mode that allows them such access and protect your process by allowing locks.

references:

How to Close

■ consider options other than full close

1. Close entire data base – required before exit from program

2. Close individual data set – to release all resources (except locks)

3. "rewind data set – to reset dynamic pointers, maintain current path

notes:

- Mode 2 close uses less overhead than a full close, but it makes no sense to close a data set if you plan to open it again. In such a case, leave it open until you no longer need it or are ready to close the entire data base.

- Mode 3 is useful before a serial read of the entire data set.

references:

Access Modes

- **serial** – essentially a simple serial access
- **directed** – random by record number
- **chained** – all items in detail with same search value
- **calculated** – locate item by particular value

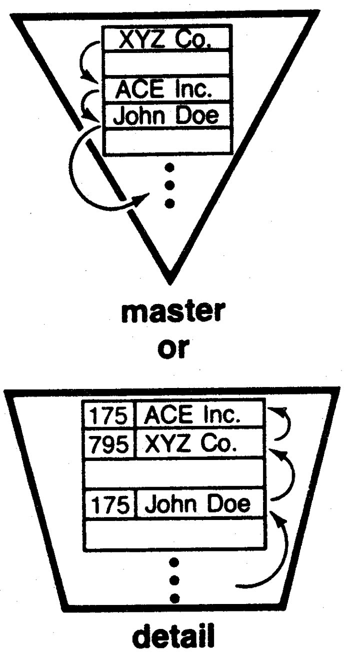
HEWLETT PACKARD

notes:

- Serial and directed access are very much the same as sequential and random access to an MPE file.
- Chained access is essentially duplicate key access. This is the type of access for which IMAGE is very well suited.
- Calculated access is for Master data sets only.

references:

Serial Access



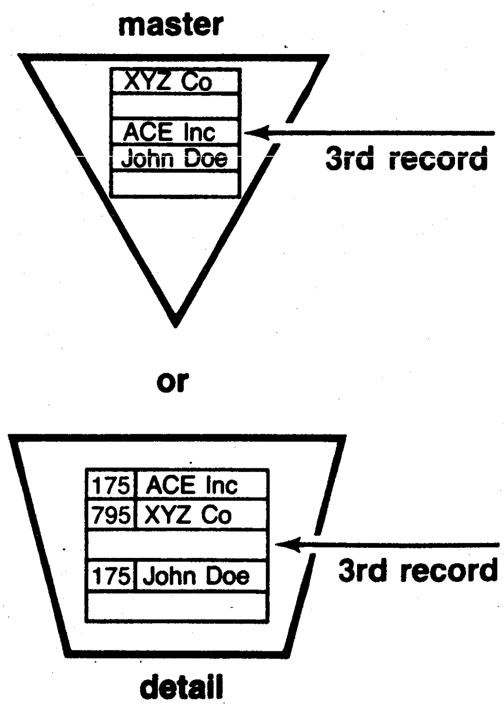
- reads sequentially through MASTER or DETAIL
- can read forward or backward
- useful to retrieve all or most of data in a data set
- use it to copy data to MPE file for subsequent sorting

notes:

- This access type reads records in the order they are stored in the data set.
- Empty records are skipped.

references:

Directed Access



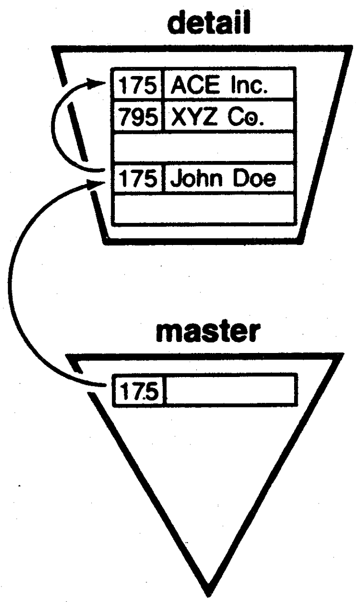
- read entry by its position in data set
- when accessing masters – records may move
- avoid unless you have exclusive access
- may access empty entry
- useful only if record number has previously been determined
- fastest access method

notes:

- This type of access can generate serious errors if used in a shared environment where adds and/or deletes can change the location of particular records.
- Note that IMAGE re-uses the space freed by delete records.

references:

Chained Access



- locates entries in "chain" with the same search item
- must first locate "chain head" in master
- useful for locating related events
- chains on primary key loaded in chronological order
  - may change since IMAGE re-uses space from deleted items

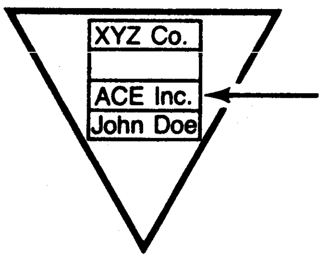
notes:

- Chains can be followed forward or backward.
- Note that this is basically a duplicate key search, the type of access for which IMAGE, unlike KSAM, is very well suited.
- Chain order of a primary search item can be physically maintained by reloading data.

references:

Calculated Access

master only



get "Ace Inc."

- locate entry with particular value
- useful for fast retrieval of data for particular use
  - find customer name/credit rating
  - find stock number/amount in stock
- best for manual master where entry contains data in addition to search item.

notes:

- This type of access is only for master sets where search item must be unique.

references:

Locking Strategies

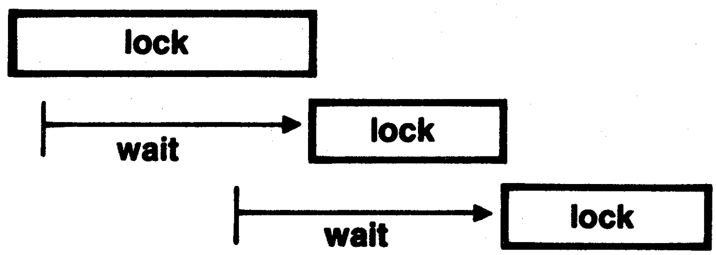
- IMAGE allows locking
  - of the entire data base
  - of individual data sets
  - of individual entries
    - by item value

notes:

- IMAGE locking, unlike MPE and KSAM locks, is enforced; it is more than a gentleman's agreement. When the data base (or data set or item) is locked, no other user can ignore the lock.

references:

① DATA BASE LOCKS



- avoid for long transactions
- saves lock/unlock overhead

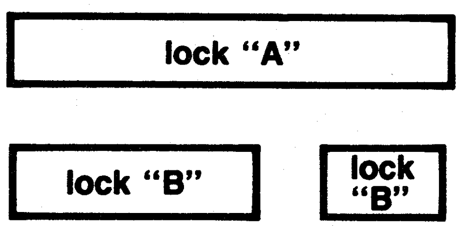
notes:

- This type of lock uses the least overhead to apply, but all sharing users must wait until the data base is released before they can access it.
- Use it mainly if short transactions mean a short wait.

references:

② LOWER LEVEL LOCKS

- **data set locks**



- lock a long transaction in data set A
- lock 2 short transactions in data set B

- **allows concurrent locking within same data base**
- **best for transactions in different data sets**

notes:

- This locking method requires more overhead.
- Provides faster response, but only if concurrent locks are applied to different data sets.

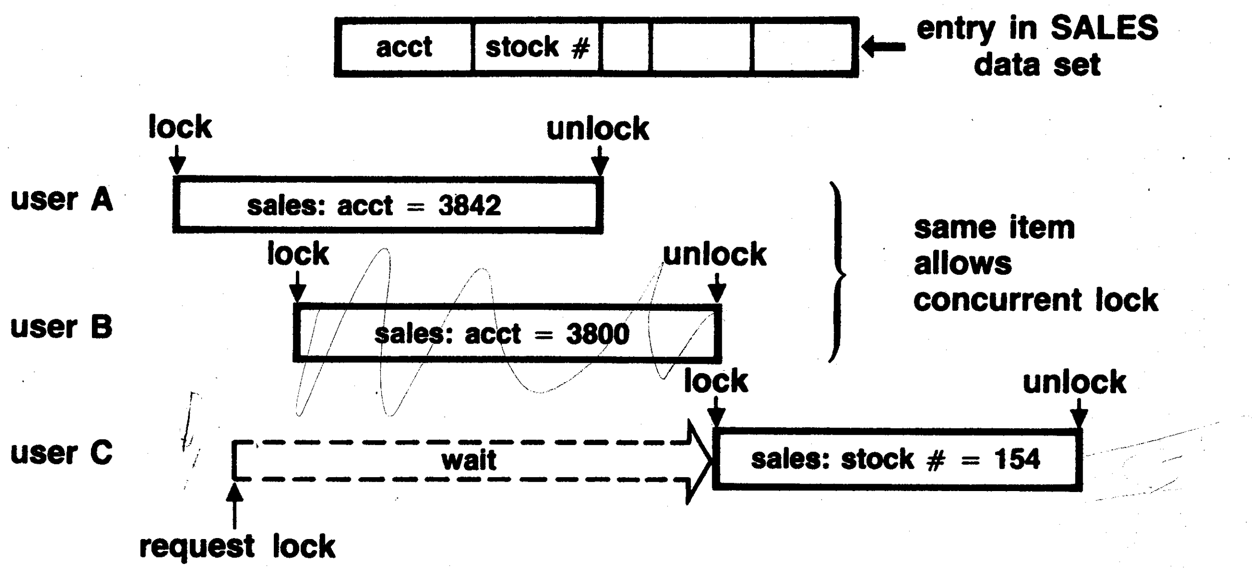
references:

③ DATA ITEM LOCKS

- every user should lock on same item

usually the search item

example:



notes:

- This locking scheme allows concurrent access to the same data set, but only if the locks are applied to the same item. Otherwise, the entire data set must be locked since there is no way to tell that the two different item values are not in the same entry.

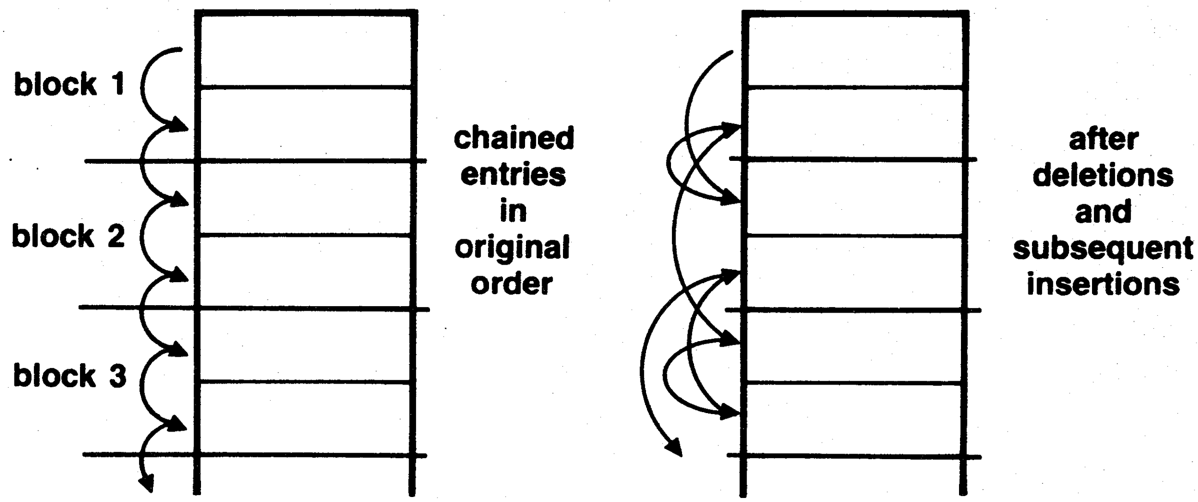
references:

Deleting Entries

- unlike KSAM, IMAGE re-uses space from deleted entries

BUT

- when entries inserted into free space:
  - loses benefits of chain
  - can effect access time when across blocks



HEWLETT PACKARD

notes:

Do chained unload to fix this

references:

IMAGE Utilities

used by Data Base Administrator to:

- **create data base**
- **maintain data base**
  - backup to tape (store/restore data)
  - restructure (unload/reload data)
  - monitor activity (logging)
  - recreate data from log tape (recover)

notes:

- Loading data must be done programmatically (only reloading provided through utility) except for small volumes of data that can be loaded on-line with QUERY.
- All data base access is also programmatic, unless QUERY is used for small-scale inquire or update.

references:

Tips on Using Image

- open file with the least capability that allows all concurrent users to function
  - open/close infrequently

253 db files per process.

- use appropriate access mode for task
- lock at lowest level that satisfies all users
  - at entry level, lock on same item
- reload data if very changeable
  - to compact chains
  - avoid disc fragmentation

separate masters & details across drives. Let a [illegible] report just fit to a system [illegible] the time for you.

HEWLETT PACKARD

notes:

- When coding IMAGE calls, reference data items by name once, to preserve data independence, then use the item number to save time.

- Use an * to back reference named items, but avoid using @ to mean "all items" since data base may change.

references:

WORKSESSION IV - 8

notes:

references:

Worksession IV-8 (using IMAGE)

1. Suppose you are the only user of the system and want to retrieve information from the data base, compare it with some data in your program and then update a non-key item. Would you open the data base for:

   a) exclusive read access
   b) read access allowing other users to read also
   c) read access allowing other users to modify the file
   d) update access allowing other users to update also
   e) exclusive modify access
   f) modify access allowing other users to read
   g) modify access allowing other users to modify also?

   Explain your answer.

2. Using the same list, how would you open the file if other users planned to read the data base? Explain.

3. Using the same list and the previous scenario, how should the other users open the data base? Explain.

Worksession IV-8 (cont.)

4. If you want to retrieve data associated with a particular search item in a master data set, what access mode would you use? Why?

   a) serial
   b) directed
   c) chained
   d) calculated

5. If you want to locate all entries with a particular order number (the order number is a search item), what access mode would you choose? Explain.

6. In this same situation (question 5), which data set(s) would you access? Why?

   a) a detail data set
   b) a master data set
   c) a master and a detail data set

Worksession IV-8 (cont.)

7. Suppose there are 15 concurrent users of the same data base, all users need to access the same data sets, and these accesses may include verification of data and subsequent changes. What level of locking would you choose? Why?

   a) data base level
   b) data set level
   c) data entry level

8. Is there anything you can do to make the locking strategy you chose more efficient?

IMAGE INTERNAL STRUCTURE

- Media records
- Synonym chains
- Sorted chains
- IMAGE as a set of files

notes:

references:

Media Records

■ include chain/path control information plus actual data

- for master data sets

	5 words	5 words	5 words		
record number	synonym chain, head or link	1st path chain head	2nd path chain head	......	entry

- for detail data sets

record number	1st backward pointer	path forward pointer	2nd backward pointer	path forward pointer	...	entry
	4 words		4 words			

notes:

- In master sets, media records define chain heads, including "synonym" chain head.

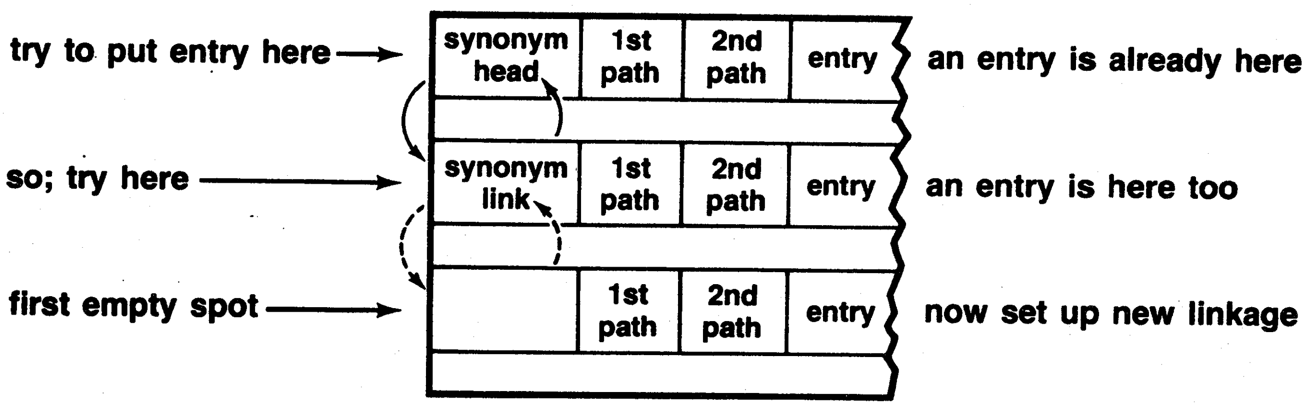
- In detail sets, media records contain backward and forward pointers for every chain.

- When data base is highly structured, pointers in media record may be longer than actual data entry.

references:

Synonym Chains

- cause migrating secondaries
- address of entry in master is calculated with algorithm
- what if same address is calculated for several unique values?



notes:

- The algorithm used to calculate the entry location in a master data set uses two variables you can control: the search item value and the number used to specify the data set capacity (how many entries are expected).

references:

Long Synonym Chains

- caused by many entries directed to same address
- make access slower
- adds/deletes much slower

reduce number of synonyms by:

- allowing 20% extra space in master
- use prime number for master capacity
- use ASCII-type search keys or integers with random values

notes:

- Access is slower because the chain must be followed to locate the correct entry. This takes time and may mean extra disc I/O.
- Adds and deletes are even slower since the synonym pointers must be modified once the correct location is found.

references:

Sorted Chains

- keep sorted chains SHORT
  - unless chain is static or
  - used primarily for inquiry

- put sort items at end of entry
  - all items following sort item included in sort

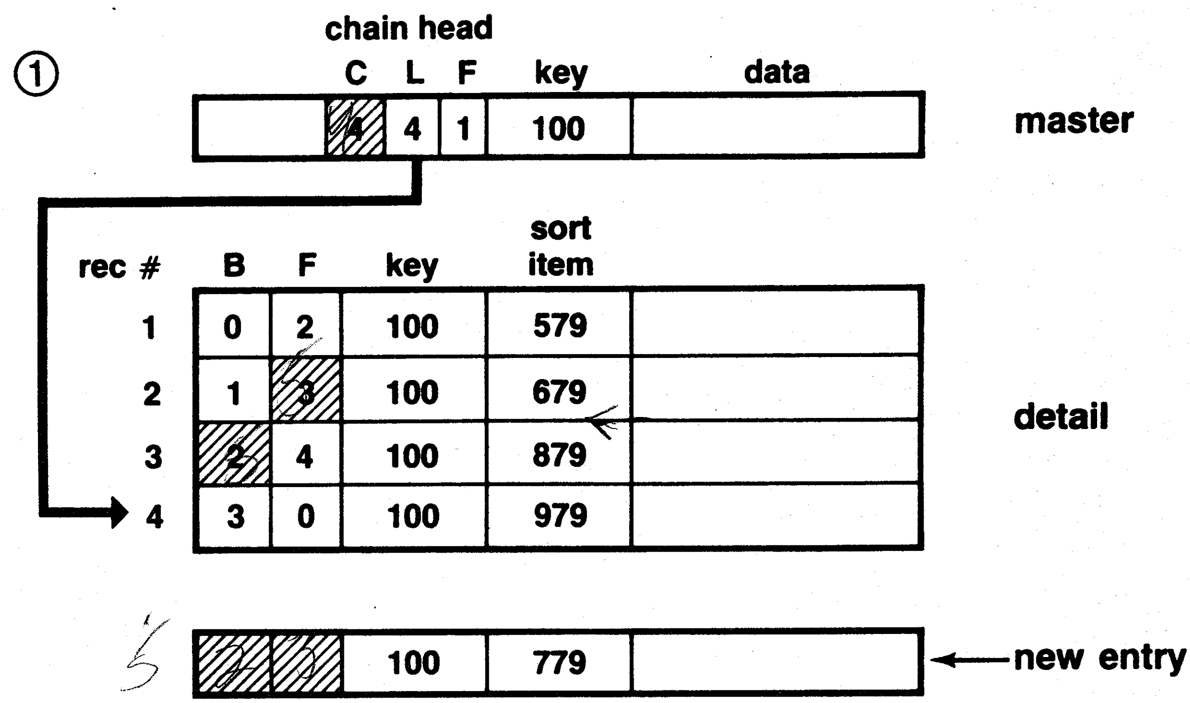
notes:

- IMAGE is not designed to manage long sorted chains. It is particularly time-consuming to add or delete sorted entries.

references:

IMAGE structure

Add a Sorted Entry



■ what pointers must be modified?

HEWLETT PACKARD

notes:

- Each of the shaded pointers must be modified; these are the count in the master set, the forward pointer logically preceding the new entry, and the backward pointer logically following the new entry. For an unsorted entry, only the count need be modified.

- If the pointers that must be changed are in different blocks more disc I/O is required.

references:

Put Sort Item At End

- to reduce sort overhead
- unless you want subsequent items to be part of sort

	pointers B	pointers F	search key	unsorted data	sort item		sort order
1	3	0	100		879	ADAMS	4
2	4	3	100		779	GREEN	2
3	2	1	100		779	JONES	3
4	0	2	100		679	BROWN	1

extended sort field

notes:

- The extended sort field does allow subsidiary sorts. In effect, items following the sort items are minor keys.

references:

Design Tips to Improve Performance

SORT ITEMS
- avoid them, but, if necessary
  - sort only sets with few values
  - put sort items at end of entry

CAPACITY
- make it realistic to save disc space, then add 20% for master data set
- make master set capacity a prime number

SEARCH ITEMS
- use type ASCII items, if possible
- if integer, use random valued items

notes:

references:

IMAGE As a Set of Files

- **consider blocksize**
  - can only be changed at create time
  - depends on type of usage

- **consider number of buffers**
  - can be changed ~~at each OPEN~~
  - increase for loading

HEWLETT PACKARD

notes:

references:

BLOCKSIZE

- default generally yields good performance
- increase or reduce size for special purpose applications

LARGE BLOCKS

- few users in batch mode
- memory is available
- most access is serial or chained (and chains are compact)

small blocks

- many users in session mode
- application is large, memory limited
- most access is direct or calculated

otes:

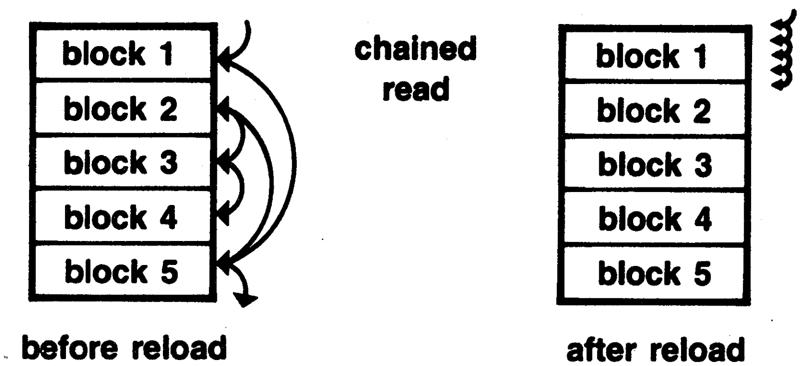
- Default is particularly suited to multi-purpose applications. Consider other block sizes if your application is devoted to one particular type of access.

ferences:

Blocks and Chained Reads

- IMAGE assigns contiguous storage to entries in a primary path

- RELOAD to force primary chains into contiguous locations – avoid crossing block boundaries



HEWLETT PACKARD

notes:

- Sort entries by primary search item value for initial load. IMAGE maintains this order at each reload, making items in primary chain contiguous.

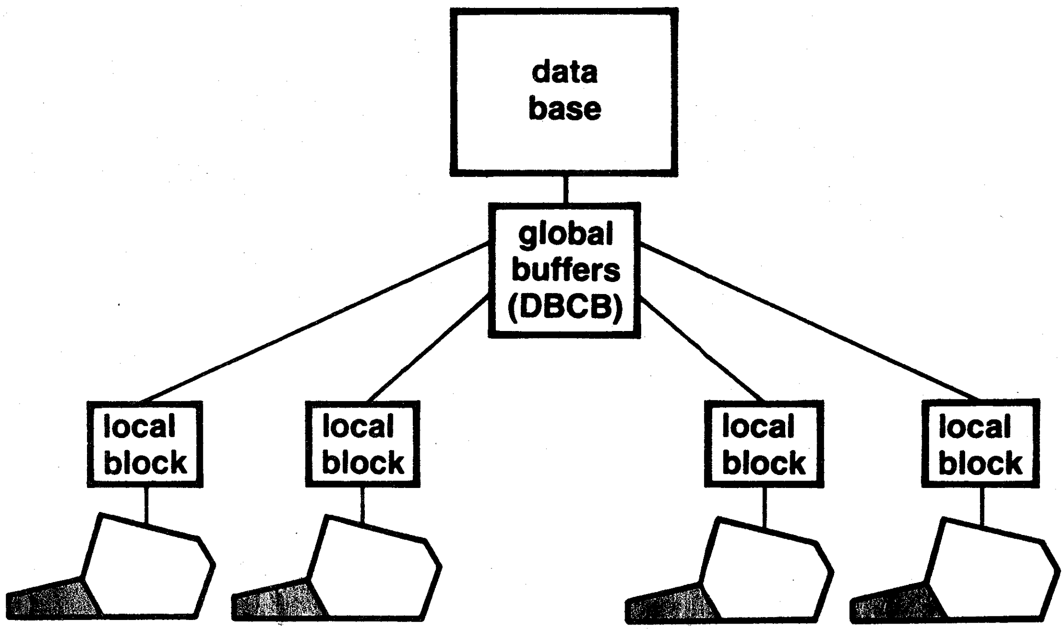
- Lots of adds and deletes wreak havoc with this order because freed space is re-used. So, reload if this occurs.

- Reload only helps the primary chain. This is why the primary path should be selected with care.

references:

BUFFERS

- global for shared access



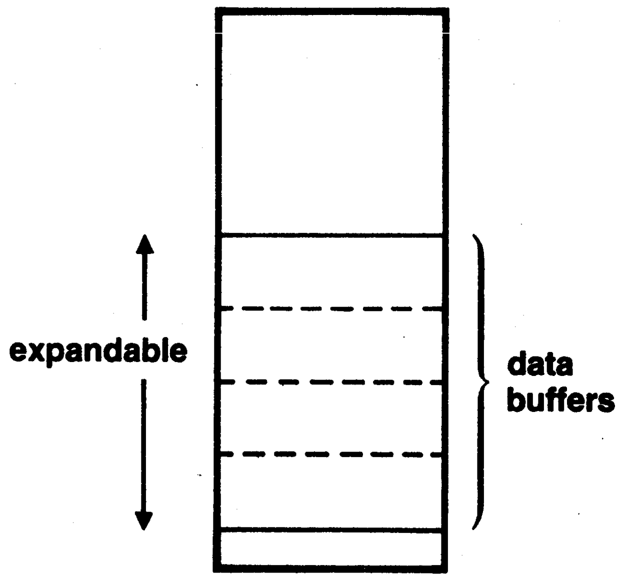
notes:

- IMAGE writes buffers to disc after every write or rewrite or deletion. Thus, it does not have true buffering.

- An output deferred option provides true buffering since it does not write buffers to the file until they are full. But, it can only be used in an exclusive environment when no other users are sharing the data base. Deferred output can be very useful to speed up after hours updates when only the update process is accessing the data base.

references:

How Many Buffers?



DBCB
(an extra data segment)

- increase number of buffers for loading data for batch jobs
- decrease number of buffers if memory size limited
- experiment!

notes:

- The data base administrator links the number of buffers to the number of users.
- The default is based on the number of search items plus the number of users.

references:

WORKSESSION IV - 9

notes:

references:

Worksession IV-9 (IMAGE structure)

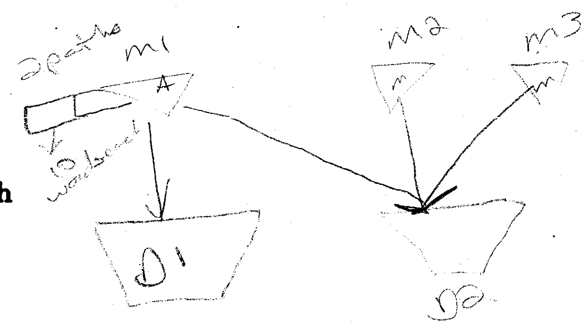
Given a data base with 5 data sets, ~~one~~ 3 masters (M), and two details (D1 and D2), with the following characteristics:

M1 is an automatic master with
1 search item,
no data items.

M2 and M3 are manual masters with
1 search item each,
1 data item each.

D1 is a detail with
1 search item linked to M1, plus
20 data items.

D2 is a detail with
3 search items, plus
12 data items, including
1 sort item.



Draw a diagram to illustrate this data base. Then answer the following questions.

1. Which of the two detail data sets (D1 or D2) will be faster to modify? Give your reasons.

D1 [illegible]

2. How many words are needed for each data set, in addition to the actual data? Explain.

m1 - 15 D1 - 4
m2 - 10 D2 - 12
m3 - 10

Worksession IV-9 (cont.)

3. Suppose the sort item is the customer's last name; what can you do

   A. to make sorted retrieval easier?

   Make the sort item the last item of the entry. Enter data in sorted order

   B. to provide a sort on the entire name?

   last, first, MI

4. How can you make a chained read of the set D1 easier?

   DB Mode 1 chained access

   Will this same technique work for D2? Explain.

5. What problems can you expect if the capacity of M1 is the nearest even number to the total expected number of entries, and the primary search item is a number that increases in increments of 1? Explain.

QUERY

- What QUERY does
- When to use QUERY

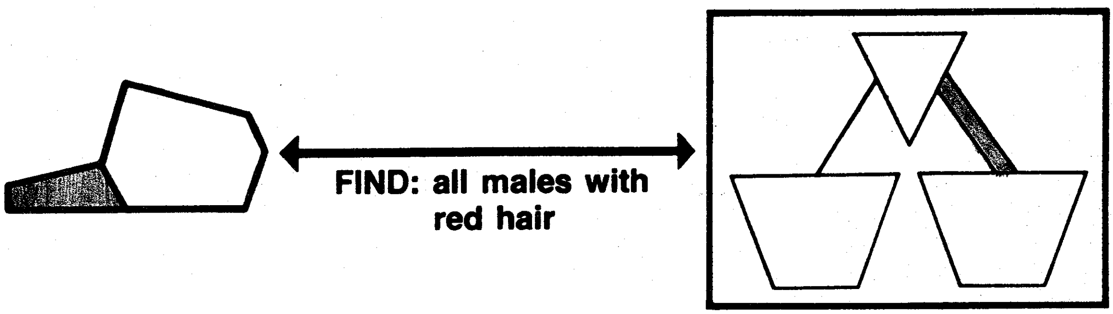
notes:

references:

Quick Retrieval

QUERY provides:

- non-programmatic, interactive access to data in an IMAGE data base



HEWLETT PACKARD

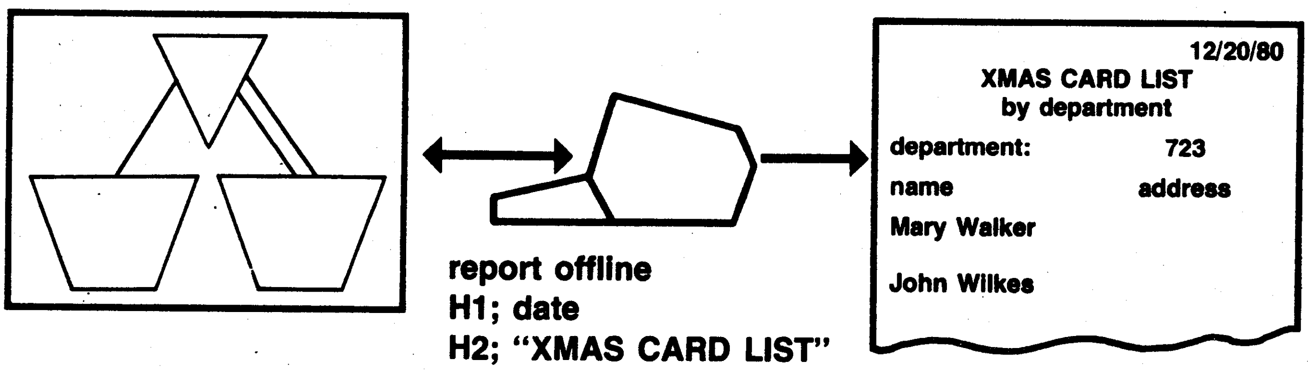
notes:

- QUERY is excellent for quick and highly specific on-line retrievals; it can look in any data set to locate entries that fit the specified criteria.

references:

Flexible Reporting

QUERY lets you format reports



- **best for impromptu, one-time reports**

HEWLETT PACKARD

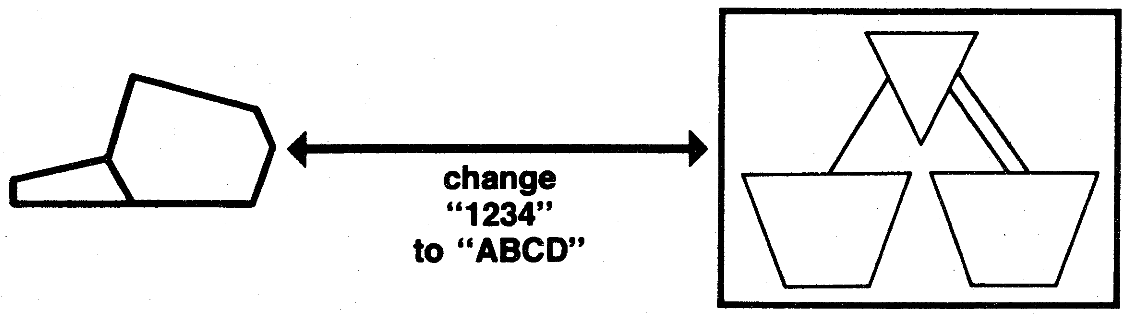
notes:

- Avoid using QUERY as your daily reporting mechanism. It is too general purpose to be efficient. Write a specific reporting routine to cut down on overhead.

references:

Low-Volume Modification

QUERY allows you to add or delete an entry or modify an existing entry



- **good for debugging and testing**

HEWLETT PACKARD

notes:

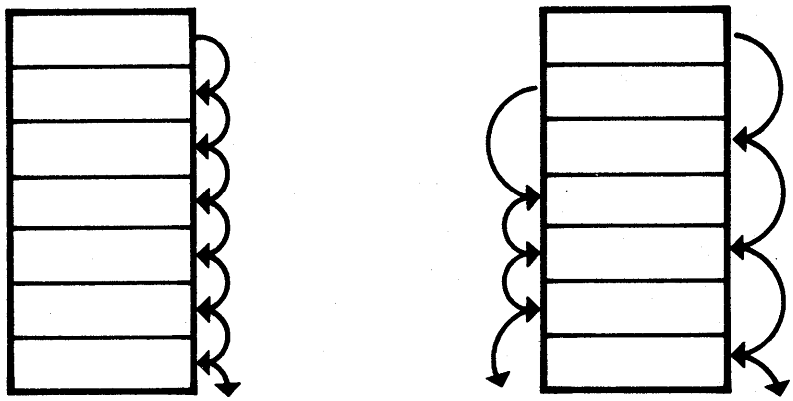
- Any regular, large-scale data base modification should be done programmatically in order to be efficient.
- QUERY is good for correcting small amounts of data, for testing changes, for verifying the structure during data base design.

references:

Access Mode

■ exact mode not easily predicted

serial read or chained read



- may look at entire file
- may have multiple chained reads

HEWLETT PACKARD

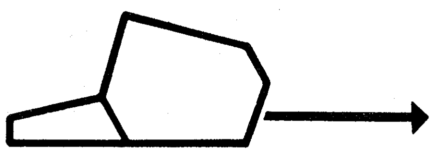
notes:

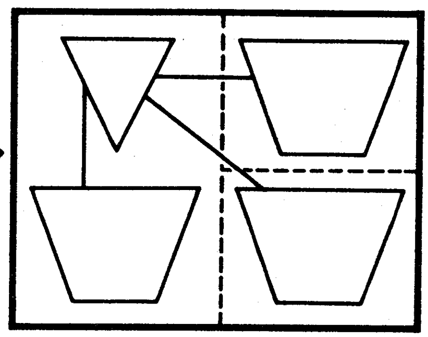
- Chained read uses more resources than serial read.
- QUERY uses serial read for non-key items, single key items, approximate items. For example:
  FIND NON-KEY="BLUE"; FIND KEY>100;
  FIND KEY=20 OR NON-KEY="BLUE"
  Each causes a serial read.
- Chained reads are chosen for multiple keys. For instance,
  FIND KEY = 1,2,3, causes 3 chained reads;
  FIND KEYA = 1 AND KEYB = 2 causes 1 chained read on KEYA.

references:

Locking

■ QUERY locks entire data ~~base~~ set for every FIND or REPORT





- while QUERY user FINDS an item and REPORTS
- all others WAIT!

- application A needs to lock data set
- applications B and C need to lock entries in another data set

HEWLETT PACKARD

tes:

- The entire data base is locked unless QUERY is told to set Locking OFF completely. This differs from user applications that can lock a data set or data entry.

ferences:

QUERY vs. User Application

- QUERY is general-purpose – not optimized for specific use
  - – works on any data base
  - – works with any data sets
- USER APPLICATION preferable for
  - – regular reports
  - – fast, streamlined access
  - – large-scale data entry, modification

HEWLETT PACKARD

notes:

references:

DEMONSTRATION

HEWLETT PACKARD

notes:

references:

SUMMARY

- when to use MPE files
- when to use KSAM files
- when to use IMAGE

HEWLETT PACKARD

notes:

references:

summary

When to Use MPE Files

- for logging and backup files
- when you expect to access entire file
- for "transaction" files
  - collect on-line updates to MPE file
  - transfer to structured file (KSAM or IMAGE) as a batch job
- for large-scale sorts

notes:

references:

When to Use KSAM

- require flexible key retrieval
  - generic or approximate keys
  - actual keys
- need data sorted in variety of ways
  - by primary or alternate key
  - in chronological order
- simple arrangements of data
  - not-hierarchical
  - no complex relations
- system 3 conversions

notes:

references:

summary

When to Use IMAGE

- if your application uses many files, consider consolidating data in data base
- need to retrieve many duplicate values
- need separate security for different user types
- need QUERY
  - for fast, structured reports
  - for debugging
  - for low-volume data entry
- need locking at entry level as well as at data set or data base level

notes:

references:

SUMMARY

SUMMARY

- **Design checklists**
  - code & data segments small, schedule
  - processing options
  - terminal options
  - data management options

- **Final remarks**

HEWLETT PACKARD

notes:

references:

CODE SEGMENTS

- stay in segment as long as possible – then stay out as long as possible
- keep segments approximately the same size
- avoid very large segments
- put seldom-executed code in a separate segment

notes:

references:

DATA STACK

- keep total stack as small as possible
- keep global area small – use dynamic area where possible
- shrink stack following unusual growth
- avoid unexpected stack growth

notes:

references:

LIBRARIES

- use Relocatable Libraries (RLs) for small, special-purpose routines private to the program
- use Segmented Libraries (SLs) for large, universal routines shared by many programs

notes:

references:

PROCESS OPTIONS

- use single process
  - to simplify development & testing
  - for small applications
  - with dynamic subprograms
- use parent/child process
  - to reduce stack size
  - to isolate end-user from system
  - for large/complex applications

notes:

references:

LANGUAGES

PASCAL

- use COBOL for applications with more I/O than computation
- use FORTRAN for applications that need efficient computation, little I/O
- use BASIC for applications that manipulate strings,arrays
- use RPG for batch-applications with reports – conversions from RPG machines
- use SPL for special-purpose routines within larger application
- use APL for array manipulation

notes:

references:

TERMINAL COMMUNICATIONS

- use character mode if
  - small amounts of terminal input
  - terminal input determines program flow
- use block mode for
  - masses of terminal input
  - data entry applications
  - cntrl-Y or break not needed
- use V/3000 for
  - on-line edits
  - easy block mode development

notes:

references:

V/3000

- make forms uniform size
- keep field specs short– use concise edits
- avoid re-painting screens
- open 1 forms file at a time
- execute with fast forms file

notes:

references:

FILE SYSTEM

- use small blocks for random access, large blocks for sequential access
- use single buffer for random access, two buffers for sequential access no buffers for fast multirec transfers
- open and close files infrequently
- lock if any sharing user changes file but – try to design so locks are unnecessary

notes:

references:

KSAM

- use KSAM for sorted, sequential access
- avoid keys that have many duplicate values
- use as few keys as possible
- select keys that don't change
- shared access requires locks around all transactions

notes:

references:

IMAGE

- use IMAGE
  - for multi-file applications
  - for data with long duplicate chains
- avoid sorted chains
- use as few paths as possible
- allow 20% extra capacity for master sets (and use prime number)
- lock at lowest level that provides
  - concurrent access to most users without too much overhead

notes:

references:

QUERY

- use for quick, one-time reports
- use to test and debug data base access
- avoid QUERY for
  - regular reporting
  - large scale data entry or retrieval

HEWLETT PACKARD

notes:

references:

General Rules

- do you really need that file? that module?
- put design effort where it counts
  - 80/20 rule
- consider when a task is needed
  - can it be batch? off hours?
- remember that others use the system
  - don't be a hog

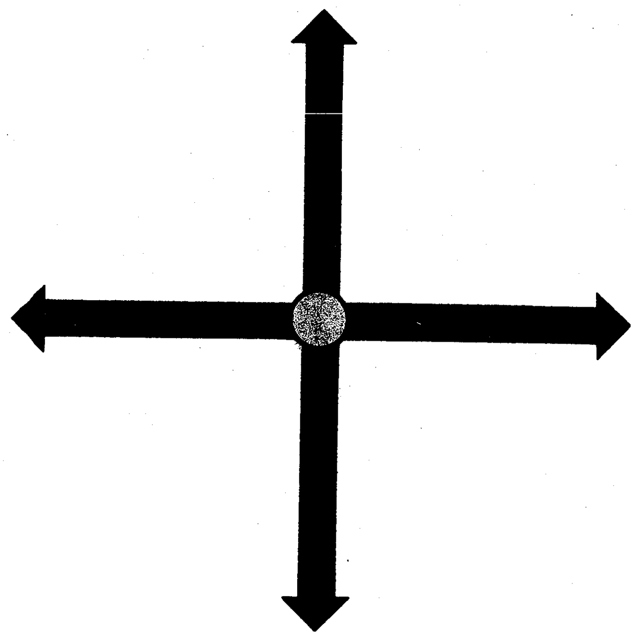
HEWLETT PACKARD

notes:

- 80/20 - 80 percent of the overhead occurs in 20 percent of the code; So, put 80 percent of your design effort in that 20 percent of the code.

references:

■ what do you really want?



HEWLETT PACKARD

notes:

references:

THE END

notes:

references:

APPENDIX A

Source Listings

1. Shrink Stack Examples

SPL subprogram
FORTRAN example
COBOL example

```
$CONTROL SUBPROGRAM
BEGIN
   PROCEDURE SHRINKSTACK(SIZE,CC);
   INTEGER SIZE,CC;
     BEGIN
       INTRINSIC ZSIZE;
       SIZE:=ZSIZE(SIZE);
       IF <> THEN CC:=-1 ELSE CC:=0;
     END;

   PROCEDURE STACKSIZE(SIZE);
   INTEGER SIZE;
     BEGIN
       PUSH(Z);
       SIZE:=TOS;
     END;
END.
```

```
C   FORTRAN Example using STACKSIZE and SHRINKSTACK
C
          PROGRAM MAIN
          INTEGER *2 SIZE,CC
C
C   Other data declarations could go here
C
C   Begin main code
C
          CALL STACKSIZE(SIZE)
          CALL BIGSUB(X,Y,Z)
C
C   Where BIGSUB is a subprogram that uses a lot of the local
C   stack.
C
          CALL SHRINKSTACK(SIZE,CC)
C
C   This returns the stack to the size it was before BIGSUB
C   was called.
C
C   End of FORTRAN example.
C
          STOP
          END
```

```
*  Sample COBOL routine using STACKSIZE and SHRINKSTACK
*
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.

     01  STACK-SIZE          PIC S9(4) COMP.
     01  CONDITION-CODE      PIC S9(4) COMP.

 PROCEDURE DIVISION.

 A100-MAIN.
*
*  Begin processing here.
*
 B100-SUB.

     CALL "STACKSIZE" USING STACK-SIZE.
     CALL "BIGSUB" USING PARMX, PARMY, PARMZ.

*  Where "BIGSUB" is a subprogram that uses a lot of
*  storage in the local area of the stack.
*
     CALL "SHRINKSTACK" USING STACK-SIZE, CONDITION-CODE.
*
*  This returns the stack to the size it was when
*  "STACKSIZE" was called.
*
*
*  End of COBOL example.
```

2. Demonstration II-1 Listings

Stream file
Source files:
  PDEMO1P
  ERRORSUB
PMAP

```
!JOB DEMO,DESIGN
!PURGE PDEMO1P
!PURGE UDEMO1U
!FILE COBTEXT=SDEMO1S
!FILE COBUSL=UDEMO1U
!FILE COBLIST=$NULL
!RUN COBOLII.PUB.SYS;PARM=5
!FILE COBTEXT=ERRORSUB
!RUN COBOLII.PUB.SYS;PARM=5
!PREP UDEMO1U,PDEMO1P;MAXDATA=11000;PMAP
!SAVE PDEMO1P
!EOJ
```

```
$CONTROL LIST, USLINIT, MAP, SOURCE
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  demo1.
 AUTHOR.  ct3000.
 DATE-COMPILED.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  error-flag                       pic S9(04)  COMP VALUE 0.
 77  exit-flag                        pic S9(04)  COMP VALUE 0.
 77  record-was-found                 PIC S9(04)  COMP VALUE 1.
 01  buffer.
     05  FILLER                       PIC X(08).
     05  buf-minus-ord                PIC X(82).
 01  image-fields.
     05  db-name                      PIC X(10)   VALUE "  ORDRET; ".
     05  list-of-items                PIC X(02)   VALUE "@;".
     05  password                     pic X(06)   VALUE "MGR;  ".
     05  mode1                        PIC S9(04)  COMP VALUE 1.
     05  mode5                        PIC S9(04)  COMP VALUE 5.
     05  mode7                        PIC S9(04)  COMP VALUE 7.
     05  search-item                  PIC X(08)   VALUE "ORD-NUM;".
     05  d-data-set                   PIC X(10)   VALUE "ITEM-DET; ".
     05  m-data-set                   PIC X(10)   VALUE "ORD-MSTR; ".
     05  db-status.
         10  cond-word                PIC S9(04)  COMP.
         10  FILLER                   PIC X(18).
 01  v-buffer.
     05  ord-num                      PIC X(08).
     05  ord-num-display              PIC X(08).
     05  m-dset-buffer.
         10  cust-name                PIC X(20).
         10  cust-street              PIC X(20).
         10  cust-city                PIC X(16).
         10  cust-state               PIC X(02).
         10  cust-zip                 PIC X(06).
         10  cust-phone               PIC X(10).
         10  purch-ord                PIC X(08).
     05  d-dset-buffer.
         10  quantity                 PIC 9(04).
         10  part-num                 PIC X(08).
         10  desc                     PIC X(30).
         10  unt-meas                 PIC X(02).
         10  price                    PIC 9(06).
 01  form-buffer
         REDEFINES v-buffer           PIC X(148).
 01  v-parameters.
     05  form-file-name               PIC X(10)   VALUE "ORDFORM1; ".
     05  term-name                    PIC X(08)   VALUE "A264X ".
     05  buf-length                   PIC S9(04)  COMP VALUE 148.
     05  message-buf                  PIC X(72).
     05  msg-length                   PIC S9(04)  COMP VALUE 72.
     05  actual-length                PIC S9(04)  COMP.
 01  field-buf.
     05  order-number                 PIC X(08).
     05  ord-num-length               PIC S9(04)  COMP VALUE 8.
 01  error-message                    PIC X(28).
```

```
 01  err-message-length             PIC S9(04)  COMP VALUE 28.
 01  comarea.
     05  vstatus                    PIC S9(04)  COMP VALUE ZERO.
     05  language                   PIC S9(04)  COMP VALUE ZERO.
     05  comarea-length             PIC S9(04)  COMP VALUE 60.
     05  FILLER                     PIC S9(04)  COMP VALUE 0.
     05  FILLER                     PIC S9(04)  COMP VALUE 0.
     05  last-key                   PIC S9(04)  COMP.
     05  FILLER                     PIC X(108)  VALUE ZEROS.
 PROCEDURE DIVISION.
 a100-start SECTION 10.
     PERFORM c100-init THRU c100-init-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO b100-exit.
     PERFORM e100-main THRU e100-main-exit.
     PERFORM d100-close THRU d100-close-exit.
 b100-exit.
     IF error-flag NOT EQUAL TO 0
         CALL "ERRORSUB" USING
                                   error-flag.
     STOP RUN.
*
*END OF ORDER RETRIEVAL PROGRAM
*
 c100-init SECTION 15.
     PERFORM c200-opendb THRU c200-opendb-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO c100-init-exit.
     PERFORM c300-openform THRU c300-openform-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO c100-init-exit.
     PERFORM c400-openterm THRU c400-openterm-exit.
 c100-init-exit.
     EXIT.
*
*RETURN TO a100-start
*
 c200-opendb.
     CALL "DBOPEN" USING
                              db-name,
                              password,
                              mode1,
                              db-status.
     IF cond-word NOT EQUAL TO 0
         CALL "DBEXPLAIN" USING db-status
         MOVE 1 TO error-flag.
 c200-opendb-exit.
     EXIT.
*
*RETURN TO c100-init
*
 c300-openform.
     CALL "VOPENFORMF" USING
                                  comarea,
                                  form-file-name.
     IF vstatus NOT EQUAL TO 0
         MOVE 2 TO error-flag.
```

```
 c300-openform-exit.
     EXIT.
*
*RETURN TO c100-init
*
 c400-openterm.
     CALL "VOPENTERM" USING
                                 comarea,
                                 term-name.
     IF vstatus NOT EQUAL TO 0
         MOVE 3 TO error-flag.
 c400-openterm-exit.
     EXIT.
*
*RETURN TO c100-init
*
 d100-close.
     PERFORM d200-closedb THRU d200-closedb-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO d100-close-exit.
     PERFORM d300-closeform THRU d300-closeform-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO d100-close-exit.
     PERFORM d400-closeterm THRU d400-closeterm-exit.
 d100-close-exit.
     EXIT.
*
*RETURN TO a100-start
*
 d200-closedb.
     CALL "DBCLOSE" USING
                              db-name,
                              password,
                              model,
                              db-status.
     IF cond-word NOT EQUAL TO 0
         MOVE 4 TO error-flag.
 d200-closedb-exit.
     EXIT.
*
*RETURN TO d100-close
*
 d300-closeform.
     CALL "VCLOSEFORMF" USING
                                  comarea.
     IF vstatus NOT EQUAL TO 0
         MOVE 5 TO error-flag.
 d300-closeform-exit.
     EXIT.
*
*RETURN TO d100-close
*
 d400-closeterm.
     CALL "VCLOSETERM" USING
                                 comarea.
     IF vstatus NOT EQUAL TO 0
         MOVE 6 TO error-flag.
```

```
 d400-closeterm-exit.
     EXIT.
*
*RETURN TO d100-close
*
 e100-main SECTION 20.
     CALL "VGETNEXTFORM" USING
                                    comarea.
     IF vstatus NOT EQUAL TO 0
         MOVE 8 TO error-flag
         MOVE 1 TO exit-flag
         GO TO e100-main-exit.
     CALL "VINITFORM" USING
                                 comarea.
     IF vstatus NOT EQUAL TO 0
         move 17 TO error-flag
         move 1 TO exit-flag
         GO TO e100-main-exit.
     CALL "VSHOWFORM" USING
                                 comarea.
     IF vstatus NOT EQUAL TO 0
         MOVE 9 TO error-flag
         MOVE 1 TO exit-flag
         GO TO e100-main-exit.
     PERFORM f100-read THRU f100-read-exit
         UNTIL exit-flag EQUAL TO 1.
 e100-main-exit.
     EXIT.
*
*RETURN TO a100-start
*
 f100-read.
     MOVE 0 TO record-was-found.
     CALL "VREADFIELDS" USING
                                   comarea.
     IF vstatus NOT EQUAL TO 0
         MOVE 10 TO error-flag
         MOVE 1 TO exit-flag
         GO TO f100-read-exit.
     CALL "VGETBUFFER" USING
                                  comarea,
                                  order-number,
                                  ord-num-length.
     IF vstatus NOT EQUAL TO 0
         MOVE 18 TO error-flag
         MOVE 1 TO exit-flag
         GO TO f100-read-exit.
     IF last-key EQUAL TO 8
         MOVE 1 TO exit-flag
         GO TO f100-read-exit.
     MOVE order-number TO ord-num,
                ord-num-display.
     PERFORM f200-findord THRU f200-findord-exit.
     IF error-flag NOT EQUAL TO 0
         MOVE 1 TO exit-flag
         GO TO f100-read-exit.
     IF record-was-found EQUAL TO 1
```

```
          MOVE SPACES TO error-message
          PERFORM f400-print THRU f400-print-exit.
 f100-read-exit.
      EXIT.
*
*RETURN TO e100-main
*
 f200-findord.
      CALL "DBGET" USING
                           db-name,
                           m-data-set,
                           mode7,
                           db-status,
                           list-of-items,
                           buffer,
                           order-number.
      IF cond-word EQUAL TO 0
          MOVE 1 TO record-was-found
          PERFORM f300-getord THRU f300-getord-exit
      ELSE
          IF cond-word EQUAL TO 17
              MOVE "order not found " TO error-message
              PERFORM f400-print THRU f400-print-exit
          ELSE
              MOVE 11 TO error-flag.
 f200-findord-exit.
      EXIT.
*
*RETURN TO f100-read
*
 f300-getord.
      MOVE buf-minus-ord TO m-dset-buffer.
      CALL "DBFIND" USING
                           db-name,
                           d-data-set,
                           mode1,
                           db-status,
                           search-item,
                           order-number.
      IF cond-word NOT EQUAL TO 0
          MOVE 12 TO error-flag
          GO TO f300-getord-exit.
      CALL "DBGET" USING
                           db-name,
                           d-data-set,
                           mode5,
                           db-status,
                           list-of-items,
                           buffer,
                           ord-num.
      IF cond-word NOT EQUAL TO 0
          IF cond-word EQUAL TO 17
              MOVE "order not found " TO error-message
              PERFORM f400-print THRU f400-print-exit
              GO TO f300-getord-exit
          ELSE
              MOVE 12 TO error-flag
```

```
            GO TO f300-getord-exit.
        MOVE buf-minus-ord TO d-dset-buffer.
    f300-getord-exit.
        EXIT.
   *
   *RETURN TO f200-findord
   *
    f400-print.
        IF record-was-found EQUAL TO 1
         AND
           error-flag EQUAL TO 0
              CALL "VPUTBUFFER" USING
                                          comarea,
                                          form-buffer,
                                          buf-length
              IF vstatus NOT EQUAL TO 0
                  MOVE 15 TO error-flag
                  MOVE 1 TO exit-flag
              ELSE
                  NEXT SENTENCE
        ELSE
              MOVE SPACES TO m-dset-buffer, d-dset-buffer
              CALL "VPUTBUFFER" USING
                                          comarea,
                                          form-buffer,
                                          buf-length
              IF vstatus NOT EQUAL TO 0
                  MOVE 15 TO error-flag
                  MOVE 1 TO exit-flag
                  GO TO f400-print-exit
              ELSE
                  CALL "VPUTWINDOW" USING
                                              comarea,
                                              error-message,
                                              err-message-length
                  IF vstatus NOT EQUAL TO 0
                      MOVE 16 TO error-flag
                      MOVE 1 TO exit-flag
                      GO TO f400-print-exit.
        CALL "VSHOWFORM" USING
                                 comarea.
        IF vstatus NOT EQUAL TO 0
            MOVE 1 TO exit-flag
            MOVE 9 TO error-flag
            GO TO f400-print-exit.
        MOVE SPACES TO error-message.
        CALL "VPUTWINDOW" USING
                                  comarea,
                                  error-message,
                                  err-message-length.
        IF vstatus NOT EQUAL TO 0
            MOVE 16 TO error-flag
            MOVE 1 TO exit-flag.
    f400-print-exit.
        EXIT.
```

```
$CONTROL LIST, SUBPROGRAM, DYNAMIC
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ERRORSUB.
 AUTHOR.  ct3000.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 LINKAGE SECTION.
 01  error-flag                          PIC S9(04)  COMP.
 PROCEDURE DIVISION
                          USING error-flag.
 a100-start.
     GO TO e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12,
           e13, e14, e15, e16, e17, e18
         DEPENDING ON error-flag.
 e1.
     DISPLAY "dbopen failure".
     GO TO end-of-sub.
 e2.
     DISPLAY "vopenformf failure".
     GO TO end-of-sub.
 e3.
     DISPLAY "vopentermf failure".
     GO TO end-of-sub.
 e4.
     DISPLAY "dbclose failure".
     GO TO end-of-sub.
 e5.
     DISPLAY "vcloseformf failure".
     GO TO end-of-sub.
 e6.
     DISPLAY "vcloseterm failure".
     GO TO end-of-sub.
 e7.
     DISPLAY "vopenformf failure".
     GO TO end-of-sub.
 e8.
     DISPLAY "vgetnextform failure".
     GO TO end-of-sub.
 e9.
     DISPLAY "vshowform failure".
     GO TO end-of-sub.
 e10.
     DISPLAY "vreadfields failure".
     GO TO end-of-sub.
 e11.
     DISPLAY "dbfind failure".
     GO TO end-of-sub.
 e12.
     DISPLAY "dbget failure".
     GO TO end-of-sub.
 e13.
     DISPLAY "startsetup failure".
     GO TO end-of-sub.
 e14.
     DISPLAY "endsetup failure".
     GO TO end-of-sub.
```

```
e15.
    DISPLAY "vputbuffer failure".
    GO TO end-of-sub.
e16.
    DISPLAY "vputwindow failure".
    GO TO end-of-sub.
e17.
    DISPLAY "vinitform failure".
    GO TO end-of-sub.
e18.
    DISPLAY "vgetbuffer failure".
    GO TO end-of-sub.
end-of-sub.
    MOVE 0 TO error-flag.
    GOBACK.
```

```
PROGRAM FILE PDEMO1P.DEMO.DESIGN

ERRORSUB            0
   NAME              STT  CODE ENTRY SEG
   ERRORSUB            1     0     0
   C'DISPLAY'FIN       4                ?
   C'DISPLAY'INIT      5                ?
   C'DISPLAY'L         6                ?
   QUIT                7                ?
   ERRORSUB'S          2     0  1071
   ERRORSUB'           3  1076  1076
   SEGMENT LENGTH         1314
E100MAIN20'         1
   NAME              STT  CODE ENTRY SEG
   E100MAIN20'         1     0     0
   VGETNEXTFORM        3                ?
   VINITFORM           4                ?
   VSHOWFORM           5                ?
   VREADFIELDS         6                ?
   VGETBUFFER          7                ?
   DBGET              10                ?
   DBFIND             11                ?
   VPUTBUFFER         12                ?
   VPUTWINDOW         13                ?
   QUIT               14                ?
   demo1               2   767   767
   DEBUG              15                ?
   COBOLTRAP          16                ?
   A100START10'       17                3
   C100INIT15'        20                2
   SEGMENT LENGTH         1224
C100INIT15'         2
   NAME              STT  CODE ENTRY SEG
   C100INIT15'         1     0     0
   DBOPEN              2                ?
   DBEXPLAIN           3                ?
   VOPENFORMF          4                ?
   VOPENTERM           5                ?
   DBCLOSE             6                ?
   VCLOSEFORMF         7                ?
   VCLOSETERM         10                ?
   SEGMENT LENGTH          300
A100START10'        3
   NAME              STT  CODE ENTRY SEG
   A100START10'        1     0     0
   ERRORSUB            2                0
   TERMINATE'          3                ?
   SEGMENT LENGTH           60

PRIMARY DB          0   INITIAL STACK    2000   CAPABILITY        600
SECONDARY DB      546   INITIAL DL          0   TOTAL CODE       3120
TOTAL DB          546   MAXIMUM DATA    25370   TOTAL RECORDS      26
ELAPSED TIME   00:00:04.350                  PROCESSOR TIME   00:01.107
```

3. Demonstration III-1 Listings

- Stream file
- Source files:
  - PDADP
  - PDEMO2P
  - SRTSETUP
  - ERRORSUB
- PMAPs for:
  - PDADP
  - PDEMO2P

```
!JOB DEMO.DESIGN
!PURGE PDADP
!PURGE UDADU
!FILE COBTEXT=SDADS
!FILE COBUSL=UDADU
!FILE COBLIST=$NULL
!RUN COBOLII.PUB.SYS;PARM=5
!PREP UDADU,PDADP;PMAP;CAP=PH
!SAVE PDADP
!PURGE PDEMO2P
!PURGE UDEMO2U
!FILE COBTEXT=SDEMO2S
!FILE COBUSL=UDEMO2U
!FILE COBLIST=$NULL
!RUN COBOLII.PUB.SYS;PARM=5
!FILE COBTEXT=ERRORSUB
!RUN COBOLII.PUB.SYS;PARM=5
!SPL SRTSETUP,UDEMO2U
!PREP UDEMO2U,PDEMO2P;MAXDATA=11000;PMAP;CAP=PH
!SAVE PDEMO2P
!EOJ
```

```
$CONTROL USLINIT, SOURCE, MAP
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  demo2dad.
 AUTHOR.  ct3000.
 DATE-COMPILED.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  HP3000.
 OBJECT-COMPUTER.  HP3000.
 SPECIAL-NAMES.
     CONDITION-CODE IS cc.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  exit-flag                       PIC S9(04)  COMP VALUE 0.
 77  error-flag                      PIC S9(04)  COMP VALUE 0.
 77  terminate-flag                  PIC S9(04)  COMP VALUE 0.
 77  son                             PIC S9(04)  COMP VALUE 1.
 77  maximum-sons                    PIC S9(04)  COMP.
 77  return-length                   PIC S9(04)  COMP.
 77  pin-num                         PIC S9(04)  COMP.
 01  programs                        PIC X(08)   VALUE "PDEMO2P ".
 01  reply                           PIC X(03).
 01  pins.
     05  pin-number
           OCCURS 3 TIMES
                                     PIC S9(04)  COMP.
 01  ldev-number                     PIC S9(04) COMP.
 01  display-son                     PIC Z(04).


 PROCEDURE DIVISION.
 a100-start.
     DISPLAY "ENTER THE NUMBER OF TERMINALS TO BE ACTIVATED".
     ACCEPT maximum-sons FREE
        ON INPUT ERROR
               DISPLAY "You must enter a number"
               GO TO a100-start.
     PERFORM b100-create-sons THRU b100-create-sons-exit
           UNTIL son > maximum-sons.
     IF error-flag NOT EQUAL TO 0
         GO TO a100-start-exit.
     PERFORM c100-print-up-message
                         THRU c100-print-up-message-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO a100-start-exit.
     MOVE 1 TO son.
     PERFORM d100-activate-sons THRU d100-activate-sons-exit
           UNTIL son > maximum-sons.
     IF error-flag NOT EQUAL TO 0
         GO TO a100-start-exit.
     PERFORM e100-reply THRU e100-reply-exit
           UNTIL terminate-flag EQUAL TO 1.
 a100-start-exit.
     STOP RUN.
*
*END OF FATHER PROGRAM
*
```

```
 b100-create-sons.
     MOVE son TO display-son.
     DISPLAY "Enter logical device number of terminal",
             display-son.
     ACCEPT ldev-number FREE
             ON INPUT ERROR
               DISPLAY "You must enter a number for the terminal"
               GO TO b100-create-sons.
     CALL INTRINSIC "CREATE" USING
                                      programs,
                                      \\,
                                      pin-number (son),
                                      ldev-number,
                                      \1\.
     IF cc LESS THAN  0
         DISPLAY "unable to create son processes"
         MOVE 1 TO error-flag
         MOVE 4 TO son
         GO TO b100-create-sons-exit.
     CALL INTRINSIC "ACTIVATE" USING
                                        \pin-number (son)\,
                                        \2\.
     IF cc NOT EQUAL TO 0
         DISPLAY "unable to activate sons"
         MOVE 1 TO error-flag
         MOVE 4 TO son
         GO TO b100-create-sons-exit.
     CALL INTRINSIC "GETPROCINFO" USING
                                           \pin-number (son)\.
     IF cc NOT EQUAL TO 0
         DISPLAY "son process aborted"
         MOVE 1 TO error-flag
         MOVE 4 TO son.
     ADD 1 TO son.
 b100-create-sons-exit.
     EXIT.
*
*RETURN TO a100-start
*
 c100-print-up-message.
     DISPLAY "*** ALL TERMINALS UP ***".
 c100-print-up-message-exit.
     EXIT.
*
*RETURN TO a100-start
*
 d100-activate-sons.
     CALL INTRINSIC "ACTIVATE" USING
                                        \pin-number (son)\,
                                        \0\.
     IF cc NOT  EQUAL TO 0
         DISPLAY "activation of son unsuccessful"
         MOVE 1 TO error-flag
         MOVE 4 TO son.
     ADD 1 TO son.
 d100-activate-sons-exit.
     EXIT.
```

```
*
*RETURN TO a100-start
*
 e100-reply.
     DISPLAY "*** RESPOND 'YES' TO END PROGRAM ***".
     ACCEPT reply.
     IF reply NOT EQUAL TO "YES" AND
        reply NOT EQUAL TO "yes"
           GO TO e100-reply-exit.
     CALL INTRINSIC "GETPROCID" USING
                                           \1\
        GIVING pin-num.
     IF pin-num EQUAL TO 0
         MOVE 1 TO terminate-flag
         GO TO e100-reply-exit.
     MOVE 0 TO exit-flag.
     PERFORM e200-reply2 THRU e200-reply2-exit
           UNTIL exit-flag EQUAL TO 1.
 e100-reply-exit.
     EXIT.
*
*RETURN TO a100-start
*
 e200-reply2.
     DISPLAY "*** SOME TERMINALS ARE STILL ACTIVE ***".
     DISPLAY "*** REPLY 'YES' TO TERMINATE THEM ***".
     ACCEPT reply.
     IF reply EQUAL TO "YES" AND
        reply NOT EQUAL TO "yes"
         MOVE 1 TO terminate-flag
     ELSE
        MOVE 0 TO terminate-flag.
     MOVE 1 TO exit-flag.
 e200-reply2-exit.
     EXIT.
*
*RETURN TO e100-reply
*
```

```
$CONTROL LIST, USLINIT, MAP, SOURCE,CROSSREF
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  demo2son.
 AUTHOR.  ct3000.
 DATE-COMPILED.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  HP3000.
 OBJECT-COMPUTER.  HP3000.
 SPECIAL-NAMES.
     CONDITION-CODE IS cc.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  error-flag                       pic S9(04)  COMP VALUE 0.
 77  exit-flag                        pic S9(04)  COMP VALUE 0.
 77  record-was-found                 PIC S9(04)  COMP VALUE 1.
 77  call-stat                        PIC S9(04)  COMP VALUE 0.
 01  buffer.
     05  FILLER                       PIC X(08).
     05  buf-minus-ord                PIC X(82).
 01  image-fields.
     05  db-name                      PIC X(10)   VALUE "  ORDRET; ".
     05  list-of-items                PIC X(02)   VALUE "@;".
     05  password                     pic X(06)   VALUE "MGR;  ".
     05  mode1                        PIC S9(04)  COMP VALUE 1.
     05  mode5                        PIC S9(04)  COMP VALUE 5.
     05  mode7                        PIC S9(04)  COMP VALUE 7.
     05  search-item                  PIC X(08)   VALUE "ORD-NUM;".
     05  d-data-set                   PIC X(10)   VALUE "ITEM-DET; ".
     05  m-data-set                   PIC X(10)   VALUE "ORD-MSTR; ".
     05  db-status.
         10  cond-word                PIC S9(04)  COMP.
         10  FILLER                   PIC X(18).
 01  v-buffer.
     05  ord-num                      PIC X(08).
     05  ord-num-display              PIC X(08).
     05  m-dset-buffer.
         10  cust-name                PIC X(20).
         10  cust-street              PIC X(20).
         10  cust-city                PIC X(16).
         10  cust-state               PIC X(02).
         10  cust-zip                 PIC X(06).
         10  cust-phone               PIC X(10).
         10  purch-ord                PIC X(08).
     05  d-dset-buffer.
         10  quantity                 PIC 9(04).
         10  part-num                 PIC X(08).
         10  desc                     PIC X(30).
         10  unt-meas                 PIC X(02).
         10  price                    PIC 9(06).
 01  form-buffer
         REDEFINES v-buffer           PIC X(148).
 01  v-parameters.
     05  form-file-name               PIC X(10)   VALUE "ORDFORM1; ".
     05  term-name                    PIC X(08)   VALUE "A264X ".
     05  buf-length                   PIC S9(04)  COMP VALUE 148.
     05  message-buf                  PIC X(72).
```

```
     05  msg-length                    PIC S9(04)  COMP VALUE 72.
     05  actual-length                 PIC S9(04)  COMP.
 01  field-buf.
     05  order-number                  PIC X(08).
     05  ord-num-length                PIC S9(04)  COMP VALUE 8.
 01  error-message                     PIC X(28).
 01  err-message-length                PIC S9(04)  COMP VALUE 28.
 01  comarea.
     05  vstatus                       PIC S9(04)  COMP VALUE ZERO.
     05  language                      PIC S9(04)  COMP VALUE ZERO.
     05  comarea-length                PIC S9(04)  COMP VALUE 60.
     05  FILLER                        PIC S9(04)  COMP VALUE 0.
     05  FILLER                        PIC S9(04)  COMP VALUE 0.
     05  last-key                      PIC S9(04)  COMP.
     05  FILLER                        PIC X(108)  VALUE ZEROS.
 PROCEDURE DIVISION.
 a100-start SECTION 10.
     PERFORM c100-init THRU c100-init-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO b100-exit.
     PERFORM e100-main THRU e100-main-exit.
     PERFORM d100-close THRU d100-close-exit.
 b100-exit.
     IF error-flag NOT EQUAL TO 0
         CALL "ERRORSUB" USING
                               error-flag.
     STOP RUN.
*
*END OF ORDER RETRIEVAL PROGRAM
*
 c100-init SECTION 15.
     CALL "STARTSETUP" USING
                               call-stat.
     IF call-stat NOT EQUAL TO 0
         MOVE 13 TO error-flag
         GO TO c100-init-exit.
     PERFORM c200-opendb THRU c200-opendb-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO c100-init-exit.
     PERFORM c300-openform THRU c300-openform-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO c100-init-exit.
     PERFORM c400-openterm THRU c400-openterm-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO c100-init-exit.
     PERFORM c500-endsetup THRU
                c500-endsetup-exit.
 c100-init-exit.
     EXIT.
*
*RETURN TO a100-start
*
 c200-opendb.
     CALL "DBOPEN" USING
                          db-name,
                          password,
                          model,
```

```
                                db-status.
     IF cond-word NOT EQUAL TO 0
         CALL "DBEXPLAIN" USING db-status
         MOVE 1 TO error-flag.
 c200-opendb-exit.
     EXIT.
*
*RETURN TO c100-init
*
 c300-openform.
     CALL "VOPENFORMF" USING
                                    comarea,
                                    form-file-name.
     IF vstatus NOT EQUAL TO 0
         MOVE 2 TO error-flag.
 c300-openform-exit.
     EXIT.
*
*RETURN TO c100-init
*
 c400-openterm.
     CALL "VOPENTERM" USING
                                   comarea,
                                   term-name.
     IF vstatus NOT EQUAL TO 0
         MOVE 3 TO error-flag.
 c400-openterm-exit.
     EXIT.
*
*RETURN TO c100-init
*
 c500-endsetup.
     CALL INTRINSIC "ACTIVATE" USING
                                            \0\,
                                            \1\.
     IF cc NOT EQUAL TO 0
         MOVE 14 TO error-flag.
 c500-endsetup-exit.
     EXIT.
*
*RETURN TO c100-init
*
 d100-close.
     PERFORM d200-closedb THRU d200-closedb-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO d100-close-exit.
     PERFORM d300-closeform THRU d300-closeform-exit.
     IF error-flag NOT EQUAL TO 0
         GO TO d100-close-exit.
     PERFORM d400-closeterm THRU d400-closeterm-exit.
 d100-close-exit.
     EXIT.
*
*RETURN TO a100-start
*
 d200-closedb.
     CALL "DBCLOSE" USING
```

```
                                  db-name,
                                  password,
                                  model,
                                  db-status.
      IF cond-word NOT EQUAL TO 0
         MOVE 4 TO error-flag.
  d200-closedb-exit.
      EXIT.
*
*RETURN TO d100-close
*
  d300-closeform.
      CALL "VCLOSEFORMF" USING
                                     comarea.
      IF vstatus NOT EQUAL TO 0
         MOVE 5 TO error-flag.
  d300-closeform-exit.
      EXIT.
*
*RETURN TO d100-close
*
  d400-closeterm.
      CALL "VCLOSETERM" USING
                                     comarea.
      IF vstatus NOT EQUAL TO 0
         MOVE 6 TO error-flag.
  d400-closeterm-exit.
      EXIT.
*
*RETURN TO d100-close
*
  e100-main SECTION 20.
      CALL "VGETNEXTFORM" USING
                                     comarea.
      IF vstatus NOT EQUAL TO 0
         MOVE 8 TO error-flag
         MOVE 1 TO exit-flag
         GO TO e100-main-exit.
      CALL "VINITFORM" USING
                                  comarea.
      IF vstatus NOT EQUAL TO 0
         move 17 TO error-flag
         move 1 TO exit-flag
         GO TO e100-main-exit.
      CALL "VSHOWFORM" USING
                                  comarea.
      IF vstatus NOT EQUAL TO 0
         MOVE 9 TO error-flag
         MOVE 1 TO exit-flag
         GO TO e100-main-exit.
      PERFORM f100-read THRU f100-read-exit
         UNTIL exit-flag EQUAL TO 1.
  e100-main-exit.
      EXIT.
*
*RETURN TO a100-start
*
```

```
f100-read.
    MOVE 0 TO record-was-found.
    CALL "VREADFIELDS" USING
                              comarea.
    IF vstatus NOT EQUAL TO 0
        MOVE 10 TO error-flag
        MOVE 1 TO exit-flag
        GO TO f100-read-exit.
    CALL "VGETBUFFER" USING
                             comarea,
                             order-number,
                             ord-num-length.
    IF vstatus NOT EQUAL TO 0
        MOVE 18 TO error-flag
        MOVE 1 TO exit-flag
        GO TO f100-read-exit.
    IF last-key EQUAL TO 8
        MOVE 1 TO exit-flag
        GO TO f100-read-exit.
    MOVE order-number TO ord-num,
               ord-num-display.
    PERFORM f200-findord THRU f200-findord-exit.
    IF error-flag NOT EQUAL TO 0
        MOVE 1 TO exit-flag
        GO TO f100-read-exit.
    IF record-was-found EQUAL TO 1
        MOVE SPACES TO error-message
        PERFORM f400-print THRU f400-print-exit.
f100-read-exit.
    EXIT.
*
*RETURN TO e100-main
*
f200-findord.
    CALL "DBGET" USING
                        db-name,
                        m-data-set,
                        mode7,
                        db-status,
                        list-of-items,
                        buffer,
                        order-number.
    IF cond-word EQUAL TO 0
        MOVE 1 TO record-was-found
        PERFORM f300-getord THRU f300-getord-exit
    ELSE
        IF cond-word EQUAL TO 17
            MOVE "order not found " TO error-message
            PERFORM f400-print THRU f400-print-exit
        ELSE
            MOVE 11 TO error-flag.
f200-findord-exit.
    EXIT.
*
*RETURN TO f100-read
*
f300-getord.
```

```
     MOVE buf-minus-ord TO m-dset-buffer.
     CALL "DBFIND" USING
                          db-name,
                          d-data-set,
                          mode1,
                          db-status,
                          search-item,
                          order-number.
     IF cond-word NOT EQUAL TO 0
         MOVE 12 TO error-flag
         GO TO f300-getord-exit.
     CALL "DBGET" USING
                         db-name,
                         d-data-set,
                         mode5,
                         db-status,
                         list-of-items,
                         buffer,
                         ord-num.
     IF cond-word NOT EQUAL TO 0
         IF cond-word EQUAL TO 17
             MOVE "order not found " TO error-message
             PERFORM f400-print THRU f400-print-exit
             GO TO f300-getord-exit
         ELSE
             MOVE 12 TO error-flag
             GO TO f300-getord-exit.
     MOVE buf-minus-ord TO d-dset-buffer.
 f300-getord-exit.
     EXIT.
*
*RETURN TO f200-findord
*
 f400-print.
     IF record-was-found EQUAL TO 1
      AND
        error-flag EQUAL TO 0
           CALL "VPUTBUFFER" USING
                                   comarea,
                                   form-buffer,
                                   buf-length
           IF vstatus NOT EQUAL TO 0
               MOVE 15 TO error-flag
               MOVE 1 TO exit-flag
           ELSE
               NEXT SENTENCE
     ELSE
           MOVE SPACES TO m-dset-buffer, d-dset-buffer
           CALL "VPUTBUFFER" USING
                                   comarea,
                                   form-buffer,
                                   buf-length
           IF vstatus NOT EQUAL TO 0
               MOVE 15 TO error-flag
               MOVE 1 TO exit-flag
               GO TO f400-print-exit
           ELSE
```

```
          CALL "VPUTWINDOW" USING
                                   comarea,
                                   error-message,
                                   err-message-length
          IF vstatus NOT EQUAL TO 0
              MOVE 16 TO error-flag
              MOVE 1 TO exit-flag
              GO TO f400-print-exit.
    CALL "VSHOWFORM" USING
                            comarea.
    IF vstatus NOT EQUAL TO 0
        MOVE 1 TO exit-flag
        MOVE 9 TO error-flag
        GO TO f400-print-exit.
    MOVE SPACES TO error-message.
    CALL "VPUTWINDOW" USING
                             comarea,
                             error-message,
                             err-message-length.
    IF vstatus NOT EQUAL TO 0
        MOVE 16 TO error-flag
        MOVE 1 TO exit-flag.
f400-print-exit.
    EXIT.
```

```
$CONTROL SUBPROGRAM
BEGIN

<<*****************************************************************
<<
<<  STARTSETUP picks up a terminal logical device number          >>
<<  from the CREATE (or RUN) PARM issued by demo2dad. It          >>
<<  converts the ldev into an ASCII string as part of a FILE      >>
<<  command.  It then issues the FILE command with the MPE        >>
<<  COMMAND intrinsic.  (COBOL II cannot call the COMMAND         >>
<<  intrinsic, which is why STARTSETUP must be in SPL.)           >>
<<                                                                >>
<<       Error return:                                            >>
<<                      =0   NO ERROR                             >>
<<                      >0   COMMAND intrinsic error number       >>
<<                                                                >>
<< CAUTION:  If this procedure is not called from the main        >>
<<           program, demo2son, the wrong PARM value will be      >>
<<           returned.                                            >>
<<                                                                >>
<<***************************************************************>>

PROCEDURE STARTSETUP(error);
  INTEGER error;

  BEGIN

     LOGICAL ARRAY w'image(0:11);
     BYTE ARRAY image(*) = w'image;
     INTEGER
             deltaq = Q,
             parm,
             cmderr,
             dumy;
     INTEGER POINTER p = S-0;

     INTRINSIC ASCII,
               COMMAND;

         MOVE w'image := "FILE A264X;DEV=        ";
         image(23) := %15;
         PUSH(Q);
         parm := p(-deltaq-4);
         DEL;
         ASCII(parm,10,image(15));
         COMMAND(image,cmderr,dumy);
         error := IF <> THEN CMDERR
                         ELSE 0;
  END;

end.
```

```
$CONTROL LIST, SUBPROGRAM, DYNAMIC
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ERRORSUB.
 AUTHOR.  ct3000.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 LINKAGE SECTION.
 01  error-flag                          PIC S9(04)  COMP.
 PROCEDURE DIVISION
                    USING error-flag.
 a100-start.
     GO TO e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12,
            e13, e14, e15, e16, e17, e18
          DEPENDING ON error-flag.
 e1.
     DISPLAY "dbopen failure".
     GO TO end-of-sub.
 e2.
     DISPLAY "vopenformf failure".
     GO TO end-of-sub.
 e3.
     DISPLAY "vopentermf failure".
     GO TO end-of-sub.
 e4.
     DISPLAY "dbclose failure".
     GO TO end-of-sub.
 e5.
     DISPLAY "vcloseformf failure".
     GO TO end-of-sub.
 e6.
     DISPLAY "vcloseterm failure".
     GO TO end-of-sub.
 e7.
     DISPLAY "vopenformf failure".
     GO TO end-of-sub.
 e8.
     DISPLAY "vgetnextform failure".
     GO TO end-of-sub.
 e9.
     DISPLAY "vshowform failure".
     GO TO end-of-sub.
 e10.
     DISPLAY "vreadfields failure".
     GO TO end-of-sub.
 e11.
     DISPLAY "dbfind failure".
     GO TO end-of-sub.
 e12.
     DISPLAY "dbget failure".
     GO TO end-of-sub.
 e13.
     DISPLAY "startsetup failure".
     GO TO end-of-sub.
 e14.
     DISPLAY "endsetup failure".
     GO TO end-of-sub.
```

```
e15.
    DISPLAY "vputbuffer failure".
    GO TO end-of-sub.
e16.
    DISPLAY "vputwindow failure".
    GO TO end-of-sub.
e17.
    DISPLAY "vinitform failure".
    GO TO end-of-sub.
e18.
    DISPLAY "vgetbuffer failure".
    GO TO end-of-sub.
end-of-sub.
    MOVE 0 TO error-flag.
    GOBACK.
```

```
PROGRAM FILE PDADP.DEMO.DESIGN

A100START00'       0
   NAME              STT  CODE ENTRY SEG
   A100START00'        1     0     0
   CREATE              3               ?
   ACTIVATE            4               ?
   GETPROCINFO         5               ?
   GETPROCID           6               ?
   C'DISPLAY           7               ?
   C'DISPLAY'FIN      10               ?
   C'DISPLAY'INIT     11               ?
   C'DISPLAY'L        12               ?
   ACCEPT'FREE'C      13               ?
   TERMINATE'         14               ?
   C'ACCEPT           15               ?
   QUIT               16               ?
   demo2dad            2  1323  1323
   DEBUG              17               ?
   COBOLTRAP          20               ?
   SEGMENT LENGTH         1470

PRIMARY DB           0    INITIAL STACK    2000    CAPABILITY          1
SECONDARY DB       310    INITIAL DL          0    TOTAL CODE       1470
TOTAL DB           310    MAXIMUM DATA        ?    TOTAL RECORDS      14
ELAPSED TIME   00:00:03.164                  PROCESSOR TIME   00:00.538
```

```
PROGRAM FILE PDEMO2P.DEMO.DESIGN

SEG'                  0
   NAME                STT   CODE ENTRY SEG
   STARTSETUP            1      0     0
   ASCII                 2                 ?
   COMMAND               3                 ?
   SEGMENT LENGTH              70
ERRORSUB              1
   NAME                STT   CODE ENTRY SEG
   ERRORSUB              1      0     0
   C'DISPLAY'FIN         4                 ?
   C'DISPLAY'INIT        5                 ?
   C'DISPLAY'L           6                 ?
   QUIT                  7                 ?
   ERRORSUB'S            2      0  1071
   ERRORSUB'             3   1076  1076
   SEGMENT LENGTH            1314
E100MAIN20'           2
   NAME                STT   CODE ENTRY SEG
   E100MAIN20'           1      0     0
   VGETNEXTFORM          3                 ?
   VINITFORM             4                 ?
   VSHOWFORM             5                 ?
   VREADFIELDS           6                 ?
   VGETBUFFER            7                 ?
   DBGET                10                 ?
   DBFIND               11                 ?
   VPUTBUFFER           12                 ?
   VPUTWINDOW           13                 ?
   QUIT                 14                 ?
   demo2son              2    767   767
   DEBUG                15                 ?
   COBOLTRAP            16                 ?
   A100START10'         17                 4
   C100INIT15'          20                 3
   SEGMENT LENGTH            1224
C100INIT15'           3
   NAME                STT   CODE ENTRY SEG
   C100INIT15'           1      0     0
   STARTSETUP            2                 0
   DBOPEN                3                 ?
   DBEXPLAIN             4                 ?
   VOPENFORMF            5                 ?
   VOPENTERM             6                 ?
   ACTIVATE              7                 ?
   DBCLOSE              10                 ?
   VCLOSEFORMF          11                 ?
   VCLOSETERM           12                 ?
   SEGMENT LENGTH             350
```

```
A100START10'        4
   NAME              STT  CODE ENTRY SEG
   A100START10'        1     0     0
   ERRORSUB            2                1
   TERMINATE'          3                ?
   SEGMENT LENGTH          60

PRIMARY DB             0    INITIAL STACK    2000    CAPABILITY            1
SECONDARY DB         553    INITIAL DL          0    TOTAL CODE         3260
TOTAL DB             553    MAXIMUM DATA    25370    TOTAL RECORDS        27
ELAPSED TIME   00:00:02.677                      PROCESSOR TIME   00:01.182
```

4. Forms File Listing

ORDFORM1

```
*****************************************************************************
                                                                            *
                     FORMSPEC  Version A.01.01                              *
                    TUE, OCT 14, 1980,  4:28 PM                             *
                                                                            *
                       ORDFORM1.DEMO.DESIGN                                 *
                                                                            *
*****************************************************************************
rms File Status
 Modified: TUE, OCT 14, 1980, 10:29 AM
 Compiled: TUE, OCT 14, 1980, 10:29 AM
 Requires 1049 + 60 = 1109 words  (add 500 for KSAMless fast forms file, or
                                   add 1300 for KSAMless slow forms file)

ad Form:
fault Display Enhancement: HI
ror Enhancement: IU
ndow Display Line: 24
ndow Enhancement: HI


ERE ARE NO SAVE FIELDS IN THIS FORMS FILE.


ere are 1   forms in this forms file:

 Form                Num Fields    Num Lines    Next Form
 FORM1               14            22           $HEAD
```

```
Form: FORM1
   Repeat Option: N

   Next Form Option: C
   Next Form: $HEAD

            this is the only form for the demo
********* ********* ********* ********* ********* ********* ********* ******
                         Enter an Order Number
             The order number must be eight digits long

               ORDER NUMBER [ordnum__]     Press ENTER key
---------------------------------------------------------------------------

Data Related to Order Number [ordnumd_]

                    Customer Name [custname____________]
                 Customer Address [custstreet__________]
                                  [custcity________]
                                  [st]             [cuszip]
                            Phone [custphone_]

       Purchase Order [purchord]

             Quantity [qty_]
          Part number [partnum_]
          Description [desc________________________]
         Unit Measure [um]

                Price [price_]        *** Press f8 to EXIT ***
********* ********* ********* ********* ********* ********* ********* ******


Field: ordnum
   Num: 1    Len: 8    Name: ORDNUM          Enh: HI    FType: R  DType: D
   Init Value:
*** PROCESSING SPECIFICATIONS ***
MINLEN 8   "Order number must be 8 digits"


Field: ordnumd
   Num: 2    Len: 8    Name: ORDNUMD         Enh: HI    FType: D  DType: D
   Init Value:


Field: custname
   Num: 3    Len: 20   Name: CUSTNAME        Enh: HI    FType: D  DType: C
   Init Value:


Field: custstreet
   Num: 4    Len: 20   Name: CUSTSTREET      Enh: HI    FType: D  DType: C
   Init Value:


Field: custcity
   Num: 5    Len: 16   Name: CUSTCITY        Enh: HI    FType: D  DType: C
   Init Value:
```

```
Field: st
   Num: 6    Len: 2     Name: ST          Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: cuszip
   Num: 7    Len: 6     Name: CUSZIP      Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: custphone
   Num: 8    Len: 10    Name: CUSTPHONE   Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: purchord
   Num: 9    Len: 8     Name: PURCHORD    Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: qty
   Num: 10   Len: 4     Name: QTY         Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: partnum
   Num: 11   Len: 8     Name: PARTNUM     Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: desc
   Num: 12   Len: 30    Name: DESC        Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: um
   Num: 13   Len: 2     Name: UM          Enh: HI    FType: D  DType: CHAR
   Init Value:

Field: price
   Num: 14   Len: 6     Name: PRICE       Enh: HI    FType: D  DType: CHAR
   Init Value:
```

5. IMAGE Schema Listing

ORDRET

```
$control list,errors=2,blockmax=512
begin data base ordret;
passwords:
    1    clerk;
    63   MGR;


items:
    cust-name,           x20     (1/63);
    cust-street,         x20     (1/63);
    cust-city,           x16     (1/63);
    cust-state,          x2      (1/63);
    cust-zip,            x6      (1/63);
    cust-phone,          x10     (1/63);
    desc,                x30     (1/63);
    ord-num,             x8      (1/63);
    part-num,            x8      (1/63);
    price,               z6      (1/63);
    purch-ord,           x8      (1/63);
    quantity,            z4      (1/63);
    unt-meas,            x2      (1/63);



sets:
name:  ord-mstr,manual(1/63);
entry:  ord-num(1),
        cust-name,
        cust-street,
        cust-city,
        cust-state,
        cust-zip,
        cust-phone,
        purch-ord;
capacity:201;


name:  item-det,detail(1/63);
entry:  ord-num(ord-mstr),
        quantity,
        part-num,
        desc,
        unt-meas,
        price;
capacity:501;

end.
```

APPENDIX B

Answers to Worksessions

Answers to Worksession II-1 (architecture overview)

1. True. The HP 3000 is a stack machine.

2. True. Code and data must be separate in a stack machine.

3. Separate code and data means that the code can be shared; only the data need be private. Shared code saves memory.

4. Virtual memory is on disc; main memory is semiconductor memory. Code cannot be executed in virtual memory; it must be in main memory.

5. False. Not ALL the code and data need be in main memory for the program to execute; the data stack and at least one code segment are required for execution.

Answers to Worksession II-2

1. Two processes. A process is a unique execution of a program at a particular time.

2. Two processes. A process is a unique execution of a program by a particular user.

3. False. Shared code is never modified. This characteristic is what allows code to be shared by many processes. It also means that the code can be re-entrant.

4. True. The main reason that data is not shared is that it must be totally private and able to be modified by each process that uses the data. As a result, each execution of the same program may have widely different data.

5. The two required ingredients of an executing process are the data stack and at least one code segment. Additional ingredients may be more code segments and extra data segments.

Answers to Worksession II-3 (code segments)

1. When code segments are variable in length, the code does not have arbitrary boundaries. This means new code can be added without crossing "page" boundaries.

2. a) Yes. The maximum segment size is 16K words.
   b) Yes. The maximum number of segments per program is 63.

3. Nothing. Transfers between segments are managed completely by the operating system. (We will see later how programmers can reduce the number of such transfers.)

4. No. An executing segment can be anywhere in available main memory (memory not dedicated to permanent MPE code). The exact location is kept track of by the system through the CPU registers: PB, B, and PL.

5. The user can control:
   a. the size of code segments,
   b. how many code segments in his program, and
   c. what code is in each segment.

Answers to Worksession II-4 (code segment design)

This worksession differs from the previous ones in that there are no simple correct answers. The following answers are guidelines; any reasonable facsimile should be acceptable.

1. A working set is the smallest set of segments that must be in main memory for a program to work efficiently. A working set is dynamic, changing as the needs of the program change over the course of its execution.

2. On the HP 3000, the longer code remains in the same segment (maintains good code locality), the less chance there is that new code segments must be brought in from virtual memory. This helps performance since it reduces the need for expensive disc access.

3. a) Stay in the same segment for as long as possible (good code locality).

   b) Keep segment size as small as possible without causing excessive transfers between segments.

   c) Try to make code segments approximately the same size since this makes it easier to find space for the code in main memory.

   d) Put infrequently used code in one (or more) separate segments.

   e) Make sure your code is written so that it can be resegmented if necessary - think about segmenting from the start.

4. Segment the illustrated program units.

   There is no one correct solution. However, there are some obvious considerations. The initialization and termination routines should probably all be in one segment since they are short, execute quickly, and each only executes once per program. The two procedure routines should probably be in a segment together since they work together. The error handling routine should be in a separate segment since it is large and may never be needed.

   Ask class for other possibilities. Have them give reasons for their choices.

Answers to Worksession II-5 (data stack)

1. False. The data stack is absolutely private.

2.
```
        DL  ---------   bottom limit (Data Limit)
            |       |
        DB  ---------   start of global data (Data Base)
            |       |
            |       |
            |       |
         Q  ---------   start of local data (dynamic base)
            |       |
            |       |
            |       |
         S  ---------   top-of-stack
         Z  |_______|   top limit (stack limit)
```

3. Mark the DL-DB and the Q-Z areas as those whose size can be directly managed by the user.

4. a) The user can set stack limits with MAXDATA or STACK. This must be done when the program is prepared or run; it cannot be done dynamically during execution.

   b) The user can change the size of the DL-DB area with the DLSIZE intrinsic, or the size of the Q-Z area with ZSIZE. Usually these intrinsics are used to reduce the expandable areas of the stack after they have expanded for a particular one-time purpose.

   c) The user can reduce the global (DB-Q) area by using this area only for truly global data. For instance, making a COBOL subprogram "DYNAMIC" insures that its data is in the local (and directly manageable) area of the stack rather than the global area. Reserve the global area for data that is shared by more than one procedure or that must be kept between procedure calls to the same procedure.

5. a) Yes. This is the typical situation where you would call ZSIZE to shrink the stack. You would call ZSIZE after "X" (the procedure that expanded the stack) has exited; otherwise, the stack will remain the size to which it was expanded by "X". Note that before you call procedure "X", you should execute an SPL procedure to determine the normal location of Z so it can be returned to that location with ZSIZE.

b) STACK is the best choice since it allocates main memory at once. Using MAXDATA would cause main memory to be allocated in increments of 1K words as it was needed, causing extra disc I/O at each increment. There is no real drawback to this choice since the space in main memory is needed at once. STACK is only wasteful if the space in main memory is not used during a large part of the process execution.

c) MAXDATA is the only choice here; STACK does not increase the size of the DL to DB part of the stack. Since it is the only choice, there is no real drawback. However, using MAXDATA is always costly in disc I/O.

6. If error messages are included in your program as data, they are placed in the global area of the stack, thereby making your stack permanently very large (the global area cannot be made smaller programmatically). If, on the other hand, error messages are placed in a separate code segment or in a subprogram that uses the dynamic area of the stack, they do not increase the global area of the stack.

Answers to Worksession II-6 (extra data segments)

1. At least two of the following differences between a data stack and an extra data segment should be mentioned:

   1. The stack must be private to a process, extra data segments may be shared by more than one process in the same job or session.

   2. The stack is created and managed by the system, extra data segments are created and managed by user programs.

   3. The stack is structured, an extra data segment is linear and unstructured.

2. Each of the three listed situations is one in which extra data segments can provide a solution. Note that there may be other solutions. For instance,

   a) An array too large for the data stack could be stored in a file; but the transfers between the stack and an extra data segment are usually less time consuming than opening and accessing a file.

   b) Data local to a procedure normally is stored in the local Q-relative area of the stack where it ceases to be available when the procedure exits. You can specify that data for procedures be stored in the global area of the stack, but this means a permanently large stack if there is a lot of such data.

   c) A MAIL facility allows one word at a time to be passed between processes. Queuing files are a special type of file (only available with MPE IV) that provide an excellent means to pass data and messages between processes in the same process tree. But, extra data segments still provide the only solution for installations that do not have MPE IV and where more than one word must be passed at a time.

   Of course, extra data segments cannot take the place of files for most data; only files provide permanent storage for data.

Answers to Worksession II-7 (using libraries)

1. A) A routine that performs a mathematical function could be in either an RL or an SL. In this case, because it is large and because it is referenced by more than one program, put it in an SL. A large routine takes up too much space in an RL; and a routine that is shared by many programs should be in an SL to reduce the number of copies of the routine.

   If the routine is referenced by a large number of programs, consider putting it in a system SL - system SLs are easier to reference since no special request is needed.

   B) These routines are very good candidates for RL routines for several reasons: They are small, taking up little room in the program file. They are essentially private to the program that calls them since they deal with that program's stack.

   C) The routine that reformats main program data must be in an RL because all data from the main program (outer block) is in the global data area of the stack. SL routines are linked to the program after the global stack is established and, thus, cannot modify global data.

2. If your program has 62 code segments already, using any RLs will bring your total number of segments to the limit of 63. If you are willing to operate with 63 code segments, it doesn't matter how many RLs you add since all RLs are placed in one segment. You should, of course, consider how large your RL segment is - follow the general code segment rules for this segment too.

3. If there are 190 SLs, you have room for only 1 more since the limit is 191. In this case, you will probably want to avoid any new SLs until you have cleaned up the segmented library, purging unused SLs, etc.

4. If your program has a lot of RLs, adding more may make the RL segment excessively large. Consider how often the RLs are needed before adding more and thereby producing a code segment that is too large to manage easily.

5. If many programs will share these libraries, then you are better off using SLs rather than RLs. This is because SLs are sharable code segments, whereas RLs must be present as separate copies in each program file that uses them. Furthermore, if any RL is changed, every program that references them must be re-prepared - possibly a horrendous

task.

6. If the routine changes a lot, put it in an SL. If it is in an RL, every program that uses it must be re-prepared to get the latest version of the RL. This can be a nuisance, particularly if the routine is shared by many programs.

Answers to Worksession II-8 (multiprogramming)

1. a) No. Program execution is never simultaneous on an HP 3000.

   b) No. For the same reason as a). It does not matter that the program is different. No two processes run at the same time on an HP 3000.

2. a) Put this program in the E queue. It should have the lowest possible priority so it does not interfere with on-line transactions.

   b) Put this program in the C queue. It is the type of on-line program to favor for execution.

   c) Put this program in the D queue. It is the type of batch program that should be more favored than the overnight printing program a), but that should not interfere with on-line programs such as b).

   d) Put this program in the C queue where it can contend for CPU time with program b). Of the two, the program that executes the longest without requiring disc I/O will probably be favored by the dispatcher.

3. Keeping the data stack small and segmenting code intelligently are the two things a programmer can do to help MPE find space in main memory for your process.

Suggested answers to worksession III-1 (transaction processing)

1. Define a "transaction".

   - A transaction is the smallest complete unit of work performed by a computer and defined by the user.

   - A transaction is a series of logical steps that accept input, process data, and generate output in order to achieve an identifiable result for the user.

2. Give at least one advantage of an interactive transaction processing system.

   - The person who uses the data is the one who interacts directly with the computer.

   - It eliminates the need for a central data processing group to enter and retrieve data from the computer.

   - It provides fast response directly to the people who need the data.

   - The users see the system as their own - are less apt to resist it.

   OR describe one disadvantage of a batch system:

   - The computer system is removed (physically and emotionally) from the people who enter data into it or depend on its output.

   - Response tends to be slower - takes longer to get to the people who need it.

   - Data processing becomes something performed in a mysterious place, the computer room, rather than at terminals in the regular work areas.

Answers to Worksession III-2 (accounting structure)

1. a) Yes, But Mary must be logged on to group COLLECT of account ACCTG since access to INVOICES is restricted to group users. Mary can log on as follows:

   :HELLO MARY.ACCTG

   She does not need to specify the group, since COLLECT is her home group. We know this since MARY is group librarian and group librarian capability is restricted to the home group of the librarian.

   b) MARY need only be logged-on as shown above in order to modify the file INVOICES. As group librarian, she has write access to INVOICES which allows her to modify the file. (Note that she cannot modify CUSTOMER since write access to that file is restricted to its creator.)

2. a) The CUSTOMER file in the COLLECT group of account ACCTG allows read access to ANY user. This means any user in the system. Therefore, a user in account SALES can access the file CUSTOMER by its full name, including the group and account to which the file belongs and the lockword assigned to the file. To illustrate, the file name is:

   CUSTOMER.COLLECT.ACCTG/LOCKWORD

   A user in the group COLLECT of account ACCTG would only have to give the name CUSTOMER and the lockword, not the group and accout names.

   b) The user in SALES cannot modify the file CUSTOMER. Write access to CUSTOMER is restricted to a single user: the file creator, BILL who is a member of COLLECT in ACCTG. No other user in that or any other group can modify the file CUSTOMER.

3. Access to INVOICES is restricted to group users. Therefore, a user in SALES cannot access INVOICES unless he logs on to group COLLECT, account ACCTG, or the file creator (JOHN) specifically releases the file for other users to access, or the account manager or file creator changes the access restrictions on the file.

4. No, a user in group OENTRY cannot run the program CUSTINV. If execution access is limited to group users, this means that only users in the group to which the program file belongs can access it. So, in order to run CUSTINV, the user in OENTRY must be able to log on to the group COLLECT with an acceptable user name and all passwords.

Suggested answers to worksession III-3 (options)

1. The "standard" MPE option has the following advantages:

   - It is simple to develop and test.
   - It requires no special capabilities.
   - Local terminal logic is straightforward.

   This option has the following disadvantages:
   - The end user must log on, run the program, and log off.
   - The overhead for logging on and off is high.
   - Interaction between terminals (global terminal logic) is non-existent.

2a. The three non-standard options we discussed are:

   1. One process per terminal with process handling.
   2. Specialized single program for multiple applications.
   3. Central terminal control with "sons" handling particular applications.

2b. Each of these options has the following advantages and disadvantages:

   The advantages of process handling, one process per terminal:
   - User is isolated from MPE commands - does not log on or off or run program.
   - Data stack and code segments tend to be small.
   - Overhead from logging on and off greatly reduced.

   The disadvantages are:
   - Special capability required (Process Handling).
   - Program testing more complex (though development may be easier).
   - Extra overhead for process creation.
   - Only COBOL II, FORTRAN, SPL can use special capability.

   Advantages of Specialized Single Program:
   - Simple communication between tasks.
   - Data stack for all applications is shared.
   - Fast terminal handling with NOWAIT I/O.

   Disadvantages are:
   - Task handling is complex.
   - Data stack can be very large.
   - Program can be very large.
   - NOWAIT I/O requires privileged mode.

Advantages of Central Terminal Control:
o Central control over all transactions.
o Individual processes make stack and code segments
  easy to manage.
o Fast multi-terminal handling with NOWAIT I/O.

Disadvantages are:
o More complicated programming required.
  - Communication between processes
  - Need SPL routines (unless coding in FORTRAN or COBOL II).
o NOWAIT I/O requires privileged mode.

Answers to worksession III-4 (languages):

1. True. The HP 3000 is a word-oriented machine.

2. False. RPG can be segmented only into fixed-length segments of 1, 2, 3, or 4K (default is 4K). APL cannot be segmented at all. For other languages, however, the answer would be true.

3. a) COBOL or RPG are best for generating formatted reports. In both languages, formatted output is simple to code, uses no extra overhead.

   b) SPL is the only language that can call machine instructions directly; both FORTRAN and COBOL II (but not COBOL '68) can call the MPE Intrinsics.

   c) BASIC is particularly well suited to manipulating character strings.

Answers to worksession III-5 (data entry)

1. a) Character mode is preferable for this task. A small amount of data is transmitted at a time (YES or NO). The program responds immediately to the entered data; it does not need to process a block of data.

   b) Block mode is preferable for this task. The program needs to process an entire block of data before returning to the user for more data. Transferring all the data at once cuts down on the number of terminal/computer transfers; the user can correct data on the screen before it is transferred which further reduces terminal I/O.

2. One terminal in each group is unable to support either block mode transfers or V/3000 (which requires block mode). These terminals are: the 2621, the 2640B, and the 3077.

Suggested answers to worksession III-6 (V/3000 design)

1. It is better forms design to have all forms approximately the same size. The internal record where each form is stored is made large enough to hold the largest form.

   You can improve the design by breaking the 18 line form into two forms, one with the header information and a second for the detail information. You may freeze the header form on the screen and append the second form to it so that the two forms appear as one to the end user.

2. FORMSPEC edits are kept in the forms file which uses stack space to store the form in memory. Therefore, if your application is short of memory, putting all edits in the code saves space. This is particularly true if the edits are long and/or complex.

   Note that disc I/O is not really a factor in this decision; it should be about the same unless the large stack causes memory contention problems that result in swapping. If errors are found, messages must be returned to the operator whether the edits are performed in FORMSPEC or in the program.

3. If all edits are in a FORMSPEC forms file, they should be kept as short and simple as possible. The less characters used, the shorter the stack. It's as simple as that. The system constants ($EMPTY, $DATE, etc) provided to help the designer are also good to keep the edits short.

4. The advantages of FORMSPEC calculations are that they avoid operator error, reduce the need for error checking, produce more accurate data, and save thinking time on the part of the operator.

   The disadvantages are that the calculations add to the size of the forms file, give the operator less control over the data, make correcting errors in entered data more complicated.

Suggested answers to worksession III-7 (V/3000 structure)

1. The code records and the data buffers for V/3000 can use up to 6K words of the DL-DB area of the stack by themselves. Add to that the regular DB positive area of the stack with all the data for the program that uses V/3000 and it becomes clear why V/3000 needs to have extra stack space allocated with the MAXDATA parameter.

   Tests showed that any program running V/3000 should have a stack capacity of at least 6K to hold the information needed to process each form. (Remember, that the smaller the form, the fewer editing specifications, the less stack space is needed. But, even with care in form design, V/3000 needs a minimum stack capacity of 6K.)

   Only MAXDATA provides extra stack space that includes the DL to DB area used for the comarea extension. The STACK parameter only increases the stack between DB and Z; it does not do anything for the DL-DB area needed by the V/3000 code records and buffers.

2. a) Yes
   b) Yes
   c) Yes
   d) Yes
   e) Yes
   f) Yes
   g) Yes

   Situations a) through g) increase the size of the form code record associated with each form. Note that everything associated with a form adds to the code record size, not just the data fields and their edits.

   The length of a field and the number of fields primarily affect the two V/3000 buffers in the DL-DB area. But, also increase the code record size if only because each each field has a name and number and is usually enhanced. In any case, the more fields and the longer each field, the more stack is used.

3. Repeating forms do not have to be brought from disc. A repeating form is simply cleared of previously entered data or "refreshed". Unless it is a repeating form, or is the only form in the file (repeating by default), each new form must be brought from disc into the user stack.

Answers to worksession III-8 (V/3000 data entry)

2. ENTRY cannot be used to take data directly from a forms file and write it to an IMAGE data base. For this purpose, you must write your own program. However, you could use ENTRY to write the data from the forms file to an MPE "batch" file and then write a program that transferred this data to an IMAGE data base. The only advantage to this method would be if you did not want to update the data base on-line. (More on this in module IV).

3. Yes, but only if you use REFSPEC and the REFORMAT program. This is exactly the type of situation that reformatting is good for. You can use ENTRY to accept the data from the new form. It will write one record containing up to 10 part numbers. You can use a REFSPEC file that breaks this record up into separate records that contain data in the form your existing application expects it. Then run REFORMAT to generate the records your application can use.

Suggested answers to worksession III-9 (V/3000 programming)

1. You need only three simple V/3000 procedures to perform this function (VGETNEXTFORM, VINITFORM, VSHOWFORM) once you have opened the forms file and the terminal.

   Therefore, it is not only more efficient but quite easy to code using V/3000 procedures in your program instead of using ENTRY. ENTRY does far more than you need. It is a data entry program with lots of fancy features, so it would be wasteful to use it for this small task.

   You probably would want to use FORMSPEC forms rather than designing them yourself since the enhancements are so easy to enter with V/3000 and are managed automatically.

   Note: the constant data is included in the forms file as initial data.

2. This is a situation in which you really don't need V/3000 at all.

   Unless the selected functions themselves require forms and block-mode transfers, you can save on general overhead and stack space by using simple character mode transfers in this situation.

   If, however, the rest of the application uses V/3000 forms control, you might as well make this initial function selection a menu type form.

   Now, if there are three or more functions to select, getting the user response requires more than one simple transfer in character mode. In this case, you might want to use a V/3000 menu form. It could be not only simpler but more efficient.

Answers to worksession IV-1 (structure)

1. Use structure in this case. Structure favors inquiry which is the main task of this application. Since the data is modified in batch mode, on-line response is not slowed down by on-line updates. The choice of IMAGE or KSAM depends on other factors we will cover later.

2. Use unstructured files. Adding data on-line requires less overheaad for MPE files than for either KSAM or IMAGE. Also, structure is primarily useful for fast on-line inquiries, not needed by this application.

   Note that the unstructured file could be either an MPE file or a stand-alone IMAGE data set.

Answers to worksession IV-2 (using MPE files)

1. a) 2 buffers - this gives you the advantage of pre-reading without hogging memory to the disadvantage of other users.

   b) many records per block - for sequential access, the more records read at a time, the fewer disc reads are needed. This advantage overrides the extra memory space for a large buffer (unless memory is severely limited).

2. a) 1 buffer - random reads gain nothing from multiple buffers, and 0 buffers (NOBUF) means deblocking records.

   b) small blocksize - Since it is unlikely that the records you want are in the same block, there is no advantage to a large block, and a large block takes up memory space.

Answers to Worksession IV-3 (shared files)

1. a) Both programs must lock the file to make sure program "B" gets accurate information.

   b) The following is a suggested solution (students may have many variations on this theme):

   Program "A" opens the file in the late afternoon, adds the day's accumulation of new employees, and closes the file. Program "B" opens the file in the morning, retrieves employee data on-line for most of the day, and then closes the file. At this point "A" can open the file again to add new employees. Since the file is never shared, it need not be locked. Note that the update program "A" can be run as a batch job.

2. Solution b) is the best strategy for this situation. Since program "B" needs the latest information, it must be able to access the file following the latest update. Solution a) is highly efficient for a single program, but it defeats the purpose of sharing the file since locking around the update loop forces program "B" to wait until "A" has completed a series of updates before it can access the file.

3. This is an ideal situation for multi-access. Since the two programs share the same buffer, the data is sure to be added in chronological order regardless of which process writes the next record. Also, because they write to the same buffer, locking is not needed for multi-access.

Answers to Worksession IV-4 (KSAM files)

1. A) The employee name because it is used more frequently (weekly rather than monthly) and because it is used as a key for several functions (full sequential access and approximate key access).

   B) Access by primary key is much faster if the data is loaded in that order. If the employee name is the primary key, then access by employee name is improved, but access by department code is not. Also, although writing records in primary key sequence improves access by primary key, it makes adding new records slower. So, if this application depends on fast on-line updates, there may be a real disadvantage to adding records in key sequence, even though it speeds up retrieval.

   C) Approximate key access (finding the first record with a key value greater than or equal to a specified value) is the type of access you would use to find the first employee whose last name starts with "K". For this purpose, you do not need to know the record number. In fact, KSAM is not really designed for direct access by record number; such access is only possible from FORTRAN or SPL programs.

2. A) The number of records in the data file that are marked for deletion may increase the time it takes to access the file. The access is slowed mainly because of the chance of crossing block boundaries to find the next record. A large number of "deleted" records means that much of the data file contains useless data that still has to be moved to and from buffers.

   B) You can compress the data file by deleting the records marked for deletion. You do this by reloading the file using FCOPY. If the number of "deleted" records is the reason access slowed down, this will improve your access time.

Answers to worksession IV-5 (selecting keys)

1. You need two keys - Customer Name and Zip Code

2. Yes - The customer name may change because of marriage or divorce. The zip code may change if the customer moves.

3. Yes - Zip Code is a duplicate key. Since the entire customer name should be a key (in order to perform function e), the last name should not be defined as a duplicate key item.

4. RDUP - Whenever possible, add duplicate keys randomly, in chronological order.

5. b) - Adding a customer means adding a new entry to the key file and adding a new record to the data file. In general, adds take more disc I/O than retrievals and non-key updates. Therefore, a) should be a less time consuming process than b).

6. d) - Simple retrieval is the least time consuming process. Retrieving records with duplicate keys - function c) takes more time than retrieval by unique keys.

7. e) - Using the entire name as a key makes the last name a partial key. With KSAM, it is easy to retrieve the first record with a partial key value and then read subsequent records in sorted order. Note: nested sorts are not otherwise available in KSAM.

8. First, ask yourself if you really need to retrieve items by Zip Code. If not, then you eliminate a duplicate key from your file. If you must perform this function, consider doing it in off-hours or as a batch job so it does not interfere with on-line activity. Second, ask yourself if you can assign the customer a unique identifier that does not change in the same way a name can change. Using such an identifier as a key means you have a static key item rather than one that may change. Any other ideas?

Answers to Worksession IV-6 (using KSAM files)

1. A) The smaller block size increased the number of levels in the B-tree for one of the keys in the file. This meant that an extra disc access might be required each time a record in the file was accessed. The number of levels in a key is directly related to the number of disc transfers that may be needed to locate a particular value for that key.

   B) Increase the block size, load data into the file, and recheck the number of levels. If the key still needs 4 levels, increase the block size again. Keep doing this until you achieve an optimum block size.

2. Changing the number of buffers for a file is easier than changing block size because block size is a permanent file characteristic. This means that you must rebuild the file then reload the data whenever you change key block size. Changing the number of buffers can be done whenver the file is opened, either in the open procedure or with a :FILE command.

3. A) Both programs must lock the file to insure that the logical record pointer is positioned to the correct record. The update procedures depend on pointer position as well as the procedures that read records in key sequence.

   B) The only way for both these programs to execute successfully without locking is to operate in an exclusive environment. In the situation as outlined, the on-line update program could execute all day, while the sequential reporting program could wait until evening (or early morning) to generate its report.

Answers to worksession IV-7 (data base definition)

Of course, there is no single correct solution. One solution is provided in the data base associated with the demonstration programs. Discuss this solution along with any other solutions the class comes up with.

In response to question 6, suggest an automatic master containing the order date.

Answers to worksession IV-8 (using IMAGE)

1. e) Exclusive modify access is the least capability that allows you to perform the specified tasks. (Note that it actually gives you more than you need since updates of non-key items require less capability than modification which implies adding or deleting entries.) You could also open for concurrent update access or any of the other modify modes - but these require more capability.

2. f) Modify, allow concurrent read is the least capability in this case. There is no reason to allow concurrent modify if the other users plan only to read data.

   You could open with h) to have the system enforce locking. This is a valid choice to insure that the latest data is read, and you don't mind the extra overhead. Note that you can lock in other modes - mode h) only enforces it.

3. c) Read, allow concurrent modify is the least capability for the other users that allows you to update. Again, if you want the system to enforce locking (and use a higher capability), they could open with the mode i).

4. d) Calculated mode lets you go directly to an entry in a master data set using the value of the search item to locate the entry.

5. c) Chained access in either forward or backward direction gives you all entries with the same search item value.

6. c) You must access both a master and a detail. You locate the search item in the master data set and then use the chain information associated with this master to locate the head (or tail) of the chain in the detail data set. The entries you actually want are in the detail.

7. c) Data entry level allows all the users to access the data set. If the users are not looking at exactly the same entry, the access can be concurrent. Note also that there are a number of users, and the transactions will probably be long (verification and update) - more reasons for choosing data entry locking.

8. In the case of data entry locking, it is essential that all users lock the same item. Otherwise, IMAGE is forced to treat the lock as a data set lock to insure that users are not accessing the same entry.

Answers to worksession IV-9 (IMAGE structure)

1. D1 is easier to modify because it has fewer paths (1 rather than 3). This means there are fewer pointers to change in both the detail and the masters when entries are added or deleted.

2. M1 needs 15 words in addition to the data; 5 for the synonym chain, and 5 for each of the two paths.
   M2 and M3 each need 10 words in addition to the data; 5 for the synonym chain, and 5 for the path head.
   D1 needs 4 words in addition to the data (1 path). D2 needs 12 words in addition to the data (3 paths).

3. a) Make the sort item the last item in each entry.
   b) Either make the sort item the entire name, or follow the sort item with other items for the first name and initial.

4. By periodically reloading the data. This will place all entries in contiguous positions on disc in the order of the chained read. This technique will work for D2 only if you are reading along the chain formed by the primary key.

5. You can expect a hashing algorithm that frequently produces the same location for a master entry. This, in turn, results in a series of secondary addresses, and long synonym chains.

STUDENT COMMENT SHEET

Application Design

22808-93001 **NOV 1980**

We welcome your evaluation of this course material. Your comments and suggestions help us improve our courses. The true test of a course is how well it serves your needs in the months following your training. We would appreciate it if, one or two months after attending this class, you would respond to the following questions concerning the course material:

How many months ago did you take this course?

1 or less ______ 1 to 3 ______ 3 or more ______

Did the material serve your needs?

Which topics were more useful to you?

Which topics were least useful to you?

What additional topics would you recommend including in future versions of this course?

Additional comments?

FROM:

Name ______________________________

Company ______________________________

Address ______________________________

____

____

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 492 CUPERTINO, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

Customer Training Manager
Hewlett-Packard Company
Information Systems Division
19447 Pruneridge Avenue
Cupertino, California 95014

FOLD

22808-93001
Printed in U.S.A. 11/80

HEWLETT PACKARD