COURSE OUTLINE

PART I
CONCEPTS
1. Objectives
   a. Course Objectives
   b. Course Structure
   c. NOS/VE Objectives
2. NOS/VE Structures
   a. Packaging
   b. Table Segments
   c. Components
   d. Memory Layout
3. Job Flow
   a. Initiation
   b. Command Processing
   c. Termination
4. File Flow
   a. Open
   b. Transfer
   c. Close

PART II
COMMUNICATION
5. Resources
   a. Documentation
   b. System Initialization
   c. Load Map
6. Internal Communication
   a. Call/Return
   b. Interrupts
   c. Monitor
   d. Traps
7. External Communication
   a. Dual State
   b. Logs and Statistics
   c. Message Generator
   d. Keypoint

PART III
JOB/PROGRAM MANAGEMENT
8. Job Control
   a. Queued File Management
   b. Job Initiation
   c. Job Termination
9. Program Execution
   a. Task Management
   b. Loader
   c. Condition Handling
10. SCL Interpreter
    a. Control
    b. Command Processors

vi
Control Data Private
PART IV

FILES

11. Permanent Files
   a. Control
   b. Set Management
   c. PF Management

12. Logical I/O
   a. File Management
   b. Basic Access Method
   c. Device Management

13. Physical I/O
   a. Page Fault Handling
   b. Device Queue Management
   c. PP Drivers
COURSE STRUCTURE

PART I  CONCEPTS
1. Objectives
2. NOS/VE Structure
3. Job Flow
4. File Flow

PART II  COMMUNICATION
5. Resources
6. Internal Communication
7. External Communication

PART III  JOB/PROGRAM MANAGEMENT
8. Job Control
9. Program Execution
10. SCL Interpreter

PART IV  FILES
11. Permanent File
12. Logical I/O
13. Physical I/O

1-3
Control Data Private
NOS/VE OBJECTIVES

OBJECTIVES
RAM
CONFIGURABILITY
EXPANDABILITY
USABILITY
CONSISTENCY
EFFICIENCY
SECURITY
MIGRATION EASE

STRATEGIES
HARDWARE
SASD
CYBEL
STANDARDS
COMMAND INTERFACE
PROGRAM INTERFACE
DUAL STATE
CP OPERATING SYSTEM
CODE ISOLATION
SYSTEM USING ITSELF

SIS queries
up subcatalog via pt system

 multis in PLT

Book: SIS

1-4
Control Data Private
PHASED RELEASES

R1 Basic Operating System
   Disk and Tape Drivers
   FORTRAN and COBOL
   Dual-State
   Conversion Aids

R2 Stand-Alone System
   Unit Record Drivers
   Interactive Facility
   Products and Utilities

R3 Competitive System
   Networks
   Applications —tran
   Etc.

Control Data Private
HARDWARE CONSIDERATIONS

- JOB STATE VS MONITOR STATE
  - Instruction Privilege
  - Interrupts

- VIRTUAL ADDRESS SPACE
  - Large
  - Segmented
  - Protected

- COMMUNICATION
  - Call/Return
  - Exchange
  - Traps

\[ \text{Call intr} = \text{non-dedicated program} \]
\[ \text{non pageable (wired pages)} \]
\[ \text{RM} = 1 \text{ (execute)} \]

\[ \text{hich proof?} \]
OS HIERARCHY

code  data

OS Modules

Task  Task....

System Job  User Job  User Job...

Job State
Monitor State

code  data

Monitor

code  data

User modules

2-3
Control Data Private
SHARED DATA

code    data
/   \    /   \    /   \
OS Modules OS Modules User modules

Task   Task....  Task   Task....
|     |          |     |          |
System Job User Job User Job...

Job State
Monitor State

Monitor

code       data
TABLE RESIDENCE

Job

Task

Task Private

Task Shared

Ring [3, 13]

Ring [2, 13]

Job Pageable

Job Fixed

Ring 1

Ring [3, 13]

Job Wired

MF Wired

MF Pageable

Segment 2

MONITOR

Working catalog connected file list
Library list

Job file table
Local name table

2-7
Control Data Private
### TABLE SEGMENT ATTRIBUTES

<table>
<thead>
<tr>
<th>Segment</th>
<th>Name</th>
<th>Rings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OSV$MAINFRAME-WIRED</td>
<td>(1,3)</td>
<td>Always in real memory. One per system. Monitor read and write.</td>
</tr>
<tr>
<td>2</td>
<td>OSV$MAINFRAME-PAGEABLE</td>
<td>(1,3)</td>
<td>Pageable. One per system.</td>
</tr>
<tr>
<td>3</td>
<td>OSV$JOB-FIXED</td>
<td>(1,3)</td>
<td>Wired when the job is active; swapped when the job is swapped. One per job. Monitor read and write.</td>
</tr>
<tr>
<td>4</td>
<td>OSV$JOB-PAGEABLE</td>
<td>(2,3)</td>
<td>One per job.</td>
</tr>
<tr>
<td>5</td>
<td>OSV$TASK-PRIVATE</td>
<td>(3,13)</td>
<td>One per task.</td>
</tr>
<tr>
<td>6</td>
<td>OSV$TASK-SHARED</td>
<td>(3,13)</td>
<td>One per job. Pageable. Shared with other tasks of the same job.</td>
</tr>
</tbody>
</table>

CPC behovert sich recht vor

Ringe ke weiger

tables ke movers neen andwe rings

2-8
Control Data Private
FUNCTIONS

User Job

Job Monitor or User Program Task

OS
R1, 2, 3

Record Mgr
Loader
File Mgr
Command Interpreter
Trap Handler

System Job

Job Monitor or Job Scheduler Task

Record Mgr
Loader
File Mgr
Command Interpreter
Trap Handler

MONITOR

- Task Dispatcher
- Physical I/O
- Page Manager

2-9
Control Data Private
## COMMUNICATIONS

### TRANSFER OF CONTROL

<table>
<thead>
<tr>
<th></th>
<th>VOLUNTARY</th>
<th>INVOLUNTARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAME STATE</td>
<td>CALL/RETURN</td>
<td>TRAP INTERRUPT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONDITION BIT SET</td>
</tr>
<tr>
<td>CHANGE TO JOB STATE</td>
<td>EXCHANGE</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>INSTRUCTION</td>
<td></td>
</tr>
<tr>
<td>CHANGE TO MONITOR STATE</td>
<td>EXCHANGE</td>
<td>EXCHANGE INTERRUPT</td>
</tr>
<tr>
<td></td>
<td>INSTRUCTION</td>
<td>MONITOR COND. BIT SET</td>
</tr>
</tbody>
</table>

---

### DATA

<table>
<thead>
<tr>
<th></th>
<th>VOLUNTARY</th>
<th>INVOLUNTARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAME STATE</td>
<td>o PARAMETERS</td>
<td>o CALL STACK</td>
</tr>
<tr>
<td></td>
<td>o TABLES</td>
<td></td>
</tr>
<tr>
<td>CHANGE TO JOB STATE</td>
<td>o SIGNAL, FLAG</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>o SHARED SEGMENT</td>
<td></td>
</tr>
<tr>
<td>CHANGE TO MONITOR STATE</td>
<td>o EXCHANGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PACKAGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o SHARED SEGMENT</td>
<td></td>
</tr>
</tbody>
</table>

---

Free flag set → trap inter
Flag handler

*bit 58 in ncr on-call
off - interrupt

2-10
Control Data Private
EXCHANGE/INTERRUPT

* EXCHANGE JUMP
The exchange jump instruction is used to change state.

Job state programs will exchange to monitor at the PVA in the monitor state XP P register. The system call bit in the MCR is set and the request will be in XO.

Monitor will find the XP of the appropriate task in the XCB entry for that task and exchange to the XP address. A system signal, a system flag or a MCR condition might indicate a special reason for the entry. In that case, monitor will set the free flag in the job state XP and execute the exchange jump. A trap will occur immediately in job state.

* EXCHANGE INTERRUPT
Exchange interrupts occur in job state when a selected monitor condition occurs. Monitor runs at the PVA in the monitor state XP.

* TRAP INTERRUPT
If traps are enabled, a trap interrupt will occur when a selected user condition occurs in the job state or a selected monitor condition occurs in monitor state. In either case there is no exchange. A stack frame is built and the trap handler is executed.
RINGS

DATA SEGMENTS (Read, Write Permission)

Ring numbers R1 and R2 are in the Segment Description Table (SDT).
Read Bracket: 1..R2.
Write Bracket: 1..R1.

Suppose R is the ring of execution, then a data segment may be read if R ≤ R2 and written if R ≤ R1.

CODE SEGMENTS (Read, Execute Permission)

Ring numbers R1 and R2 are in the Segment Description Table (SDT). R3 is in the binding segment entry. The linker constructs the binding segment and will insert R3 only if the procedure is gated.
Read Bracket: 1..R2
Execute Bracket: R1..R2
Call Bracket: R2+1..R3

Suppose that R is the ring of execution, then a call to a procedure in the range R1..R2 is valid and the procedure will run in Ring R; a call to a procedure in the range R2+1..R3 is valid and the procedure will run in ring R2.
## DATA

<table>
<thead>
<tr>
<th>ACCESS FROM</th>
<th>RINGS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>P, Rn</td>
<td>1, 3</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>2, 3</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>3, D</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>B, B</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>X</td>
</tr>
</tbody>
</table>

## CODE

<table>
<thead>
<tr>
<th>CALLED FROM</th>
<th>RINGS</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P, Rn</td>
<td>1, 1, 3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2, 2, 3</td>
<td></td>
<td></td>
<td>X</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2, 3, D</td>
<td>X</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1, D, D</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>B</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2, D, D</td>
<td>X</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>B</td>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B, B, B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2-15
Control Data Private
SUBMIT JOB

/collect text ABC
login JHW81
execute DEF
Logout
**
/submit job ABC

---PMP$EXECUTE---

---JMP$ROUTE---
1

JOB ENTRY

POINTS OF ENTRY
1. Interactive Facility (LOGIN)
2. Remote Host Facility
   (Card reader or NOS Route)
3. SUBMIT

3-3
Control Data Private
JOB QUEUEING

* Queued file manager is part of task services. It processes the jmp$route request.

* Queued files are validated and registered in the $SYSTEM catalog and queued through the known job list (KJL).

* The KJL entry for a job is linked into a thread which represents one of the following states.

  "Deferred"  waiting for a time interval to elapse
  "Queued"    waiting to be initiated
  "Initiated"  active, inactive or swapped out but available for execution
  "Terminated" completed but output files queued for disposition
JOB SCHEDULER

* Job scheduler executes as a task in the system job.
* Job scheduler determines:
  - Order in which jobs in the input queue should be initiated
  - When a job should be swapped into or out of memory
* Some examples of scheduling criteria are:
  - Current priority within job class
  - Job resource requirements
  - Job class and status
  - Current system resource availability
* Job scheduler monitors the available mix of queued and initiated jobs and prioritizes them based on current system usage.
INITIATE JOB

System Job
Scheduler Task

User Job
Job Monitor Task

SCL Interpreter

JCB

JMP$JOB-BEGIN

XCB

AJL

KJL

dispatcher

JMP$CREATE-JOB

PTL

Dispatch Table

3-6 Control Data Private
JOB INITIATION

* When a job is selected, it is given an entry in the Active Job List (AJL)

* Initiate job with the help of monitor initialized the CSS$job fixed segment for the new job. The Job Control Block (JCB) is built. The Execution Control Block (XCB) is initialized with the XP for the first task to run (Job Monitor)

* Monitor creates a Primary Task List (PTL) entry and logs the job monitor task into the dispatch table. The new job waits its turn. Eventually the dispatcher gives the new job its first time slice.

* Job begin initializes CSS$job pageable and CSS#task-shared segments. The command file, output file, and job log are also initialized.

* The SCL interpreter interprets the first command (LOGIN). The user is validated and the prolog is executed.
3

COMMAND PROCESSING

NOS

NOS/VE

User Job

SCL

ABC

JOB

MONITOR

3-8

Control Data Private
SCL INTERPRETER

* SCL reads the command from the $COMMAND file.
* SCL searches for the command in the command list, by default:
  $LOCAL
  $SYSTEM
* If SCL finds the command it runs the CYBIL procedure which must have been provided to process it. This procedure can run as part of the current task or as a new task.
* If the command is a file name call, it might be a program or an SCL procedure file.
* In all cases, the SCL Interpreter passes the command parameter list to a processor. The processor can now use other SCL interfaces to crack the command.
Program Execution Example

Caller Task

Job Monitor

- Job Begin
- SCL Interpreter - EXECUTE

Task Initiator
Task Terminator
Loader

Called Task

User Program

- Entry Point
- Exit

Task Initiator
Task Terminator
Loader

MONITOR

Bount in job fixed

En free control block for new child

Monitor loops DCT

Until DCT

En waarschijnlijk parent task completed.
* Program or command requests program execution
  * Calls task initiator
  * Builds tables for the new task

* Exchanges to system monitor to request task initiation
* Links new task into CPU dispatch list
* CPU is dispatched to new task

* Loader loads object module
  * Loader passes control to initial entry point
* New task executes asynchronously to caller task
* New task calls exit interface

* Cleans up task
  * Exchanges to system monitor to request task termination
* Remove task entries from dispatch list
  * Informs caller that callee has terminated

Parent Task Services .Task initiator
System Monitor
Child Task Services .Program loader
Child Task
Child Task Services .Task terminator
System Monitor
JOB TERMINATION

System Job

Job Terminator
Task

User Job

Job Monitor
Task

Command Processor
"LOGOUT"

SCL

Interpreter

JCB

KOL

XCB

JMP$TERMINE
ATE-JOB

Dispatcher

JMP$END-JOB

R1

Job fixed + XCB

SCT collapse
environment.

Known output list

Control Data Private
## JOB FLOW TABLES

<table>
<thead>
<tr>
<th>Command List</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KJL-jmt$known_job_list_entry (JMDKJL)</td>
<td>This list will be searched by SCL interpreter. If the command is found, a command processor will be called.</td>
</tr>
<tr>
<td>KOL-jmt$known_output_list_entry (JMDKOL)</td>
<td>All jobs in the system have an entry on this table.</td>
</tr>
<tr>
<td>AJL-jmt$active_job_list (JMDAJL)</td>
<td>All output files waiting for routing have entries on this table.</td>
</tr>
<tr>
<td>XCB-ost$execution_control_block (OSDXCB)</td>
<td>All jobs that have been initiated and are not swapped have entries on this list.</td>
</tr>
<tr>
<td>SDT-mmt$segment_descriptor_table (MMDSDT)</td>
<td>Every task in a job has an XCB. This table contains the tasks exchange package.</td>
</tr>
<tr>
<td>TCT-tmt$dispatch_control_table (TMDDCT)</td>
<td>One SDT per task. Every segment in every task has an entry in an SDT. The SDT is used by hardware to relate VM address to real memory addresses.</td>
</tr>
<tr>
<td>PTL-tmt$primary_task_list (TMDPTL)</td>
<td>The dispatcher organizes tasks in this table by priority and chooses the appropriate candidate for execution.</td>
</tr>
<tr>
<td>JCB-jmt$job_control_block (JMDJCB)</td>
<td>This monitor table contains global information about every task in the system.</td>
</tr>
</tbody>
</table>

( ) Common Deck Name

---

3-14

Control Data Private
CREATE FILE

CREATE_FILE_Ex
CYBIL ...
LGO

AMP$FILE ('EX', attributes...
AMP$OPEN ('EX', amc$record,
    attributes, fid...

AMP$PUT-NEXT (fid....
AMP$CLOSE (fid....
PME$EXIT (status)

DETACH_FILE_Ex
INITIATE FILE

CREATE_FILE
CYBIL
LGO

AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
PMP$EXIT

SCL
Command Processor

amp$file

R3 DETACH_FILE

PF
Manager

File
Manager

Catalog

LNT

JFT

Device Manager

SFT

R2

R1

JOB
MTR

Coordinating access to all files from job

Control Data Private

...
FILE INITIATION

* If the first mention of the file is on a command, then the Job File Table (JFT), the Local Name Table (LNT), and the System File Table (SFT) are built. The commands are:

REQUEST - TAPE
REQUEST - TERMINAL
PRINT
FILE
Any PF Command

* For amp$file and some other requests, an auxiliary request table is built. The file tables are built when the file is opened for the first time.
OPEN FILE

CREATE_FILE
CYBIL
LGO

AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
FMP$EXIT

DETACH_FILE

User
Access Method
FAPs

TFT
BAP's open
Open
(File Manager)

JFT

SFT
Device Manager

R1

R2

JOB
MTR

Physical I/O

File Label

4-5
Control Data Private
OPEN

* When the file is opened, the information from commands, program interface requests and AMP$OPEN will be used. The precedence is:

1. AMP$OPEN
2. Commands
3. Requests

* Open entails various processing depending on the file residence and direction of transfer. For example:

DISK File attributes are read or written
TAPE Labels are checked or created (R2)
TERMINAL Attributes are sent to the interactive facility
ACCESS LEVELS

* PUT-NEXT
The access method gets records from the user's buffer and puts them in the file segment that is opened for that purpose (i.e., the system does segment level access). The paging mechanism will take care of real memory and device manager will make sure that space is allocated on the disk. When the filled pages are needed by the system, page manager will instruct the physical I/O component to transfer them.

* GET-NEXT
Again the access method opens a file segment. Page faults will occur when the needed data is not in real memory. But the access methods is not aware of that; it simply copies the records from the file segment to the user's record buffer.

* Segment Level Access
If the user opens the file for segment level access, the file segment is directly addressable by the user.

* Page Manager
After Systems initiation, all disk I/O is initiated by page manager which is a part of system monitor. Page manager runs as the result of a page fault (page seek without file condition).

On a read, page manager will find a free page, initiate physical I/O and suspend the task. When I/O is complete, the task is reactivated. On a write, page manager will find a free page and return to the faulting task. The hardware will mark the page modified when data is written to it. Page manager will write the modified page to backing store (disk) eventually.

4-3
Control Data Private
CLOSE FILE

CREATE_FILE
CYBIL
LGO
AMP$FILE
AMP$OPEN
AMP$PUT_NEXT
AMP$CLOSE
FMP$EXIT
DETACH_FILE

User
Access Method
FAPs

Task Manager
Close
(File Manager)
Device Manager

Dispatcher

TFT
BAM FAP

4-9
Control Data Private
CLOSE/RETURN

* AMP$CLOSE is a request to close this instance of open. The Task File Table (TFT) will be dismantled if there are no other opens. At task termination (EMP$EXIT, for example) all files in the task will be closed once for each instance of open. Job and System File Tables will remain.

* RETURN will cause all references to the file to be deleted. Examples of file disposition are:
  DISK  Temporary files will no longer be accessible. Permanent files will be known through the user's catalog only.
  TAPE  Trailer labels will be processed (R2). The volume will be returned.
  TERMINAL Disconnected and returned.

* At job termination all files are closed and returned.
FILE FLOW
PACKAGING

Run Anywhere
R3

Command Processors

Basic Access Method

PF Manager

File Manager (Open/Close)

TFT

Catalogs

LNT

JFT

R2

Device Manager

FAT

FMD

SFT

R1

Allocate/Deallocate

Physical I/O

Page Manager

JOB

MTR

4-11
Control Data Private
FILE FLOW TABLES

TFT-bat$task_file_table  (BADTFT)  All files opened by a task are controlled by this table. Entries contain pointers to record and block descriptors, file attributes, user request tables, and file access procedures.

LNT-fmt$local_name_table  (FMDLNT)  This table controls the files known to a job by name. It keeps track of the request and attribute info which is global to the job.

JFT-fmt$job_file_table  (FMDJFT)  This table has information about all the files known to the job including unnamed segments like stack and binding.

Catalogs  Each user has a master permanent file catalog.

SFT-dmt$file_descriptor_table  (DMDSFD)  These tables have entries for all files in the system at a given time. Entries point to tables which describe the file on the device.

FMD-dmt$ms_file_medium_descriptor  (DMDFMD)  This table lists the volumes on which a file has been allocated. The portion of the file on a particular volume is called a subfile.

FAT-dmt$ms_file_allocation_table  (DMDFAT)  There is one FAT per subfile. It describes the physical location of the file on the device.

(  ) Common Deck Name

4-12

Control Data Private.
**NCS/VE Project Organization**

**Development: 6-8**

- **Design Team**
  - Project Leader
  - Design Team Reps
  - T/I Reps
  - Members

**Test & Integration**

**Designers**
- Job Mgmt
- Program Mgmt
- I/O
- Dual State
- Deadstart

**Development Groups**
- PFS
- Physical I/O
- Logical I/O
- Dual State Communication
- Logs
- Program Control
- Program Execution
- Job Mgmt
- Command Language
- Monitor
- Maintenance
- Deadstart

*5-2 Control Data Private*
architectural objectives/requirements.

MATERIALS

ERS
  - Prog. Int.
  - Com. Int.
  - Installation

SIS
  NOS/VE
  Procedure & Conv.*

CYBIL

GID
  - Direction
  - Design Analysis
  - DFDs

SES

Data Dictionary
  Design Specification
    - Structure Charts
    - Packaging*
    - Internal Interface*

Source Code

Prog. Lib

Object Lib.
  Load Map
  Integration Notes
  Installation H.B.

User's Guide

Ref. Man.

*Class Handout
NOS/VE PROCEDURES & CONVENTIONS

1. Introduction
2. Design Team
3. Document Review Process
4. Product Identifiers
5. Design Documentation
6. Procedure Interface Conventions
7. NOS/VE Program Library Conventions
8. CYBIL Coding Conventions
9. Keypoint Usage
10. Code Submittat Process
11. NOS/VE Document Maintenance
12. Data Dictionary Conventions
13. Yourdon Methodology

5-4
Control Data Private
PREFIX NOMENCLATURE

SYNTAX:

XXC$ . . . . = Constant
XXT$ . . . . = Type
XXE$ . . . . = Error Code
XXP$ . . . . = Procedure
XXM$ . . . . = Module

XXV$ . . . . = Variable
XXK$ . . . . = Keypoint
XXS$ . . . . = Segment
XXF$ . . . . = File

ID CODE, (XX):

AM = Access Methods
CL = Command Language
IC = Interstate Communication
IF = Interactive Facility
JM = Job Management
OF = Operator Facility
CS = Operating System
PF = Permanent Files
PM = Program Management
RM = Resource Management
SF = Statistics Facility
MM = Memory Management
FM = File Management
MT = Monitor
LO = Loader
CY = CYBIL
IO = Input/output
CC = Object Code

PU = PF Utilities
SR = Conversion Services
DB = Debug
BA = Basic Access
RH = Remote Host
ML = Memory Link
II = Interactive Interface
DP = Display
SY = System
ST = Sets
TM = Task Management
DM = Device Management
LG = Logs
AV = Accounting/Validation
DS = Deadstart
CM = Configuration Management
JS = Job Swapper

3-5
Control Data Private
DECK NAMING CONVENTION

pptzzzz
pp = two character identifier
t = deck type
zzzz = mnemonic name

DECK TYPES

M = CYBIL
P = PP Assembler
A = CP Assembler
X = XREF declarations*
D = Type and Constant declarations*
H = Documentation Header*
I = In-line procedure*
E = Example

* = common deck
INTERNAL INTERFACE

- Chapter Descriptions
- Procedure Descriptions
  - Request Description
  - Parameter Description
  - XREF Declarations
  - Common Deck Calls
- Common Deck Expansions
- Topics
  - CP MONITOR
  - Job Management
  - Resource Management
  - Segment/Memory Mgmt
  - Memory Mgmt
  - Queued Files
  - Program Mgmt
  - Preemptive Communication
  - File Mgmt
- Intrinsics

PF Mgmt
SCL
Interstate Com.
Memory Link
Log Mgmt
System Access
Accounting
Operator Facility

Control Data Private
1 #CALLER_ID (ID)
2 #CALL_MONITOR (REQBLK)
3 #COMPARE (S1, S2): RESULT
4 #COMPARE_COLLATED (S1, S2, TABLE): RESULT
5 #COMPARE_SNAP (LOCK, INITIAL, NEW, ACTUAL, RESULT
6 #DISABLE_TRAPS (OLD_TE)
7 #ENABLE_TRAPS (OLD_TE)
8 #FREE_RUNNING_CLOCK (PORT): INTEGER
9 #HASH_SVA (SVA, INDEX, COUNT, FOUND)
10 #INTERRUPT_PROCESSOR (PORT_SELECTOR)
11 #KEYPOINT (CLASS, EXPRESSION, CODE)
12 #MOVE (SOURCE, DESTINATION, LENGTH)
13 #OFFSET (PVA): INTEGER
14 #PREVIOUS_SAVE_AREA: POINTER
15 #PROGRAM_ERROR
16 #PTR: (DISP, BASE_POINTER): CELL
17 #PURGE_BUFFER (OPTION, ADDRESS)
18 #READ_REGISTER (REGID): REGISTER_VALUE
19 #REAL_MEMORY_ADDRESS (PVA, RSA)
20 #REL (POINTER, BASE_POINTER): INTEGER
21 #RING (PVA): 0..15
22 #RESTORE_TRAPS (OLD_TE)
23 #SCAN (SELECT, STRING, INDEX, FOUND)
24 #SEGMENT (PVA): ...4995
25 #STORE_BIT (BIT_VALUE, BIT_VARIABLE)
26 #TEST_ALARM_CONDITION_REG (GELOPT, BITNUM, BRANCH_EXIT)
27 #TEST_SET_BIT (BIT_VARIABLE, PREVIOUS_VALUE)
28 #TRANSLATE (TABLE, SOURCE, DESTINATION)
29 #WRITE_REGISTER (REGID, REGISTER_VALUE)

Note: see Internal Interface

5-8
Control Data Private
SYSTEM INITIALIZATION PROCESS

1. Library Generation

2. System Generation

3. System Initialization
## CODE SEGMENTS

<table>
<thead>
<tr>
<th>R7-D</th>
<th>1.D.D</th>
<th>2.D.D</th>
</tr>
</thead>
<tbody>
<tr>
<td>R4-6</td>
<td></td>
<td>2.6.6</td>
</tr>
<tr>
<td>R3</td>
<td>1.3.D</td>
<td>1.3.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.3.D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.3.6</td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td>2.2.3</td>
</tr>
<tr>
<td>R1</td>
<td>1.1.3</td>
<td></td>
</tr>
</tbody>
</table>

### JOB STATE

### MTR STATE

### CP MONITOR

XLMTR

### SYSTEM CORE

(TASK MONITOR & CP MONITOR)

XLSnnn

### JOB TEMPLATE

(TASK SERVICES)

XLSnnn

5-11

Control Data Private
## FILE DESCRIPTIONS

<table>
<thead>
<tr>
<th>NAME</th>
<th>AREA</th>
<th>TYPE</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOSVEPL</td>
<td>PL</td>
<td></td>
<td>Contains all code &amp; data source for NOS/VE except program interface.</td>
</tr>
<tr>
<td>OSLPI</td>
<td>PL</td>
<td></td>
<td>Program Interface</td>
</tr>
<tr>
<td>XLMNTR</td>
<td>Monitor</td>
<td>Object Lib.</td>
<td>Library of monitor modules</td>
</tr>
<tr>
<td>XLSnnn</td>
<td>System Core</td>
<td>Object Lib.</td>
<td>Library to run in rings (n,n,n) Task monitor.</td>
</tr>
<tr>
<td>XLJnnn</td>
<td>Job Template</td>
<td>Object Lib.</td>
<td>Library to run in rings (n,n,n) Task services.</td>
</tr>
<tr>
<td>SCMLCB</td>
<td>Monitor</td>
<td>Directives</td>
<td>System Core/Monitor State Linker Control Block</td>
</tr>
<tr>
<td>SCJLCB</td>
<td>System Core</td>
<td>Directives</td>
<td>System Core/Job State Linker Control Block</td>
</tr>
<tr>
<td>JOBLCB</td>
<td>Job Template</td>
<td>Directives</td>
<td>Linker Control Block</td>
</tr>
<tr>
<td>OST</td>
<td></td>
<td></td>
<td>Outboard Symbol Table. List of gated entry points.</td>
</tr>
<tr>
<td>OSTSJxx</td>
<td>System Core</td>
<td>OST</td>
<td>System Core/Job State OST. System with id=xx.</td>
</tr>
<tr>
<td>MTRHDR</td>
<td>Monitor</td>
<td>Segment Files</td>
<td>This HDR describes a collection of &quot;seed&quot; files with names MTR101, MTR102, etc.</td>
</tr>
<tr>
<td>SYSHDR</td>
<td>System Core</td>
<td>Segment Files</td>
<td>This HDR file describes a collection of &quot;seed&quot; files with names SYS101, SYS102, etc.</td>
</tr>
<tr>
<td>JOBHDR</td>
<td>Job Template</td>
<td>Segment Files</td>
<td>This HDR file describes a collection of &quot;seed&quot; files with names JOB101, JOB102, etc.</td>
</tr>
</tbody>
</table>

5-13
Control Data Private
<table>
<thead>
<tr>
<th>NAME</th>
<th>AREA</th>
<th>TYPE</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDSYSC</td>
<td>System Core</td>
<td>Directives</td>
<td>Load Directives for use by the Virtual Environment Generator (VEGEN) to build the system core memory image.</td>
</tr>
<tr>
<td>LDJOE</td>
<td>Job Template</td>
<td>Directives</td>
<td>Load Directives for use by VEGEN to build a job template memory image.</td>
</tr>
<tr>
<td>SYSxx</td>
<td>System Core</td>
<td>Memory Image</td>
<td>This core is sufficient to initialize a system with id=xx in 1M bytes.</td>
</tr>
<tr>
<td>JOBxxyy</td>
<td>Job Template</td>
<td>Memory Image</td>
<td>This template will run under the system with id=xx. The id of the template is yy.</td>
</tr>
<tr>
<td>PP Code</td>
<td></td>
<td></td>
<td>Set of peripheral drivers to run in the PPs.</td>
</tr>
<tr>
<td>DSDIR</td>
<td></td>
<td>Directives</td>
<td>Control the building of the DS Tape.</td>
</tr>
<tr>
<td>DSxxyy</td>
<td></td>
<td>DS Tape</td>
<td></td>
</tr>
</tbody>
</table>
LOADMAP

For each module
- Name
- Section Description
- Entry Point List
- External List

For each file
- Name
- Segment Attributes
- Length
- Address

5-15
Control Data Private
LOAD MODULE

**FILE** = XLJ220  1982/07/18  12:11:53

| WORKING STORAGE = RE_200 | 398 | 2 010 00053989 (2,0,0) |
| WORKING STORAGE = RE_XXX | 210 | 2 01B 00018840 (2,0,0) |
| WORKING STORAGE = CLS$JOB_PAGED_LITERAL | AA | 2 00B 0001DE60 (2,0,0) |
| WORKING STORAGE = CLS$PDT | 4C | 2 00B 0001DF13 (2,0,0) |
| WORKING STORAGE = CLS$PDT_PARAMETERS | 360 | 2 00B 0001DF60 (2,0,0) |
| WORKING STORAGE = CLS$PDT_NAMES_AND_DEFAULTS | 24C | 2 00B 0001DE20 (2,0,0) |

**ENTRY POINT DEFINITIONS**

- CLP$DECLARE_VARIABLE_COMMAND
- CLP$REMOVE_VARIABLE_COMMAND
- CLP$READ_VARIABLE
- CLP$WRITE_VARIABLE
- CLP$PROCESS_ASSIGNMENT
- CLP$DEREFERENCE_VARIABLE
- CLP$DEREFERENCE_NAME
- CLP$COMPLETE_VARIABLE_SCAN

**EXTERNAL ENTRY POINTS REFERENCED**

- CLP$TEST_RANGE
- CLP$CONVERT_INTEGER_TO_STRING
- CLP$CONVERT_TOHexString
- CLP$INITIALIZE_VARIABLE
- CLP$CHECK_NAME_FOR_BOOLEAN
- CLP$CREATE_VARIABLE
- CLP$GET_SET_COUNT
- CLP$DELETE_VARIABLE
- CLP$SCAN_PARAMETER_LIST

5-16  Control Data Private
## SEGMENT DESCRIPTION
### PART 1

<table>
<thead>
<tr>
<th>FILE NAME</th>
<th>LOAD/</th>
<th>RING</th>
</tr>
</thead>
<tbody>
<tr>
<td>PJBX101</td>
<td>1000</td>
<td>* 005 00000000 (2,3,3)</td>
</tr>
<tr>
<td></td>
<td><strong>004</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX102</td>
<td>3AE4</td>
<td>* 005 00000000 (3,0,0)</td>
</tr>
<tr>
<td></td>
<td><strong>005</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX103</td>
<td>1070</td>
<td>* 006 00000000 (3,0,0)</td>
</tr>
<tr>
<td></td>
<td><strong>006</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX110+</td>
<td>318</td>
<td>* 007 00000000 (8,8,8)</td>
</tr>
<tr>
<td></td>
<td><strong>007</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX105</td>
<td>5EBB0</td>
<td>* 01A 00000000 (2,2,3)</td>
</tr>
<tr>
<td></td>
<td><strong>01A</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX106</td>
<td>1FD02</td>
<td>* 01B 00000000 (2,0,0)</td>
</tr>
<tr>
<td></td>
<td><strong>01B</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX107</td>
<td>BAD50</td>
<td>* 01C 00000000 (2,3,0)</td>
</tr>
<tr>
<td></td>
<td><strong>01C</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX108</td>
<td>AB9F5</td>
<td>* 01D 00000000 (2,0,0)</td>
</tr>
<tr>
<td></td>
<td><strong>01D</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX109</td>
<td>17750</td>
<td>* 01E 00000000 (8,8,8)</td>
</tr>
<tr>
<td></td>
<td><strong>01E</strong></td>
<td><strong>00000000</strong></td>
</tr>
<tr>
<td>PJBX111J</td>
<td>6EBB8</td>
<td>* 008 00000000 (2,0,0)</td>
</tr>
<tr>
<td></td>
<td><strong>008</strong></td>
<td><strong>00000000</strong></td>
</tr>
</tbody>
</table>

**Control Data Private**

5-17
SYSTEM INITIALIZATION

1. Deadstart (x.NVEffff)
   System device established
   Memory partitioned
   Page table built
   System Core copied to Real Memory
   NOS exchanges to Monitor (Dispatcher)

2. System Job
   Configuration established
   Job Template established
   System Tasks started
   - Job scheduler & terminator
   - Operator Communication
   - Memory Link
   - Interactive Executive & Remote Host

3. User Job
COMMUNICATION

MECHANISMS

Call
Return
Exchange Jump
Exchange Interrupt
Trap Interrupt

PROCESSORS

Monitor Interrupt Processor (MIP)
Request Processors

Trap Handler
Signal Handler
System Flag Handler
Monitor Fault Handlers
STACK DATA MAPPING

1. SFSA—stack frame save area
   o Typically words 0–4, length in word 2
   o See diagram and CYBIL definition

2. Automatic Variables
   o First two words not used
   o Each variable starts a new word
   o Array and record components are byte aligned unless packed
   o Packed components are bit aligned except characters, integers, and pointers

3. Parameters
   o Each parameter starts a new word
   o VAR parameters are passed as pointers
   o The pointer to the parameters is in A4

6-3
Control Data Private
DATA MAPPING
EXAMPLE

MODULE x;
PROCEDURE ABC,
VAR
  i: 1..10,
  c: string(10),
  a: array 1..3 of string(6),

  XYZ(i,c);

PROCEND ABC;
PROCEDURE XYZ (i:1..10; VAR s:string(10));

SFSA

variables
  i
c

parameters
  i
  c

SFSA
  P
  A0
  A1
  A2
  A3
  A4

Control Data Private
TRANSFER OF CONTROL

AMPSOPEN

USER STACK

- SFSA
- frame
- SFSA

R3 STACK

- frame
- SFSA
- BAP$OPEN

R2 STACK

- frame
- FMP$OPEN

R1 STACK

MONITOR STACK

- frame
- BAP$OPEN

MONITOR XP

- no change

HARDWARE XP

- B $ off
- 3 $ off
- 2 $ off
- 3 $ off
- B $ off

1. User makes a request using program interface e.g. AMPSOPEN.
2. AMPSOPEN checks parameters and calls BAP$OPEN.
3. BAP$OPEN creates task tables and calls FMP$OPEN to update job files.
4. FMP$OPEN returns.
5. BAP$OPEN returns.
6. AMPSOPEN returns.

6-6

Control Data Private
<table>
<thead>
<tr>
<th>Word No.</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VMID*</td>
<td>A0</td>
</tr>
<tr>
<td>2</td>
<td>Flags Traps Enables</td>
<td>A1</td>
</tr>
<tr>
<td>3</td>
<td>User Mask</td>
<td>A2</td>
</tr>
<tr>
<td>4</td>
<td>Monitor Mask</td>
<td>A3</td>
</tr>
<tr>
<td>5</td>
<td>User Condition</td>
<td>A4</td>
</tr>
<tr>
<td>6</td>
<td>Monitor Condition</td>
<td>A5</td>
</tr>
<tr>
<td>7</td>
<td>Kypt Class LPID*</td>
<td>A6</td>
</tr>
<tr>
<td>8</td>
<td>Keypoint Mask</td>
<td>A7</td>
</tr>
<tr>
<td>9</td>
<td>Keypoint Code</td>
<td>A8</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>A9</td>
</tr>
<tr>
<td>11</td>
<td>Process Int. Timer</td>
<td>AA</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>AB</td>
</tr>
<tr>
<td>13</td>
<td>Base Constant</td>
<td>AC</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>AD</td>
</tr>
<tr>
<td>15</td>
<td>Model Dependent Flags</td>
<td>AE</td>
</tr>
<tr>
<td>16</td>
<td>Segment Table Length</td>
<td>AF</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>XF</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Model Dependent Word</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Segment Table Address</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>Untranslatable Pointer</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>Trap Pointer</td>
</tr>
<tr>
<td>37</td>
<td>Debug Index Debug Mask</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Debug List Pointer</td>
</tr>
<tr>
<td></td>
<td>Largest Ring Number</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Top of Stack Ring No. 1</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Virtual Machine Identifier
** Untranslatable Virtual Machine Identifier
*** Last Processor Identification

6-7
Control Data Private
TRANSFER OF CONTROL

1. User calls AWAIT_ACTIVITY_COMPLETION.
2. AWAIT_ACTIVITY_COMPLETION calls PMP$DELAY
3. DELAY processor exchanges with an RB request.
4. Monitor Interrupt processor delays the task.
5. Monitor returns to the task.
6. Delay processor returns.
7. AWAIT_ACTIVITY_COMPLETION returns.

6-8
Control Data Private
<table>
<thead>
<tr>
<th>Event</th>
<th>Program Interface Request</th>
<th>P.I. request 4 amp. AP return</th>
<th>R1 returns</th>
<th>R3 returns</th>
<th>Page Fault</th>
<th>Page fault (on READ)</th>
<th>Call interrupt handler</th>
<th>Call Dispatcher</th>
<th>Return from Dispatcher</th>
<th>Exchange to new task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor XP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor Stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 Stack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2 Stack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3 Stack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All Stack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware XP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T0</td>
<td>6# n</td>
<td>b.frame</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>5#F n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>1# F</td>
<td>empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>3#F</td>
<td>empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>B# n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T5</td>
<td>T6</td>
<td>1# n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>T7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>T8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>T9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>T11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>T12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td>T13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Control Data Private
CALL DISPATCHER?

YES:
- CHECK PERIODIC ACTIVITIES
  - PROCESS I/O COMPLETIONS
  - CONSOLE INPUT MESSAGE
  - SMU RESPONSES

NO:
- Disp. Call is called if:
  - call flag is set
  - time slice has elapsed
  - SIT has run out

EXCHANGE MIGHT BE TO DOS 170 IN DUAL STATE.
SYSTEM CALL PROCESSING

RING 3 of 1

INTRINSIC

I#call_monitor (REQBLK)

XP

code
status
params

RB: Request Block

Job

Monitor REQUEST CODE TABLE

code

high-ring
local priv.
proc

allowed

count
time

Monitor INTERRUPT PROCESSOR

SYSTEM CALL MGR

... processors

PROCEDURE [XDCL] xxp$yyyy (VAR rb:request-block)

mcr ~ system call bit set in XP (High in X0)

6-11

Control Data Private
REQUEST BLOCK

MEMORY MANAGER REQUEST DEFINITIONS
1 MMT$RB ADVISE
2 MMT$RB ASSIGN FLAWED MEMORY
3 MMT$RB ASSIGN REAL PAGE
4 MMT$RB FLAW_PAGE
5 MMT$RB FREE_FLUSH
6 MMT$RB UNFLAW_PAGE

TASK MANAGER REQUEST DEFINITIONS
1 TMT$RB INITIATE JOB
2 TMT$RB INITIATE_TASK
3 TMT$RB CYCLE
4 TMT$RB DELAY
5 TMT$RB EXIT_JOB
6 TMT$RB EXIT_TASK
7 TMT$RB SEND_SIGNAL
8 TMT$RB WAIT SIGNAL
9 TMT$RB CHANGE_SEGMENT_TABLE

Note: see Internal Interface.

6-12
Control Data Private
<table>
<thead>
<tr>
<th>Time</th>
<th>Hardware XP</th>
<th>R1 Stack</th>
<th>R3 Stack</th>
<th>R2 Stack</th>
<th>R1 Stack</th>
<th>Monitor Stack</th>
<th>Task XP Area</th>
<th>Monitor XP Event</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>b# n1</td>
<td>b.frame1</td>
<td>empty</td>
<td>empty</td>
<td>empty</td>
<td>m.frame1</td>
<td>Program starter</td>
<td>MIP</td>
<td>Access Violation</td>
</tr>
<tr>
<td>T1</td>
<td>m# n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B# n1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MIP</td>
<td>Exchange</td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>b# n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>b# n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SFSA</td>
<td>Trap handler - XCB, UCR</td>
<td>pop processing needed</td>
<td>Call default Condition Handler</td>
</tr>
<tr>
<td>T5</td>
<td>3# n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3-frame1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T6</td>
<td>3# n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SFSA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T7</td>
<td>3# n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3-frame2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T8</td>
<td>m# n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m.frame2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T9</td>
<td>m# n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m.frame2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T10</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m.frame1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T11</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T12</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T13</td>
<td>&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TRAP HANDLING

- Trap Handler runs at the ring of the interrupted program.
- Dispose-of-traps runs in ring 1.
- Dispose-of-traps checks the reason for entry in this order:
  1. Monitor Faults
     Segment Access Conditions, MCR conditions and User conditions are merged together. If a user condition handler exists it will be found in the stack. The default condition handler will abort the task in all cases.
  2. Preemptive Conditions
     All signals and flags will be processed if the free flag is set. If the ring of execution is lower than the recognition ring, the critical frame flag will be set in the first stack frame above the recognition ring. In all other cases the signal or flag handler will be called.
  3. Critical Frame Flag
     If the critical frame flag indicates a delayed signal or flag it will be resolved by dispose-of-preemptive-conditions. If the user has established a block exit handler, the handler will be found in the stack frame and called.

- A Daley diagram of these modules is included in the chapter on program management.
SIGNAL PROCEDURES

SEND SIGNAL

PMP$SEND_SIGNAL(recipient, signal, status)

SIGNAL HANDLER

ppP$HANDLE_SIGNAL_***(originator, signal)

DEFINE HANDLER (for test only)

PMP$DEFINE_SIGNAL_HANDLER(id, handler, recog_ring, status)

SIGNALS

<table>
<thead>
<tr>
<th>Memory Link</th>
<th>MLP$ handle_signal interprets 'sub_signals' and calls a handler.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interactive</td>
<td>IFP$handle_signal passes info between the interactive exec., job monitor, and user tasks.</td>
</tr>
<tr>
<td>Callend</td>
<td>PMF$child_terminator_handler.</td>
</tr>
<tr>
<td>Scheduler</td>
<td>JMF$ handle_qfm_ia_signal processes the signal from QF manager that interactive job has been routed.</td>
</tr>
</tbody>
</table>

Memory link

interactive

suspend child

Scheduler

Control Data Private
SYSTEM FLAGS

SET FLAG

PMP$SET_SYSTEM_FLAG(flag_id, recipient, status)

FLAG HANDLER

ppP$HANDLE_FLAG_xxx(flag_id)

DEFINE HANDLER (for test only)

PMP$DEFINE_SYSTEM_FLAG_HANDLER(id, handler, recog_ring, st)

FLAGS

<table>
<thead>
<tr>
<th>Statistics</th>
<th>AVP$monitor_statistics_handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminate</td>
<td>PMP$terminate_flag_handler</td>
</tr>
<tr>
<td>Drop</td>
<td>JMP$handle_drop_job_flag</td>
</tr>
<tr>
<td>Linked Signals</td>
<td>TMP$dispose_mainframe_signals</td>
</tr>
<tr>
<td></td>
<td>This flag indicates that a signal occurred while the task was swapped.</td>
</tr>
</tbody>
</table>
MONITOR FAULTS

FAULT HANDLER

PPF$HANDLE_FAULT_xxx(fault, save_area)

DEFINE HANDLER (for test only)

PMP$DEFINE_MONITOR_FAULT(id, handler, status)

FAULTS

Instruction Specification Error
Address Specification Error
Access Violation
Environment Specification Error
Outward Call/Inward Return

SEGMENT ACCESS CONDITIONS

Read beyond EOI
Write beyond msl
Segment access error
Key lock violation
Ring violation
I/O read error

6-19
Control Data Private
<table>
<thead>
<tr>
<th>Job Mode</th>
<th>TRAP ENABLED</th>
<th>F/F SET</th>
<th>DELAY</th>
<th>F/F CLEAR</th>
<th>MONITOR MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONITOR</td>
<td>HALT</td>
<td>HALT</td>
<td>STACK</td>
<td>HALT</td>
<td>STACK</td>
</tr>
<tr>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
</tr>
<tr>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
</tr>
<tr>
<td>MONITOR</td>
<td>HALT</td>
<td>HALT</td>
<td>STACK</td>
<td>HALT</td>
<td>STACK</td>
</tr>
<tr>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
<td>EXCH</td>
</tr>
<tr>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
<td>TRAP</td>
</tr>
</tbody>
</table>

**BIT NUMBER AND DEFINITION**

- 0: Processor Detected Malfunction
- 1: Memory Detected Malfunction
- 2: Invalid Instruction
- 3: Instruction Specification Error
- 4: Invalid Address
- 5: Access Violation
- 6: Exchange Request
- 7: External Interrupt
- 8: Page Table Search Without Find
- 9: System Call
- 10: System Interval Timer
- 11: Invalid Segment
- 12: Outward Call/Inward Return
- 13: Soft Error Log
- 14: Trap Exception

**Status:** This bit is a flag only and does not cause any hardware action.
<table>
<thead>
<tr>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
<th>TRAP ENABLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>F/F CLEAR OR</td>
<td>F/F SET AND</td>
<td>F/F CLEAR OR</td>
<td>F/F SET AND</td>
<td>F/F CLEAR OR</td>
<td>F/F SET AND</td>
<td>F/F CLEAR OR</td>
<td>F/F SET AND</td>
<td>F/F CLEAR OR</td>
<td>F/F SET AND</td>
<td>F/F CLEAR OR</td>
<td>F/F SET AND</td>
<td>F/F CLEAR OR</td>
<td>F/F SET AND</td>
<td>F/F CLEAR OR</td>
</tr>
<tr>
<td>TRAP MODE</td>
<td>MONITOR MODE</td>
<td>TRAP MODE</td>
<td>MONITOR MODE</td>
<td>TRAP MODE</td>
<td>MONITOR MODE</td>
<td>TRAP MODE</td>
<td>MONITOR MODE</td>
<td>TRAP MODE</td>
<td>MONITOR MODE</td>
<td>TRAP MODE</td>
<td>MONITOR MODE</td>
<td>TRAP MODE</td>
<td>MONITOR MODE</td>
<td>TRAP MODE</td>
</tr>
<tr>
<td>JOB</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
<td>MEH</td>
</tr>
<tr>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
</tr>
<tr>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
<td>STACK</td>
</tr>
</tbody>
</table>

**USER CONDITION REGISTER**

<table>
<thead>
<tr>
<th>BIT NUMBER AND DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

Control Data Private
VIRTUAL ENVIRONMENT PARTITIONING

- The system resources are partitioned between CYBER 170 and CYBER 180 logical machines

- CPU is partitioned using the VMID field in the exchange package. Determines how the CPU will
  - Fetch and interpret instructions
  - Interpret the register file
  - Interpret interrupts

- CPU access to central memory
  - CYBER 170 addresses map into real memory addresses 0–N
  - CYBER 180 addresses map into (N+1) – (memory size–1)

- PPU access to central memory
  - PPUs are assigned to either 170 system or NOS/VE (dedicated)
  - IOU bounds register limits write access to CM

- Channels are software partitioned to access only CYBER 170 or CYBER 180 peripheral devices (except maintenance channel).
MEMORY LINK
INTERNAL INTERFACES

MLP$SIGN_ON
    (name, max_msgs, unique_name, status)

MLP$SIGN_OFF
    (name, status)

MLP$ADD_SENDER
    (name, sender_name, status)

MLP$DELETE_SENDER
    (name, sender_name, status)

MLP$CONFIRM_SEND
    (name, destination_name, status)

MLP$SEND_MESSAGE
    (name, info, signal, message_area, message_length, destination
     name, status)

MLP$FETCH.Receive_LIST
    (name, sender_name, list, count, status)

MLP$RECEIVE_MESSAGE
    (name, info, signal, message_area, msg_length, msg_area
     length, receive_index, sender_name, status)

Control information

7-5
Control Data Private
The NOS program uses CYBIL procedures or COMPASS macros or FORTRAN subroutines to communicate with the NOS/VE job. These translate to 017 instructions which are trapped by the NOS trap handler (NTH).

When an 017 instruction is executed, the NOS trap handler runs in NOS/VE instruction mode. It moves the message into a circular buffer. The entries are called request blocks. Trap handler issues a monitor request to ready the link_helper task.

MLI helper transfers the message from the circular buffer to the mainframe pageable segment using MLI transfer requests. Some NOS/VE applications are signaled when there is a message received for that application.

MLI manages the queue and services the requests issued by MLI helper, Interactive Facility, and so on.

The NOS/VE program transfers the message into its buffer using MLI requests. Applications which use the facility are:

- RHF180
- Interactive Exec.
- Interactive Facility
- Operator Facility
- Interstate Communication (users)

7-3
Control Data Private
ML HELPER

NOTE - The request block is a circular buffer in C170 memory.
REMOTE HOST FACILITY
(Permanent Files)

ENVIRONMENT:
Task in any job.

ACTIVATED:
By command interfaces get_file and replace_file.

PARTNER:
The NVE subsystem routes a NOS job which is brought to a control point. This job signs on to the memory link and begins communication with the remote host facility running on behalf of a NOS/VE task.

PROCESSING:
NOS/VE initiates the transfer. The records are transferred one at a time. Usually the file undergoes some conversion. The conversion is always performed on the NOS/VE side.
REMOTE HOST FACILITY  
(Queued Files)

ENVIRONMENT:

RHIN and RHOUT are tasks in the system job.

ACTIVATED:

By the deadstart procedure.

PARTNER:

NOS system jobs which are always at a control point.

PROCESSING:

NOS/VE asks NOS if there is an input job for NOS/VE. NOS scans the output queue. If there is such a job, it is sent a record at a time. The z-records are converted to v-records by NOS/VE. The file is given to the queued file manager to be entered in the KJL.

PROCESSING (RHOUT):

NOS/VE tells NOS that there is a file in the NOS/VE output queue (KOL). NOS/VE converts the v-records to the z-records and sends them to NOS a record at a time.
INTERACTIVE COMMUNICATION

ENVIRONMENT:
The interactive executive runs as a task in the system job. The interactive FAP is part of task services for every task.

ACTIVATED:
The interactive executive is activated by the deadstart procedure. The interactive FAP is associated with interactive files by command language interpreter.

PARTNER:
PASSON is an application of NAM. PASSON handles all information for the VEIAF application. LOGIN and interactive break are passed to the memory link application called interactive executive. Other commands are passed to the user task.

PROCESSING (INTERACTIVE EXECUTIVE):
When the interactive executive is executed by the job monitor of the system job, it is possible for users of NOS/VE to login. When they do, the executive calls queued file manager to make entries in the input queue (KYL).

PROCESSING (INTERACTIVE FAP):
Eventually the job is scheduled and the job monitor runs. Job monitor executes the command language interpreter (CLI). When CLI opens an interactive file, the interactive FAP is linked and entered. The FAP signs on to the memory link and establishes communication with PASSON. Now each time CLI wants to communicate with the user, it writes or reads the communication file and the FAP handles the memory link transfers.
OPERATOR FACILITY

ENVIRONMENT:
A task in the system job.

ACTIVATED:
By the deadstart procedure.

PARTNER:
NVE is a NOS subsystem. It must occupy a control point from before NOS/VE deadstart until NOS/VE termination.

NVE includes the K display driver. In this role it is similar to PASSON. It transfers the NOS operator's console K display to and from the operator facility via the memory link.

PROCESSING:
The operator console is treated like a terminal. For input, the operator facility uses a FAP which is nearly the same as the interactive FAP. CLI reads and writes the communication file and processes the commands in the same manner as it would for any interactive user. The only difference is that this task of the system job has privilege which extends to command repertoire.

For output, the operator facility uses a collection of FAPs. the K display is divided as shown below. When a procedure writes to a part of the screen, it writes to the lfn which has the FAP for that part of the screen. This FAP adds the information to a screen buffer.

Periodically, the screen buffer is written. Another FAP makes the memory link requests to transfer it to the partner.

---

Diagram:

```
+------------------+
|   K              |
| HEADER           |
|                  |
| OUTPUT           |
|                  |
| RESPONSE         |
|                  |
| PROMPT           |
+------------------+
```

7-18
Control Data Private
INTERSTATE COMMUNICATION

ENVIRONMENT:
Any task in any job.

ACTIVATED:
When OPEN finds icp$fap_control as the FAP attribute of the link file.

PARTNER:
The partner is written by the user in FORTRAN, CYBIL or COMPASS. Requests are available which synchronize the communication with the NOS/VE task.

PROCESSING:
The NOS/VE program does normal I/O (open, get, put) on the file. On OPEN the string variable named in the user_info field of the link file attributes is transferred by the FAP to the NVE application. NVE routes the string (presumably a command stream). The partner starts.

On get (or put), the FAP transfers from (or to) the partner. The FAP enforces a serial protocol; that is, each 'message' must be received before another can be sent.
LOG MANAGER (LGM)

SCL

COMMAND PROCESSORS

INTERNAL LOGGING INTERFACE

LOCAL LOGS_CTL

LOCAL LOG_MANAGER

GLOBAL_LOG_CTL

GLOBAL LOG_MANAGER

INT LOGS_NAMES ARRAY

Control Data Private
LOG MANAGER
INTERNAL INTERFACE

Any functional area
Log manager

LGP$ADD_ENTRY_TO_BINARY_LOG
(log, entry_address, log_address, cycle, status)

LGP$APPEND_JOB_LOG_TO_OUTPUT
(status)

LGP$BUILD_DISPLAY_OF_ASCII_LOG
(log, scroll_size, status)

LGP$INTERCEPT_LOG_IO_REQUEST
(fid, call_block, layer_no, status)
This is a FAP.

LGP$SETUP_ACCESS_TO_LOCAL_LOGS(status)

LGP$SETUP_ACCESS_TO_GLOBAL_LOGS(logs, status)
PROBE

A PROBE IS THAT PORTION OF SOFTWARE RESPONSIBLE FOR COLLECTING AND EMITTING A STATISTIC TO THE STATISTICS FACILITY.

- PROBES ARE EMBEDDED IN KEY AREAS OF THE SYSTEM, BUT ARE NOT SUBJECT TO GUIDELINES LIKE KEYPOINTS.
- THE PRECISE LOCATION OF PROBES AND THE INFORMATION REPORTED WILL BE DETERMINED BY THE REQUIREMENTS OF THE COMPONENTS IN WHICH THEY LIE.
- THE FREQUENCY AT WHICH A PROBE EMITS A STATISTIC TO THE STATISTICS FACILITY IS DETERMINED BY THE SUPERORDINATE COMPONENT.
- A PROBE DOES NOT ASCRIBE ANY INHERENT QUALITIES TO A STATISTIC.
- THERE SHOULD BE A ONE-TO-ONE CORRESPONDENCE BETWEEN A PROBE AND STATISTIC.

NOS/VE STATISTIC

A NOS/VE STATISTIC HAS THREE COMPONENTS:

- STATISTIC CODE: AN ORDINAL THAT UNIQUELY IDENTIFIES THE STATISTIC.
- DESCRIPTIVE DATA: A STRING INDICATING THE OCCURRENCE OF A SYSTEM OR JOB EVENT.
- COUNTERS: A SEQUENCE OF COUNTERS CONTAINING REPORTED VALUES OF SYSTEM OR JOB VARIABLES.

7-25
Control Data Private
PRODUCT STATISTICS COLLECTED BY NOS/VE

In general, the O/S is responsible for collecting job step statistics that can be determined external to the product, that is statistics that the O/S is capable of determining.

For each product identified in SIS section 4.1 that is directly invoked by the user, e.g., via command or as a program initiated task, NOS/VE will record resources used per invocation. Resources accounted for include:

- Total CP-time
- Maximum virtual memory used
- Maximum real memory used
- Average working set size
- CP-time per memory size used
- Number of I/O requests
- Number of data read/written to files

Additional data to be collected for each invocation of a product include:

- Origin of job step - batch command, terminal command, procedure file, executing job.
- Type of termination - normal, product error, time limit, invalid memory request, operator drop, and so on. A recovered condition does not cause product termination.
- Abnormal conditions recovered from.
- Average interactive response time for interactive products - the average elapsed time between input data available and output data issued to terminal.
- The fact that the product was invoked (added to count of the number of separate invocations).
- Number of modules loaded (input units for the loader)
- Source languages of modules loaded (added to language usage count).

7-26
Control Data Private
# STATISTICS MANAGER TABLES

## GLOBAL BINARY LOG FORMAT

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY DATE AND TIME</td>
<td>Statistic Code</td>
</tr>
<tr>
<td>STATISTIC CODE</td>
<td>Job Sequence Number</td>
</tr>
<tr>
<td>STATISTIC IDENTIFIER</td>
<td>Global Task ID</td>
</tr>
<tr>
<td>JOB SEQUENCE NUMBER</td>
<td>Condensing Frequency</td>
</tr>
<tr>
<td>GLOBAL TASK ID</td>
<td>Number of Counters</td>
</tr>
<tr>
<td>CONDENSING FREQUENCY</td>
<td>Descriptive Data Size</td>
</tr>
<tr>
<td>NUMBER OF COUNTERS</td>
<td><strong>COUNTER_1</strong></td>
</tr>
<tr>
<td>DESCRIPTIVE DATA SIZE</td>
<td><strong>COUNTER_2</strong></td>
</tr>
<tr>
<td><strong>COUNTER_N</strong></td>
<td><strong>DESCRIPTIVE DATA</strong></td>
</tr>
</tbody>
</table>

## ACCUMULATION CONTROL

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATISTIC CODE</td>
<td>Accumulation Control Type</td>
</tr>
<tr>
<td>ACCUMULATION CONTROL TYPE</td>
<td>Accumulation Address</td>
</tr>
<tr>
<td>ACCUMULATION ADDRESS</td>
<td>Frequency Address</td>
</tr>
<tr>
<td>FREQUENCY ADDRESS</td>
<td>Threshold</td>
</tr>
<tr>
<td>THRESHOLD</td>
<td>Forward Link</td>
</tr>
<tr>
<td>FORWARD LINK</td>
<td>Backward Link</td>
</tr>
</tbody>
</table>

## GLOBAL AND LOCAL ROUTING CONTROL TABLE

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATISTIC CODE</td>
<td>Identifier</td>
</tr>
<tr>
<td>IDENTIFIER</td>
<td>Routing Control Type</td>
</tr>
<tr>
<td>ROUTING CONTROL TYPE</td>
<td>Enabled</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Condensing Address</td>
</tr>
<tr>
<td>CONDENSING ADDRESS</td>
<td>Threshold</td>
</tr>
<tr>
<td>THRESHOLD</td>
<td>Interval Size</td>
</tr>
<tr>
<td>INTERVAL SIZE</td>
<td>Interval End Time</td>
</tr>
<tr>
<td>INTERVAL_END_TIME</td>
<td>Log Cycle</td>
</tr>
<tr>
<td>LOG_CYCLE</td>
<td>Forward Link</td>
</tr>
<tr>
<td>FORWARD_LINK</td>
<td>Backward Link</td>
</tr>
</tbody>
</table>

---

**7-27**

Control Data Private
FEATURES

CONDENSING

The first counter of a statistic can be condensed, that is, the information will be collected in the counter until either time runs out or a certain number occur. When the condensing threshold is reached, a new entry is logged and collecting starts again. This might be used to count page faults or total monitor time.

ACCUMULATING

Accumulation also involves collecting occurrences of an event. When the threshold (limit) is reached, some action is taken. Typically the job monitor will be signaled and will take further action. Currently this is being used for CP time and SRUs.

BREAKOUT

Sometimes it is necessary to seek local and global statistics of the same thing. An example might be job time. It would be necessary to have total job time as well as the time for individual jobs to compute standard deviation. If breakout is established, the statistics manager will enter in both the local and the global logs.
PROCEDURE [XREF] sfp$establish_system_statistic (identifier: sft$statistic_identifier;
statistic_code: sft$statistic_code;
log_name: pmt$global_binary_logs;
bayout: boolean;
condensing_control: sft$condensing_control;
VAR status: ost$status);

PROCEDURE [XREF] sfp$enable_system_statistic (statistic_group: sft$statistic_group;
VAR status: ost$status;

PROCEDURE [XREF] sfp$disable_system_statistic (statistic_group: sft$statistic_group;
VAR status: ost$status);

PROCEDURE [XREF] Sfp$disestablish_system_stat (identifier: sft$statistic_identifier;
statistic_code: sft$statistic_code;
log_name: pmt$global_binary_logs;
bayout: boolean;
VAR status: ost$status);

PROCEDURE [XREF] sfp$emit_system_statistic (identifier: sft$statistic_identifier;
statistic_code: sft$statistic_code:
descriptive_data: sft$descriptive_data;
counter: sft$counters;
VAR status: ost$status);
ACCOUNTING

AVP$BEGIN_ACCOUNT
- Establish Accounting Stats

AVP$MONITOR_STATISTICS_HANDLER (every2FFFFFF microsec)
- Emit Accounting Stats
- Emit System Stats
- If Accounting Stat exceeds limit, signal job monitor

AVP$END_ACCOUNT
- Call AVP$MONITOR_STATISTICS_HANDLER
- Establish Local Statistic
- Emit System Statistic
- Disestablish Local Statistic

7-30
Control Data Private
STATUS

AM

amc$conflicting__access__level

MYPHY
AMPSREAD
AMC$RECORD

MESSAGE LIBRARY

E File +F1: +P2
issued but opened
for +P3 access.

TEMPLATE CODES

<table>
<thead>
<tr>
<th>+ Fn</th>
<th>+ Xn</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Pn</td>
<td>+ Nn</td>
</tr>
<tr>
<td>+ T</td>
<td>+ En</td>
</tr>
<tr>
<td>+ I</td>
<td>+</td>
</tr>
<tr>
<td>+ C</td>
<td>++</td>
</tr>
</tbody>
</table>

ERROR File MYPHY:
AMPSREAD issued but
opened for AMC$RECORD
access.

System Interface of Program Interface
7-32
Control Data Private
KEYPOINT INSTRUCTIONS

- Special instructions in C180 state which may be collected through PMF hardware or cause a software trap (if selected).
- Program register 22 bit → register 21 → high keypoint enable flag.
- Puts four basic items into 64-bit FIFO buffer entry:
  - Overflow indicator
  - Time stamp
  - Keypoint class
  - Data value

- Obviously requires cooperation from software to work. Conventions are described in SIS/180 and in NOS/VE Procedures and Conventions. Additional data available in Integration Procedures Notebook.

- Control and implementation through common decks and intrinsics.

#KEYPOINT(class, data, id)
KEYPOINT CLASS

- 16 classes to allow hierarchical priority of keypoint collection.
- Class usage defined in SIS/180.

<table>
<thead>
<tr>
<th>Class</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>O/S data</td>
</tr>
<tr>
<td>1</td>
<td>O/S unusual</td>
</tr>
<tr>
<td>2</td>
<td>O/S procedure entry</td>
</tr>
<tr>
<td>3</td>
<td>O/S procedure exit</td>
</tr>
<tr>
<td>4</td>
<td>O/S debug</td>
</tr>
<tr>
<td>5</td>
<td>Reserved for O/S</td>
</tr>
<tr>
<td>6</td>
<td>Product set data</td>
</tr>
<tr>
<td>7</td>
<td>Product set unusual</td>
</tr>
<tr>
<td>8</td>
<td>Product set procedure entry</td>
</tr>
<tr>
<td>9</td>
<td>Product set procedure exit</td>
</tr>
<tr>
<td>10</td>
<td>Product set debug</td>
</tr>
<tr>
<td>11-14</td>
<td>Reserved for end users</td>
</tr>
<tr>
<td>15</td>
<td>PMF hardware start/stop control</td>
</tr>
</tbody>
</table>

Gated procedures

Major internal procedures

XDCI door

Meerkat function works

geometric.
PMF SOFTWARE

- Data collection
  - Operates under NOS/170 with both CP and PP code.
  - Various options to handle counter and event control capabilities.

- Data reduction
  - Keypoint processor
    Common to all keypoint data reduction processing. Accepts keypoint data from three sources:
    - 180 Simulator Keypoint File
    - NOS/VE Software Keypoint File
    - PMF hardware through PMF data collection
  - Data reduction
    There will be multiple data reduction programs as the facility is implemented. Those now underway are:
    "Current" Data Reduction (on the fly)
    provides system level reporting on keypoints and associated data (mainly resource usage information).
    Entry/Exit Analysis Reports
    Provides data on major software modules called by task and associated resource usage.

In the future, there will be a database implementation allowing long term tracking and correlation of data from various collection runs and sources.
FILE NAMES: 1. user_job_name  
2. system_job_name  
   AAAA$, AAAB$, ...

RECOVERY: The $SYSTEM catalog is recovered like any PF catalog. Information in the system file labels (SFL) of the files is sufficient to reconstruct the KJL and the KOL.

Queue file = PF met attributes.
SWAP file = file die on-stant (leeg) als job file wordt gehouden.
SCHEDULING OVERVIEW

SCHEDULING CLASSES

Currently, jobs can be divided into three classes: system, batch, and interactive. Scheduler's class attribute table is used to delineate the classes.

Low, high, and initial priorities are defined as are memory values. The exclude class flag will inhibit the initiation of jobs from this class. The self terminating capability will allow queued jobs of a class to be initiated even though the maximum active jobs for that class have been exceeded. The job will be up long enough to bring itself down. Currently interactive class jobs have this capability.

SWAP CONTROL

a) The maximum number of swapped jobs in a class.
b) The maximum overall number of swapped jobs.

Swapping is initiated as a result of three conditions:

a) If the scheduler determines that the system is thrashing, a candidate will be chosen and swapout will be performed. The two rules given above will be overridden.
b) The scheduler will periodically examine the input and active job queues. If a job in input has a higher priority than one executing, a swap request will be issued for the active job. This swap request obeys the two parameters governing the swap function.
c) If memory contention is high and a terminal break occurs, that job will be swapped.

page aging interval adjustment.

Control Data Private
PRIORITY ADJUSTMENT

Job priority adjustment is limited to aging queued jobs, aging swapped jobs, and adjusting the priorities of executing jobs.

The aging function will increment job priority based on values local to the class. There are two aging increments for each class: one for input list and the other for swap list. The aging function will be performed on a periodic basis.

Executing jobs will have their priorities adjusted according to several factors. If the job has just been swapped in, it will be given a priority boost to prevent it from being swapped out immediately. If the job's ready task count falls to zero, it will lose priority points. (This may or may not initiate swapping.) If the job's time or memory limits are exceeded, it will be switched to another internal class. Currently there are secondary interactive and batch classes.

When a swapped job receives a signal, the scheduler will increase that job's priority which will result in the job being activated sooner.

WORKING SET

The task working set is the collection of pages that are assigned to a particular task at a particular time. Pages are aged by memory manager and the least recently used pages are removed from the task WS. When a new page is needed, the OS finds a free page and adds it to the task WS. Thus the task WS vacillates. For best performance the WS should be fairly constant. This depends on certain programming techniques and memory manager's aging algorithm. See Ch. 13 for more detail.

The job working set is the collection of all the working sets of all the tasks in the job plus the pages shared by the job (e.g., job fixed). When a job is swapped, the job working set is swapped.
SCHEDULING TABLES

**jmt$job_scheduler_table**
- Adjust priority timing
- Age timing
- Page fault max
- WS max
- WS min
- Max AJL entries
- Max swapped jobs

**jmt$class_attribute_table**
- Priority range
- Initial priority
- Max jobs
- Working set size
- Page fault rate
- Time slice: 50 msec
- Aging
- Swap priority

Memory management:
- Reduce aging rate
- Wired + needed free pages
- Zero pages: 8 megabyte

Page aging per page job
SCHEDULING PROCESS

1. Check for Thrashing
   - Add Working Set (ws) from all AJL entries.
   - If the sum is in the thrashing range, swap jobs till the sum is out of that range. Start with the job with largest ws.
   - Stop.

2. Check page fault rate (R2)
   - If page fault rate > page fault max in jnl$job_scheduler_table, increase memory manager's aging interval.

3. Fill Free Memory
   - Built temporary queues for each state (active, queued, swapped) for each class (batch, interactive, system, etc.).
   - Calculate the number of free pages between the current value and ws_max.
   - Select the algorithm (proc). The only R1 algorithm gets the highest priority queued job from each class and compares it with the highest priority swapped job. If the queued job wins it is initiated, otherwise swap. Continue until ws_min <= ws < ws_max.
   - Stop.

Control Data Private
Scheduler creates all segments.
Scheduler initializes Job Fixed.
Initialize_Job_Environment initializes other segments.
JOB CONTROL TABLES

MP
jmt$known_job_list_entry

Name
AJL Ordinal
Scheduling Thread
Job Type
Job Class
Job Mode
Priority
Drop Attribute
Input Source
Label Info
Time Stamp

Label

system catalog

MW
jmt$active_job_list_entry

Lock
Entry Status
KJL Ordinal
Swap Status
Statistics

JF
jmt$job_control_block

Lock
Names/id.
AJL Ordinal
Sense Switches
Input Source
Keyboard bfr.
Accounting Info
Statistics

8-10
Control Data Private
EXECUTION CONTROL BLOCK

XCB
- Exchange Package
- MCR Selections
- Lock
- Flags
- Wait Inhibited
- Task Rethreaded
- Give up CPU
- Task ID
- Priority
  - XCB
  - TCB
  - ST
  - STX
- End Time Out
- Quantum
- Quantum Left
- Monitor Faults
- Paging Stats
- Signals

SDT
- 0 - Page Table
- 1 - Mainframe Wired
- 2 - Mainframe Paged
- 3 - Job Fixed
- 4 - Job Pageable
- 5 - Task Private
- 6 - Task Shared
- 7 - Task Private Rll
- 8 - System Dayfile
- 9 - Job Dayfile

TCB
- Task control block
  - parent/child

SDTX

8-11
Control Data Private
TASK DISPATCHING TABLES

mp$primary_task_list

PTL

mt$primary_task_list_entry

PTL Thread Status (task)
AJL Ordinal
XCB
AJL Thread Swap Status

Mw

tmt$dispatch_control_table

DCT

1
2
3
4
5
6
7
8
9
10

10 Threads

with ladder

8-12
Control Data Private.
TASK DISPATCHING

* Currently (R1) all tasks are on DGT thread 4 unless they have a system table locked. Tasks with a system table locked are put on thread 2, and the rethreaded field in the XCB is marked true.

* All tasks on the highest priority thread get 50 m-sec time slice in a round robin fashion as long as there are active tasks on that thread.

* In future releases, all 10 threads will be used. Different threads will have different time slices. These algorithms have not been defined yet.

* NOS - NOS/VE scheduling is done in NOS and MIP. If the current NOS job has higher priority, NOS runs; if the current NOS/VE task has higher priority, NOS/VE runs. If the priorities are equal (NOS job default = NOS/VE task = 30) then the CPU is toggled between states. Currently 50 ms are awarded to each side but that can be changed to favor one side or the other. NOS trap handler does the timing. Idle is in NOS.
In broer of zuster heeft geen siblins.

draad alleen naar siblings aflopen.

kind. does abort => rethread.
PARENT/CHILD REQUESTS

PMP$VERIFY_CURRENT_CHILD (tid, current)

PMP$SIGNAL_ALL_CHILD_TASKS (signal, status)

PMP$FLAG_ALL_CHILD_TASKS (flag, status)

PMP$REVOKE_PROGRAM_TERMINATION

9-5
Control Data Private
TASK INITIATION

PWSFUNC TASK INITIATION 2.3.0
PWSEXECUTE (prog_desc, prog_parameters, wait, task_id, task_status, status);

CALL Task Services Ring

parent ring = caller ring

validate Program Description

PWS CREATE TASK ENVIRONMENT

PWS INITIATE CHILD TASK

wait

RETURN

old task

PWS TASK BEGIN;

job monitor?

initialize task private

enable preempt common

find executing task id

dbg on?

establish starting proc

initialize object list

initialize module list

initialize library list

from program description or nil

setup loader options

LDP$ LOAD PROGRAM

dsp sys... ERROR

Outward call error
TASK INITIATION (Continued)
TASK TERMINATION LEVELS

1. Unwinding
   - Revoke program termination (Debugger)
   - Pop stack frames—block exit processing
   - Close files at each 'active ring' to ring 3
   - Child Task Cleanup
     - Abnormal—kill all child tasks
     - Normal—await child termination
   - Clean up task environment
   - SCL Clean up

2. Unwinding Impossible
   - Stack, for example, is bad
   - Child task cleanup
   - Clean up task environment

3. Broken Task
   - Monitor detects user fault with traps disabled
   - Fix trap handler tables to see broken task flag

4. Monitor Kill.
LOADER EXECUTIVE

LOM$LOADER_EXECUTIVE
LOP$LOAD_PROGRAM (object_file_list,
module_list,
execute_library_list,
job_library_list,
starting_procedure,
parent_ring,
loader_options,
code_base_pointer,
status)

CALL

determine
initial_ring

addprog_load_libraries

load_object_files

load_module_list

establish_transfer_symbol

satisfyexternals

fix_program_seq_attr.

finish_load_map

RETURN

execute libraries
job libraries

1. Loaded Modules
2. Execute Libraries
3. Job Libraries
4. OSFS TASK SERVICES_LIBRARY - outbound symbol Table
   and JBTemplate

Code - X
Data R, N
Ring Key/Lock
Binding etc.

binding section
OBJECT MODULE INTERNAL FORMAT

* Each object module is a set of records on the object file
* The object record descriptor contains
  o Item type
  o Record length
* Item types
  IDR: Identification of module and attributes
  LIB: Libraries from which to satisfy external references
  SDC: Length and attributes of each section, code, working storage, binding, and all common blocks
  TEX: Text to be placed in each section
  RPL: Text to be repetitively placed in each section
  BIT: Inserts bit-level data into a section
  EPT: Defines an address in a section as an entry point
  RIF: Identifies addresses that must be relocated by the library generator when binding modules together
  ADR: Allows PVAs to be built at load time (when ring, segment number, and offset are known)
  XRL: List of external references to be satisfied
  BTI: Binding template describes the contents of a location in the binding section
  TRA: Terminates the object module and gives the primary entry point
LOCAL FILE LGO R1=11, R2=11, R3=11

<table>
<thead>
<tr>
<th>IDR</th>
<th>LIB</th>
<th>SDC</th>
<th>SDC</th>
<th>SDC</th>
<th>SDC</th>
<th>SDC</th>
<th>TEXT, RPL BIT, REL ADR, XRL EPT, BIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>FTNLIB</td>
<td>CODE</td>
<td>BINDING</td>
<td>WORKING STORAGE</td>
<td>COMMON BLOCKS</td>
<td>CODE, BINDING AND WORKING STORAGE SECTIONS</td>
<td></td>
</tr>
<tr>
<td>TIME &amp; DATE CREATED</td>
<td>ETC.</td>
<td>SECTION</td>
<td>SECTION</td>
<td>SECTIONS</td>
<td>SECTIONS</td>
<td>SECTIONS</td>
<td></td>
</tr>
<tr>
<td>END OF MODULE</td>
<td>END OF MODULE</td>
<td>END OF MODULE</td>
<td>END OF MODULE</td>
<td>END OF MODULE</td>
<td>END OF MODULE</td>
<td>END OF MODULE</td>
<td></td>
</tr>
</tbody>
</table>

USER COMMAND STREAM (VALIDATED FOR RING 11)

FTN, I=MAIN, B=LGO
FTN, I=SUB, B=LGO
LGO

OBJECT MODULE FOR MAIN

OBJECT MODULE FOR SUB

9-12
Control Data Private
MODULE SQACI;

CONST
  MIN = 1,
  MAX = 30;

TYPE
  T_ARRAY = ARRAY [MIN .. MAX] OF INTEGER;

PROCEDURE SQUARER (B: T_ARRAY;
  VAR SQ: T_ARRAY);

VAR
  J: MIN .. MAX;

FOR J := MIN TO MAX DO
  FOREND;
  PROCEND SQUARER;

PROGRAM MAIN;

VAR
  BASE: T_ARRAY;
  SQUARE: T_ARRAY;
  I: MIN .. MAX;

FOR I := MIN TO MAX DO
  BASE [I] := I;
  FOREND;
  SQUARER (BASE, SQUARE);
  PROCEND MAIN;
  MODEND SQACI;

9-13
Control Data Private
<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>INFO RETURNED</th>
<th>SCOPE</th>
<th>USER</th>
<th>DESCRIPTION</th>
<th>SYSTEM</th>
<th>BLOCK</th>
<th>JOB RESOURCE</th>
<th>SEGMENT ACCESS</th>
<th>PROGRAM INTERNAL TIMER</th>
<th>COMBINATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Condition Description passed on FRAME_CAUSE_CONDITION</td>
<td>Current Ring</td>
<td>Selector Name</td>
<td>Selector Handler</td>
<td>Selectors Set of MCR, UCR Loop Prevention Handler</td>
<td>Selectors Set of reason OFF Handler</td>
<td>Selectors Set of reason Segment Number Loop Prevention Handler</td>
<td>Selectors Set of Category Handler</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>All Rings</td>
<td>Interactive</td>
<td>Selector 0-255</td>
<td>Selector 0-255</td>
<td>Selector 0-255</td>
<td>Selector 0-255</td>
<td>Selector 0-255</td>
<td>Selector 0-255</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Frame</td>
<td>All Rings</td>
<td>Current Ring</td>
<td>All Rings</td>
<td>Current Ring</td>
<td>All Rings</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Return, Pop or non-local</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Condition</td>
<td>Current Ring</td>
<td>Selector Handler</td>
<td>Save Area of Frame that caused the condition</td>
<td>Exit</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td>Save Area of Frame that caused the condition</td>
<td></td>
</tr>
</tbody>
</table>

NOTE: See Program Interface
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous block</td>
<td>TS</td>
</tr>
<tr>
<td>caller ring</td>
<td>TS</td>
</tr>
<tr>
<td>child task list</td>
<td>TS</td>
</tr>
<tr>
<td>interpreter mode</td>
<td>TS</td>
</tr>
<tr>
<td>variables</td>
<td>TS</td>
</tr>
<tr>
<td>path</td>
<td>TS</td>
</tr>
<tr>
<td>command source</td>
<td>TS</td>
</tr>
<tr>
<td>parameter value table</td>
<td>TS</td>
</tr>
<tr>
<td>pvt area</td>
<td>TS</td>
</tr>
<tr>
<td>when condition file</td>
<td>TS</td>
</tr>
<tr>
<td>line identifier</td>
<td>TS</td>
</tr>
<tr>
<td>line index</td>
<td>TS</td>
</tr>
<tr>
<td>being executed</td>
<td>TS</td>
</tr>
<tr>
<td>exit position</td>
<td>TS</td>
</tr>
<tr>
<td>label</td>
<td>TS</td>
</tr>
<tr>
<td>kind name</td>
<td>TS</td>
</tr>
<tr>
<td>kind end name</td>
<td>TS</td>
</tr>
<tr>
<td>kind</td>
<td>TS</td>
</tr>
<tr>
<td>command</td>
<td>TS</td>
</tr>
<tr>
<td>for</td>
<td>TS</td>
</tr>
<tr>
<td>if</td>
<td>TS</td>
</tr>
<tr>
<td>input</td>
<td>TS</td>
</tr>
<tr>
<td>proc</td>
<td>TS</td>
</tr>
<tr>
<td>when</td>
<td>TS</td>
</tr>
<tr>
<td>repeat</td>
<td>TS</td>
</tr>
<tr>
<td>while</td>
<td>TS</td>
</tr>
<tr>
<td>task</td>
<td>TS</td>
</tr>
<tr>
<td>block</td>
<td>TS</td>
</tr>
<tr>
<td>loop</td>
<td>TS</td>
</tr>
<tr>
<td>subparameters</td>
<td>TS</td>
</tr>
<tr>
<td>utility</td>
<td>TS</td>
</tr>
</tbody>
</table>

10-6
Control Data Private
Block Stack

Command Source

Next entry
Search mode
global
restricted
exclusive
Kind
catalog
library
Sub commands
name
A commands
A functions
System

Name proc name

Name proc name

10-7
Control Data Private
<table>
<thead>
<tr>
<th>Command</th>
<th>For</th>
<th>IF</th>
<th>Input Proc When</th>
<th>Repeat While</th>
<th>Task</th>
<th>Block Loop</th>
<th>Sub Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kind</td>
<td>Variable value</td>
<td>Condition met</td>
<td>Inherited block</td>
<td>Control exp.</td>
<td>Tid</td>
<td>Kind</td>
<td>Link</td>
</tr>
<tr>
<td></td>
<td>limit</td>
<td>Else</td>
<td>Input</td>
<td></td>
<td></td>
<td>Parent</td>
<td>Current block</td>
</tr>
<tr>
<td></td>
<td>Increment</td>
<td>allowed</td>
<td>block</td>
<td></td>
<td></td>
<td>Tid</td>
<td>Display</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>input</td>
<td></td>
<td></td>
<td>Kind</td>
<td>log</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>lfn</td>
<td></td>
<td></td>
<td>Link</td>
<td>index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fid</td>
<td></td>
<td></td>
<td>Parent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ba</td>
<td></td>
<td></td>
<td>Current block</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>data</td>
<td></td>
<td></td>
<td>Tid</td>
<td>Display</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>device</td>
<td></td>
<td></td>
<td>Link</td>
<td>log</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>prompt</td>
<td></td>
<td></td>
<td>Parent</td>
<td>index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Previous command</td>
<td></td>
<td></td>
<td>Current block</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Status</td>
<td></td>
<td></td>
<td>Tid</td>
<td>Display</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When</td>
<td></td>
<td></td>
<td>Link</td>
<td>log</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>condition</td>
<td></td>
<td></td>
<td>Parent</td>
<td>index</td>
</tr>
</tbody>
</table>

2f1 task shared.

Allen hier de Gip.
BLOCK STACK EXAMPLE

1 login  "job monitor set up" C = command
2 my_proc x

3 PROC name (pl, status)
4 WHEN condition
   file_name
   WHEND"KEY"END
5 IF $value (pl) = specified THEN
6 FOR i = 1 to $set_count (pl) DO
     display_file
7 FOREND
8 ELSE
    display_file all
9 IFEND
PROCEND

8 copy_file a b
   ...
   ...

Block Stack
1 I = command
2 C (...)
3 Proc I = my proc
4 C Proc
5 IF C
6 F W WHENEND
7 F C WHENEND
8 ELSE
9 C copy_file

10-9
Control Data Private
Command List/Block Stack Example

Command List

login
:
:
copy_file a b 2
display_value 7+93
SCU 4
SCU display_library
compare_file a b
edit_file=utility

<table>
<thead>
<tr>
<th>I</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>C</td>
</tr>
<tr>
<td>L</td>
<td>C</td>
</tr>
<tr>
<td>L</td>
<td>T</td>
</tr>
<tr>
<td>L</td>
<td>U</td>
</tr>
</tbody>
</table>

Block Stack

<table>
<thead>
<tr>
<th>I</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td></td>
</tr>
</tbody>
</table>

edit
edit
edit

creol
CREOL
CREOL
display_new_library
quit
end
create_variable my lib
set_command_list my lib
my_proc proc

10-10
Control Data Private
CATALOG TREE STRUCTURE

SYSTEM
  \|-- SET 1
     \|-- FAMILY 1
     \|-- FAMILY 2
        \|-- FAMILY N

     \|-- USER 1
     \|-- USER 2
        \|-- USER N

             \|-- FILE 1
             \|-- CATALOG 11
             \|-- FILE 2
                \|-- CATALOG 12

                    \|-- FILE A
                    \|-- FILE 3
                       \|-- CATALOG 21

                           \|-- FILE A
                           \|-- FILE 5
                              \|-- FILE 6
                                 \|-- FILE 7
                                    \|-- FILE N

11-3
Control Data Private
File allocation is
niet voor set managers.
Kan wel van hondenhok volumes voor
verdubbel en licht van konditaakt volumes voor
device management.
**SET MANAGER TABLES**

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>stt$active_set_table STDAST</td>
</tr>
<tr>
<td>JAST</td>
<td>stt$job_active_set_table STDJAST</td>
</tr>
<tr>
<td>VST</td>
<td>stt$vol_set_table STDVST</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Master VSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member VSNs</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set Owner</th>
<th>Number of Jobs Using Set Root object list locator</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Set Owner Root object list locator</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>VSN</th>
<th>Name Member Master VSN Master</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member VSNs</td>
<td></td>
</tr>
<tr>
<td>VST heap</td>
<td>Segment size fixer</td>
</tr>
</tbody>
</table>

Control Data Private
SET INTERNAL INTERFACES

FROM OUTSIDE SET MGR.

STP$CREATE_SET
STP$ADD_MEMBER_VOL_TO_SET
STP$PURGE_SET
STP$REMOVE_MEMBER_VOL_FROM_SET
STP$ASSOCIATE_CATALOG

FROM WITHIN SET MGR.

STP$CREATE_VOL_SET_TABLE
STP$GET_ROOT_OBJECT_LOCATOR
STP$GET_SET_OWNER
STP$CHECK_CATALOG_ASSOCIATION
STP$CHANGE_ACCESS_TO_SET
STP$SET_END_JOB
FILE MANAGEMENT COMPONENTS

USER

AM

BA

TP

TFT

FM JOB FILES

FM LOCAL NAME

JP

LNT

JP

JFT

SFT

MW

SFT

DM

IO

MM

UIT

QP

PP DRIVER

Control Data Private
BASIC ACCESS METHOD

USER

AM

USER FAP

SYSTEM FAP

8AM FAP

File position info

ART

FTD

TFT

ATTR

FAP CONTROL

FPI

FM JOB FILE MGR

FM LOCAL FILE MGR

LNT

COMMAND TAB.

SFL

DM

12-3
Control Data Private
FILE ACCESS PROCEDURES

1. USER
   Interstate Communication
   User Defined

2. CONNECTED FILE

3. SYSTEM
   Advanced Access Method
   Connected File
   Operator Facility
   Interactive Facility
   Interstate Communication
   Logging

4. BASIC ACCESS METHOD = laadte FAP
ATTRIBUTES

* Permanent attributes are established on the first open of a new file.

* Permanent attributes are never changed (R1).

* Source of permanent attributes:
  - FAP Request
  - Open Request
  - Commands
  - Other program interface requests
  - Defaults

* Source of temporary attributes:
  - Store request
  - Open
  - Commands
  - Other program interface requests
  - SFL
  - Defaults
FILE MANAGER TABLES

Indexed by job file id.

Indexed by file name.

<table>
<thead>
<tr>
<th>JFT</th>
<th>LNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>jfid</th>
<th>tid</th>
<th>global name</th>
<th>permit options</th>
<th>sfd</th>
<th>usage selections</th>
<th>ring attr.</th>
<th>open count</th>
<th>file type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>lnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>segment</td>
</tr>
<tr>
<td>device class</td>
</tr>
<tr>
<td>pf</td>
</tr>
<tr>
<td>global name</td>
</tr>
<tr>
<td>sfd</td>
</tr>
<tr>
<td>jfid</td>
</tr>
<tr>
<td>~ request desc.</td>
</tr>
<tr>
<td>~ label desc.</td>
</tr>
<tr>
<td>~ file desc.</td>
</tr>
<tr>
<td>~ route desc.</td>
</tr>
<tr>
<td>~ SPL</td>
</tr>
<tr>
<td>~ FPI</td>
</tr>
<tr>
<td>ring attr.</td>
</tr>
</tbody>
</table>

640

12-9
Control Data Private
DEFINITIONS

MAU—The minimum addressable unit is the quantum of data transfer between a driver and a mass storage device. It is a constant 2048 bytes in length. Standard software is released with page size $\geq 2048$ bytes (MAU). Special systems could have page size $< 2048$ bytes but page size could never be changed without file conversion.

DAU—The device allocation unit is the quantum of device allocation. It is a device dependent, integral multiple of MAU.

ALLOCATION UNIT—A power of 2 multiples of contiguous DAUs on a device. An allocation unit does not span cylinders on a device. A physical I/O request does not span allocation units. Expressed as A1, A2, A4, A8, A16, A32, A64, A128, A256. Current default = A16 ($16^k$) = 2 pages

<table>
<thead>
<tr>
<th>Capacity</th>
<th>844-4x</th>
<th>885-1x</th>
<th>885-4x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cylinders/Spindle</td>
<td>823</td>
<td>843</td>
<td>843</td>
</tr>
<tr>
<td>Tracks/Cylinder</td>
<td>19</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>MAU/DAU (bytes)</td>
<td>(4096) 2</td>
<td>(4096) 2</td>
<td>(4096) 2</td>
</tr>
<tr>
<td>Total (**10^6 bytes)</td>
<td>151.6</td>
<td>552.5</td>
<td>552.5</td>
</tr>
</tbody>
</table>

Performance

| Seconds/Revolution | .0167 | .0167 | .0167 |
| Transfer rate (bytes/sec) | .589x10^6 | .981x10^6 | 3.924x10^6 |

Allocation

| DAU/A1 (Bytes) | (4096) | 1 | (4096) | 1 |
| DAU/A2 (Bytes) | (8192) | 2 | (8192) | 2 |
| | | | | |
| | | | | |
| DAU/A32 | 32 | 32 | 32 |
| DAU/A64 | (180224) | 44 | 64 | 64 |
| DAU/A128 | 44 | 128 | 128 |
| DAU/A256 | 44 | (655360) | 160 | (655360) |

12-11
Control Data Private
DM TABLES

SFT  SYSTEM FILE TABLE  1 entry/file
FMD  FILE MEDIUM DESCRIPTOR  1 entry/subfile
FAT  FILE ALLOCATION TABLE  1 entry/allocation

DVL  DEVICE LABEL  1/volume
DFD  DEVICE FILE DIRECTORY  1 entry/device file
DFL  DEVICE FILE LIST  1 entry/subfile
DAT  DEVICE ALLOCATION TABLE  1 entry/AU

AVT  ACTIVE VOLUME TABLE  1 entry/volume
MFL  MAINFRAME FILE LIST  1 entry/new file
MAT  MAINFRAME ALLOCATION TABLE  1 entry/available AU

12-13
Control Data Private
DEVICE MANAGER USERS

- **FILE MANAGER**
  - Locally Named File Mgr.
  - File Allocation
  - Set Mass Storage Limit
  - Job File Mgr.
    - Create File
    - Assign File to Device
    - Destroy File

- **MEMORY MANAGER**
  - Store ASID in SFT for Sharing
  - Provide transfer unit offset and length

- **PHYSICAL IO**
  - Device Address for I/O transfers
  - Check initial write of new allocation
  - Flaws

- **MANAGE SETS**
  - Add volume to Set
  - Remove volume from Set

- **MANAGE PFS**
  - Get FMD for storage in PF Catalog
  - Manage FMD on attach/detach
  - Destroy PF
  - Lock and Unlock Catalog File

12-14
Control Data Private
DM FILE TABLES

GLOBAL SFT

lock
global id
ASID
eoi
file limit
FMD

FMD

Header
file type
lock
preset
requested
style

Subfile 1
AVT index
DFL index
FAT

Subfile n

FAT

FAT

FAT

LOCAL SFT

FMD

FAT

FAT
MAGNETIC TAPE PROCESSING

The user opens the file in the usual way and makes get_next and put_next requests in the usual way. The system provides device independence. The only restrictions are that the file can be opened only once and that it must be a sequential file opened for record level access. Labels are not supported on Release 1.

BAM opens the file for segment level access. On the first access, a page fault occurs. The system assigns real memory pages for 10 blocks and allocates a window file on disk to hold 10 blocks. Ten blocks are transferred by the PP driver to real memory. When the transfer is complete, the program continues.

Often the "window" stays in real memory until the program exhausts it. If the pages must be paged out, they are paged to the window file on the disk. Page faults occur when the page resides on disk. The window file never changes size. Device manager advances the segment offset so that the file looks like a normal sequential file.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Count/Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTL</td>
<td>Primary Task List-TM</td>
<td>1 entry/task</td>
</tr>
<tr>
<td>DCT</td>
<td>Dispatch Control Table-TM</td>
<td>1/mainframe</td>
</tr>
<tr>
<td>PT</td>
<td>Page Table-SY Hardware</td>
<td>1 entry/active page</td>
</tr>
<tr>
<td>PFT</td>
<td>Page Frame Table Software</td>
<td>1 entry/page</td>
</tr>
<tr>
<td>PQL</td>
<td>Page Queue List PFT tops of threads</td>
<td>1/mainframe</td>
</tr>
<tr>
<td>AST</td>
<td>Active Segment Table AST index -&gt; ASID</td>
<td>1/active segment</td>
</tr>
<tr>
<td>FMD</td>
<td>File Medium Descriptor</td>
<td>1/file</td>
</tr>
<tr>
<td>FAT</td>
<td>File Allocation Table</td>
<td>1/subfile</td>
</tr>
<tr>
<td>LUT</td>
<td>Logical Unit Table</td>
<td>1/drive</td>
</tr>
<tr>
<td>UIT</td>
<td>Unit Information Table IO request queue</td>
<td>1/drive</td>
</tr>
<tr>
<td>PPIT</td>
<td>PP Interface Table</td>
<td>1/drive</td>
</tr>
</tbody>
</table>

Control Data Private
MODULES

MONITOR INTERRUPT HANDLER

- Receive Page Fault
- Call Memory Manager to process fault
- Call Physical IO Mgr to process completion

DISPATCHER

- Adjust wait status
- Pick next task to execute

MEMORY MANAGER

- Process Page Fault
- Manage Working Set (aging)
- Lock/Unlock pages

PHYSICAL IO

- Link requests
- Alert PP
- Process IO completion status

DEVICE MANAGER

- Provide physical addresses
- Allocate space

PP DRIVER

- Function and status the device
- Read/Write the device
- Read/Write Real Memory

13-4
Control Data Private
PHYSICAL IO

1. PROCESS PAGE FAULT

2. INITIATE PHYSICAL IO

3. PROCESS IO COMPLETION
Software hashes the AST index to assign the ASID.

Hardware hashes the ASID and page offset to find the page table index. A sequential search of the next 32 entries might follow.
PAGING TABLES

link
queue index
AJL ordinal
age
PT index
Active IO count
time stamp
Locked page
task queue
SVA (for debug)

PTL

PT

PFT

POL

FREE
AVAIL.
AVAIL.MOD
SHARED
WIRED

Jn
FIXED
Jn
SHARED
Jn
IO ERROR
Jn
WORKING

Jn AJL

AJL

4*AJLO*max+5

4*AJLO

ordinal (AJLO)

13-8
Control Data Private

Voor zelfde gebruiker
het is een reclaimed
page

iedere entry heeft
nu een link

met in

te ready free, waarin
bekend
prefix
write only at
reassignment

across jobs
monitor node
non mainframe wired

hardware krijgt het op
welke raad

L → PFF = page fault frequency
COMPLETE IO REQUEST

TASK

MTAS$ MONITOR INTERRUPT_HANDLER

TMM$ DISPATCHER

IOP$ PROCESS IO COMPLETIONS

DMP$ WRITE INITIALIZE_TERM

MMM$MEM. MGR. MTR. MODE UNLOCK

PP DRIVER

PPIT

RESP. Q

NO ERRORS—Ready Task
PF ERROR—Notify PF Manager
READ ERROR—Abort Task
WRITE ERROR—Leave page in memory

13-12
Control Data Private
APPENDIX A

PACKAGING
### NOS/VE DESIGN SPECIFICATION

**PART III**

**SYSTEM PACKAGING**

**TABLE OF CONTENTS**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>SYSTEM STRUCTURE</td>
<td>1-1</td>
</tr>
<tr>
<td>1.1</td>
<td>GENERAL STRUCTURE ELEMENTS</td>
<td>1-1</td>
</tr>
<tr>
<td>1.1.1</td>
<td>JOB ELEMENT</td>
<td>1-2</td>
</tr>
<tr>
<td>1.1.2</td>
<td>TASK ELEMENT</td>
<td>1-3</td>
</tr>
<tr>
<td>1.1.3</td>
<td>MODULE ELEMENT</td>
<td>1-5</td>
</tr>
<tr>
<td>1.2</td>
<td>NOS/VE STRUCTURE</td>
<td>1-7</td>
</tr>
<tr>
<td>1.2.1</td>
<td>CPU MONITOR ENVIRONMENT</td>
<td>1-7</td>
</tr>
<tr>
<td>1.2.2</td>
<td>NOS/VE MODULES ENVIRONMENT</td>
<td>1-8</td>
</tr>
<tr>
<td>1.2.2.1</td>
<td>Task Services Modules</td>
<td>1-8</td>
</tr>
<tr>
<td>1.2.2.1.1</td>
<td>TASK SERVICES REQUEST HANDLING</td>
<td>1-9</td>
</tr>
<tr>
<td>1.2.2.2</td>
<td>Task Monitor Modules</td>
<td>1-9</td>
</tr>
<tr>
<td>1.2.3</td>
<td>OPERATING SYSTEM TASKS</td>
<td>1-9</td>
</tr>
<tr>
<td>1.2.4</td>
<td>OPERATING SYSTEM COMMUNICATION</td>
<td>1-10</td>
</tr>
<tr>
<td>1.2.5</td>
<td>OPERATING SYSTEM ENVIRONMENT SUMMARY</td>
<td>1-11</td>
</tr>
<tr>
<td>1.2.6</td>
<td>SEGMENT USAGE</td>
<td>1-12</td>
</tr>
<tr>
<td>1.2.6.1</td>
<td>Ring Assignment for a User Task</td>
<td>1-12</td>
</tr>
<tr>
<td>1.2.6.2</td>
<td>Segment Assignments for User Modules</td>
<td>1-13</td>
</tr>
<tr>
<td>2.0</td>
<td>SYSTEM TABLES AND INTERFACES</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1</td>
<td>GENERAL GUIDELINES</td>
<td>2-1</td>
</tr>
<tr>
<td>2.2</td>
<td>TABLES AREAS</td>
<td>2-2</td>
</tr>
<tr>
<td>2.3</td>
<td>TABLES AREA GUIDELINES</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.1</td>
<td>JOB PRIVATE FIXED</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.1.1</td>
<td>Job Private Fixed Static Section</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.1.2</td>
<td>Job Private Fixed Dynamic Section</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.2</td>
<td>JOB PRIVATE PAGEABLE</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3.2.1</td>
<td>Job Private Pageable Static Section</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.2.2</td>
<td>Job Private Pageable Dynamic Section</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.3</td>
<td>TASK PRIVATE</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.3.1</td>
<td>Task Private Static Section</td>
<td>2-4</td>
</tr>
<tr>
<td>2.3.3.2</td>
<td>Task Private Dynamic Section</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.4</td>
<td>MAINFRAME PAGEABLE</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.4.1</td>
<td>Mainframe Pageable Static Section</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.4.2</td>
<td>Mainframe Pageable Dynamic Section</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3.5</td>
<td>MAINFRAME WIRED</td>
<td>2-6</td>
</tr>
<tr>
<td>2.3.5.1</td>
<td>Mainframe Wired Static Section</td>
<td>2-6</td>
</tr>
<tr>
<td>2.3.5.2</td>
<td>Mainframe Wired Dynamic Section</td>
<td>2-6</td>
</tr>
</tbody>
</table>
1.0 SYSTEM STRUCTURE

A basic objective is to provide a well defined system structure which will result in a highly reliable system and one that can grow over time in an orderly and cost effective manner.

In order to meet this objective, a set of hardware and software conventions are imposed on both user and system code. This allows the normal protection, debugging, loading, code maintenance, accounting, and error handling methods of the user and the system to be the same. This also facilitates movement of services between user and system.

1.1 GENERAL STRUCTURE ELEMENTS

Jobs, tasks and modules represent the basic structure elements for all services provided by NOS/VE. They have the general relationship shown in figure 1. Each element has a set of unique execution attributes, interface conventions and resource requirements. System and application programmers make services available to users with combinations of these elements.

```
System
   \---------\---------\-------
   |          |          |       |
   | Job(1)   | Job(2)   | Job(N) |
   \---------\---------\-------
       |          |          |       |
       | Task(1)  | Task(2)  | Task(N) |
       \---------\---------\-------
           |          |          |       |
           | Module(1)| Module(2)| Module(N) |
```

Figure 1 - Structure Elements

Each level contains a system element which monitors the progress of other elements within that level. The job level contains a system job which schedules, initiates, and terminates (normal or abnormal) user and system jobs. Within each job resides a system task which initiates and terminates tasks of the job. Within each task resides a collection of system modules which assist in the initiation and termination of the task.
1.1.1 JOB ELEMENT

The general facility for presenting work to the system is a job. Jobs run on behalf of a specific user whose identification is the basis of the system access control mechanisms. In addition to batch or interactive jobs that are submitted by end users, the operating system and various subsystems not initiated by end users also run as jobs. Since all jobs are protected and compete for resources via the same mechanism, it is anticipated that the addition of new subsystem jobs will be quite straightforward.

Every job consists of multiple tasks. An important characteristic of a job is that all tasks executing within the job share a common set of operating system services that are determined at the time of job initiation. These service modules, called task services, are the mechanism through which operating system functions are made available. They are constructed from a job template that is selected based on job type. This allows different jobs to have different services.

1.1.2 TASK ELEMENT

A task is the execution of a program. A program is a set of modules organized to perform some specific function (e.g., compile COBOL statements, copy a file). Tasks are protected from one another, can be dynamically created and destroyed, can communicate with other tasks and can execute asynchronously with other tasks. Tasks are the only asynchronous execution unit supported by NOS/VE.

Tasks then are the environment for providing functions that are natural to place outside of the requesting environment. Tasks are requested via an operating system request. They have their own (clock) accounting, scheduling, and execution characteristics. Tasks can come and go independently and represent a mechanism which is used to control memory usage (e.g., each pass of a compiler as a separate task). Protection is enforced by different segment descriptor tables for the caller and callee.

The figure below illustrates a task environment.
Every task looks similar to MOS/VE in that it has an exchange package which defines execution status, a segment descriptor table which defines protection, a queue which defines a communication path and a collection of modules which define the program. The collection of modules can include "user" modules, application or run time service modules and operating system modules. The address space of each task is subdivided by a ring protection hierarchy. An attribute of a module is its ring of execution. Each task will include modules which are protected from each other by executing in different ring brackets.

Company Private Rev 4 October 1980
All tasks, regardless of the type of function they perform, have the same appearance as illustrated below.

<table>
<thead>
<tr>
<th>USER TASK</th>
<th>PRODUCT SET TASK</th>
<th>OS TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER PROGRAM MODULARS</td>
<td>COMPILER MODULARS</td>
<td>OS PROGRAM MODULARS</td>
</tr>
<tr>
<td>PROTECTED</td>
<td>PROTECTED</td>
<td>PROTECTED</td>
</tr>
<tr>
<td>RUN TIME MODULARS</td>
<td>RUN TIME MODULARS</td>
<td>RUN TIME MODULARS</td>
</tr>
<tr>
<td>OS MODULARS</td>
<td>OS MODULARS</td>
<td>OS MODULARS</td>
</tr>
</tbody>
</table>

1.1.3 Module Element

Modules are the environment for the set of services that are natural to place within the environment of the caller. These services are provided as procedures and are interfaced via the standard procedure call. They have the same (clock) accounting, scheduling, and execution characteristics as the caller. Examples include file access methods, loading, table handling, and Fortran object time. The available services can be added dynamically by explicit requests of the loader. Protection enforced by the ring hardware may exist between the caller and callee.
1.2 NOS/VE STRUCTURE

NOS/VE utilizes the task and module structure elements to package the operating system services. Some of its tasks execute as part of the "user" jobs and some execute as part of NOS/VE system jobs. NOS/VE also collects together a set of modules that perform the lowest level operating system functions into a special environment called the CPU Monitor. The operating system services are provided within three basic environments:

- CPU Monitor (one per system)
- NOS/VE Modules (modules within each task)
- Operating System Tasks (executing within "user" jobs, and executing within "system" jobs)

Every request a user makes of the system is translated into communication with one or more of these environments. Whenever operating system extensions are being implemented, the conventions and interfaces of these environments must be understood and used.

1.2.1 CPU MONITOR ENVIRONMENT

CPU Monitor is that portion of the operating system that is most directly related to the hardware environment. It provides:

- Basic intertask communication (signals)
- CPU Dispatching
- Basic CPU Scheduling
- Changing Task Status
- Interrupt Handling
- Page Management
- Basic Physical I/O Management

CPU Monitor is interrupt driven, nonpageable, and represents the most thoroughly debugged, least frequently changed code within the operating system.

1.2.1.1 CPU Monitor Request Handling

CPU monitor requests are only made by Task Services and Task Monitor functions. These requests are made using the hardware exchange instruction. Parameters are passed in the hardware registers.

Company Private Rev 4 October 1980

A-6
1.2.2 NOS/VE MODULES ENVIRONMENT

NOS/VE modules are the set of operating system modules that execute within the environment of a task. These modules perform the operating system functions that are most directly related to the requestor’s environment. To provide for maximum protection and RAM these modules are divided into Task Services modules and Task Monitor modules.

1.2.2.1 Task Services Modules

Task services modules provide the user interface to NOS/VE capabilities for:

- File Management
- Access Methods
- Program Management
- Job Management
- Resource Allocation

Task services is a collection of protected procedures. These procedures are directly callable by user code via the call instruction. The call causes a change in privilege for the called procedure, allowing these operating system services to execute with more or different privileges than the calling procedure. This type of structure allows protected operating system services to execute within the user environment. Task services provide a central interface for all requests and responses made and received by a task. If the requested service is not supported directly by task services, the request is passed on to CPU Monitor or to an operating system task. Task services occupies rings 3 to 6 within each address space. Only ring 3 is used for release 1 of NOS/VE.

1.2.2.2 TASK SERVICES REQUEST HANDLING

There are multiple task service entry points gated to requestors. Every call to a task service must supply a status variable of type esistatus. The parameter rules will conform to those of CYBIL.

1.2.2.2 Task Monitor Modules

Task monitor modules perform the more privileged functions of NOS/VE and execute at rings 1 and 2. These modules are a collection of procedures that interface to NOS/VE basic system tables (e.g. segment table, system file tables, catalogs, execution control tables) and to the CPU Monitor. The ring 2
procedures manage job global tables (i.e. accessible in all tasks of a job). The ring 1 procedures manage system wide tables (i.e. accessible in all tasks of all jobs) and are more privileged and critical to the integrity of the system. Task Monitor procedures are not directly callable by "users"; only NOS/VE Task Services procedures can directly interface to Task Monitor procedures.

1.2.3 OPERATING SYSTEM TASKS

Operating system tasks are those portions of the operating system that are relatively independent of the requestor's environment. They may execute asynchronously to the requestor and provide major portions of:

- Job Management
- Job Scheduling
- Operator Communications
- Device Drivers
- Hardware Maintenance

Execution of a system task is triggered by a signal passed into its communication queue. Tasks may execute in different processors. The device drivers, for example, are system tasks which execute on the IOU.

1.2.4 OPERATING SYSTEM COMMUNICATION

The operating system functions communicate using a basic signal handling service. The signals have a fixed format, a maximum size and are used by the operating system primarily for communication between address spaces. CPU Monitor is responsible for placing signals into the proper signal queue and for notifying the proper Task Monitor that a signal exists. Task Monitor is responsible for taking signals out of the communication queue and passing it to a Task Services signal handler. Routing, based on signal type, to a signal processor within Task Services will be effected by the Signal Handler.

Company Private Rev 4 October 1980

A-8
1.2.5 OPERATING SYSTEM ENVIRONMENT SUMMARY

The following figure summarizes the basic environments and interfaces of NOS/VE.

![Diagram of task and CPU interfaces]

1 - INTERFACED VIA THE CALL INSTRUCTION, CYBIL PARAMETERS FOR COMMUNICATION, RINGS FOR PROTECTION

2 - INTERFACED VIA THE SYSTEM CALL, SIGNALS FOR COMMUNICATION, SEGMENT TABLES FOR PROTECTION

3 - INTERRUPTS ARE PROCESSED BY CPU MONITOR OR ARE TRANSLATED INTO SIGNALS
1.2.6 SEGMENT USAGE

1.2.6.1 Ring Assignment for a User Task

<table>
<thead>
<tr>
<th>AREA</th>
<th>DATA PORTION</th>
<th>CODE PORTION</th>
<th>WHEN CREATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER APPLICATION PROGRAM</td>
<td>WORKING STORAGE, STACK, USER DATA</td>
<td>APPLICATION PROGRAM</td>
<td>AT LOAD TIME ACCORDING TO LIBRARY LIST IN PROGRAM DESCRIPTOR</td>
</tr>
<tr>
<td>PROTECTED RUN TIME MODULES</td>
<td>WORKING STORAGE, STACK</td>
<td>DATA BASE MANAGER</td>
<td></td>
</tr>
<tr>
<td>TASK SERVICES/TASK MONITOR MODULES</td>
<td>WORKING STORAGE, STACK, TABLES FOR JOB, TABLES FOR SYSTEM</td>
<td>RECORD MANAGER LOADER, PROGRAM COMM., TRAP HANDLING</td>
<td>A JOB TYPE TEMPLATE SUPPLIED BY SYSTEM GENERATION WHICH IS USED BY JOB INITIATION</td>
</tr>
</tbody>
</table>

This diagram illustrates:

1. Examples of code which exist at each ring bracket
2. Examples of private data at each ring bracket
3. When the data and code segments are created

Entry points to task services are created by system generation within the loader symbol table and are dynamically linked to external references from user and protected run time procedures by the loader.
1.2.6.2 Segment Assignments for User Modules

The following example demonstrates how the loader allocates and initializes segments based on information contained in compiler generated object text.

- Object Text Topology

  RECORD TYPE
  
  (Identifier record) name, date, generator name
  (section definition) code, binding, working storage, protection
  (interpretive text) text, replication, bit, entry, external
  (transfer...end of text)

- Generated Object Text

  CODE SECTION (R,X)
  . Non selfmodifying instructions

  BINDING SECTION (B)
  . Base address of other sections
  . All procedure descriptions

  STATIC SECTION (R,W)
  . Modifiable data

  LITERAL SECTION (R)
  . Constant data

  DYNAMIC WORKING STORAGE SECTIONS (R,W)
  . Common blocks
  . Data allocated at run time

Company Private Rev 4 October 1980
Mapping sections to segments (assume 2 modules) providing an executable entity.

<table>
<thead>
<tr>
<th>LGO file</th>
<th>Module 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Module 2</td>
</tr>
<tr>
<td></td>
<td>EDF</td>
</tr>
</tbody>
</table>

**Segments**

<table>
<thead>
<tr>
<th>Segment N+1 (R,X)</th>
<th>Segment N+1 (B)</th>
<th>Segment N+2 (R,W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Section M(1)</td>
<td>Binding Section M(1)</td>
<td>Static Data M(1)</td>
</tr>
<tr>
<td>Code Section M(2)</td>
<td>Binding Section M(2)</td>
<td>Static Data M(2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Any Named Common</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Literals M(1)</td>
<td>Universal Heap (Grow)</td>
<td>Run Time Stack (Grow)</td>
</tr>
<tr>
<td>Literals M(2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The binding segment contains pointers to static, literals, code and other binding sections. The advantages of using segments include:

- Independent growth
- Integrity by separation
- Supports code sharing
- Non rewrite of code and constants (paging or swapping)

R - Read
E - Extensible
B - Binding
W - Write
X - Execute

Company Private Rev 4 October 1980

A-12
2.0 SYSTEM TABLES AND INTERFACES

2.1 GENERAL GUIDELINES

The operating system is dependent on the use of tables to provide interfaces between different system modules and between the system and the user, and to describe the basic objects supported by the system and how these objects are related. When a table is defined within the system, consideration must be given to the following six general characteristics.

- **Protection** - Should the information be protected by hardware from inadvertent write operations? Must the information be protected from malicious write/read operations?

- **Scope** - Should the information be local to a user or should it be made global and shareable by other users? In general, information should be globally defined only when required. Keeping information local to a user has two advantages: 1) this information is private and no other user can interfere with it, and 2) if most of the tables required by a job are collected locally, it is easier for the system to keep track of a user (swapping, restart, paging critical tables, etc.).

- **Residence** - Should the information be pageable or locked down? Whenever possible, information should be pageable. It should be locked down only when an obvious efficiency case exists. Three points can be made: 1) System Monitor cannot tolerate access interrupts, so any information referenced by System Monitor must be in real memory at the time of reference, 2) I/O channels use absolute addresses and require that real memory exists when in operation, and 3) there are degrees of paging; that is, some information must be present if a task is to use the CPU and can only be explicitly removed.

- **Life Cycle** - When will the table come into existence and when will it disappear? The data to describe a job is divided into environments which will go away, when the job terminates, when a task terminates, when the system crashes, and environments which will live forever unless explicitly removed.

- **Crash Resistance** - When the system crashes, how will the tables be reconstructed? What impact will there be on recovery if the tables cannot be reconstructed? Will the corrupting of the tables cause a system crash? What protection will be provided to detect corruption?
Structure - The general structure of each of the NOS/VE Tables Area is the same and allocation of entries within a particular table is the same.

The contents (entries) of NOS/VE are position independent, that is,

a) the order and number of static entries in tables areas can vary from build to build;
b) the order and number of static entries in tables areas (task and job private) can vary among job types; and
c) the order and number of dynamic entries in the tables areas can vary among instances of execution.

The allocation of entries in NOS/VE tables should require minimal interaction among development projects; is controlled at the source level; via CYBIL; and is managed by execution and the system generator.

The general structure, allocation technique or order, value assignment tactics of NOS/VE tables should not impose undue constraints on the structure of entries contained in tables areas.

The allocation of entries and the assignment of values to entries in NOS/VE tables should be postponed as long as is feasible - priority order:

a) execution time
   1) first use time
   2) task initiation time
   3) job initiation time
   4) system initiation time
b) system generation time
c) source (compile) time.

### 2.2 TABLES AREAS

<table>
<thead>
<tr>
<th>TABLES AREA</th>
<th>RL, RL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TASK SHARED</td>
<td>3, 13</td>
</tr>
<tr>
<td>TASK PRIVATE</td>
<td>3, 13</td>
</tr>
<tr>
<td>JOB PRIVATE PAGEABLE</td>
<td>2, 13</td>
</tr>
<tr>
<td>JOB PRIVATE FIXED</td>
<td>1, 3</td>
</tr>
<tr>
<td>MAINFRAME PAGEABLE</td>
<td>1, 3</td>
</tr>
<tr>
<td>MAINFRAME WIRED</td>
<td>1, 3</td>
</tr>
</tbody>
</table>

Company Private Rev 4 October 1980

A-14
2.3 TABLES AREA GUIDELINES

2.3.1 JOB PRIVATE FIXED

The Job Private Fixed tables area is the container for tables shared among monitor and all tasks of a job. Job Private Fixed tables reside in non-pageable memory because of monitor access. Therefore, care should be exercised to minimize the amount of space allocated to entries which are not accessed by monitor.

2.3.1.1 Job Private Fixed Static Section

The Job Private Fixed static section is the container for statically allocated tables entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Job Private Fixed tables area. Statically allocated table entries are those which are somewhat constant in nature for the duration of the job. Such entries may also be "root" pointers to dynamically allocated entries in the Job Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.1.2 Job Private Fixed Dynamic Section

The Job Private Fixed dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) tables entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the job. Dynamic entries whose lifetime is less than that of the job must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.2 JOB PRIVATE PAGEABLE

The Job Private Pageable tables area is the container for tables shared among all tasks of a job. Table entries residing in this tables area are not accessible by monitor.

Company Private Rev 4 October 1980

A-15
2.3.2.1 Job Private Pageable Static Section

The Job Private Pageable static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Job Private Pageable tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the job. Such entries may also be "root" pointers to dynamically allocated entries in the Job Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.2.2 Job Private Pageable Dynamic Section

The Job Private Pageable dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the job. Dynamic entries whose lifetime is less than that of the job must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.3 TASK PRIVATE

The Task Private tables area is the container for tables shared among procedures in task services and task monitor of a task. Task Private is pageable. Table entries residing in this tables area are not accessible by other tasks or monitor.

2.3.3.1 Task Private Static Section

The Task Private static section is the container for statically allocated tables entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Task Private tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the task. Such entries may also be "root" pointers to dynamically allocated entries in the Task Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

Company Private Rev 4 October 1980

A-16
2.3.3.2 Task Private Dynamic Section

The Task Private dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the task. Dynamic entries whose lifetime is less than that of the task must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

2.3.4 MAINFRAME PAGEABLE

The Mainframe Pageable tables area is the container for tables shared among all jobs in the system. This tables area is writable by all task monitors and readable up to task services. The mainframe pageable tables area is not accessible to monitors.

2.3.4.1 Mainframe Pageable Static Section

The Mainframe Pageable static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Mainframe Pageable tables area.

Statically allocated table entries for those which are somewhat constant in nature for the duration of the system. Such entries may also be "root" pointers to dynamically allocated entries in the System Private tables area.

The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.4.2 Mainframe Pageable Dynamic Section

The Mainframe Pageable dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the system. Dynamic entries whose lifetime is less than that of the system must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.

Company Private Rev 4 October 1980

A-17
2.3.5 MAINFRAME WIRED

The Mainframe Wired tables area is the container for tables shared among monitor and all jobs in the system. The Mainframe Wired tables reside in wired memory due to monitor access. Therefore, care should be exercised to minimize the amount of space allocated to entries which are not accessed by monitor.

2.3.5.1 Mainframe Wired Static Section

Only monitor software can allocate static table entries in the Mainframe Wired static section.

The Mainframe Wired static section is the container for statically allocated table entries. Static entries are allocated at compile time, via CYBIL static variable declarations, which specify the Mainframe Wired tables area.

Statically allocated table entries are those which are somewhat constant in nature for the duration of the system. Such entries may also be "root" pointers to dynamically allocated entries in the System Private tables area. The allocator of a static entry is responsible for the initial value assignment to that entry.

2.3.5.2 Mainframe Wired Dynamic Section

The Mainframe Wired dynamic section is the container for dynamically allocated (CYBIL allocate or next statements) table entries. Dynamic entries vary in number and size - their lifetime is often less than the life of the system. Dynamic entries whose lifetime is less than that of the system must be freed (CYBIL free statement) when their lifetime expires - the responsibility for freeing lies with the ultimate allocator.
DAY 1 ASSIGNMENTS

1. Read the SYSTEM PACKAGING tract in Appendix A of the handout.

2. Get copies of the following:
   a. NOS/VE Procedures & Conventions (1/person)
   b. Loadmap in the DEV1 catalog (1/4 people)
   c. Hints and Integration Procedures Notebook (Optional)
   d. Sorted list of NOS/VE Modules (1/person)
   e. Internal Interface (1/person)

3. Answer the following questions
   a. What is a task? What does each task have uniquely? (tables, modules, etc) What does each task share with other tasks of the same job?
   b. What is a job template? How does it relate to the system core?
   c. How do jobs interface to one another; tasks; modules?
   d. Note the privilege of each of the following components; name a function of each:
      1. CPU Monitor
      2. R1 (Task Monitor)
      3. R2 (Task Monitor)
      4. R3 (Task Services)
      5. Operating system task
   e. The Execution Control (XCB) contains the exchange package for a task. Its residence is in the Job Fixed (JF) segment. SCL Interpreter's block stack is changed whenever a command is processed. It is resident in the Task Shared (TS) segment.

Discuss the choice of residence for these tables based on the following criterion:
Protection
Scope
Residence
Life Cycle

When is each table initiated?
DOCUMENT RETRIEVAL

1. NOS/VE Procedures & Conventions
   SES,MAD,LISTPC
2. LOADMAP
   SES,DEV1,NVEMAP PMPXX TWO PRINT
   SES,DEV1,NVEMAP PMPXXYY TWO PRINT
2. Helpful Hints
   SES,DEV1,LISTHINTS
2. Sorted List of NOS/VE Modules
   SES,DEV1,NVEREP PRINT
1. Internal Interface
   SES,MAD,LISTCII or
   SES,MAD,LISTNII
2. Integration Procedures Notebook
   ATTACH,IPNDOC/UN=DEV1
   SES.PRINT,IPNDOC
2. Selected Modules
   SES,DEV1,LISTNVE..
   (list of deck names)..
   PRINT
2. Tables
   SES,DEV1,COMLIST..
   id=(list of 2-char ids)..
   api=NOSVEPL un=INT2..
   PRINT
2. Selected Tables - 'tab' is a
   CYBIL module with 'call's.
2. Common Deck Cross Reference
   SES,SCL,XREFCO
2. General Internal Design
   ATTACH,GIDP1R6/UN=MAD
   SES,FORMAT GIDP1R6 TXTFORM

NOTES:
1. Documented in NOS/VE Procedures and Conventions
2. Documented in Integration Procedures Notebook
1. Use a LOADMAP to complete the table on the next page.

2. Read Ch. 2 (CPU Monitor) in Internal Interface, Ch. 8 (Program Mgmt) in Internal Interface, Ch. 30 (Intrinsics) in Internal Interface, Ch. 8 (Program Library Conventions) in NOS/VE Procedures & Conventions, Ch. 1 (NOS/VE System Overview) in Integration Procedures Notebook, Ch. 2 (Overview of Integration Process,(Sections 1-4) in Integration Procedures Notebook.)

3. Fill in the events on the provided time line
   a. User program is running (R11)
   b. User establishes a condition handler for arithmetic overflow (R3)
   c. pmp$establish_condition_handler returns.
   d. User generates the overflow condition; trap handles runs.
   e. Trap handler calls user condition handler.
   f. User condition handles returns.
   g. Trap handler returns.

4. Fill in the events on the provided time line
   a. SCL Interpreter reads 'ATTACH_FILE...' from SCOMMAND (R11) and calls the command processor (clp$attach_command) (R11)
   b. clp$attach_command calls pfp$attach (R11)
   c. pfp$attach calls pfp$r2_attach (R2)
   d. pfp$r2_attach calls pmp$delay because file is not available (R2)
   e. pmp$delay exchanges (to monitor)
      ...
   f. Monitor exchanges after delay
   g. pmp$delay returns
   h. pfp$r2_attach calls fmp$attach_job_file since file is available; LNT and JFT entries will be built.
   i. fmp$attach_job_file returns
   j. pfp$r2_attach returns
   k. pfp$attach returns
   l. clp$attach_command returns
   m. SCL Interpreter reads from SCOMMAND.
<table>
<thead>
<tr>
<th>Rings</th>
<th>(1,1,3)</th>
<th>(1,3,3)</th>
<th>(1,1,1)</th>
<th>(1,1,3)</th>
<th>(1,3,3)</th>
<th>(1,3,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCBB</td>
<td>1006 0000</td>
<td>1208 0000</td>
<td>8010 0000</td>
<td>8350 0000</td>
<td>3580 0000</td>
<td>1850 0000</td>
</tr>
<tr>
<td>Do $A 0000</td>
<td>1 6 $</td>
<td>6</td>
<td>1 6 1 2 0</td>
<td>8</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>PMA-</td>
<td>1 6</td>
<td>1 6</td>
<td>2 0</td>
<td>1 6</td>
<td>1 6</td>
<td>2 0</td>
</tr>
<tr>
<td>Procedure/Variable Name</td>
<td>TMP $ delay</td>
<td>PMP $ cycle</td>
<td>MTP $ Begin</td>
<td>FMTP $ Begin</td>
<td>FMTP $ Begin</td>
<td>FMTP $ Begin</td>
</tr>
<tr>
<td>Module Name</td>
<td>PMP $ compiler</td>
<td>PMP $ compiler</td>
<td>XDEL</td>
<td>XDEL</td>
<td>XDEL</td>
<td>XDEL</td>
</tr>
<tr>
<td>Notes</td>
<td>PMP $ compiler</td>
<td>PMP $ compiler</td>
<td>XDEL</td>
<td>XDEL</td>
<td>XDEL</td>
<td>XDEL</td>
</tr>
<tr>
<td>CUPS Scan command file</td>
<td>CUPS Scan command file</td>
<td>CUPS Scan command file</td>
<td>CUPS Scan command file</td>
<td>CUPS Scan command file</td>
<td>CUPS Scan command file</td>
<td>CUPS Scan command file</td>
</tr>
<tr>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
</tr>
<tr>
<td>Interpreter command file</td>
<td>Interpreter command file</td>
<td>Interpreter command file</td>
<td>Interpreter command file</td>
<td>Interpreter command file</td>
<td>Interpreter command file</td>
<td>Interpreter command file</td>
</tr>
<tr>
<td>Inter-</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
</tr>
<tr>
<td>Variable</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
</tr>
<tr>
<td>Notes</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
<td>Load command file</td>
</tr>
</tbody>
</table>

**Note:** The table contains various entries related to system configurations and commands, but the text is not clearly legible due to the handwriting style.
<table>
<thead>
<tr>
<th>Event</th>
<th>Monitor</th>
<th>Task</th>
<th>Program</th>
<th>Call</th>
<th>Trap Handler</th>
<th>User Trap Handler</th>
<th>Return</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepare parameter on stack</td>
<td>prepare parameter on stack</td>
<td>prepare parameter on stack</td>
<td>prepare parameter on stack</td>
<td>prepare parameter on stack</td>
<td>prepare parameter on stack</td>
<td>prepare parameter on stack</td>
<td>prepare parameter on stack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
<th>Time</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
<th>T8</th>
<th>T9</th>
<th>T10</th>
<th>T11</th>
<th>T12</th>
<th>T13</th>
<th>T14</th>
<th>T15</th>
<th>T16</th>
</tr>
</thead>
<tbody>
<tr>
<td>XP</td>
<td>monitor</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A3</td>
<td>task</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A2</td>
<td>program</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ST3A</td>
<td>call</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ST3A</td>
<td>return</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Note: The table above is a simplified representation of a computer system's event timeline with corresponding actions for each event.*
<table>
<thead>
<tr>
<th>Time</th>
<th>HARDWARE XP</th>
<th>R1 Stack</th>
<th>R2 Stack</th>
<th>R3 Stack</th>
<th>Monitor Stack</th>
<th>Task XP Area</th>
<th>Monitor XP Area</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₀</td>
<td>B # n₁</td>
<td>B H₁₁</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>m ≡ tr₁</td>
<td>m ≡ ip</td>
<td>Sel runs ring B job monitor</td>
</tr>
<tr>
<td>T₁</td>
<td>B # n₂</td>
<td>SFSA₁</td>
<td>B H₁₂</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td>CLP$ ATTACH_command</td>
</tr>
<tr>
<td>T₂</td>
<td>B # n₃</td>
<td>SFSA₂</td>
<td>B H₁₃</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td>PFP$ ATTACH₁</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SFSA₃</td>
<td></td>
<td>3 + tr₁</td>
<td>2 + tr₁</td>
<td></td>
<td></td>
<td>PFP$ R₁ ATTACH</td>
</tr>
<tr>
<td>T₃</td>
<td>2 n₁</td>
<td>SFSA₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PMP$ delay 1DD</td>
</tr>
<tr>
<td></td>
<td>2 n₂</td>
<td>SFSA</td>
<td></td>
<td></td>
<td>2 + tr₁</td>
<td></td>
<td></td>
<td>each → inter</td>
</tr>
<tr>
<td>T₄</td>
<td>m n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₅</td>
<td>2 n₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₆</td>
<td>2 n₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>m ≡ ip</td>
<td></td>
</tr>
<tr>
<td>T₇</td>
<td>2 n₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>each from monitor</td>
</tr>
<tr>
<td>T₈</td>
<td>2 n₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₉</td>
<td>2 n₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₁₀</td>
<td>B n₃</td>
<td>SFSA₂</td>
<td>B H₁₃</td>
<td></td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₁₁</td>
<td>B n₂</td>
<td>SFSA₁</td>
<td>B H₁₂</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₁₂</td>
<td>B n₁</td>
<td>B H₁₁</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T₁₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DAY 3 ASSIGNMENTS

1. Read
   Ch. 14 (Memory Link Interface) in Internal Interface.
   Ch. 10 (Key point Usage) in NOS/VE Procedures & Conventions.
   Ch. 11 (Message Handling) in NOS/VE Procedures & Conventions.

2. Describe SCL Interpreter's block stack entries for the following command sequence:

   LOGIN
   GET_FILE CSOURCE A6
   CYBIL CSOURCE
   LGO
   REPLACE_FILE report A6
APPENDIX C

DUMP
DUMP QUESTIONS

1. Orientation
   a. Where did the processor stop?
      (NOS, NOSVE monitor state, NOSVE Job state)
      \[
      P = 1006.0081, 1538 \rightarrow \text{Ring 1}
      \]
      MPS begins in NOSVE monitor state.

   b. Was there a hardware error?
      \[
      SS, P^2 = 2x \quad SS.IOY = 10
      \]

   c. Three exchange packages are dumped. What state is each? What program is 'represented' by each? What real address are they at?
      page 2  MPS at 252000
      page 4  JPS at 653260
      page 1  XJ in registers.

   d. Why was monitor entered? (Look at MCR)
      MCR 0020 Job Monitor

   e. How many active stack frames are there in monitor's stack in the current job stack?
      \[
      T1 100F 0000 000Z \\
      T2 7010 0000 0002 \\
      T3 3011 0000 000F \\
      TB 8058 0000 1F02
      \]

   f. Mark the end of the stack in the dump of the stack segments. Use the TCS registers from the current job exchange package.
      [Note: not present]
2. Call stack of current job.
   a. If your load map matched the dump (which it doesn't) which procedure exchanged to monitor? Trace the previous stack frame (PSA) address back 3 frames.

   b. What module was the first to run in ring B? the last?

   c. What segment is the binding segment for rings 1, 2, 3, B?

   d. Find AMMSRETURN in program interface and the loadmap. Use the dump of stack frame 7 to determine what file is being returned.
3. Job fixed for current job.
   a. Find jnv$sjmr$xcb and jnv$sjcb in the loadmap. What is the value of these variables?

   b. What is the system generated name for the job?
      
      
      AAKT

   c. What does the P-address field in the exchange package in the XCB contain?
      
      301D      AB276

   d. The segment description table (SDT) address starts at 2ED (in the SCB). How many entries appear to be in the SDT? (They are 1 word long and end at AEE.
      
      \[ 17 \times 4 = 68 = 1048 = 1000100 \]
DISPLAY EXCHANGE PACKAGE

LABEL = EXCHANGE_PACKAGE_AT_JPS

A REGISTERS

AO = 100F 0000 0308
A3 = 100F 0000 0260
A6 = 100F 0000 0367
A9 = 100F 0000 0240
AC = 100F 0000 0240
AF = 2042 0000 2240

X REGISTERS

X0 = 2101 100D 0002 0000
X3 = 0000 FFFF 0000 0000
X6 = 0000 C000 0000 AFEC
X9 = 0000 C000 0000 FFFA
XC = 0000 C000 0000 0000
XF = 0000 C000 0000 1001

TOS REGISTERS

T1 = 100F 0000 0308
T4 = 0000 0000 0000
T7 = 0000 0000 0000
TA = 0000 0000 0000
TD = 0000 0000 0000
T2 = 2010 0000 05C0
T5 = 0000 0000 0000
TB = 0000 0000 0000
TD = 0000 0000 0000
TE = 0000 0000 0000

LABEL = EXCHANGE_PACKAGE_AT_JPS

LABEL = EXCHANGE_PACKAGE_AT_JPS

LABEL = EXCHANGE_PACKAGE_AT_JPS

LABEL = EXCHANGE_PACKAGE_AT_JPS