NCR CONTROL DATA CORPORATION
ADVANCED SYSTEMS LABORATORY

CHAPTER 7

STORAGE MANAGEMENT

Doc. No. ASL00282

Rev. 04

Copy No. 87

NCR/CDC PRIVATE

REV 06/23/75

TABLE OF CONTENTS

1.0 INTRODUCTION 1-1
1.1 BASIC CONCEPTS AND DEFINITION OF TERMS 1-1
1.2 STORAGE SYSTEM'S FUNCTIONS 1-7
1.2.1 SEGMENT MANAGEMENT FUNCTIONS 1-9
1.2.2 MEMORY MANAGEMENT FUNCTIONS 1-10
1.3 GENERAL STRUCTURE 1-11

2.0 PAGED MEMORY AND SCHEDULING IMPLICATIONS 2-1
2.1 PAGING ALGORITHMS 2-1
2.2 PAGING ALGORITHM'S EFFECTIVENESS 2-4
2.3 IPLOS' PAGING CONSIDERATIONS 2-6
2.4 IPLOS' MEMORY MANAGEMENT ENVIRONMENT 2-7
2.4.1 JOB WORKING SET SWAP 2-7
2.4.2 JOB SWAP 2-8
2.5 PAGE CONTROL FUNCTIONS 2-9
2.5.1 SYSTEM MONITOR'S RESPONSIBILITIES 2-9
2.5.2 PAGE QUEUES 2-10
2.5.3 USED BIT SCAN ROUTINE (SCAN) 2-12
2.5.4 PAGING ALGORITHM 2-13
2.5.5 SHARED PAGE MANAGEMENT 2-14
2.5.6 PAGING STRATEGY CONSIDERATIONS 2-15

3.0 DESIGN CONSIDERATIONS 3-1
3.1 STORAGE HIERARCHY 3-1
3.2 SHARING IMPLICATIONS 3-3
3.3 RECOVERY AND RESTART 3-5

4.0 STRUCTURE 4-1

5.0 JOB SCHEDULER AND DATA MANAGEMENT INTERFACE 5-1

6.0 USER INTERFACE AND FACILITIES 6-1
6.1 SEGMENT CONTROL REQUESTS 6-3
6.1.1 SEGMENT CONTROL AND ACCESS RIGHT CONVENTIONS 6-4
6.1.1.1 Segment Control Right Conventions 6-4
6.1.1.2 Access Control Conventions 6-6
6.1.2 REQUEST DESCRIPTIONS 6-9
6.1.2.1 Segment Initiation/Termination Requests 6-9
6.1.2.1.1 SC#INITIATE_SEGMENT 6-10
6.1.2.1.2 SC#TERMINATE_SEGMENT 6-19
6.1.2.1.3 SC#MAP_IN 6-23
6.1.2.1.4 SC#MAP_OUT 6-25
6.1.2.2 Segment Descriptor Table Manipulation Requests . . 6-27
6.1.2.2.1 SC#EXPAND_SDT 6-27
6.1.2.2.2 SC#CONTRACT_SDT 6-28
6.1.2.2.3 SC#RESERVE_SDT_ENTRY 6-29
6.1.2.2.4 SC#RETURN_SDT_ENTRY 6-31
6.1.2.2.5 SC#CHANGE_ACCESS 6-32
6.1.2.3 Segment Attribute Manipulation Requests 6-33

6.1.2.3.1 SC#SET_MAX_LENGTH 6-33
6.1.2.3.2 SC#EXPAND_SEGMENT 6-34
6.1.2.3.3 SC#CONTRACT_SEGMENT 6-35
6.1.2.3.4 SC#TRUNCATE_SEGMENT 6-36
6.1.2.3.5 SC#RELEASE_PVA 6-37
6.1.2.3.6 SC#STATUS_SEGMENT 6-38
6.1.2.4 Segment Sharing Requests 6-39
6.1.2.4.1 SC#MAKE_GLOBAL 6-39
6.1.2.4.2 SC#MAKE_LOCAL 6-40
6.1.2.4.3 SC#CREATE_TRANSACTION_LIST 6-41
6.1.2.4.4 SC#REMOVE_TRANSACTION_LIST 6-41
6.1.2.4.5 SC#REGISTER_TRANSACTION_SET 6-41
6.1.2.4.6 SC#START_TRANSACTION 6-41
6.1.2.4.7 SC#EXIT_TRANSACTION_SET 6-41
6.1.2.4.8 SC#LOCK_PVA_RANGE 6-41
6.1.3 REQUEST STATUS INFORMATION 6-42
6.1.4 REQUEST PROCESSING SEQUENCES 6-42
6.2 PAGE CONTROL REQUESTS 6-43
6.2.1 REQUEST DESCRIPTIONS 6-43
6.2.1.1 Page Usage Prediction Requests 6-44
6.2.1.1.1 MC#ADVISE_IN 6-44
6.2.1.1.2 MC# ADVISE_OUT 6-46
6.2.1.1.3 MC#CLEAR_PVA 6-49
6.2.1.1.4 MC#SET_USAGE_LEVEL 6-51
6.2.1.2 Memory Management Requests 6-53
6.2.1.2.1 MC#LOCK_PVA 6-53
6.2.1.2.2 MC#UNLOCK_PVA 6-55
6.2.1.2.3 MC#FIX_MEMORY 6-57
6.2.1.2.4 MC#RELEASE_MEMORY 6-60
6.2.1.3 Status Requests 6-61
6.2.1.3.1 MC#STATUS_WS 6-61
6.2.1.3.2 MC#STATUS_PAGE 6-61
6.2.1.3.3 MC#STATUS_REQUEST 6-61
6.2.2 REQUEST STATUS INFORMATION 6-62
6.2.3 REQUEST PROCESSING SEQUENCES 6-62
6.3 SYSTEM INTERFACE REQUESTS 6-63

1.0 INTRODUCTION

IPLOS Storage Management provides the mechanism through which a user of the system can create, organize and manage Storage Structures.

This section specifies the IPL Storage System's external interface, describes the design considerations leading to it and gives an overview of the supporting internal structure.

An IPL user's programs and data, partially forming a Job (see Chapters 4 and 9), are contained within an Address Space. Its extent is delinlated by a Segment_Descriptor_Table. Control_Points define the number of potentially parallel executions of code within this Address Space. Data, Program, Job, Storage, etc., Management procedures of Task Services, within the Address_Space, assist the user, and each other, with organizing, controlling and managing a Job, with interfacing to System/Subsystem/User Jobs and with the controlled sharing of Storage Structures within and among separate Address_Spaces.

System/Subsystem Jobs, in conjunction with System Monitor, manage and control User Jobs and schedule the system's resources to accomplish work, and hence, to complete User Jobs.

1.1 BASIC CONCEPTS AND DEFINITION OF TERMS

File - A file is a named entity consisting of an ordered set of elements. The Storage System is only aware of files stored on direct access mass-storage (disk, drum) devices.

File Control Block - A File Control Block is a symbolically addressable table that contains a description of the logical properties of a file.

Physical File Descriptor - The Physical File Descriptor is an internal system table that contains a description of the physical properties of a file.

Physical Space Map - A Physical Space Map represents the storage space allocated to a mass-storage file and is recorded within

its Physical File Descriptor.

Temporary File - A temporary file is one for which a description of its logical and physical characteristics exists only in the File Control Block, the Physical File Descriptor and other internal system tables.

Permanent File - A permanent file is a mass storage file for which the description of its logical and physical characteristics has been recorded in a catalog.

Sector - The smallest addressable block of storage on a mass-storage device. Standard IPL sector sizes are 512, 1K, 2K, 4K and 8K bytes. page_frames are not supported initially by IPL's Central Processors).

Real Address - An address generated by an IPL processor in real addressing mode (see Model Independent Processor-Memory GDS).

Real Memory - Real Memory, or just memory, or real storage, or main memory is memory addressable by an IPL processor via a real address.

Page Frame - A page frame is a block of real memory. Real memory is managed in units of page frames. Standard IPL page frame sizes are 512, 1K, 2K, 4K, 8K and 64K bytes (64K sized page_frames are not supported initially by IPL's Central Processors).

External Storage - Storage Space not addressable by an IPL processor as real memory.

Physical Record - A physical record is the smallest unit of data transfer between memory, and external storage. Sector and page-frame are examples of physical records.

Block - A block is the logical unit of transfer between memory and the mass storage space allocated to a file. Standard IPL block sizes are 2**N multiples of the minimum page frame size.

Input/Output Subsystem - The hardware/firmware/software components of the peripheral configuration.

Device Driver - Device Drivers manage the interface between IPLOS and other members of the Input/Output Subsystem.

1.0 INTRODUCTION
1.1 BASIC CONCEPTS AND DEFINITION OF TERMS

Block Manager - Block Management procedures provide a relatively device independent interface between elements of IPLOS and Device Drivers.

Address Space - c.f., 1.2.

Segment - c.f., 1.2.

Virtual Address - c.f., 1.2.

Virtual Memory Mechanism - VMM is a system feature of IPL which changes a virtual address into a real memory address, and performs dynamic access control, during an execution of an instruction.

Page - A page is a fixed length block of instructions, data or both, that can be transferred between real storage and external storage.

Paging - Paging is the process of transferring pages between real memory and external storage. It is the method of IPLOS' memory management and the standard method for accessing a file's storage space. A user can request that the storage system performs all transfers in groups of pages equal to the file's block size or in multiple pages equal to the system's current page frame size.

Paging Storage - External Storage Space suballocated and managed by the Storage System.

Direct Segment - A mass-storage file made directly addressable as a segment, within an Address Space. The file's block-size must be a power of two multiple of the minimum IPL page size (512 bytes). The segment's virtual address range is associated with the storage space, represented by the file's Physical Space Map, on a continuous byte-to-byte basis. Storage space is allocated dynamically, on a one-to-one basis without gaps, to the corresponding virtual address range, up to and including the highest block accessed, which must be within the currently defined file size. Address of the last block written is recorded by Block Manager within the Physical File Descriptor. All intervening blocks (e.g., blocks between the last written and the currently accessed one) are initialized to Null (i.e., all zeroes) by the Storage System. Since this is a relatively high overhead operation, Direct Segments should be accessed initially and allocated storage space only sequentially. In this case, no extra output transfers are generated. A block accessed for

1.0 INTRODUCTION
1.1 BASIC CONCEPTS AND DEFINITION OF TERMS

the first time is cleared to Null within real memory and the corresponding page(s) eventually written to external storage due to normal paging operations. Direct Segments can be associated with Permanent or Temporary Files. Paging is performed directly between real memory and the associated file's storage-space.

Mapping - Mapping is a technique used to associate virtual address and real address. Virtual Memory Mechanism is the system feature (software and hardware) which accomplishes mapping.

Fetch Policy - The Fetch Policy determines when information is to be loaded into real memory from external storage.

Replacement Policy - The Replacement Policy determines which information is removed from real memory to external storage.

Placement Policy - Placement Policy determines which unallocated region of memory will contain the incoming information.

Page Table - Page Table indicates whether a page is in real storage and correlates virtual with real storage addresses. There is one Page Table in an IPL System.

Thrashing - Thrashing is a phenomenon which occurs in a multiprogramming system when memory is overcommitted.

Fragmentation - Fragmentation occurs when a system suffers from a loss of usable storage.

Pool File - Paging Storage is specified to the Storage System as one or more Pool Files. The Storage System suballocates the external storage space, represented by a Pool File's Physical Space Map. A Storage System initiated paging operation is performed as per a Pool File's Physical File_Descriptor. Therefore standard Block Management and Device Driver interface conventions apply. A Pool File can be temporary or permanent, shared globally, or local to an Address Space.

Temporary Segment - A Temporary, or Transient, Segment cannot survive beyond the creating Address Space's existence. Several Temporary Segments can be associated with a file (e.g., a Pool_File). Storage System maintains the segment-to-file storage space maps within its internal tables. Storage space is allocated dynamically, on a block-by-block basis, whenever the corresponding virtual memory is accessed for the first time. Each new block is

recorded as a separate entry within the Segment Control Table. Temporary Segment's storage space can contain 'gaps' and thus it is not necessarily allocated continuously up to the highest block accessed. Temporary Segments provide a convenient, low overhead mechanism for allocating and managing storage for temporary data structures, i.e.,

- o Stacks
- o Working Storage
- o Heaps
- o Binding Segment
- o Object Module Code Segments
- o Local LNS Segments
- o Etc.

A Temporary Segment can not be saved and made permanent, like a Temporary File, since no File Control Block is defined which contains the description of its logical attributes.

Physical Tables Heap - PTH is a one-per-system segment which contains internal system tables, used and managed by the File and Storage System. It has Read/Write access in the System Job's Address Space and Read only access in all other Address Spaces.

Transient Segment - See Temporary Segment.

Segment Control Table - A SCT is a table in the Physical Tables Heap that contains the description of the physical properties, or attributes, associated with the segment.

Segment Descriptor Table - Software name for the Process Segment Table. It defines the Address Space of a job and contains its segment descriptors. (See Model Independent Processor-Memory GDS).

Indirect Segment - A Direct Segment whose modified pages/blocks are held within Paging Storage. An explicit SC#Map_Out request (c.f., 6.1.2.1.4) must be made to the Storage System before any data is transferred to the associated file's storage space. Indirect Segments enable a user to exercise precise control over file updating, even while it is being used/modified in parallel by several tasks/jobs. However, more table space is required by Indirect Segments than Temporary or Direct ones. SC#Map_Out requests might also generate additional I/O transfers, over and beyond those required by normal paging operations. For these reasons, they should only be used when the increase in system

resiliency warrants the extra overhead incurred.

Buffer Segment - A Temporary Segment whose associated page-frames, in real memory, are used by Block Management procedures to transfer data to/from;

- o Non standard files
- o Archival devices (magnetic tape like)
- o Remote devices (via communication links)
- o Unit-record devices,
- o Mass-storage devices not conforming to IPL standards,
- o etc.

Page of a Buffer Segment is locked into real memory while any Block Manager initiated transfers are in progress, and/or outstanding, for the page-frame occupied by that page. At all other times the page's memory/paging-storage residency is controlled by the standard Fetch/Replacement Policies. Page-frames occupied by a Buffer Segment's pages are not allocated contiguously in real memory. The Input/Output Subsystem provided scatter-read/gather-write capability is utilized to maintain uninterrupted data transfers across physical records.

Storage Structure - All identifiable entities known and controlled by the Storage System are called Storage Structures. Pages, Segments, Segment_Control_Tables, Physical_File_Descriptors, Segment_Descriptors and Segment_Descriptor_Tables are examples of Storage Structures.

Working Set - WS is the set of pages currently being actively used by cooperating Tasks within one Address_Space.

Validation Level - Validation Level is the ring number of the Argument_Pointer_Register (A6) on entry into a request processor of the Storage System. (See Model Independent Processor-Memory GDS, 2.6.1.2).

Memory Region - IPL processors have two memory ports and thus might be connected to a region of an installation's memory which is not accessible by all the processors of the system. System and subsystem jobs can request specific assignment of segment pages to page frames of a Memory Region.

1.2 STORAGE SYSTEM'S FUNCTIONS

The primary function of the IPL Storage System is to implement Job Address Spaces.

The Address Space of a real processor in an IPL system is a Segmented Address Space. A segment is an entity defined by:

- A Name which uniquely identifies the segment.
- A Descriptor which describes the properties or attributes associated with the segment.
- A Body which is an array of consecutive elements. The Body of a segment is an ordered set of elements, called bytes (or cells in SWL terminology), each of which is identified within the segment body by an integer i, its index. The number of elements in the body is called the length of the segment.

A Job's Address Space, or virtual memory, is an ordered set of as many as 2**12 segments each consisting of as many as 2**31 bytes.

The IPL hardware interpretable virtual address is a pair of integers consisting of a segment number and a location within that segment. In order for a program to execute a reference to data, it must have an address of this form. Hence, it is the Storage_System's responsibility to map files, which are stored on direct access Mass Storage devices (disks, drums, etc.) into/out-of the hardware Address Space so that the Job owned tasks can address them directly. In addition to transforming a symbolic reference into a machine address, the Storage System must be concerned with moving data from external storage into memory in order for it to be directly referenceable.

In most contemporary systems on-line information, stored on mass storage devices, is generally accessed as copies of files thru buffers locked into real memory. In contrast, all on-line information in an IPL system can be directly referenced via the segmented virtual memory mechanism. All on-line information can be now controlled and managed by the Storage System. As far as a user is concerned, data transfers to/from off-line devices are into/out-of the System's Storage Space and not his own local memory. Residency of a user's code and data, in real or external storage, or between levels of the storage hierarchy (cache -> Bipolar Memory -> MOS Memory -> Magnetic Bubble Memory -> Fixed

Head Disks -> Moving Head Disks -> Tape Library -> Off-line Devices...), is goverened by usage and scheduling requirements on a global level. The globally managed Storage Space has to be utilized in a uniform manner by both the system and its users. The distinction between traditional mass-storage files, managed by the user himself, and directly addressable memory has now almost disappeared.

Non on-line archival (magnetic tape like!), unit-record, remote (via communication links) and mass storage devices not conforming to IPL standards are addressed via Buffer Segments. Buffer management software within user Address_Spaces will control the allocation and useage of buffers in Buffer Segments, however, physical memory is only locked down during the duration of the explictly initiated transfers. At all other times, the Storage System controls the buffers memory/mass-storage residency.

As far as possible, logical control and management of data is performed within user Address Spaces. Physical data-container and data-flow functions of the operating system are processed outside user Address Spaces, to maintain a well controlled fire-wall between misbehaving elements of the software/hardware complex. Another important reason for the separation between logical/physical elements of the operating system is the global scheduling environment. Since resource control has to be performed globally, effective system utilization can only be guaranteed by separating out and exactly defining necessary control information.

The Operating System functions listed below are major users of, or used extensively by, the Storage System. Internal system Interface conventions, data and control-flow are also documented within the following subsections.

- Interpretive access to data for direct, indirect or buffer segments, is controlled by File Access Procedures.
- Data transfer between real memory and peripheral devices is managed by Block Management and Device Drivers in a System Job.
- Device or Volume_Set allocation and all resource/job scheduling is performed by Job Management .
- Volume (tape, disk) mounting/demounting and label processing is the responsibility of Mount Control within a System Job. It is functioned by requests via signals from

other jobs.

Storage System consists of two major divisions which are responsible for Segment Management and Memory Management, respectively. The major function of each division is described below.

1.2.1 SEGMENT MANAGEMENT FUNCTIONS

Segment Control is responsible for the following storage structures:

- o Direct, Indirect and Temporary Segments
- o Buffer Segments, for buffering files not accessed via the virtual memory mechanism of IPL.

Access to segments is controlled by hardware. Segment Control checks all explicit requests directed at Buffer Segments for:

- o Access rights
- o Buffer residency

and assists Block Management by performing;

- o Retrieval from external-storage pages of Buffer-Segments when a Virtual-Buffer is not present in memory
- o Conversion of Process Virtual into System Virtual and/or Real Memory Address
- o Locking Pages for duration of peripheral transfers
- o Keeping account of outstanding data transfers, into/out-off Address_Space local Buffer Segments, on a per Control_Point basis.

Segment Control Functions: (in addition to Block Management assistance)

- o Assist the File System in the Creation, Deletion, Labelling and Cataloging of Segments/Files held on mass-storage and archival devices.

- o Manage the allocation of space to segments on external storage (Space Control).
- o Organize page Working-Set swapping and move Job-Private Storage and System Tables to/from external storage (Swap Control).
- o Organize the movement of System-Global Storage_Tables between memory and external storage according to usage and scheduling considerations (GST Control).
- o Assist System Monitor with the creation and deactivation of Address_Spaces (Address_Space Control).
- o Organize the migration of storage structures between levels of the storage hierarchy (Migration Control).
- o Support the shared usage of storage structures (Share Control).
- o Perform primary level error recovery, in case of memory parity errors, and assist in memory reconfiguration (Recovery Control).
- o Maintain Storage System restart information (Recovery Control).
- o Provide a set of procedures for the management of Storage System Tables (SST Control).

1.2.2 MEMORY MANAGEMENT FUNCTIONS

The unit of memory management is a page_frame. Page-frame size is set at System Deadstart and can vary between a minimum of 512 bytes and maximum of 64k bytes.

Page Control is responsible for the management of real memory according to usage and scheduling requirements.

Page Control Functions:

- o Maintain a record of the status of each page in memory and process any requests involving the allocation, locking and fixing in real storage, release or change in status of memory page frames.

- Maintain jobs' page Working-Sets (WS) for Job Scheduler and Swap Control.
- Control memory occupancy based on usage and scheduling requirements.
- Process all requests for movement of pages from real memory to external storage and vice-versa in segment dependent block size (block size must be 2**N multiple of minimum page frame size).
- Perform second level error processing, in conjunction with Block Management, in case of errors due to paging operations.
- Perform second level error processing in case of memory parity errors.

1.3 GENERAL STRUCTURE

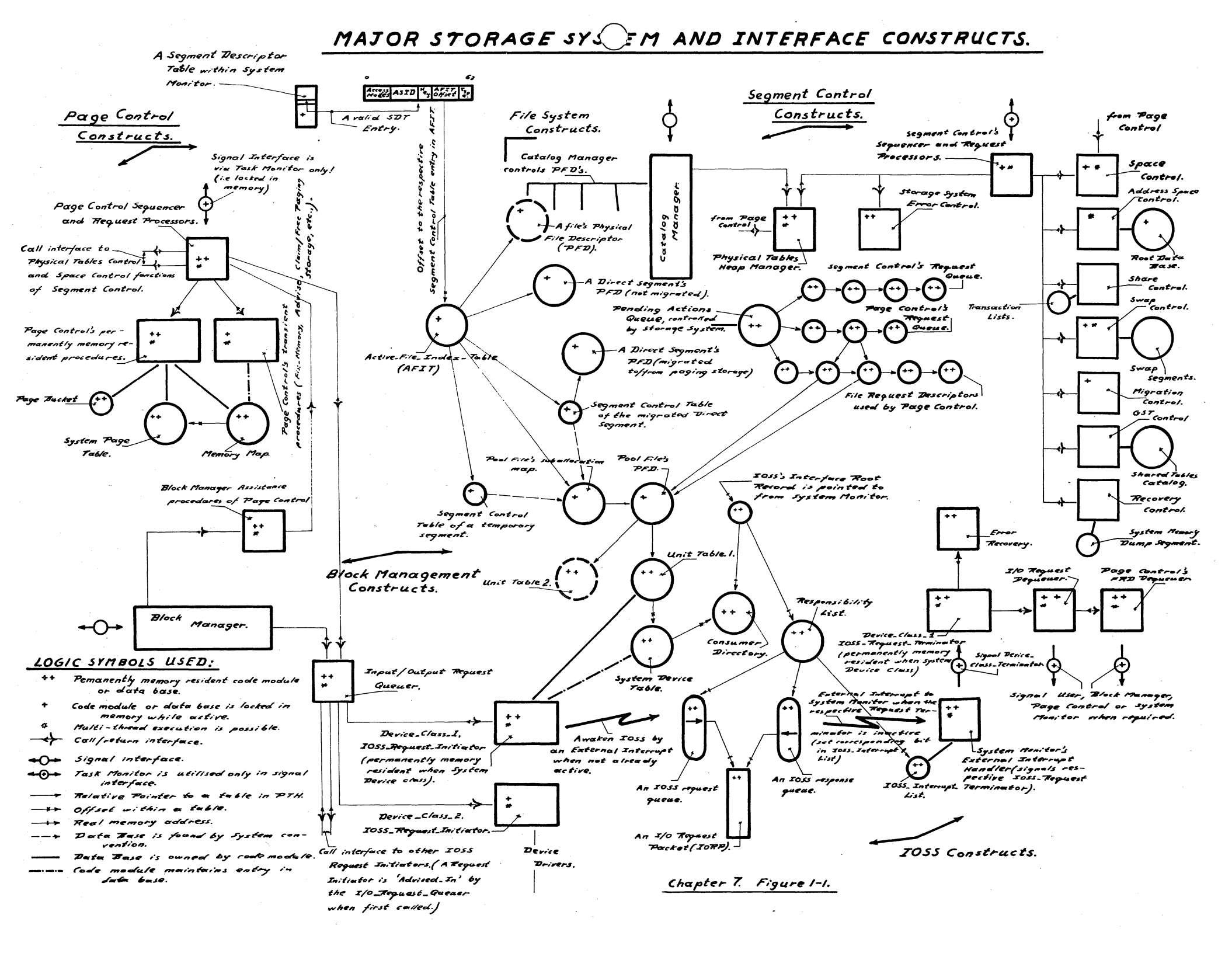
Page Control and most of Segment Control function as procedures within a System Job. (See Chapter 9, Figure 1.2-1). Most of Page Control procedures are permanently memory resident.

User and Data Management interface functions of Page and Segment Control are performed by request processors within Task Services of the requesting Job.

Standard means of communication between Storage Management procedures, within a System Job, and other Jobs is via signals. Task Services level request processors are provided for creating and routing signals to the System Job, therefore, request processor interfaces can conform to SWL calling convensions.

Structure Overview

Figure 1-1


MAJOR STORAGE SYSTEM AND INTERFACE CONSTRUCTS.
A Segment Descriptor Table within System Monitor.
A valid SDT Entry.
Page Control Constructs.
Signal Interface is via Task Monitor only! (i.e locked in memory)
Page Control Sequencer and Request Processors.
Call interface to Physical Tables Control and Space Control functions of Segment Control.
Page Control's permanently memory resident procedures.
Page Control's transient procedures (Pic-Memory Address Claim/Free Paging Storage, etc.).
Page Packet
System Page Table.
Memory Map.
Block Manager Assistance procedures of Page Control
Block Manager.
Block Management Constructs.
Offset to the respective Segment Control Table entry in AFIT.
File System Constructs.
Catalog Manager controls PFD's.
A file's Physical File Descriptor (PFD).
Catalog Manager
A Direct Segment's PFD (not migrated).
Active_File_Index_Table (AFIT)
A Direct Segment's PFD (migrated to/from paging storage)
Segment Control Table of the migrated Direct Segment.
Pool File's sub-allocation map.
Pool File's PFD.
Segment Control Table of a temporary segment.
Unit Table 1.
Unit Table 2.
Segment Control Constructs.
Segment Control's Sequencer and Request Processors.
from Page Control
Storage System Error Control.
Physical Tables Heap Manager.
Segment Control's Request Queue.
Pending Actions Queue, controlled by storage system.
Page Control's Request Queue.
Transaction Lists.
File Request Descriptors used by Page Control.
Space Control.
Address Space Control.
Root Data Base.
Share Control.
Swap Control.
Swap Segments.
Migration Control.
GST Control
Shared Tables Catalog.
Recovery Control.
System Memory Dump Segment.
IOSS's Interface Root Record is pointed to from System Monitor.
Error Recovery.
I/O Request Dequeuer.
Page Control's PRD Dequeuer.
Responsibility List.
Consumer Directory.
System Device Table.
Device_Class_1 IOSS_Request_Terminator (permanently memory resident when system Device Class)
Signal Device_Class_Terminator.
Signal User, Block Manager, Page Control or System Monitor when required.
Input/Output Request Queuer.
Device_Class_1, IOSS_Request_Initiator (permanently memory resident when System Device class).
Awaken IOSS by an External Interrupt when not already active.
External Interrupt to System Monitor when the respective Request Terminator is inactive (set corresponding bit in IOSS_Interrupt List)
System Monitor's External Interrupt Handler (signals respective IOSS_Request Terminator).
IOSS_Interrupt List.
An IOSS request queue.
An IOSS response queue.
An I/O Request Packet (IORP).
Device_Class_2. IOSS_Request_Initiator.
Device Drivers.
Call interface to other IOSS Request Initiators. (A Request Initiator is 'Advised_In' by the I/O_Request_Queuer when first called.)
IOSS Constructs.
LOGIC SYMBOLS USED:
++ Pemanently memory resident code module or data base.
+ Code module or data base is locked in memory while active.
Multi-thread execution is possible.
Call/return interface.
Signal interface.
Task Monitor is utilised only in signal interface.
Relative Pointer to a table in PTH.
Offset within a table.
Real memory address.
Data Base is found by System convention.
Data Base is owned by code module.
Code module maintains entry in data base.
Chapter 7. Figure 1-1.

2.0 PAGED MEMORY AND SCHEDULING IMPLICATIONS

Paging is a memory management scheme that organizes memory into fixed size blocks called page frames and organizes the address space into fixed size blocks called pages. Any page of an address space can potentially reside in any page frame in memory.

Paged virtual memory is a feature of IPL. Virtual memory, or storage, consists of at least two levels; to be accessible to the processors, information must reside in the highest (fastest), or real storage/memory.

IPLOS will use a variant of what is known as demand paging in which a page/block is brought into memory only when needed, a policy which under certain circumstances (seldom satisfied in real systems) is optimal. The operating system's scheduling/memory management routines accept advice requests from users on intended future virtual memory usage and block-swap a job's pages on termination or start of a major time slice.

The page placement policy, which is how to select the page frame in which to place an incoming page, is simple under demand paging. If there is an empty page frame, use it; if not, remove some page (according to the replacement policy) from memory, and use the page frame thus made available. Every time an attempt is made to access a page which is not in memory, a page interrupt is said to occur, and this page is brought into memory. A page interrupt is costly in terms of lost processing time, since the processor must execute the page replacement and page placement policy algorithms, and then may often have to wait (if there is no other task to run) while that page is fetched. It is desirable therefore, and the objective of a paging algorithm, to minimize the number of page interrupts. The paging algorithm under a demand paging policy is equivalent to the page replacement policy.

2.1 PAGING ALGORITHMS

Most paging algorithms, with respect to the amount of memory space per task/job they assume, fall into two classes:

1) Assume that the amount of space a program has is fixed;

   - Random (RAND)
   - First_In_first_out (FIFO)
   - Least_recently_used (LRU)
   - Least_Frequently_Used (LFU)
   - MIN

2) The remaining algorithms all allow the number of page frames used by a given program in a multiprogramming environment to vary;

   - Multics
   - CP-67
   - Page Fault Frequency (PFF)
   - Working-Set (WS)
   - Damped-Working-Set (VMOS - Univac)
   - IBM OS/VS2-1
   - Burroughs' B6700 (not frames but space varies)

These algorithms also differ substantially in their effectiveness and in their difficulty of implementation, for example, FIFO and RAND are easy to implement but inefficient, while LRU and Working Set, which are relatively efficient but costly to implement.

Generally, paging supervisors maintain a pool of free page frames. Whenever the number of free page frames falls below a predetermined value, the paging supervisor replenishes the free page queue by taking pages from a program.

Among the above algorithms the more important are the following:

LRU (Least-Recently-Used) - Remove the page from memory that has not been referenced for the longest amount of time.

Multics - This algorithm uses a pointer that rotates through the page frames in memory. With every frame is associated a use bit, which is turned on when the page frame is referenced. When a page frame is needed, the page frame pointed to by the pointer is examined. If the use bit is on, the bit is turned off, the pointer is advanced and the page frame now pointed to is examined. If it is off, the page in that page frame is removed and the new page is placed in there. Multics' algorithm can be viewed parametrically such that at one extreme it has a page removal strategy of FIFO and at the other extreme a removal strategy of LRU. The CP-67

operating system uses almost the same algorithm.

IBM OS/VS2-1 With every page frame is associated a used and modified bit. Page frames are selected for reclaiming according to the settings of the used and modified bits. There is an active page queue for each possible configuration of the used and modified bits.

- The page has not been used since the last inspection, and has not been modified.
- The page has not been used since the last inspection, but was modified at some previous time.
- The page has been used since the last inspection, but has not been modified.
- The page has been used since the last inspection, and has been modified.

Each queue is maintained in a first-in, first-out order, thus preserving a record of the comparative longevity for each page. The use and modify bits are set automatically by the CPU. Since the pages are constantly being referenced and changed, the bit settings may not be the same when interrogated as when the page frame was first placed on a particular queue. As the paging routine serially searches the queues, it steals those page frames whose status remained unchanged. Those page frames whose status has changed are moved to the appropriate queue.

To prevent too much time spent in paging, a condition known as thrashing, a task disabling algorithm is periodically implemented. At predetermined intervals, the paging routine compares the number of Read I/O operations generated by paging requirements against an installation dependent high and low paging and reclamation thresholds. If too much paging and reclaiming has been done during the last interval, one of the lower priority active tasks is disabled. If more paging and reclaiming should be allowed during the next interval, a previously disabled task is reactivated.

Damped-Working-Set (DWS) - The DWS algorithm has two parameters. T, the working set parameter, and M, a multiplier less than 1. The working set algorithm is itself a special case of DWS in which M=1. This algorithm functions as follows:

- Any page not referenced within the preceding T program/task execution time is removed from memory.
- At the time of a page interrupt, if the time since the last reference to the least recently used page in memory is greater than M X T (M less than 1), then replace the least recently used page by the new page, otherwise increase the number of allocated page frames by one.

Page-Fault-Frequency (PFF) - If the program's execution time since the last page interrupt by the currently interrupted task/program is less than T, assign an additional page frame to the task from the free page pool. If there are no page frames free, deactivate a task, and free its page frames. If the time since the last page interrupt is greater than T, free all page frames not used by the interrupted task/program since the last page interrupt and assign one of these now free frames (if any) to the new page. If there are no frames freed in this way, obtain a frame in the same manner as if the time since the interrupt was less than T.

2.2 PAGING ALGORITHM'S EFFECTIVENESS

As a general rule, it can be said that those algorithms which allow the space occupied by a program to vary (e.g., PFF, DWS) do better than those operating with a fixed number of page frames allocated (e.g., FIFO, LRU).

When page replacement algorithms were first used for memory management, they were applied globally across the memory. This means, whenever a replacement decision must be made, the replacement algorithm is applied to all pages in memory, irrespective of the task/program to which they belong (e.g., Multics, CP-67, IBM OS-VS2/1). This kind of memory management can lead to severe performance degradation. In fact, every global replacement algorithm may eventually lead to an unduly high page fault frequency since there is no way to guarantee that a task has been assigned enough page frames to run efficiently.

If a program's page references were randomly distributed over all pages according to a uniform distribution, it would not matter what page is chosen for replacement. The page fault frequency would only depend on the number of allocated page frames. However, programs tend to reference pages unequally; they tend to cluster references to certain pages in short time intervals. This property is exhibited to a varying degree by all

practical programs and commonly known as 'locality'.

The Working-Set model of program behavior is based on the assumption that practical programs behave according to the 'locality' principle and takes into account the varying memory requirements. Those pages are kept in memory that have been accessed during the last T references. The performance of the WS algorithm still depends on the correct choice of the working set parameter T. A strict implementation would require a setting of a flag at this point; that is, an indication that the corresponding page frame can be used for a different page of any task. Another problem is to detect the exact time when a page has not been referenced during the last T msec. Hence, it appears to be rather expensive to implement the WS algorithm in its pure form.

Correct choice of the working set parameter requires future knowledge of a program's behavior which is impossible to do in a practical implementation. We require an adaptive replacement algorithm which is largely independent of program behavior and input data, and is simple to implement. Page-Fault-Frequency (PFF) algorithm satisfies the above requirements.

PFF memory management policy uses the page fault frequency (the frequency of those instances at which an executing program requires a page that is not in main memory) as an adaptive parameter to control the decision process of the replacement algorithm. Since the page fault frequency reflects the actual memory requirements of a program at execution time, the PFF algorithm can be applied to arbitrary programs without prior knowledge about program behavior.

There is one difficulty with both the WS and PFF algorithms which up till now has not been mentioned. Occasionally during the execution of a program, a sudden change of locality will occur, e.g., when a large subroutine is called or between different phases of a compiler. The problem is that when such a change of locality happens, a time interval must elapse before the pages belonging to the old locality are completely flushed. In the meantime, a number of pages belonging to the new locality have been brought into memory. This peaking behavior is also a characteristic of the Multics and OS-VS2/1 algorithms which under some circumstances would tend to steal excessive number of pages from other tasks when locality changes. PFF is the worst offender in this regard, since it would be far slower to flush the pages belonging to the old locality than would WS. The extra page-frames used to satisfy the peak demand must either be taken from a pool of free pages, or are frames belonging to another

task. In the latter case, it may even be necessary to deactivate a task/job to satisfy the peak demand. The DWS algorithm corrects the peaking behavior of WS.

2.3 IPLOS' PAGING CONSIDERATIONS

IPLOS' Page Management is based on PFF for the following reasons:

- PFF is an efficient algorithm in execution time, control table size and minimizes memory space-time products.
- Memory is allocated according to the dynamically changing memory requirements of a program.
- No prior knowledge of program behavior is required and it can be applied to programs of different types and sizes.
- Using a PFF algorithm in a multiprogramming environment, the system scheduler has control over the efficiency and memory requirements of all tasks/jobs.

To satisfy IPLOS' requirements extensions are made to PFF in the following areas:

- Compensate for the peaking behavior during changes of locality by adjusting ageing rate according to page-fault-frequency.
- PFF is a program/task local page replacement algorithm. IPLOS is designed to permit sharing of code and data among tasks of a job and between separate jobs. The 'global' usage pattern requires that shared pages be removed from memory according to a 'global' policy. Its implementation should be such that the global policy only applies to shared pages without interference with the program/task local management of its private pages.
- PFF only frees pages while a page fault is being serviced. A compute bound program might go without a page interrupt for long time intervals, while some pages could leave its working-set and should be removed from memory.

2.4 IPLOS' MEMORY MANAGEMENT ENVIRONMENT

Memory management is performed within IPLOS' Storage System Hierarchy (for details see 3.0). Pages owned by Active Jobs, and shared segments, are managed according to IPLOS' Paging Algorithm. Memory management, as applied to inactive/deferred Jobs, is pre-empt/resume strategy on a JOB (not Task) basis. Similarly, Page_Control maintains Job Working-Sets based on the assumption that the contained tasks' work is closely related within the same areas of locality of the virtual address space. Not closely related 'work' should be performed by separate jobs. A job can easily submit other jobs, share segments/files/private disk-packs, etc., and exchange messages/signals. A submited job is scheduled independently, according to its own resource requirements.

A running job is delineated by a Segment_Descriptor_Table defined Address Space and entries within various system tables, i.e.,:

o System Monitor entries

- Running_Job_Ordinal_Table_Entry
- Control_Point(s)
- Signal Buffer(s)
- Timing_Signal Selection(s)
- Etc.

o Segment_Control entries

- Physical File Descriptors
- Active Segment Control Tables
- Pool_File Allocation Map(s)
- Etc.

2.4.1 JOB WORKING SET SWAP

Within a job several tasks can be simultaneously executed by the multiprocessing system controlled processors. Periodicly, when a job's tasks used a certain amount of system resources (CPU time, memory space, channel time, etc.) the Job Scheduler instructs Segment_Control, via System Monitor, to inactivate and swap to a Swap-File:

o All private pages in the job's Working-Set.

o All currently used execute and globally read only pages.

(For a detailed description of Working-Set-Swap conditions, see Chapter 9, 2.10.1.1.3).

2.4.2 JOB SWAP

An inactive job might be further 'deferred' for the following reasons:

- o One of its tasks goes into a 'long wait' (i.e., waits for terminal input, new tape/disk to be assigned or queues for the exclusive use of an interlocked file/segment) and all the other tasks are also waiting.
- o Used its share of system resources for the current 'schedule cycle' in the Running Job Mix.
- o Too many running jobs in inactive state (inactive jobs tie down locked memory by their respective entries in system tables).
- o Etc.

On a Defer_Job request, Overseer and Segment Control transfer to the Swap_File an inactive job's owned:

- o Content of entries in System Monitor Tables.
- o Private Physical File Descriptors and Segment Control Tables. Shared Physical File Descriptors and Segment Control Tables are 'aged' from system tables and eventually removed by Segment Control to the one-per-system Shared_Descriptors_Catalog.

Segment_Control requests the allocation of contiguous space on rotating, direct access mass storage so that a swap operation can be streamed with minimal movement of the recording heads.

2.5 PAGE CONTROL FUNCTIONS

Page Control is activated by a signal. The following can cause an activation signal:

- Task services, on an Advice, Lock, etc. request.
- System Monitor, following a Page_Table_Search_Without_Find page interrupt.
- System Monitor, on a Memory Malfunction interrupt.
- Job-Scheduler, to instruct re-evaluation of a job's Working-Set and initiate possible stealing of inactive pages.
- System Overseer, to initiate page stealing from a job which has not had a page interrupt for a certain time interval.

2.5.1 SYSTEM MONITOR'S RESPONSIBILITIES

The functions and parameters supported by System Monitor for Page Control consists of:

- TET - Task_Execution_Time. CPU time used by a task, normalized to P0 time, is maintained within its Control_Point.
- LPIT - Last_Page_Interrupt_Time. On a page interrupt System Monitor calculates the approximate sum of the job's Task_Execution_Times and records it as LPIT in the Running_Job_Ordinal_Table (RJOT).
- PATT - Page_Age_Tick_Time. A parameter set by Job Scheduler in RJOT, determines the aging-rate of pages belonging to a job's Working Set.

On a Page Interrupt Page_Interrupt_Request_Processor is activated within System Monitor. It performs the following actions:

- Sums the Task_Execution_Times (STET). Tasks belonging to the same job could be running in parallel on other processors and only the last recorded TET's can be obtained. (The other processors would have to be interrupted and interrogated by a signal for exact values.)
- Obtains LPIT from RJOT.
- Calculates Age_Interval_Increment (AII):

  AII = (STET-LPIT)/PATT

- Calculates Page_Fault_Frequency (PFF):

  PFF = 1111/(STET-LPIT msec) page interrupt/sec

- Records STET as LPIT in RJOT.
- Sends a signal to Page_Control which contains:
  - Control Point Identifier
  - Page Identifier
  - AII (rounded)
  - PFF

2.5.2 PAGE QUEUES

Page Control maintains several page frame queues according to usage and status. These are the following:

- Job Working Set Queues. Page frames belonging to a job are maintained in approximate LRU order (see Scan below).
- Shared Page Queue. When a task interrupts to a page within a shared segment it is counted as being part of the interrupted jobs working set. After being used for a certain time period it's valid and used bits are cleared, it is taken from the Job_WS and placed into the Shared Page Queue.
- Free Page Queues. When a page is removed from the active queues its page-frame is placed into one of the Free Page Queues. Any page which has not been modified is linked to the Available_Free Queue, any page which has been modified is linked to the Modified_Free Queue. Modified pages must be written to mass storage before the page frame can be used.

Whenever a page is 'stolen' from a job's Working Set or the Shared Page Queue, its valid bit is reset and the page is linked to the appropriate Free Page Queue.

When a free page is required to satisfy a page interrupt, Page_Control always tries to take a page frame first from the Available_Free_Queue. When the number of entries in the Available_Free_Queue drops below some threshold, pages in the Modified_Free_Queue are rewritten to their respective home positions on Mass Storage. Thus, the write operation does not have to be performed while some task is waiting for the page.

Before a page-frame can be reallocated, all processor's Map Buffer entries pointing to the contained page must be cleared. Cache Buffer entries, corresponding to page frames in the Free Queues, also have to be purged if retrieval from the Cache is by Real Memory Address (RMA) and not System Virtual Address (SVA). A processor always checks its Map Buffer before the Cache Buffer is accessed. Since no entry can correspond in the Map to a page not in memory, there is no conflict until an SVA or RMA is reused. An RMA is immediately reused when a page frame is allocated to a new page. Active Segment Identifiers are only reused infrequently, and then all Cache Maps are cleared (See ASID management in 4.0 below).

Page Control time stamps the page when it is placed into either free queues. All processors are periodically interrupted, the respective maps cleared by System Monitor and the time recorded. Page_Control compares the page-frame's time-stamp to the last Buffer_Flush_Time-s (there could be several processors!) and if it is older than the least recent one the page frame can be reused. Otherwise, Page_Control requests System Monitor to have the buffers cleared of entries corresponding to the current Free Queue's pages. On return from System Monitor Page_Control can now assume that all free page frames are available for reuse. Frequency of Buffer_Flush_Time depends on paging_rate/Minimum_Free_Queue_Size ratio, thus, the scheduler can make the CPU versus Idle-memory overhead tradeoff (for details see Replenish_Free_Queue operation in 4.0 below).

Page Table locations corresponding to Free_Queue entries are only cleared on actual reassignment of a page frame, e.g., just before a page is read from mass-storage or when a page is set to all zeroes. Page_Control always looks first in the Page Table when a page interrupt is being serviced and retrieves a still useable entry from the Free_Queues or Shared_Queues.

2.5.3 USED BIT SCAN ROUTINE (SCAN)

In order to determine the recency of use of a job's pages and to take away from it any pages which are not being used, the used bits of all pages which belong to the job and have their valid bit set are examined (at times indicated below). The Scan routine's input parameters are the following:

- o AII (from Page_Interrupt_RP)
- o Age_Interval_Ceiling (AIC)
- o Age_Interval_Floor (AIF)

AIC and AIF are obtained by Page Control from a static table (Job Scheduler can change it according to global scheduling consideration, see 2.5.6 and 5.0 below), called Unused_Age_Interval_Table, which is maintained by Job_Scheduler. It relates PFF to AIC/AIF.

Scan performs the following actions:

- o Use Bit On - The page has been used recently. The used bit is turned off and the Age_Count field set to zero within the corresponding Memory_Map entry.

- o Use Bit Off - Age_Count is incremented by AII. The magnitude of the Age_Count, since the use_bit has been off, determines the specific action:

  d. Age_Count $\geq$ AIC

  The page has been unused for more than AIC time interval; thus, it is no longer considered as belonging to the Job's Working_Set. It is removed from the Working_Set by having its valid_bit turned off and being linked to the Free_Frame_Queue. It's page table entry is only deleted (set to all zeroes) when the page frame is actually reused. If the page is re-accessed before the page_frame is reused, a mass storage read operation is avoided.

  b. AIF $\leq$ Age_Count < AIC

  The page has been unused long enough to be eligible to be taken from the Job's Working Set and its page frame is reused by a new page of the job, if required. During the scan process, the pages are chained into an approximate LRU list, the one with the highest

Age_Count at the head of the list.

c. Age_Count < AIF

The page has not been unused long enough and may not be taken from the job. Once a page is given to a job, it is protected for at least the AIF time interval.

2.5.4 PAGING ALGORITHM

IPLOS' Paging Algorithm is based on PFF with extensions from DWS. System Monitor's Page_Interrupt_RP calculates Page_Fault_Frequency and Age_Interval_Increment. PFF is used by Page_Control to obtain values for Age_Interval_Ceiling and Age_Interval_Floor from the Unused_Age_Interval_Table.

The following sets of actions comprise the algorithm:

1. Compare AIC to the Age_Count of the jobs LRU page:

a. Age_Count ≥ AIC:

- o Take a page from the Free_Queue
- o Initiate Page_Pull Sequence.
- o If no page on Free_Queue, use job's LRU page and clear processor maps accordingly.
- o Place LRU page on Free_Queue when not used.
- o Exit to Step (2).

b. AIF ≤ Age_Count < AIC:

- o Take a page from the Free_Queue.
- o Initiate Page_Pull Sequence.
- o Exit to Step (2).

- If there is no page on the Free_Queue:

- o Take LRU page.
- o Clear Processor Maps.
- o Initiate Page_Pull Sequence.
- o Exit to Step (2).

c. Age_Count < AIF:

- o Take Page from Free_Queues.
- o Initiate Page_Pull Sequence.

- o Exit to Step (2).

- If there is no page on the Free_Queues and AII ≥ 1;

- o Scan jobs Working_Set.
- o Set AII = 0
- o Repeat Step (1).

- When AII is zero and there are no pages on the Free_Queues:

- o Reschedule processing of this page interrupt.
- o Schedule Replenish_Free_Queue Operation.

- If the Free_Queue is successfully refilled, the page interrupt is processed again, otherwise Job Scheduler is notified.

2. If AII ≥ 1 Scan job's Working Set and Exit to Step (3).

3. Change Control-Point_Status when page becomes available.

2.5.5 SHARED PAGE MANAGEMENT

When a task interrupts to a shared page it is counted as being part of the containing job's Working Set. Its active age within the Working Set is maintained by Page Control as a count, the Active_Age_Count. This is updated by the Used_Bit_Scan_Routine whenever it is activated for the respective Address Space. After being used for a certain length of time, the Active_Age_Limit period, its valid and used bits are cleared, it is taken from the job's Working Set and placed into the Shared_Paged_Queue. On a subsequent page interrupt, it is again placed/replaced into the Working Set of the interrupting task's Address Space and thus, an active page is moved from Working_Set to Working_Set of the jobs sharing it. Jobs are charged for the amount of real memory utilized by their Working Sets and, since shared pages are moved around periodically, it is now possible to assign a shared page's usage time equitably.

While a shared page is within an Address Space's Working Set, it is aged by the Scan routine just like a non-shared one, however, the Page Fault Frequency used for retrieving its respective AIC/AIF values from the Unused_Age_Interval_Table is a function of both the containing job's PFF and the System's

Shared_PFF. Shared_PFF is maintained by System Monitor. It is the average PFF of page interrupts to shared pages vs. total CPU time accumulated by the currently active jobs. The 'eviction-rate' of shared pages from real memory is thus a function of the global usage pattern.

Active_Age_Count of shared pages within the Working Set of I/O bound jobs might never reach the Active_Age_Limit. Page Thrashing Monitor in System Monitor's Overseer compensates for this anomaly by activating SCAN whenever the Shared_PFF falls below a Job Scheduler set minimum threshold. SCAN attempts to take Shared pages from the jobs' Working Sets, and place them into the Shared_Page_Queue, until Shared_PFF rises above the prescribed threshold.

2.5.6 PAGING STRATEGY CONSIDERATIONS

IPLOS' Paging Strategy is a combination of PFF and DWS. Unreferenced pages are automatically 'evicted' from memory after the elapse of a time-interval, Age_Interval_Ceiling (AIC). An approximate Age_Count indicates how long a page has not been referenced. AIC is similar to T, the Working-Set Parameter. When AIC is set too high memory becomes under-utilized and the multiprogramming-level limited. A too low value for AIC can result in thrashing, pages to be used soon could be removed from memory and then immediately reloaded. Age_Interval_Floor sets the lower limit to an unreferenced page's right to stay in memory since there is a high probability that it will be referenced again during this interval. When a job needs a page, one with an age greater than AIC is a good candidate for replacement. However, pages with ages between AIC and AIF should only be used when:

- A program is in steady state and new code or data pages are only required occasionally.
- A program is moving to a different locality (e.g., Working-Set) and the old pages should be rapidly flushed from memory.

The 'instantaneous' page-fault-frequency is used by IPLOS Paging Algorithm to differentiate between the above conditions. Values of AIC and AIF are constant for an exact implementation of DWS. PFF does not take into consideration the relative age of unreferenced pages. In IPLOS AIC and AIF are a function of the Page_Fault_Frequency, and chosen in such a way, as to maintain

the PFF at an optimum level.

Optimum PFF is configuration and usage pattern dependent. Maximum paging rate of a system is influenced by:

- Memory size, e.g., multiprogramming level.
- I/O Channel's: Data Rate
  Number
  Memory Access Interference
- Number of simultaneous: Transfers
  Seek-overlaps
- CPU overhead incurred in Page and Block Control.

Differing patterns of use (i.e., transaction, interactive, batch) require optimization for:

- Minimum response time to transactions.
- Job thru' put.
- CPU utilization.

Generally it is possible to trade-off CPU overhead for response time; however, this can only be achieved dynamically within a continuously balanced paging system.

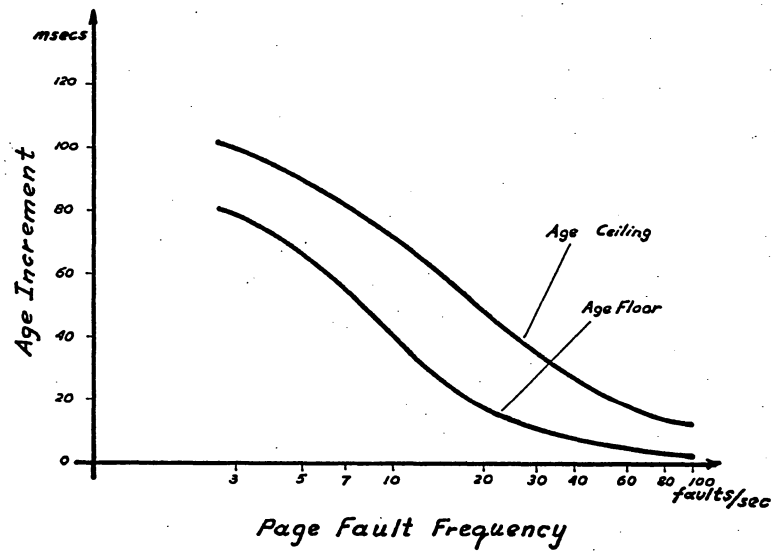
On Figure 2-1 'possible' Age_Ceiling/Floor curves are shown as a function of PFF. An optimum PFF of 20 is assumed. At high paging rates pages are protected for shorter time periods and become candidates for reassignment earlier. Since the algorithm tries to 'steal' pages from the page-faulting job first, non regular 'thrashing' programs are naturally limited in their rate of expansion. System Overseer periodicly examines individual RJOT entries which contain:

- PFCI - Page_Interrupt_Count_Increment. Number of page faults since last examination.
- PIAS - PICI_Accumulation_Start time in job execution time.

From the above parameters System Overseer deducts if the job's paging rate is below the desired minimum and instructs Page Control to activate Scan.

Age_Interval vs. PFF curves should be constructed empirically by observing system behavior during a set of benchmarks. Job Scheduler maintains the Unused_Age_Interval_List

which contains tabulated values of AIF and AIL vs. PFF. PATT, Page_Age_Tick_Time, is also set by Job-Scheduler; thus, it can control how often should a job's Working-Set be aged. Accuracy of WS_Size evaluation should be weighed against the overhead of the more frequent evaluation.



Age_Ceiling/Floor vs. PFF

Figure 2-1

3.0 DESIGN CONSIDERATIONS

The following topics had such a significant influence on the general design of the Storage System that are judged sufficiently important to justify their appearance in this GDS.

3.1 STORAGE HIERARCHY

As computing grows increasingly more data oriented, the speed of data handling, and especially of storage functions, becomes the limiting factor in the overall performance of modern computers. Storage systems typically use several technologies, which are linked together with the objective of effectively utilizing the advantages of each technology (high speed, low cost).

The early memory hierarchy designs attempted to integrate the speed/cost characteristics of electronic and electro-mechanical storage with rather disastrous result. The slow access times of drums, and/or fixed head disks, resulted in a significant degradation of the CPU performance when a large number of references to auxiliary storage were made. To compensate for such disparity in access time, the main memory had to contain nearly complete programs and their working storage. In addition, the cost/performance characteristics of the three level disk/drum/electronic-memory hierarchy has been rather poor due to the high cost, low capacity and slow access time of drum like devices. To date, hierarchy has been used successfully only in the main memory system where the speed/cost characteristics of different electronic, and not mechanical, storage technologies have been integrated (e.g., bipolar cache vs MOS memory).

There are a number of technologies being currently pursued that can lead to storage devices falling between the traditional high cost, high performance main storage and lower cost, low performance electo-mechanical storage. Many of these intermediate storage approaches have been successfully demonstrated and the most promising ones shuld be in full production before the first IPL system is delivered. These new devices are of electronic nature with the resultant cost/performance and reliability benefits. Their development

completely alters the feasability and desirability of automatically managed memory hierarchies.

IPL's Storage System will include the capability of managing and functioning within a three level storage hierarchy (e.g., mass-storage, paging storage and real memory). The design of this feature is being influenced by the following main objectives:

- o A user should be able to control where in the Storage Hierarchy his data is.
- o Elements of the Storage Hierarchy should be managed/controlled similarly to any other mass storage device.
- o Storage Hierarchy's Control Software should be removed/included at System Generation as an option.

The next release of this GDS will include a detailed description of the Paging Storage Management/Control algorithms.

3.2 SHARING IMPLICATIONS

A major design objective of the IPL system is to provide for the sharing of data structures within directly addressable segments. IPL hardware's Virutal Memory Mechanism and the Storage System supported data structures accomplish the sharing of an execute, read-execute or read-only segment in a relatively straight forward manner. The only requirement is to establish a Segment Descriptor within the accessing job's Segment Descriptor Table for the respective segment, manage its pages in memory and transport them to/from mass storage.

However, support for concurrent, in place, updating and simultaneous reading of arbitrary data structures places some additional requirements on the Storage System. These are the following:

1) Provide for the locking of arbitrary data structures for exclusive update.

2) Prevent concurrent update of inter-locked data structures.

3) Permit simultaneous, shared reading of data structures, inter-locked for exclusive update, in their original, non-modified form.

4) Provide queuing logic to permit proper sequencing of conflicting demands.

5) Detect an existing deadlock.

6) Provide for the automatic roll-back of the effect of a set of updates to remove a deadlock or correct a system or user error.

7) Provide the above facilities for controlled, simultaneous access to data structures within a set of segments.

8) Provide the above facilities without excessive overhead and with minimum penalty to those users who do not subscribe to their use.

Requirements (1), (2), (3), (6), and (8) naturally follow from the basic structure of the Storage System. To meet requirements (4), (5), and (7) additional code and table space has to be utilized within the Working Set of those jobs which subscribe to this service.

The relevant requests, their sharing and usage options plus a detailed description of the individual request's internal operation is described in 6.1.2.4 and 6.1.4.

3.3 RECOVERY AND RESTART

Storage Management is responsible for managing the residency of a job's code, working storage and data segment's pages/blocks within IPL's Storage Hierarchy. It performs the respective functions automatically, without its users explicit knowledge, and thus it must be also responsible for guaranteeing the validity of the information entrusted to its care. Internal design of the Storage Management must reflect this requirement, within the cost/performance constraints (e.g., real memory and processing time overheads, additional external storage space used, etc.) imposed on IPLOS.

Storage Management will have the following capabilities in support of the system resiliency requirements:

- o Integrity of a Deferred Job's code, working storage and of data segments local to it's Address Space are guaranteed over System Restarts.

- o In case of a system power outage, it will dump sufficient number of its tables and data pages to provide for:

  - Identifying the afflicted jobs and pages/blocks of shared data segments.

  - Memory resident pages of 'critical' data segments (c.f., 6.1.2.1.1 Usage (Critical) parameter) are saved.

- o A Recovery Subsystem is provided which attempts to salvage the Storage System's tables from a System Dump and assits other elements of IPLOS with their respective recovery/restart responsibilities after a system outage.

4.0 STRUCTURE

4.0 STRUCTURE

This section is being continuously revised and contains implementation details. An up-to-date copy can be obtained by listing the file R7, under User Number MAD, within ASL's SES System.

5.0 JOB SCHEDULER AND DATA MANAGEMENT INTERFACE

To be supplied when the internal design of the Storage System is completed.

6.0 USER INTERFACE AND FACILITIES

This section contains descriptions of the Storage System requests. These are grouped into the following categories:

- Segment Control Requests:
  - Initiate/terminate Segments
  - Manipulate Segment Descriptor Table Entries
  - Control Segment Attributes
  - Control sharing of Segments
- Page Control Requests:
  - Predict memory usage
  - Manage memory
- System Interface Requests:
  - Assist other elements of the Operating System
  - Define tables/parameters to be used by the Storage System procedures

The following conventions are used in the request descriptions:

- Requests follow standard operating system conventions, as specified in Structure Overview section of this GDS
- Requests are specified as SWL macroes
- On return from the Storage_System_Request_Processors, the Request_Status parameter is always immediately available.
  - If the request was accepted, the requestor can determine if it was completed or just initiated
  - If the request was rejected, the requestor can determine the reasons

  Fields in the request status provide information about the conditions of completion or initiation, specify parameter

errors or reasons for rejection. Examples; No available memory, installation threshold has been reached, tables are full, etc.

- Option field specifies variants of a standard request and, in case of error conditions, provide instructions for alternate action.

6.1 SEGMENT CONTROL REQUESTS

Segment Control Requests are used to create, manipulate and control Storage Spaces which are made addressable via IPL's Virtual Memory Mechanism. They are grouped into the following categories:

A. Segment Creation/Termination requests:

- o SC#Initiate_Segment
- o SC#Terminate_Segment
- o SC#Map_In
- o SC#Map_Out

B. Segment Descriptor Table Manipulation requests:

- o SC#Expand_SDT
- o SC#Contract_SDT
- o SC#Reserve_SDT_Entry
- o SC#Return_SDT_Entry
- o SC#Change_Access

C. Segment Attribute manipulation requests:

- o SC#Set_Max_Length
- o SC#Expand_Segment
- o SC#Contract_Segment
- o SC#Truncate_Segment
- o SC#Release_PVA
- o SC#Status_Segment

D. Segment sharing requests:

- o SC#Make_Global
- o SC#Make_Local
- o SC#Create_Transaction_List
- o SC#Remove_Transaction_List
- o SC#Register_Transaction_Set
- o SC#Start_Transaction
- o SC#Exit_Transaction_Set
- o SC#Lock_PVA_Range

6.1.1 SEGMENT CONTROL AND ACCESS RIGHT CONVENTIONS

The following objectives influenced the design of the segment access and ownership control procedures supported by Segment Control:

- o A user's control over hardware features should only be limited by system integrity and privacy considerations.
- o Provide for the controlled sharing of directly addressed segments, within and among separate Address Spaces, without the requirement that only permanently cataloged Direct Segments can be shared globally.
- o Permit the Operating System provided "access monitor" procedures like Program, Job and Data Management to exercise control over their specific areas of responsibility without preventing a user from explicitly controlling his own environment.
- o Use only a minimum of software and system table space to achieve the above objectives.

6.1.1.1 Segment Control Right Conventions

Program and Job Management procedures create active entities like

- o Address Spaces - Jobs
- o Control Points - Tasks, Programs

whose direct access to each other and to elements of the virtual Address Space is controlled by hardware. Segment Descriptor Tables define a job's Address Space, ring/key combinations define task/program accessable segments within this Address Space.

The key/lock mechanism is used by Segment Control to define and identify a set of related segments (e.g., owned and/or controlled/used by the same entity within one Address Space) via their Segment Descriptors. A limited set of ownership and usage control information is maintained within Segment Descriptors, in addition, to the hardware enforced access control data. These Control_Attributes are encoded into the respective Segment Descriptor's Type field (See 4.0 for details). The Control Attributes can be specified by the SC#Initiate_Segment and

SC#Change_Access requests (c.f., 6.1.2.1.1 and 6.1.2.2.5).

A Segment's Control_Attribute can be at one of the following Control_Levels:

- o Owner
- o Pass
- o Copy
- o Use

Owner: Owner Attribute can be given to an entity over a Segment Descriptor (SD) in the following cases:

1. It is the initial Segment Descriptor Table entry of the newly created segment by the entity.

2. It is the File Access Procedure, as defined by the File Control Block of a Direct/Indirect Segment.

3. It is the owner of the Direct/Indirect Segment, as recorded within its File Control Block.

4. Requestor's Validation Level is that of Task Services.

Owner of an SD (e.g., the requestor's key matches that in the SD and it has Owner Attribute recorded within its Type Field) can change its access right fields or initiate a new SD with the following attributes:

- o Lower/higher access rights,
- o In any Address Space,
- o With any Control Attributes

Pass: Pass Attribute, with respect to a Segment Descriptor, can be given to an entity by its owner or another entity with Pass right.

Pass right, with respect to a Segment Descriptor, enables an entity to change its access right fields or initiate a new SD with the following attributes:

- o Lower/higher access rights,
- o In the current Address Space,
- o With Pass, Copy or Use Attributes.

Copy: Copy Attribute, with respect to a Segment Descriptor, enables an entity to change its access right fields or

initiate a new SD with the following attributes:

- o Lower or higher access rights,
- o With the same key/lock,
- o With Copy or Use Attribute.

Use: Use Attribute, with respect to a Segment Descriptor, enables an entity to change its access right fields, or initiate a new SD with the following attributes:

- o Lower or same access rights,
- o With the same key/lock.

6.1.1.2 Access Control Conventions

A Segment Descriptor's (for details see Model Independent IPL Processor/Memory GDS) Access_Control Fields are initialized/modified by Segment Control according to the set of conventions detailed below.

A. The following restrictions apply when a requestor's Validation Level is that of Task Services or greater (e.g., the call is if from a higher, less capable ring);

- o A Global_Privileged segment can only be created from within a System Job's Task Services.

- o A segment can only be initiated with an Execute Attribute when its write ring (R1) is greater or equal to that of the requestor's Validation Level. This restriction prevents a user from directly creating an executable segment which can execute within more privileged rings.

B. The following additional restrictions apply when a requestor's Validation Level is greater than that of Task Services;

1. General Restrictions

   - o No Segment with Execute Attribute of Local_Privileged can be initiated.

   - o No Segment with the Read Attribute of Binding Section can be initiated. This restriction prevents a user from directly creating inter segment linkages which might by-pass gated entry points into more

privileged rings.

- o A segment, initiated with execute attribute, inherits its initiator's Global/Local Flags and key. This restriction permits Segment Control to maintain the intra Address Space isolation firewalls provided by the Key/Lock feature.

- o A requestor's key must be the same as that within the Segment_Descriptor specified by a Storage System request when one of the following actions is requested:

  - Initiate a new SD per specified one,
  - Change its access control or ownership attributes,
  - Use it in any Page Control requests.

- o When an existing global segment is made accessible by a new Segment Descriptor and the requesting entity has no Owner rights than the new access attributes must be definable by the following transformation:

Read/Write -> Execute/Read -> Execute

The above rule prevents damaging a globally shared Read/Execute only segment

2. When an existing file is made directly accessable as a Direct Segment the following fields of the File_Control_Block (See section IV, Appendix A) might restrict access:

- o fapring - The initiated segment's write ring (R1) and read ring (R2) must be the same as the fapring when a File_Access_Procedure is defind.

- o acclevel - must permit Segment Level access.

- o Write ring - defines the highest write ring (e.g., R1 max). If it is Null, and no FAP is defined, user specified ring is used.

- o Read ring - defines the highest read ring (e.g., R2 max). If it is Null, and no FAP is defined, user specified ring is used.

- o User Identifier - when this field is not that of the user initiating a new Segment Descriptor then its Control Attribute can be only of Use type.

6.1.2 REQUEST DESCRIPTIONS

The following subsection contains a description of requests accepted by Segment Control. Segment Control requests generally execute serially with respect to the requestor. An exception is always noted within the individual request description.

Request Status codes, which are set before returning from each request, and request processing are described in section 6.1.3 and 6.1.4, respectively.

6.1.2.1 Segment_Initiation/Termination Requests

These requests initiate/create or terminate/destroy Direct, Indirect, Temporary or Buffer Segments. The created segment's physical attributes are recorded within the one-per-segment Segment_Control_Table. One-or-several Segment Descriptor(s), within Segment Descriptor Tables of those Address Spaces where this segment has been initiated, point to its Segment Control Table. Segment Descriptors contain hardware enforced access control attributes of the segment and the Segment Control used/enforced ownership attributes. A unique identifier is assigned to the Segment on its creation (e.g., when the first SC#Initiate_Segment request led to the building of its Segment Control Table). The segment is known by this "name", or Locator Record, while its descriptors exist within the Storage System.

6.1.2.1.1 SC#INITIATE_SEGMENT

The purpose of this request is to make an existing segment known by a new process segment number, or enable direct addressing of an existing file as a Direct Segment, or create a new Direct, Temporary, or Buffer Segment within a job's segmented Address Space.

The macro format is as follows:

SC#Initiate_Segment (Seg_No, Seg_Specifier, Control, Access, Job_Id, Usage, Scope, Attribute, Supervisor, Status)

Seg_No: (Input, Output) - The Seg_No parameter is a variable of ^Cell type. It specifies either the Segment Number to be used for the initiated segment or, when it is set to Nil initially, a pointer to the newly allocated segment (with byte number of zero and ring number set to the callers validation level) is returned. A non Nil Seg_No parameter must correspond to a segment number reserved by a previous SC#Reserve_SDT_Entry (c.f., 6.1.2.2.3) request or by the Reserve_Option of an SC#Terminate_Segment (c.f., 6.1.2.2) request.

Seg_Specifier: (Input) - The Seg_Specifier parameter is a variable of union (Descriptor, Seg_PVA, FCB_Name, Fcbid) type.

Descriptor parameter is a pointer variable of Locator type. A Physical_File_Descriptor's or Segment_Control_Tables's position within the one-per-system Physical_Tables_Heap Segment is defined by that particular descriptor's Locator record (see 4.0 for details).

Seg_PVA parameter is a pointer variable of ^Cell type. It must specify a cell within an existing segment of the requestor's Address_Space.

FCB_Name parameter is a pointer variable of ^LNS_Name type. A File_Control_Block is defined as per the specified LNS_Name.

Fcbid parameter is a variable of File_Control_Block_Identifier type. It must specify a defined File_Control_Block.

The Seg_Specifier parameter is used to specify that:

either

1. A new Temporary_Segment is to be created;

   o Descriptor = Nil is specified (the associated Pool_File is specified by the Attribute parameter).

   or

2. An existing mass-storage file is to be made directly addressable as a Direct Segment;

   o A valid Descriptor is specified.

   Note: Only Task Services' procedures are permitted to use this form of the Seg_Specifier parameter. (This is the standard method for initiating a Direct Segment by the File System and Program Management).

   or

3. An existing Direct/Indirect/Temporary/Buffer Segment is to be made accessible via a new segment number and access attributes;

   o Seg_PVA is specified.

   or

4. A new File_Control_Block (FCB) is to be defined, the corresponding temporary mass-storage file created and made directly addressable as a Direct Segment;

   o The new FCB's LNS_Name is specified.

   Note: File System requests are used by the Storage System to:

   o Define and merge parameters into the FCB

   o Create a temporary mass-storage file as per the defined FCB.

   or

5. A file, specified by a File_Control_Block_Identifier (Fcbid), is to be created (when the FCB is not locked, e.g., no corresponding file exists) and/or made directly addressable as a Direct Segment;

   o An Fcbid is specified.

   Note: File_System requests are used by the Storage System:

   o To merge parameters into a non locked FCB

   o Create a temporary mass-storage file as per the defined FCB.

   An FM#Save_File requst can be used to convert a temporary file to a permanent file and record its attributes in the File Catalog of the appropriate Volume_Set. An FM#Attach_File request retrieves a pemanent file and initializes the corresponding FCB so that a subsequent SC#Initiate_Segment request can make it directly addressable (for details, see Section V of this GDS).

Control: (Input) - The Control parameter is a variable of Seg_Control = (Owner, Pass, Copy, Use) type. It is used to specify the Control Rights of the entity identified by the key field of the Segment Descriptor created by this request. (See 6.1.1 for detailed specification of Segment Access and Control Right Rules).

Access: (Input, Optional) - The Access parameter is a variable of ^Access_Descriptor type. An Access_Descriptor record is defined by the folloing SWL structure:

```
type

Access_Descriptor = record
  R1 : 0..15, "Write_Ring"
  R2 : 0..15, "Read_Ring"
  By_Pass_Cache : Boolean, "True for cache bypass"
  Execute : (Not_E, Execute, Local_Privileged_Execute,
       Global_Privileged_Execute)
  Read : (Not_R, No_Key_Control_Read, Key_Control_Read,
       Binding_Section)
  Write : (Not_W, No_Key_Control_Write, Key_Control_Write)
  Key/Lock : 0..63;
```

Global_Flag : Boolean, "True for global key/lock"
Local_Flag : Boolean "True for local key/lock"
recend;

The Initiated_Segment's Segment_Descriptor Entry (for details see Model_Independent IPL Processor/Memory GDS) is initialized by Segment Control according to the fields of the Access_Descriptor record and the access rules defined by 6.1.1.

An Access = Nil parameter is assumed to specify that standard system default values are to be used. These are the following:

- o R1 := Requestor's Validation_Level.
- o R2 := R1; e.g., both R1 and R2 ring numbers are set to the callers validation level.
- o By_Pass_Cache := False, e.g., the cache is not by-passed (Segment Descriptor's VL field is set to 10).
- o Execute := Not_E, e.g., the initiated segment is not executable (Segment Descriptor's XP field is set to 00).
- o Read := No_Key_Control, e.g., reading of the initiated segment is not controlled by Key/Lock and reading is permitted (Segment Descriptor's RP field is set to 10).
- o Write := No_Key_Control, e.g., writing of the initiated segment is not controlled by Key/Lock and it is permitted (Segment Descriptor's WP field is set to 10).
- o Key/Lock := Requestor's Key/Lock, e.g., the initiated segment's Key/Lock becomes that of the initiator.
- o Global_Flag := False, e.g., no global lock.
- o Local_Flag := False, e.g., no local lock.

Job_Id : (Input, Optional) - The Job_Id parameter is an integer variable of Job_Identifier type. The new Segment Descriptor is to be entered within the Segment Descriptor Table of the Address Space specified by this parameter.

Job_Id = Null is assumed to specify that the Job_Id is that of the requestor.

Only System Job(s) are permitted to create Segment Descriptors directly within Segment Descriptor Tables of other Address Spaces (e.g., with a Seg_No = Nil parameter). User and Subsystem Jobs must explicitly cooperate;

- o The target Address Space must explicitly reserve an entry within its own Segment Descriptor Table.
- o Notify the Address Space making this request.
- o The reserved Segment Descriptor's Seg_No is specified by the requestor in the SC#Initiate_Segment request.

Usage : (Input, Optional) - The Usage parameter is a variable of Seg_Usage = Set of (Block, Serial, Lock, Non-migrate, Critical, Reliable, Public, Buffer) type. It is used to specify the usage environment of a newly created segment.

Block parameter specifies that pages within a block of the segment should always be returned together to external storage. File updating is thus guaranteed to take place on a block-by-block basis. (Note: A block's pages are always fetched as a unit). Overuse of this option can result in larger job Working Set size then otherwise since inactive pages of a block can not be removed individually from real memory.

Serial parameter specifies that the segment is to be accessed serially. Storage System retrieves one more block then the one interrupted to and returns the previous block (when modified) to external storage thus bypassing the normal paging algorithm. The extra block's pages are only made valid in the page table when they are also interrupted to, therefore the "circular_buffer_scheme" (e.g., of memory page_frames) can be repeated and the segment's blocks fetched ahead of use.

Lock parameter specifies that the segment's pages should be locked into real memory immediately after retrieved from external storage. The locked down pages are bypassed by the paging algorithm, however, they are swappable when owned by an Address Space local segment. An SC#Advise_Out request can remove the lock and return them to external storage. It is used by Storage System to control its own

internally used segments.

Non-migrate parameter specifies that the segment's pages should not be migrated to paging storage. Storage System migrates frequently used pages of a segment to/from fast paging storage (e.g., Drum, Micro-Bit Memory, etc.) when available and specified to do so at System Deadstart or, on a job-to-job basis, by Job Scheduler (see 3.1 for details).

Critical parameter specifies that the Segment's storage space should be immediately updated when one of its modified pages leaves a job's Working Set. This option guarantees regular updating of critical data bases even while its more frequenly used pages are migrated to paging storage. Careless use of this option can cause needless output transfers.

Reliable parameter specifies that page-frames within highly reliable zones of memory should only be allocated to pages of this segment. System or Subsystem Jobs are only permitted to use this option.

Public parameter specifies that this segment might be accessed from several Address Spaces and thus its paging storage space must be suballocated from a globally shared Pool File.

Buffer parameter specifies that a newly created Temporary Segment is to be used as a Buffer Segment. This option is only accepted when the requestor's validation level is that of Task Services.

When $Seg_Usage [] is specified, the Storage System assumes that standard default conditions apply. These are the following:

- Pages are treated individually by the paging algorithm.
- No pre-fetching of blocks is performed.
- Pages are not locked into memory on initial access.
- Frequently used pages are migrated to/from paging storage.
- No automatic updating of storage space.

- Page_frames are allocated as available.
- Job's Local Pool File is used for paging storage.
- Newly created Transient Segments are non Buffer type.

Scope: (Input) - The Scope parameter is a variable of Segm_Scope = (Local, Global, System, Default) type. It is used to specify the initial usage scope of a newly created segment, e.g.,

- It is to be used only within the creating Address_Space?
- It is to be shared among Address Spaces?

Local parameter specifies that the newly created segment is to be accessed within the creating Address Space (e.g., it is not shared initially among Address Spaces). A local Segment's Segment Control Table is immediately removed from memory to external storage whenever the containing Address Space assumes Deferred State.

Global parameter specifes that the segment is to be shared by several Address Spaces. A segment's Scope can only become Global when it was created with Usage (Public) attribute. When a segment is initiated (e.g., Made_known) in more than one Address_Space its Scope is automatically changed to Global. It can only be made to revert to Local status by the SC#Make_Local request (c.f., 6.1.2.4.2). Segment Control Table of a Global Segment can only be removed from memory to external storage when it is not known by any of the currently Running Jobs.

System parameter specifies that a special global segment is to be created. A System Job can only initiate a segment with this attribute, its segment Control Table is permanently locked into real memory. Segments owned and used by the Storage System have this attribute.

Default parameter must be specified when this request is used to re-initiate an already existing segment with a new segment number.

Attribute: (Input, Optional) - The Attribute parameter is a variable of ^Attribute Descriptor type. It is used to specify the physical atributes of a new segment. An Attribute_Descriptor record is defined by the following

6.0 USER INTERFACE AND FACILITIES
6.1.2.1.1 SC#INITIATE_SEGMENT

SWL structure:

```
type

Attribute_Descriptor = record
  Block_Size : integer,
  Max_Length : integer,
  Memory_Region : (Default, Area1, Area2, "..," Area7)
  Pool_File : FileId
  recend;
```

Block_Size parameter specifies the segment's block size in bytes. It must be 2**N multiple of the minimum page frame size (512 bytes).

Max_Length parameter specifies the segment's size in bytes. It must be a multiple of Block_Size. Storage space is automatically allocated, when referenced, up to and including this size. Segm_Length_Error On_Condition is raised within the erring Control_Point's stacks when virtual memory is referenced beyond Max_Length. No new storage space is assigned, unless the segment is a Stack_Segment. In this case, the following procedure is followed:

1) Error is not from ring of Task Services;

   - o Dynamic Space Pointer (A0) is retrieved from the offending Control_Point's Exhange Package.
   - o A0 is set to Max_Length (e.g., when larger).
   - o An extra block or 1024 bytes (which ever is greater) worth of storage space is allocated to the segment, so that trap processing can be initiated.
   - o Page Control is signalled within the Control_Point. Page Control will cause the Segm_Length_Error On_Condition.

2) Error from Task Services; Follow special System Error Recovery procedure (c.f., 6.1.4.1)

Memory_Region parameter specifies the memory area to be used for allocating page_frames to this segment's pages. This parameter can only be set to Default (e.g., specifying the default area) within User Jobs. System or Subsystem Jobs can request other memory areas. Storage

6.0 USER INTERFACE AND FACILITIES
6.1.2.1.1 SC#INITIATE_SEGMENT

system is provided with the memory_area_ordinal vs. address range list on System Deadstart.

Pool_File parameter specifies the FileId of the file which should be used in place of the Job's Pool_File for paging storage. The user must have block level access to this file. When set to null, the default Pool File is used.

When Attribute = Nil is specified the Storage System assumes that standard default conditions apply. These are the following:

- o Max_Length := <System Default Value>,
- o Block_Size := <System Default Value>,
- o Memory_Region := <Default Area>,
- o Pool_File := <Job's Pool_File).

Supervisor: (Input, Optional) - The Supervisor parameter is a variable of ^Segm_Supervisor type. The Segm_Supervisor record is defined by the following SWL construct;

```
type

Segm_Supervisor = record
  Supervisor_CP : Control_Point_Address,
  Queue : qcb_pointer
  recend;
```

Control_Point specified by the above record's Supervisor_CP field is notified of all error or access-violation events associated with the supervised segment via standard signalling mechanism. Signals are enqueued into the queue defined by the Queue field.

Supervisor = Nil is assumed to specify that the segment is not to be be supervised.

Supervisor_Event Signal List;

To be defined.

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.1.2 SC#TERMINATE_SEGMENT

The purpose of this request is either to remove one or all Segment Descriptors associated with a Direct, Indirect, Temporary, Buffer Segment from the requestor's Segment Descriptor Table or unconditionally terminate a Segment Control Table and thus sever its association with the Segment Descriptors within the requestor's or all other Address Spaces. Local Segment Descriptor's are immediately removed on terminating the associated Segment Control Table. Segment Descriptor's within other job's (e.g., where the global segment is known) are also removed when those jobs assume Running Status. This request can also delete a temporary file, or detach a permanent file when they are affiliated with the segment being terminated. Segment termination procedures execute asynchronously with respect to the requestor (see Queue parameter).

The macro format is as follows:

SC#Terminate_Segment (Seg_Specifier, Drop_Scope, Drop_Mode, Queue, Reserve, Status)

Seg_Specifier : (Input) - The Seg_Specifier parameter is a variable of union (Descriptor, Seg_PVA, FCB_Name, Fcbid) type.

Descriptor parameter is a pointer variable of Locator type. A Physical_File_Descriptor's or Segment_Control_Table's position within the one-per-system Physical_Tables_Heap Segment is defied by that particular descriptor's Locator record (See 4.0 for details). Task Services' procedures are permitted only to use this form of the Segm_Specifier parameter.

Seg_PVA parameter is a pointer variable of ^Cell type. It must specify a cell within an existing segment of the requestor's Address Space.

FCB_Name parameter is a pointer variable of ^LNS_Name type. It must specify the LNS name of a File_Control_Block within the requestor's Address_Space.

Fcbid parameter is a variable of File_Control_Block_Identifier type. It must specify a File_Control_Block within the requestor's Address_Space.

The Seg_Specifier parameter is used to specify which Segment Descriptor is to be removed and/or the segment to

be terminated.

Drop_Scope : (Input) - The Drop_Scope parameter is a variable of Segm_Drop_Scope = (Single, Group, Job, All) type.

Single parameter specifies that only the Segment Descriptor defined by the Seg_PVA parameter is to be removed from the requestor's Segment_Descriptor_Table. The requestor's Control Attribute must be of Use Level or higher, or the request's Validation Level must be that of Task Services otherwise the request is rejected.

Job parameter specifies that all Segment Descriptors associated with the segment specified by Seg_PVA, Descriptor or via the FCB_Name/Fcbid parameter are to be removed from the requestor's Segment_Descriptor_Table. The requestor's Control Attribute must be of Pass Level or higher, or that of Task Services, otherwise the request is rejected.

Group parameter specifies that all Segment Descriptor's whose Key/Lock field is that of Seg_PVA should be removed from the requestor's Segment_Descriptor_Table. The requestor's Control Attribute must be of Copy Level or greater, or that of Task Services. Otherwise the request is rejected.

All parameter specifies that the segment is to be unconditionally made unknown in all Address Spaces and the respective Segment Descriptors removed system wide. The requestor's Control Attribute must be of Owner Level or that of Task Services otherwise the request is rejected. Forced_Seg_Terminate On_Condition will be caused by the Storage System in all other jobs (e.g., where the specified segment is currently known) but that of the requestor.

When Seg_Specifier :^: Descriptor, or Seg_Specifier :^: FCB_Name, or Seg_Specifier :=: Fcbid is specified Drop_Scope should be set to Job or All otherwise the request is rejected.

Drop_Mode : (Input) - The Drop_Mode parameter is a variable of Segm_Drop_Mode = Set of (Pages, Segment, File) type.

This parameter is ignored unless the requestor's Control Attribute statisfies one of the following conditions:

- It is of Use or higher Level and the Segment is only known by one Segment Descriptor.

- It is of Copy of higher Level, Drop_Scope (Group) is specified and the segment is only known by Segment Descriptors whose Key/Lock field is that of Seg_PVA.

- It is of Pass or higher Level, Drop_Scope (Job) parameter is specified and the segment is only known in the requestor's Address_Space.

- It is of Owner Level or the requestor's validation Level is that of Task Services.

Pages parameter is used to specify that all pages of the segment should be deleted from real memory and paging storage without updating the segment's storage space in external storage. The requestor should have write access to the segment otherwise the request is rejected.

Segment parameter is used to specify that the Segment_Control_Table, and hence the segment, should be terminated.

File parameter is used to specify that the file affiliated with a Direct Segment should be deleted, when it is a temporary file, or detached from the requestor's Address_Space when it is a permanent file.

When $Segm_Drop_Mode [Segment] is specified and a Direct or Indirect Segment is being terminated, then the storage space of the associated file is automatically updated from memory. All modified pages are also copied from paging storage, including pages directly controlled by the SC#Map_Out request (see 6.1.2.1.3). In case the requestor does not want to return the modified pages of an Indirect Segment, then these should be released by a previous SC#Release_PVA request (c.f., 6.1.4.5). The 'Page' parameter should be used with care since it also deletes those pages which are still in paging storage but have been removed from the requestor's direct control by a SC#Map_Out request. Automatic page updating is not performed when the associated Temporary file is terminated (e.g., 'File' is specified), a permanent file is always updated unless 'Page' is specified.

Queue : (Input) - The Queue parameter specifies a variable of qcb_pointer type. It defines the Queue to be used to

receive request status signals and the final request completion signal. An SC#Terminate_Segment request is considered completed when all relevant pages are returned to external storage, or released, and system tables cleared. The requestor may use the Program Management event monitoring procedures (see Chapter IV) to keep informed of the progress of this request. When the Queue parameter is Nil, request completion is not signalled. The requestor is notified of unrecoverable errors by causing the respective On_Conditions within his stacks (see 6.1.3 for details).

Reserve: (Input) - The reserve parameter is a variable of SD_Key type, where

type

SD_Key = 0..63;

It is used to reserve the terminated Segment Descriptor entry, within the respective Segment Descriptor Table(s), for subsequent use by the SC#Initiate_Segment (see 6.1.2.1.1) request.

SD_Key := 0, indicates that the descriptor entry(s) should not be reserved.
SD_Key := [1..63] indicates the descriptor entry(s) should be reserved for use by an entity with the specified key/lock.

Status: (Output) - The status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.1.3 SC#MAP_IN

The purpose of this request is to make a Direct Segment into an Indirect one.

The macro format is as follows:

SC#Map_In (Seg_No, Seg_PVA, Permit_Map_Out, Access, Control, Status)

Seg_No: (Input, Output) - The Seg_No parameter is a variable of ^Cell type. It specifies either the Segment Number to be used for the Indirect Segment or, when it is set to Nil initially, a pointer to the newly allocated segment descriptor is returned (see 6.1.2.1.1 for exact definition). Seg_No can be set to be the same as Seg_PVA (see below), in this case, no new Segment Descriptor is created.

Seg_PVA: (Input) - The Seg_PVA parameter is a pointer variable of ^Cell type to a data structure within an existing Direct Segment in the requestors Address Space. It is used to specify the subject Direct Segment.

Permit_Map_Out: (Input) - The Permit_Map_Out parameter is a variable of Boolean type.

Permit_Map_Out = True is used to specify that Map_Out (c.f., 6.1.2.1.4) access is permitted to the storage space of the subject Direct Segment by an entity when it is accessed via the Segment Descriptor specified by Seg_No.

Permit_Map_Out = False inhibits updating the subject Direct Segment's storage space by a Map_Out request.

This request is rejected when the requestor specified Permit_Map_Out = True and it has no write access to the subject Direct Segment, unless its Control_Level is Copy or higher, or its Validation Level is that of Task Services.

Access: (Input) - The Access parameter is a variable of ^Access_Descriptor type. Access_Descriptor record's definition can be found in 6.1.2.1.1 (see Access parameter). The specified Segment Descriptor's access control fields are initialized by Segment Control according to the fields of the Access_Descriptor record, specified by this parameter, and the access rules defined

by 6.1.1., with the following exception:

- The new Segment Descriptor's access control field can have Write_Attribute even when the requestor's initial access was Read_Only to the Direct Segment and its Control_Attribute is at Use_Level. This alteration of the basic access_control rules is permissible since the Write_Access only applies to pages within temporary paging storage.

Control: (Input) - The Control parameter is a variable of Seg_Control = (Owner, Pass, Copy, Use) type. It is used to redefine the Control Rights of the entity identified by the key field of the subject Segment Descriptor initialized/reinitialized by this request. (See 6.1.1 for detailed specification of Segment Access and Control Right rules.)

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.1.4 SC#MAP_OUT

The purpose of this request is to return an Indirect Segment into its original Direct form and, in addition, transfer a selected set of pages/blocks from the Indirect Segment associated paging storage to the segment's external storage space.

The macro format is as follows:

SC#Map_Out (Seg_PVA, PVA_Pairs, Terminate_SD, Status)

Seg_PVA: (Input) - The Seg_PVA parameter is a pointer variable of ^Cell type to a data structure within an existing Indirect Segment in the requestor's Address Space. It is used to specify the subject Indirect Segment.

PVA_Pairs: (Input) - The PVA_Pairs parameter is a pointer variable of ^PVA_Pairs_List type. PVA_Pairs_List is described by the following SWL constructs:

```
type

  Address_Pair = record
    Spva : ^Cell,
    Lpva : ^Cell
  recend,

  PVA_Pair_List = Array [0..*] of Address_Pair,
```

Spva parameter is a pointer variable of ^Cell type to a data structure in the first page/block of a set of consecutive pages, within the segment specified by the Seg_PVA parameter, to be transferred from paging storage to the respective segment's external storage space.

Lpva parameter specifies a pointer variable to a data structure within the last page/block of a set of consecutive pages to be transferred.

PVA_Pairs = Nil is used to specify that no pages/blocks within paging storage should be transfered to the external storage space.

This parameter is ignored by the Storage System when the respective SC#Map_In request, creating the Segment Descriptor specified by Seg_PVA, had its Permit_Map_Out parameter set to False.

Terminate_SD: (Input) - This parameter is a variable of Boolean type.

Terminate_SD = True is used to specify that the Segment Descriptor corresponding to the Seg_PVA parameter should be removed from the requestor's Segment Descriptor Table.

Terminate_SD = False reverts the specified Segment Descriptor's Write_Access field to the state previous to that of the respective SC#Map_In request.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.2 Segment Descriptor Table Manipulation Requests

These set of requests control an Address Space owned Segment Descriptor Table's size, reserve/release one or more Segment Descriptor entries and alter the segment access attributes recorded within Segment Descriptors.

6.1.2.2.1 SC#EXPAND_SDT

The purpose of this request is to expand the size of an Address Space owned Segment Descriptor Table.

The macro format is as follows:

SC#Expand_SDT (Free_Entry_Cnt, Status)

Free_Entry_Cnt: (Input, Output) - The Free_Entry_Cnt parameter is variable of integer type. It is used to specify the number of unused entries required within the requestors Segment Descriptor Table. When the number of the currently free entries (e.g., not allocated or reserved) is less than that requested Segment Control Expands the Segment Descriptor Table. The number of the new available entries is returned in this parameter on request completion.

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.2.2 SC#CONTRACT_SDT

The purpose of this request is to contract the size of an Address Space owned Segment Descriptor Table.

The macro format is as follows:

SC#Contract_SDT (Free_Entry_Cnt, Status)

Free_Entry_Cnt: (Input, Output) - The Free_Entry_Cnt parameter is a variable of integer type. It is used to specify the number of unused entries required to remain within the requestors Segment Descriptor Table. When the number of currently free entries is larger than that requested Segment Control attempts to reduce the size of the Segment Descriptor Table. A Segment Descriptor's position can not be changed while it is pointing to a valid Segment Control Table, therefore the SDT's size is only reduced down to the highest allocated entry. The remaining free entry count is returned on request completion within this parameter.

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.2.3 SC#RESERVE_SDT_ENTRY

The purpose of this request is to reserve one or more entries within a job's Segment Descriptor Table (SDT).

The macro format is as follows:

SC#Reserve_SDT_Entry (Location, Entry_Cnt, Group, Key, Job, Status)

Location: (Input, Output) - The Location parameter is a variable of ^Cell type. It is used to specify the process segment number of the entry, or of the first entry within a group whose members should have consecutive segment numbers, to be reserved within the SDT. When the requested SDT entry(s) is not free, the request is rejected and the appropriate status code returned.

A Location = Nil parameter specifies that the first (e.g., of a group), or only, entry can be reserved anywhere in the SDT. The reserved entry's segment number is returned on request completion as the <Segment_number> field of this parameter, with its ring number set to the caller's validation level and byte number field of Null.

Entry_Cnt: (Input) - The Entry_Cnt parameter is a variable of integer type. It is used to specify the number of SDT entries to be reserved. The Storage System automatically expands the Segment Descriptor Table of the requestor when the number of free entries is less than that requested.

Group: (Input) - The Group parameter is a variable of Boolean type.

Group = True is used to specify that a consecutive set of segment numbers should be reserved, in increasing order, starting with the one given in the Location parameter, or when this is Nil, anywhere in the SDT.

Group = False specifies that the segment numbers (e.g., when Entry_Cnt > 1 is specified) do not have to be reserved as a consecutive group.

Key: (Input) - The key parameter is a variable of integer type. It is used to specify the key of the entity which can make use of the reserved entry(s) in the SC#Initiate_Segment or SC#Return_SDT_Entry requests. When Key = 0 is specified any entity can use the reserved

entry.

Job: (Input) - The Job parameter is a variable of Job_Identifier type. It is used to specify the job's identifier which can make use of the reserved entry(s) in SC#Initiate_Segment request (see Job_Id parameter in 6.1.2.1.1). When Job_Identifier = Null is specified segment control reserves the entry(s) for the requestor's job.

Status: (Input) - The Status parameter is a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.2.4 SC#RETURN_SDT_ENTRY

The purpose of this request is to return one or more reserved entry(s) within the requestor's Segment Descriptor Table to non-reserved state.

The macro format is as follows:

SC#Return_SDT_Entry (All, Location, Entry_Cnt, Status)

All: (Input) - The All parameter is a variable of Boolean type. All = True specifies that all reserved entries whose key field is the same as that of the requesting entity are placed into non-reserved state. When All = False is specified unreserving is performed as per the Location and Entry_Cnt parameters of this request.

Location: (Input) - The Location parameter is a variable of ^Cell type. It is used to specify the process segment number of a particular entry, or the first entry of a consecutive group of entries whose members should be returned to non-reserved state.

Entry_Cnt: (Input, Output) - The Entry_Cnt paramter is a variable of integer type. It is used to specify the number of entries in a consecutive group to be returned to non-reserved state. When All = True is specified the number of unreserved entries is returned in Entry_Cnt on request termination.

Status: (Input) : The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.2.5 SC#CHANGE_ACCESS

The purpose of this request is to change the access attributes of a Segment Descriptor within the requestor's Segment Descriptor Table.

The macro format is as follows:

SC#Change_Access (Seg_PVA, Access, Control, Status)

Seg_PVA: (Input) - The Seg_PVA parameter specifies a pointer variable of ^Cell type. It must specify a Cell within an existing segment of the requestor's Address Space. Its <segment_number> field is used by Segment Control to locate the respective Segment Descriptor whose access attributes are to be changed.

Access: (Input) - The Access parameter is a variable of ^Access_Descriptor type. Access_Descriptor records' definition can be found in 6.1.2.1.1 (see Access parameter). The specified Segment Descriptor's access control fields are reinitialized by Segment Control according to the fields of the Access_Descriptor record, specified by this parameter, and the access rules defined by 6.1.1.

Control: (Input) - The Control parameter is a variable of Seg_Control = (Owner, Pass, Copy, Use) type. It is used to redefine the Cotrol Rights of the entity identified by the key field of the subject Segment Descriptor reinitialized by this request. (See 6.1.1 for detailed specification of Segment Access and Control Right rules.)

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.3 Segment Attribute Manipulation Requests

The following set of requests alter the physical attributes of a segment or provide information on its current status.

6.1.2.3.1 SC#SET_MAX_LENGTH

The purpose of this request is to set a new limit on the size of a segment. The requestor must have owner rights or its validation level must be that of Task Services.

The macro format is as follows:

SC#Set_Max_Length (Seg_PVA, Length, Status)

Seg_PVA: (Input) - The Seg_PVA parameter is a pointer variable of ^Cell type to a segment within the requestor's Address Space. It specifies the segment whose Length is to be changed.

Length: (Input, Output) - The Length parameter is a variable of integer type. It specifies the segment's Max_Length in bytes. It must be a multiple of Block_Size. This request can not reduce a segment's Max_Length below the highest block written. The previous Segment length is returned in this parameter on request completion.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.3.2 SC#EXPAND_SEGMENT

The purpose of this request is to allocate additional storage space to a segment beyond its highest allocated block.

The macro format is as follows:

SC#Expand_Segment (Seg_PVA, Expansion_Increment, Status)

Seg_PVA: (Input) - The Seg_PVA parameter is a pointer variable of ^Cell type to a segment within the requestor's Address Space. It specifies the segment whose size should be expanded.

Expansion_Increment: (Input, Output) - The Expansion Increment is a variable of integer type. It is used to specify the additional storage space to be allocated in bytes. Segment Control rounds this up to multiple of block-size for Temporary and Buffer Segments and to the Physical File Descriptor dependent file Expansion_Increment for Direct and Indirect Segments. The allocated storage space's size is returned in this parameter on request completion in bytes.

Status: (Input) - The Status parameter specifies a pointer to a standard status request. It must always be specified or the request is not accepted for processing.

6.1.2.3.3 SC#CONTRACT_SEGMENT

The purpose of this request is to reduce the size of the preallocated, not_used (e.g., not initialized or written to) storage space of a segment beyond its highest block written.

The macro format is as follows:

SC#Contract_Segment (Seg_PVA, Reduction, Status)

Seg_PVA: (Input) - The Seg_PVA parameter is a pointer variable of ^Cell type to a segment within the requestor's Address_Space. It specifies the segment whose size should be contracted.

Reduction: (Input, Output) - The Reduction parameter is a variable of integer type. It is used to specify the reduction in allocated storage space in bytes. Segment Control rounds this down to multiple of block-size for Temporary and Buffer Segments and to the Physical File Descriptor dependent file Expansion_Increment for Direct and Indirect Segments. The amount of storage space deallocated is returned within this parameter on request completion in bytes. Reduction = Null specifies that all not used storage space is to be deallocated.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.3.4 SC#TRUNCATE_SEGMENT

The purpose of this request is to release all not used (e.g., not written to or initialized) storage space of a segment.

The macro format is as follows:

SC#Truncate_Segment (Seg_PVA, Storage, Status)

Seg_PVA: (Input) - The Seg_PVA parameter is a pointer variable of ^Cell type to a segment within the requestor's Address Space. It specifies the segment which is to be truncated.

Storage: (Output) - The storage parameter is a variable of integer type. On request completion Segment Control returns the subject segment's allocated storage space size in bytes in this parameter.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.3.5 SC#RELEASE_PVA

The purpose of this request is to release a set of consecutive blocks (e.g., return them into the free storage pool) from the storage space of a Temporary or Buffer Segment. The requester must have write access to the segment or its Control_Attribute must be at least of Copy Level.

The macro format is as follows:

SC#Release_PVA (Lpva, Spva, Status)

Lpva: (Input) - The Lpva parameter is a variable of ^Cell type to a data structure within the first block to be released from storage and deleted from real memory.

Spva: (Input) - The Spva parameter is a variable of ^Cell type to a data structure within the last block to be released from storage and deleted from real memory.

Pages within the specified virtual address range and currently within paging storage are also deleted.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.3.6 SC#STATUS_SEGMENT

The purpose of this request is to permit a user to retrieve the a status and current attributes of a segment.

The macro format is as follows:

To be supplied when detailed internal design of the Storage System is completed.

6.1.2.4 Segment Sharing Requests

Segment sharing requests provide the general capablty for controlled access to sets of pages/blocks in a multi-thread environment. Consecutive pages/blocks can be reserved for exclusive update and shared read only access within one or more segments simultaneously. The Storage System supports internal queuing primitives for awaiting the release of reserved virtual memory areas. It also detects existing deadlocks. A rollback facility can be invoked to reverse the effect of a set of updates, in case of a deadlock, internal system or user error.

6.1.2.4.1 SC#MAKE_GLOBAL

The purpose of this request is to make an Address_Space local segment globally shareable among Address Spaces. The request is rejected unless the requestor's Control_Attribute is own or its Validation Level is that of Task Services and the segment was initially initiated with Usage (Public) attribute.

The macro format is as follows:

SC#Make_Global (Seg_PVA, Status)

Seg_PVA: (Input) - The Seg_PVA parameter specifies a variable of ^Cell type. It must specify a Cell within an existing segment which is current Local to the requestor's Address Space. Segment Control changes the segment's Local status to Global, thereby making it accessable from other Address Spaces.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.1.2.4.2 SC#MAKE_LOCAL

The purpose of this request is to return a globally shareable segment to local status with respect to the requestor's Address Space. The request is rejected unless the requesting entitle's Control_Attribute is Own or its Validation Level is that of Task Services and the segment is not known in any other Address Spaces beside that of the requestor.

Segment_Control_Table and Physical_File_Descriptor of an Address Space local segment is automatically removed from memory to paging storage when the respective job is placed into Deferred State, thereby enhancing memory utilization (see 4.0 for details).

The macro format is as follows:

SC#Make_Local (Seg_PVA, Status)

Seg_PVA: (Input) - The Seg_PVA parameter specifies a variable of ^Cell type. It must specify a Cell within an existing globally shareable segment.

Status: (Input) - The Status parameter specifies a pointer to a stadard status record. It must always be specified or the request is not accepted for processing.

The following six request's external interface specification is closely dependent on certain internal design trade-offs made within the Storage System (i.e., the flexibility of the table space maintenance procedures, recovery/restart capability, external storage space/memory/CPU overhead used, etc.). They will only be defined when the Storage System's internal design is completed on July 31, 1975.

6.1.2.4.3 SC#CREATE_TRANSACTION_LIST

To be defined.

6.1.2.4.4 SC#REMOVE_TRANSACTION_LIST

To be defined.

6.1.2.4.5 SC#REGISTER_TRANSACTION_SET

To be defined.

6.1.2.4.6 SC#START_TRANSACTION

To be defined.

6.1.2.4.7 SC#EXIT_TRANSACTION_SET

To be defined.

6.1.2.4.8 SC#LOCK_PVA_RANGE

To be defined.

6.1.3 REQUEST STATUS INFORMATION

To be supplied when the internal design of the Storage System is completed.

6.1.4 REQUEST PROCESSING SEQUENCES

To be supplied when the internal design of the Storage System is completed.

6.2 PAGE CONTROL REQUESTS

Page Control Requests are used to manage virtual memory usage and page frame residency. They are grouped into the following categories:

A. Usage Prediction Requests:

- MC#Advise_In
- MC#Advise_Out
- MC#Clear_PVA
- MC#Set_Usage_Level

B. Memory Management requests:

- MC#Lock_PVA
- MC#Unlock_PVA
- MC#Fix_Memory
- MC#Release_Memory

C. Status requests:

- MC#Status_WS
- MC#Status_Page
- MC#Status_Request

6.2.1 REQUEST DESCRIPTIONS

The following subsection contains a description of requests accepted by Page Control.

Request status codes, which are set before returning from each request, and request processing are described in sections 6.2.2 and 6.2.3, respectively.

6.2.1.1 Page Usage Prediction Requests

Page usage prediction request enable a user to advise the system of intended virtual memory access patters, when known in advance, thereby inproving system efficiency. These requests also permit close control over the updating of external storage space, when required, thus enhancing system resiliency.

6.2.1.1.1 MC#ADVISE_IN

The purpose of this request is to transfer a set of consecutive pages/blocks, within one segment, from external storage to memory. This request executes asynchronously with respect to the requestor (see Queue parameter).

The macro format is as follows:

MC#Advise_In (Spva, Lpva, Queue, Rid, Option, Status)

Spva: (Input) - The Spva parameter specifies a pointer variable ^Cell type to a data structure within the first page/block to be retrieved

Lpva: (Input) - The Lpva parameter specifies a pointer variable of ^Cell type to a data structure within the last page/block to be retrieved.

Queue: (Input) - The Queue parameter specifies a pointer variable of type qcb_pointer. It defines the Queue to be used to receive request status signals and the final request completion signal. An Advise_In request is considered completed when all relevant pages have been retrieved from mass-storage. When the Queue parameter is Nil, request completion is not signalled. The requestor may use the Program Management event monitoring procedures (see Chapter IV) to keep informed of the progress of this request.

Status Signal List:

To be defined.

Rid: (Output) - The Rid parameter specifies an integer variable of Request_Identifier type. Rid is returned on request initiation to allow subsequent monitoring of this request by a MC#Status_Request.

Options: (Input, Optional) - The Options parameter specifies a variable of Advise_In_Option = Set of (Extend_Segment, WSS_Override, Error_Report) type indicators.

Extend_Segment indicator inhibits rejection of this request when the requested transfer extends beyond the current segment length. Automatic mass-storage space allocation takes place up to the maximum permitted segment size. The new pages are initialized to Null.

WSS_Override indicator specifies that this request is not to be processed when either

- o The number of free page frames is insufficient to satisfy its requirements.

or

- o The job's WSS Limit might be exceeded during processing.

Error_Report indicator specifies that a full error report should be returned via the Queue when any recoverable or unrecoverable transfer/memory-parity errors occur during request processing.

Note The requestor is notified of unrecoverable errors by signalling the respective On_Conditions within his Stacks. (see 6.2.3 for details).

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.2.1.1.2 MC# ADVISE_OUT

The purpose of this request is to transfer a set of consecutive pages/blocks, within one segment, from memory to external storage and/or move pages/blocks from paging storage to the associated mass-storage space of an Indirect Segment. This request executes asynchronously with respect to the requestor (see Queue parameter).

The macro format is as follows:

MC#Advise_Out (Spva, Lpva, Queue, Rid, Option, Status)

Spva: (Input) - The Spva parameter specifies a pointer variable of ^Cell type to a data structure within the first page/block to be removed.

Lpva: (Input) - The Lpva parameter specifies a pointer variable of ^Cell type to a data structure within the last page/block to be removed.

Queue (Input) - The Queue parameter specifies a pointer variable of type qcb_pointer. It defines the Queue to be used to retrieve request status signals and the final request completion signal. An Advise_Out request is considered completed when all relevant pages have been transferred to external storage. Request completion is not signalled when the Queue parameter is set to Nil. The request may use the Program Management event monitoring procedures (See Chapter VI) to keep informed of the progress of this request.

Status Signal List

To be defind.

Rid: (Output) - The Rid parameter specifies an integer variable of Request_Identifier type. Rid is returned on request initiation to allow subsequent monitoring of this request by an MC#Status Request.

Option: (Input, Optional) - The Option parameter specifies a variable of Advise_Out_Option = Set of (Update, Update_Storage, Keep_WS, Keep_Lock, Error_Report) type indicators.

When $Advise_Out_Option [] is specified, Page Control assumes that the requestor completed using the indicated

pages/blocks and thus wants them removed from the job's Working Set. Page Table entries corresponding to the page-frames occupied by the pages, within the requestor's Working Set, are set invalid and the pages placed into the Free or Shared Queues (see 2.5.2). The Initial_Lock (see 6.1.2.1.1 parameter Usage: Lock_Option) is also removed from these pages. Effected pages could be in other job's Working Set when the target Segment is globally shared. In this case, the Initial_Lock is again removed, Use_Bits in the affected Page Table entries are reset, and the page-frame's Age_Count set to AIC (see 2.5.3) thus the pages are made candidates for leaving immediately the respective Working Set. Exact control over their future memory/paging storage/mass-storage residency is determined by the condition of the specific optional indicators.

Update indicator specifies that the external/paging storage space corresponding to the virtual address range should be immediately updated from memory. Page Table entries are marked invalid until request completion and placed into the Free/Shared Queues with Initial_Lock removed, or left within their original Working Sets as described in the previous paragraph.

Update_Storage indicator's effect is similar to that of the simple Update, however, when it is used on an Indirect Segment or while automatic page migration is in force, the storage space of the associated file is also updated. The requestor thus can enforce the updating to take place in well defined groups of pages/blocks thereby enhancing the resiliency of update processing. Pages/blocks within paging storage can be removed by a subsequent SC#Release_PVA request.

Keep_WS indicator inhibits removal of the effected pages from the requestor's Working Set. Their Use_Bit is cleared, Initial_Lock removed, Age_Count set to AIF and thus made candidates for reuse by a subsequent page-interrupt or SC#Advise_In request. This option enables a user to kep complete control over his Working Set Size.

Keep_Lock indicator inhibits removal of the Initial_Lock. (It is going to be used by the Storage System to keep precise control over its own internal Table Space).

Error_Report indicator specifies that a full error report should be returned via the Queue when any recoverable or

unrecoverable transfer/memory parity errors occur during request processing.

Note: The requestor is notified of unrecoverable errors by signalling the respective On_Conditions within his stacks (See 6.2.3 for details).

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be spcified or the request is not accepted for processing.

6.2.1.1.3 MC#CLEAR_PVA

The purpose of this request it to clear a set of consecutive pages/blocks within one segment.

The macro format is as follows:

MC#Clear_PVA (Spva, Lpva, Option, Status)

Spva: (Input) - The Spva parameter specifies a pointer variable of a ^Cell type to a data structure within the first page/block to be cleared.

Lpva: (Input) - The Lpva parameter specifies a pointer variable of ^Cell type to a data structure within the last page/block to be cleared.

Option: (Input) - The Option parameter specifies a variable of Clear_Option = Set of (Clr_Modified, Clr_Partial, Clr_Direct) type indicators.

When $Clear_Option [] is specified unmodified pages/blocks within the indicated virtual address range, and currently in memory, are deleted.

Clr_Modified parameter is used to specify that modified pages should be cleared and the external storage space of a Temporary Segment or an Indirect_Segment's Paging storage, corresponding to the virtual address range, should be treated as not initalized in any subsequent accesses. The requestor must have write access to the segment when this option is used otherwise the request is rejected. A Direct Segment's external storage space is not reset (see Clr_Direct Option).

Clr_Partial parameter specifies that pages/blocks not completely within the address range should be also partially cleared. Currently memory resident pages (maximum of two, one at each end of the range) are set to Null as specified. A Temporary Segment's non-memory resident pages, or Indirect Segment's page(s) in paging storage, are retrieved, partially cleared and placed into the Free or Shared Queue. The requestor must have write access to the segment when this option is used otherwise the request is rejected. A Direct Segment's external storage space is not reset (See Clr_Direct option).

Clr_Direct parameter must explicitly specify that the Storage

Space associated with the virtual address range of a Direct or Indirect Segment should be reset to Null. Since a Physical File Descriptor only contains the address of the highest block written to, the Storage System must explicitly overwrite the external storage space with Nulls, thus performing the equivalent of the Clr_Modified/Clr_Partial options for a Direct/Indirect Segment. This is a high overhead operation and should be used with caution (clearing/freeing Temporary Segments' pages/blocks can be done efficiently since only internal tables need to be updated). The requestor must have Write access to the segment or the request is rejected.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.2.1.1.4 MC#SET_USAGE_LEVEL

The purpose of this request is to advise the system of intended Working Set Usage pattern. Its use can result in improved memory utilization.

The macro format is as follows:

MC#Set_Usage_Level (Usage_Specifier, Status)

Usage_Specifier parameter is a pointer variable of ^Usage_Array type. Usage_Array is described by the following SWL constructs;

```
type

 Usage_Record = record
  Spva : ^Cell,
  Lpva : ^Cell,
  Usage : Boolean
 recend,

 Usage_Array = Array [0..*] of Usage_Record;
```

Spva parameter is a pointer variable of ^Cell type to a data structure in the first page/block of the page-set within a segment whose Usage_Level is to be respecified.

Lpva parameter is a pointer variable of ^Cell type to a data structure within the last page/block of the page-set whose Usage_level is to be respecified.

Usage parameter equal to False specifies that the page-frames associated with the pages within the page set, in the virtual address range Spva..Lpva, should be made candidates for reuse (e.g., if still within the job's Working Set) and thus the pages will be removed from the WS when a new page frame is required.

Usage := True specofies that pages within the virtual address range Lpva..Spva should remain within the Job's WS or included, if still in memory and the resulting increase in WS size is below the permitted WSS max.

Usage_Array specifies page-set's disposition, within the job's WS, with respect to their estimated future usage. It is used to advise the system of any gross alternations in membership of the Working Set.

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.2.1.2 Memory Management Requests

Memory Management requests enable a user to control the real memory residency of his code/date pages/blocks and to gain access to specific real memory locations.

6.2.1.2.1 MC#LOCK_PVA

The purpose of this request is to suspend paging operation on a set of consecutive pages, within one segment. The specified pages/blocks are retrieved from external storage, when not already in memory, and the respective page-frames' lock_count fields updated within the Storage System's Internal Memory Map (see 4.0 for details). The requestor must be a System Job or its 'profile' must contain Lock_PVA capability (see Chapter 9, 2.10.1.1.8) otherwise the request is rejected. Job_Identity/page_frames_locked information is maintained by the Storage System to permit control of lock removal/restoration on swapping or job termination. It is possible to create segments (see 6.1.2.1.1) whose pages are always locked while in use (see 6.2.1.1.2) or during the owning job's major time-slice.

Block Management and Device Driver routines always transfer data into/out-of already locked memory (see 1.2.1 and 5.0) and do not have to use this request.

This request executes asynchronously with respect to a requestor (see Queue Parameter).

The macro format is as follows:

MC#Lock_PVA (Spva, Lpva, Rid, Queue, Option, Status)

Spva: (Input) - The Spva parameter specifies a pointer variable of ^Cell type to a data structure within the first page/block to be locked.

Lpva: (Input) -The Lpva parameter specifies a pointer variable of ^Cell type to a data structure within the last page/block to be locked.

Queue: (Input) - The Queue parameter specifies a pointer variable of qcb_pointer type. It defines the Queue to be used to retrieve request status and the final request completion signal. A Lock_PVA request is considered completed when all relevant pages/blocks have been retrieved from external storage and locked into the

associated page-frames. Request completion is not signalled when the Queue parameter is set to Nil. The requestor may use the Program Management event monitoring procedures (see Chapter IV) to keep informed of the progress of this request.

Status Signal List:

To be defined.

Rid: (Output) - The Rid parameter specifies an integer variable of Request_Identifier type. Rid is returned on request initiation the allow to subsequent monitoring of this request by a MC#Status_Request.

Option: (Input) - The Option parameter specifies a variable of Lock_PVA_Option = Advise_In_Option type indicators. See 6.2.1.1.1 for detailed specification.

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.2.1.2.2 MC#UNLOCK_PVA

The purpose of this request is to restart paging operation on a set of consecutive pages/blocks within one segment. Unless a previous MC#Lock_PVA operation has been performed by the requesting Job on all pages/blocks specified by this request, the request is rejected. Pages can be unlocked on a page-by-page basis.

The macro format is as follows:

MC#Unlock_PVA (Spva, Lpva, Option, Status)

Spva: (Input) - The Spva parameter specifies a pointer variable of ^Cell type to a data structure within the first page/block to be unlocked.

Lpva: (Input) - The Lpva parameter specifies a pointer variable of ^Cell type to a data structure within the last page/block to be unlocked.

Option: (Input, Optional) - The Options parameter specifies a variable of Unlock_PVA_Option = Set of (Remove_WS, Release_Pages) type indicators.

When $Unlock_PVA_Option [] is specified, Page Control assumes that the requestor intends paging operations to restart. The one-per-page lock count is decreased, within the Storage System's internal tables, and when it becomes zero, the affected Page Table entries Use bit is cleared. The associated page frame's Age_Count is set to AIF (see 2.5.3) and, unless accessed soon, made a candidate for eventual reuse.

Remove_WS indicator specifies that pages within the indicated virtual address range should be removed from the job's Working Set.

Release_Pages indicator specifies that pages within the indicated virtual ddress range should be deleted from real memory and the asssociated page-frames freed. The requestor must have write access when any of the affected pages has been modified. A page partially included in the virtual address range is not deleted and only cleared as specified.

Status: (Input) - The Status parameter specifies a pointer to a standard status record. It must always be specified or

the request is not accepted for processing.

6.2.1.2.3 MC#FIX_MEMORY

The purpose of this request is to allocate permanently (e.g., for the life-time of the owning job or until released) a continuous section of real memory and associate it with a virtual address range within one segment.

Principal users of this request are Diagnostic Subsystem, I/O Subsystem, Storage System, Block Manager, System Monitor which either want access to specific 'real' memory addresses or require a non-pagable continuous real memory area (i.e., IORB's and Segment Descritor Tables).

Storage System can not guarantee instantaneous allocation, and in turn, it is a user of System Monitor, Block Management and the I/O Subsystem. These operating system modules must maintain enough reserved space to permit Page Control to function while freeing up memory.

A Job issuing this request must be a System Job or have Fix_Memory capability.

This request can execute asynchronously with respect to a requestor (see Queue parameter).

The macro format is the following:

MC#Fix_Memory (Memory_Location, Spva, Length, Queue, Rid, Option, Status)

Memory_Location: (Input, Output) - The Memory_Location parameter is a variable of Real_Address: integer type. It is used to specify the address of the real memory section to be allocated in bytes.

When Real_Address = 2**31-1 is specified Page Control assumes that it is free to allocate a memory area of sufficient size anywhere in real storage with the least amount of 'global allocation cost'. Its address is returned within the Memory_Location parameter for a synchronous request (e.g., Queue = Nil is specified, see Queue parameter below). It is part of the request completion signal for the asynchronous case.

When Real_Address is not equal to 2**31-1 is specified Page Control attempts to allocate memory at the requested address. It performs paging or memory_move operations to free the required area unless this includes locked or

fixed page-frames. In this case, it rejects or queues this request (see Enqueue option).

Spva: (Input, Output) - The Spva parameter specifies a pointer variable of ^Cell type to a data structure within the first page/block to be fixed.

When Spva's <byte-offset> = 2**31-1 is specified Page Control assumes that the process virtual address' byte offset within the associated segment should be equivalent to the real memory address of the fixed area. Spva is set and returned accordingly for a synchronous request (see Queue parameter below). It is part of the request completion signal for the asynchronous case.

Length: (Input, Output) - The Length parameter is a variable of integer type. It is used to specify the length of the memory area to be fixed in bytes. Page Control can only assign whole multiple of pages, therefore, Length is rounded-up accordingly and the size of the allocate memory is returned within this parameter for a synchronous request. It is part of the request completion signal for the asynchronous case (see Queue parameter below).

Queue: (Input) - The Queue parameter specifies a pointer variable of qcb_pointer type. It defines the Queue to be used to retrieve request status and the final request completion signal. A Fix_Memory request is considered completed when a memory area of specified size and location is found and assigned fixed status within the Storage System's internal tables. Pages corresponding to the virtual address range of the fixed_area in external storage are not allocated or retrieved when exist. Request completion is not signalled when the Queue parameter is set to Nil and the request is processed synchronously with respect to the requestor (e.g., it is put in wait state).

The requestor may use the Program Management event monitoring procedures (see Chapter IV) to keep informed of the progress of this request.

Status Signal List

To be defined.

Rid: (Output) - The Rid parameter specifies an integer variable of Request_Identifier type. Rid is returned on

request initiation to allow subsequent monitoring of this request by a MC#Status_Request.

Option: (Input) - The Option parameter specifies a variable of Fix_Option = Set of (WSS_Override, Enqueue) type indicators.

WSS_Override indicator specifies that this request is not to be processed when the job's WSS Limit might be exceded during processing. This parameter is only relevant for a subsystem job since System Job(s) WSS is only limited by the size of the available real memory.

Enqueue indicator specifies that this request should be enqueued until a memory area of sufficient size and/or at the specified location becomes available. An intermediate signal is returned when Queue is not equal to Nil and the request cannot be satisfied immediately (e.g., not large enough continuous area is available or locked/fixed pages exist at the specified real memory address). A subsequent Status_Request can be issued to cancel this request, when its requirements are not met, after a sufficient time interval.

Status: (Output) - The Status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.2.1.2.4 MC#RELEASE_MEMORY

The purpose of this request is to release a continuous section of real memory from fixed status allocated by a previous Fix_Page request. A continuous area of memory can be returned in subsections, however, these must be at the beginning or end of the initially allocated area. The virtual address range corresponding to the released memory area is treated as if a Clear_PVA operation has been performed.

The macro format is as follows:

MC#Release_Memory (Spva, Length, Status)

Spva: (Input) - The Spva parameter specifies a pointer variable of ^Cell type to a data structure within the first page to be released from fixed status.

Length: (Input, Output) - The Length parameter is a variable of integer type. It is used to specify the length of the memory area to be released in bytes. Page Control can only fix/release whole multiple of pages, therefore Length is rounded-up accordingly and the size of the released memory area is returned in bytes.

Status: (Output) - The status parameter specifies a pointer to a standard status record. It must always be specified or the request is not accepted for processing.

6.2.1.3 Status Requests

These requests permit a user to retrieve the status of individual pages/page-frames, or that of job Working Sets. The processing of asynchronous requests can be monitored and terminated forcibly.

6.2.1.3.1 MC#STATUS_WS

The purpose of this request is to retrieve the status of pages within a job's Working Set.

The macro format is as follows:

To be supplied when the detailed internal design of the Storage System is completed.

6.2.1.3.2 MC#STATUS_PAGE

The purpose of this request is to retrieve the status of a page or a page-frame.

The macro format is as follows:

To be supplied when the detailed internal design of the Storage System is completed.

6.2.1.3.3 MC#STATUS_REQUEST

The purpose of this request is to monitor the progress of an asynchronous request being processed by Page Control.

The macro format is as follows:

To be supplied when the detailed internal design of the Storage System is completed.

6.2.2 REQUEST STATUS INFORMATION

To be supplied when the internal design of the Storage System is completed.

6.2.3 REQUEST PROCESSING SEQUENCES

To be supplied when the internal design of the Storage System is completed.

6.3 SYSTEM INTERFACE REQUESTS

System Interface Requests are used to define the Storage System's operational environment and the Utility, Restart and Recovery Functions performed for, or in conjunction with, other elements of IPLOS. They are grouped into the the following categories:

A. Table Definitions

B. Utility Functions

C. Restart and Recovery Functions

To be supplied when the detailed internal design of the Storage System is completed.