IPLOS GDS - STRUCTURE/OVERVIEW/CONVENTIONS

NCR CONTROL DATA CORPORATION

ADVANCED SYSTEMS LABORATORY

CHAPTER 02

IPLOS

Structure, Overview

and

System Conventions

Doc. No. ASL00282

Rev. 04

Copy No. 87

STRUCTURE/OVERVIEW/CONVENTIONS

This document represents the structure, overview, and conventions for V 1.0 of the Operating System.

This document will not describe an entire design but rather a set of interfaces and conventions which should be followed and improved upon during the detailed design/implementation phase. This is based on several beliefs:

- The system will change over time.
- A defined structure which can evolve is more important than features or speed.
- During the push for an operational system, violation of good coding/design/implementaion practices will occur. Having a uniform project understanding will help everyone recognize when this is happening.
- There should be a set of Logical interfaces to physical resources which give the implementor some insulation from the way the physical resource looks or is being managed.
- There will be multiple variants of the Operating System with all Product Set, User and Command Language (Operator/User) Codes transportable between variants.

The next update of this document will concentrate on a more detailed description of tables in Section 2.4.

1.0 SYSTEMS COMMAND LANGUAGE (SCL)

1.0 SYSTEMS COMMAND LANGUAGE (SCL)

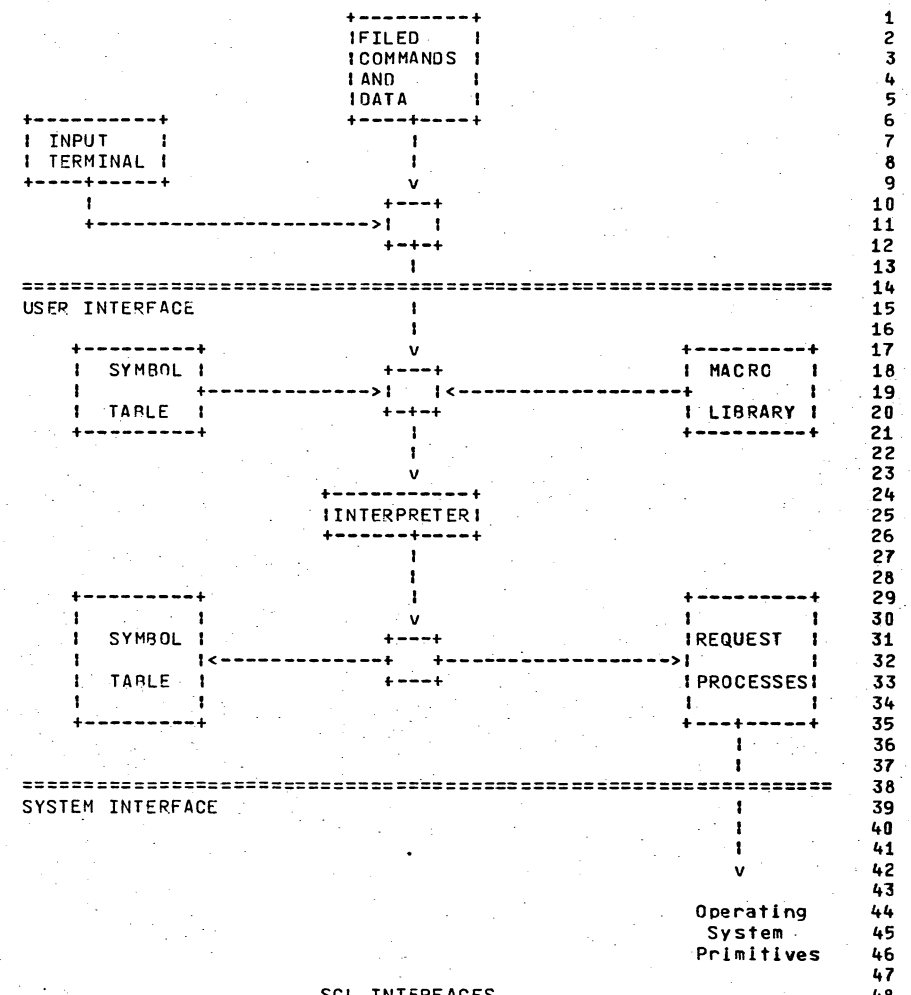
The Command Language is an externalization of the Operating System. Almost all Users (system, application, operations) will come in contact with this interface. A major objective of the external design is tailoring to the type of user, while a major objective of the internal design is modularity and maintainability.

Several characteristics of the Command Language include:

- o It is a true language with enough capability to solve simple problems.
- o Most Command Language statements are MACRO calls and result in the execution of several Operating System primitives.
- o User to System communication is through a symbol table (Logical Name Space).
- o Input to the Command Language interpreter can be redirected to previously filed commands.
- o It is invariant to the mode of access, e.g., Local Batch, Interactive, Remote Batch.
- o Some services supported by the system will be implemented using Command Language.

This command language interface can isolate the end user from underlying system requests providing a level of consistency and reliability above that available if these requests were directly called. Command Language syntax and parameter rules must be followed by all subsystems (debug, edit, online maintenance, operator, etc.).

1.0 SYSTEMS COMMAND LANGUAGE (SCL)



SCL INTERFACES

1.0 SYSTEMS COMMAND LANGUAGE (SCL)

While it is true that the normal user can express his problem in a shorter amount of time and in a more reliable fashion through the use of a high level language, it is also true that not all problems can be expressed effectively in a high level language. This is because the amount of control the user has over the physical resources of the system is reduced by automating the mapping of his logical requirements onto the physical resources of the system. This problem is typically solved by providing a machine language escape of some sort in the case of programming languages and can be solved by providing an Operating System escape in the case of a command language.

The approach outlined above is the one that has been adopted for the IPL System Command Language. Three logical environments are defined, one for the normal user, one for the system user and one for the system operator. These environments are controlled by three repertoires called the User, System and Operator Repertoires respectively. These three repertoires are not mutually exclusive and therefore a system user can have both the System Repertoire and the User Repertoire attached at the same time.

1.0 SYSTEMS COMMAND LANGUAGE (SCL)

User Repertoire Summary

LOGIN/JOB	gain access to the system
LOCK	lock the terminal preventing inadvertant logout
LOGOUT/JOBEND	relinquish access to the system
LIMIT	limit the amount of resources the job can consume
CLAIM	set maximum usage for a class of devices for a job
CHANGE	increase or decrease number of units of a class
RESERVE	register the requirement for a non-preemptible resource
CANCEL	cancel all previous RESERVE requests
ACQUIRE	satisfy all previous RESERVE requests
RETURNR	return a file, volume set, or unit set to a job
DECLARE	declare an lns entry
REMOVE	remove an lns entry
LNS	add or delete entries from lns segment list
LNSBLOCK/LNSEND	delimit an LNS naming context
FORTRAN	invoke the FORTRAN compiler
COBOL	invoke the COBOL compiler
SWL	invoke the SWL compiler
COMPILE	invoke default compiler or use file data type attribute
LIBRARY	add or delete entries on a Library List
OBJECT	add or delete entries from a Object List
EXECUTE	cause the named program to be loaded and executed as a task
ATTACH	attach a permanent mass storage file to a job
DETACH	detach a permanent mass storage file from a job
CREATE	allocate mass storage space for a temporary or permanent file
EXPAND	allocate additional mass storage space for existing temporary or permanent file
CONTRACT	release part or all the mass storage space allocated to temporary or permanent file
SAVE	convert a temporary mass storage file to a permanent file
DELETE	deletes a temporary or permanent file

	from the system	1
REDEFINE	redefine some of the logical	2
	characteristics of an existing permanent	3
	mass storage file	4
PERMIT	gives or modifies access(x) to file(y)	5
	for user(z)	6
PROHIBIT	removes access(x) to file(y) for user(z)	7
CONNECT	establish connection between stream and	8
	file	9
DISCONNECT	remove connection between stream and	10
	file	11
ROUTE	transmit a file	12
DIRECT	alter the destination of a file	13
RETRACT	remove the effect of a previously issued	14
	DIRECT request	15
PRINT	print a file	16
PUNCH	punch a file	17
SEND	send a message to a user or set of users	18
MAIL	mail a message to a user or set of users	19
DELETE	delete the contents of a user's mailbox	20
TIMER	select a time condition	21
WHEN/WHENEND	associate a series of commands with a	22
	specified event	23
WAIT	await the occurrence of a specified	24
	event	25
CAUSE	cause the occurrence of a specified	26
	event	27
CLEAR	clear a specified event	28
ENABLE	enable a specified event	29
DISABLE	disable a specified event	30
SUBMIT	submit a new job to the system	31
STOP	stop processing of task or job	32
START	restart processing of task or job	33
TERMINATE	terminate processing of task or job	34
DISPLAY	display the contents of a file or ins	35
	value	36
CALL	process the SCL text contained in a file	37
RETURN	return control from a called file	38
IF/IFEND	delimit conditionally processed commands	39
GOTO	transfer control to the label specified	40
ACCEPT	read data from the standard input	41
COLLECT	collect data from the input stream	42
COPY	copy data	43
DEFINE	define new commands and redefine system	44
	supplied commands	45
BEGIN/END	delimit a command block	46
FOR/FOREND	delimit a FOR loop	47
LABELBLOCK/LABELEND	delimit an SCL label block	48

MICRO	define a micro	1
SET	set or clear SCL toggles	2
BASEFILE	appoint file as the base file	3
		4
		5
		6
		7
		8
		9
		10
		11
		12
		13
		14
		15
		16
		17
		18
		19
		20
		21
		22
		23
		24
		25
		26
		27
		28
		29
		30
		31
		32
		33
		34
		35
		36
		37
		38
		39
		40
		41
		42
		43
		44
		45
		46
		47
		48

1.0 SYSTEMS COMMAND LANGUAGE (SCL)

System Repertoire Summary

The system repertoire is comprised of a set of commands which represent an externalization of the Operating System requests available to a running program. These commands allow the user to invoke most of the Operating System request processors directly. The command descriptions are contained in the various sections of the GDS at the points where the requests are defined.

1.0 SYSTEMS COMMAND LANGUAGE (SCL)
1.1 OPERATOR(S) COMMAND LANGUAGE

1.1 OPERATOR(S) COMMAND LANGUAGE

The primary objective of the Operator Communications System (OCS) is to provide a facility whereby both the system and users may converse with the human operator(s) of the system or network. Specific objectives of OCS are:

- The system should be flexible and easily extended at the site.
- The system will have several logical operators, each with different responsibilities and privileges.
- The set of logical operators may be mapped onto one or more physical operators.
- The hardware and software necessary to support the input and output streams of a physical operator should not be different from that of any other job doing terminal I/O.
- The system should be capable of operating in a default mode without any physical operators.

Operator Repertoire Summary

REPLY	respond to message issued by a OC#CONVERSE request.
INFORM	send unsolicited input to a Job(s).
STATUS	display the status of a system recognized entity.
SEIZE	take over complete control of a hardware resource.
UNSEIZE	return a previously seized resource.
MEMORY	display/alter real or virtual memory.
PATCH	temporary replacement of a system procedure.
SET	set various system operational valves.
SHUTDOWN	close down a site in an orderly manner.
HOLD	suspend activity at natural boundary
STOP	suspend an activity immediately.
START	restart previously suspended activity
RESET	reset an activity to beginning point.
CANCEL	stop an activity and purge it from the system.
ALTER	change/alter Job's operational values
ONSYSTEM	assign a device to the system job.
OFFSYSTEM	release a device from the system job.
BACKSPACE	back up an output listing.
FORESPACE	skip forward an output listing.
PAGE	print a specified number of pages.
ONLINE	add a device to the configuraton.
OFFLINE	delete a device from the configuration.

Command Availability Summary

The following table illustrates the availability of commands to different levels of operators and the activities that may be affected by them. The following abbreviations are used in the table.

ok	no restrictions
na	not available
-->	included via lower level repertoire
system	entire system
q	input or output queue
opq	output queue
ropq	remote batch output queue
job	job
rjob	remote origin job
opqjob	job in output queue or function
ropqjob	job in remote batch output queue or function
dev	device
sysdev	"onsystem" device
rdev	remote batch terminal device
rsysdev	remote batch terminal "onsystem" device

To use the table, locate the intersection of the command name and the logical operator job name. The activities that may be referenced by the command when used by the particular logical operator are all those to the right of the intersection in the row for that command. If an arrow (-->) is encountered, follow it.

1.0 SYSTEMS COMMAND LANGUAGE (SCL)
1.1 OPERATOR(S) COMMAND LANGUAGE

	CE#OP	SE#OP	SYSTEM#OP	MASS#OP TAPE#OP BATCH#OP	REMOTE#OP
reply	ok	ok	ok	ok	rjob
inform	ok	ok	ok	ok	rjob
status	ok	ok	ok	ok	ropq rjob rdev
seize	ok	na	na	na	na
unseize	ok	na	na	na	na
memory	-->	ok	na	na	na
patch	-->	ok	na	na	na
set	-->	-->	ok	na	na
shutdown	-->	-->	system	na	na
hold	-->	-->	system q job	opq opqjob sysdev	ropqjob rsysdev
stop	-->	-->	system q job	opq opqjob dev	ropqjob rdev
start	-->	-->	system q job	opq opqjob dev	ropqjob rdev
reset	-->	-->	job	opqjob sysdev	ropqjob rsysdev
cancel	-->	-->	q job	opq opqjob sysdev	ropqjob rsysdev
alter	-->	-->	job	opqjob	ropqjob

continued...

1.0 SYSTEMS COMMAND LANGUAGE (SCL)
1.1 OPERATOR(S) COMMAND LANGUAGE

	CE#OP	SE#OP	SYSTEM#OP	MASS#OP TAPE#OP BATCH#OP	REMOTE#OP
onsystem	-->	-->	-->	dev	rdev
offsystem	-->	-->	-->	dev	rdev
backspace	-->	-->	-->	sysdev	rsysdev
forespace	-->	-->	-->	sysdev	rsysdev
page	-->	-->	-->	sysdev	rsysdev
online	-->	-->	-->	dev	rdev
offline	-->	-->	-->	dev	rdev

1.0 SYSTEMS COMMAND LANGUAGE (SCL)
1.1 OPERATOR(S) COMMAND LANGUAGE

```
JOB WJH "COBOL COMPILE AND EXECUTE"                          1
COLLECT SOURCE                                               2
.                                                            3
          .  COBOL SOURCE DECK APPEARS HERE                  4
                    .                                        5
**                                                           6
COBOL I=SOURCE, O=OBJECT, L=LISTING, S=ERR                   7
IF ERR.LEVEL GT 0                                            8
                    PRINT LISTING                            9
                    JOBEND                                  10
                    IFEND                                   11
SAVE OBJECT                                                 12
OBJECT ADD=OBJECT                                           13
COLLECT DATA UNTIL = /.                                     14
.                                                           15
          .  DATA DECK APPEARS HERE                         16
                    .                                       17
/.                                                          18
EXECUTE PROG=MAIN, PARAM=DATA                               19
JOBEND                                                      20
                                                            21
                                                            22
JOB WHJ "COBOL EXECUTE ONLY"                                23
OBJECT ADD=OBJECT                                           24
COLLECT DATA UNTIL = /.                                     25
.                                                           26
          .  DATA DECK APPEARS HERE                         27
                    .                                       28
/.                                                          29
EXECUTE PROG=MAIN, PARAM=DATA                               30
JOBEND                                                      31
                                                            32
                                                            33
                                                            34
                                                            35
                                                            36
                                                            37
                                                            38
                                                            39
                                                            40
                                                            41
                                                            42
                                                            43
                                                            44
                                                            45
                                                            46
                                                            47
                                                            48
```

2.0 STRUCTURE

2.1 INTRODUCTION

As the number of concurrent users has increased, Operating System structures have become more sensitive to integrity problems.

A need to protect and share data on a logical unit which is quite small (table, byte, bit) has resulted. Typically, groups of these logical units are collected together (in order to reduce fragmentation and complexity) within a system supported protection container (segment file, etc.) and then accessed interpretively (via procedure). For the same reasons, procedures are collected together within a system supported protection container (cannot afford a unique procedure per function per user). This need to control access to critical data via 'procedure sets' is not unique to the Operating System. Any large multi-user application will have the same requirements and therefore it is imperative to externalize the solutions used by the O.S. to a subsystem writer.

2.1.1 MULTIPLE MONITOR CONCEPT

The Operating System is trying to generalize the User-Supervisor situation. This concept of multiple monitors, each in charge of a single shared data base, is analogous to the approach used for many large resource management problems: i.e.,

- Implement levels of responsibility and isolation where each level is knowledgeable and responsive to requests in a local environment.
- Requests at the same level may operate concurrently with other environments.
- Requests which cannot be processed at a local level are automatically routed to a more global level.
- Reduction in responsibility of any one level reduces the impact of failure.

As the concept of multiple monitors has evolved, two accepted techniques have appeared:

- Separate address space for each monitor (Master, OS/MVT, Cyber)
- Monitor(s) within each address space (PL, Multics, OS/VS2)

Recent Operating Systems contain both interfaces, with the degree of support being heavily influenced by the sophistication of the hardware.

2.1.2 HISTORY

Many of the ideas being used in the IPL Operating System implementation are based on the Basic Time Sharing (BTS) papers published by the Control Data advanced development group. The first BTS-based implementation for the CDC 3300/3500 computer (MASTER) reached production status and is a standard CDC offering today. This was followed by an implementation using CDC peripheral processors with private memory and a large shared common memory (MOS) and an implementation on CDC STAR hardware (PLOS). Neither of the latter implementations reached product status. The IPL Operating System is being evolved from the current implementation of PLOS.

Two characteristics which have changed drastically over the time period of these systems are the advances made in storage technology (size, cost, speed, reliability of Disk Memory, plus the hardware assisted techniques for storage management) and the great increase in the services expected of an Operating System.

These two factors have impacted the O.S. evolution in the following way:

BTS advocates translating all requests for service into a Task which an EXECUTIVE will apply to an available processor. The EXECUTIVE has three basic types of services: processor assignment, call routing (a call is a request for work made by one task on another task), and signal handling (inter-task communication). The main problem encountered is the overhead of the executive. BTS advocates implementing these basic services in the processor in order to gain the necessary efficiency.

Since there is a reluctance to implement specific O.S. services in hardware, a slightly different approach must be taken.

One way to reduce the overhead induced by the executive is to not use it. This means a set of O.S. services are placed in the environment of the requesting program, thereby eliminating the trip through the executive that was previously required in order to access the services.

This concept must not be used to the extreme or a new set of problems will result as the Multics Operating System demonstrates (i.e. global resources, such as processors and physical memory, should be managed outside of the requestors environment).

The IPL Operating System will provide services implemented in both local and global environments.

2.1.3 IMPLEMENTATION

For the multiple monitor concept to work, a formal set of hardware and software conventions must be imposed on both user and system code. This will allow the normal debugging, linking/loading, code maintenance, accounting and checkout methods of the user and the system to be the same. This also facilitates the movement of code between levels and allows recursive use of the system. These conventions will cover:

- o Entry/Exit
- o Parameter Passing
- o Parameter Validation
- o Address Space Management
- o Memory Management
- o Protection Environment
- o Naming Environment
- o Code Generation

Most large applications allow binding of code and data to occur at compile, load and execution time. The Operating System is complicated by the fact that most binding occurs as a result of both loading and execution and is between system-supplied and user-supplied units. Many of the conventions are intended to enforce correct binding and to allow unbinding to occur in an orderly manner. The most difficult situations exist in a program when the unpredictable occurs and orderly termination is necessary. Several Operating System examples include Login/Logout (job), open/close (file), call/return (procedure) and declare/remove (variable). In each example, system defined conventions exist for initiation and termination, whether normal or abnormal. The importance of these conventions is greatly magnified by the desire to share code and data within the system and between users of the system.

CONVENTION IMPLICATIONS

ENTRY/EXIT

- o Sharing code requires that the control path is remembered outside of the code body.
- o Protection implies that separate stacks are maintained for each level of privilege and that correct entry to each level can be guaranteed.
- o Externalization to subsystem writers implies the existence of a way to add or delete protected entry points associated with the subsystem without doing a system generation.
- o Asynchronous operation implies separate machine environment, scheduling information (status, priority), and monitor intervention for coordination.

PARAMETER PASSING

- o Asynchronous operation implies the need for either synchronization primitives (if parameters are in shared space) or movement of parameters through a neutral storage area.
- o Protection requires no inadvertant destructions of communication areas which may imply copying, hierarchical permissions and read only solutions.
- o Location transparency (separate address space, separate systems) implies the existence of intervening procedures to collect/dispose and transmit/receive parameters.

PARAMETER VALIDATION

- Parameter validation requires that the identity of the caller is guaranteed to be correct.
- Parameter validation requires the type of parameter to be known. This in turn implies that descriptive information exists for all data and is protected separately from the actual data.
- Parameter validation requires the executing procedure to determine its own level of privilege.

ADDRESS SPACE MANAGEMENT

- Movement of functions within the system implies the need for consistent address space management. Care should be used when considering dedicated addresses, acquiring and manipulating full pointers, and reusing addresses.
- Segmentation, with dynamic assignment of segment numbers, implies that most information should be accessed and used via offsets.

MEMORY MANAGEMENT

- Virtual Memory should permit most knowledge of physical memory characteristics to be removed from both Operating System and user programs.

PROTECTION ENVIRONMENT

- The fundamental protection unit is a segment. The segment is an extension of the notion of a file, and can be used to address the active portion of the file base. The segment descriptor entry provides basic read, write, and execute control.
- Two additional descriptors are defined which allow finer protection control:

  a. code binding section for known entry points (hardware controlled)

  b. data binding section for known table entries (software controlled)

  Both of these descriptors contain type information, access control information and pointers to data.

NAMING ENVIRONMENT

- If users and the system are to communicate, share, and control access to information they must use consistent naming methods.
- The active (open files, library lists, etc.) environment will be described via LNS descriptors. LNS segments can exist at the system or job level, thus allowing known qualifications for known entities.
- The file system will provide a convention for uniquely identifying permanently catalogued files.

CODE GENERATION

- Sharing implies the need for pure procedures which may be executed reentrantly.
- Pure procedures require code and data to be separated with different protection applied to each.
- Pure procedures imply that code segments and read only data segments do not need to be updated when memory is reassigned, thereby reducing I/O and cache clearing operations.

2.1.3.1 3300 IMPLEMENTATION

The multiple monitor concept was supported in the form of tasking on the CDC 3300 Master system. The system-supported envelope (task) could contain user or system code. Formal interfaces existed for call/return (synchronous and asynchronous), communication and identification purposes.

The "OS Task" typically had access to parts of the O.S. data base which had to be protected from the user. The 3300 system mechanized this interface by maintaining formal call/return links, switching page tables (for protection), keeping access information as a task attribute, and serializing the entire 'procedure set' when working on shared data.

The Task interface was used by the subsystem writer when adding new functions to the system. This is a valid solution and will exist as one type of interface in IPL O.S. but contains too much overhead for frequently used services such as get/put.

2.1.3.2 PL IMPLEMENTATION

The PL Operating System provided an additional interface for synchronous requests which solved some of the overhead problems:

- All code is shared and serialization occurs on data rather than code.
- The 'procedure set' (Task Monitor) resides in each user's space, thus simplifying communication between the system and the user.
- A call/return/parameter mechanism was defined for the system and user, making system code and user code compatible.
- The hardware distinguishes between local (traps) and global (interrupts) fault conditions, thus reducing the number of trips to a central monitor.
- The system provides software segmentation on top of a hardware-supported linear virtual memory with paging.

The Task Monitor interface provides a job mode interface through which all requests to the system and responses from the system are passed. Some resulting benefits are listed below.

- Requests/response processing can initially (in some cases, entirely) proceed as a part of the requesting task, thus increasing the potential for concurrent processing.
- Accounting is simpler since no context switch has occurred.
- When requests which require resources ask for them while executing as a part of the requesting task, rejection is simpler since remembering who the request is for is inherent (current control point).
- Simpler recovery and consistency codes can be implemented since they will execute in the same environment as the requestor.
- It is easier for the system to use itself resulting in a more compact and better debugged system.
- Increased predictability - most systems have formal

conventions for passing information out of an address space but relax that formality when returning status or setting completion indicators. The Task Monitor interface requires formal conventions to be followed when passing control information into the address space as well.

Since most of this interface was software enforced (dedicated addresses for user/system area, dedicated entry point, passwords, etc.) it would have been very difficult to allow subsysem writers to add additional protected procedure sets to the system.

This type of interface will exist in the IPL Operating System (task services) and will be generalized to allow subsystem writers to introduce a similar interface into a users address space, thereby changing the services available to that address space.

2.1.3.3 IPL IMPLEMENTATION

A very positive characteristic of the IPL implementation is the hardware enforcement which allows the Task Monitor interface to be externalized to the subystem writer. The following section reviews the advantages, the IPL hardware support and how the Operating System intends to use this interface.

ADVANTAGES AND HARDWARE

Performance

Calls to protected procedures use the same structural mechanism (Call/Return instructions) as calls to unprotected procedures with the same cost in execution time. Thus, a programmer does not need to consider whether he is calling a protected procedure when estimating performance on new designs.

Reliability

Information in the storage system (online mass storage) can be read and written by mapping it into virtual memory, and then using load and store instructions whose validity is checked by the descriptor mechanism. In addition, the addressing privileges of the current protected procedure are governed by its identification, which is located in the descriptor of the segment which supplied the most recent instruction. Every transfer of control to a different procedure is thus guaranteed to automatically produce valid addressing.

Flexibility

If virtual memory is used, a program can be moved more easily to another environment within the system.

Resource Management

Management of resources at a local level (via traps, local clock, local and global segment attributes) reduces the number of calls to the central monitor, thus reducing conflicts and increasing concurrency and responsiveness.

Many system procedures called in a string of references set interlocks or record partial results. If these conditions are not restored properly in case of failure, continued

system operation may not be possible. The dynamic stack represents an efficient way to record and recover these conditions.

Since the system can use itself recursively (with help of the Dynamic Stack and Call/Return instructions), duplication of function is avoided resulting in a more compact system.

2.2 BASIC CONCEPTS

2.2.1 SEGMENTS

An underlying concept of the IPL system is a segmented virtual address space. The address space contains all the bytes directly addressable by the user. Segmentation organizes the address space into two dimensions. That is, all addresses are specified by two components (N,i) where N is a segment number and i is a byte number within N.

The principle benefits of segmentation are:

a. Convenient presentation of very large address space to the user.

b. Different access attributes can be defined for different segments.

c. Procedure and data segments may be shared.

d. Each segment within the address space can be independently, dynamically expanded.

The IPL virtual address space provides a range of addresses considerably larger than the real memory available in the computer. A combination of software and hardware is responsible for mapping the virtual address space into real memory. The technique used to perform this mapping is called paging. That is, real memory is divided into uniform units called page frames and segments are divided into units of matching size called pages. Only pages which are required at a given point in time need occupy page frames in real memory.

2.2.2 LOGICAL NAME SPACE (LNS)

The IPL Operating System is dependent on the use of tables to provide interfaces between different system modules as well as between the System and the User.

Most systems provide methods for the definition, access and allocation/freeing of tables. This interface is typically optimized as a module to module interface. IPL provides an additional facility which is optimized as a human (Module to User) interface and allows symbolic access to table entries and/or items within the entry.

LNS began as a symbol table for the Command Language interpreter and grew to a general symbolic access method and table manager supporting simple and complex data types.

This evolution was based on the belief that the system would appear more consistent if descriptors for the basic objects of the system (File, Task, Job, Unit) were defined, and uniform actions applied to these descriptors (START, STOP, DISPLAY, STATUS) LNS-managed structures now represent a major portion of the environment for any user or Operating System module within the IPL system.

The LNS is composed of user and system supplied segments containing user and system defined entries. A list, called the LNS segment list, is maintained for each job known to the system. The LNS segment list contains the names of the segments which are to be searched for LNS entries and the order in which they are to be searched. When an LNS entry is sought, each segment whose name appears in the LNS segment list is searched until the entry has been found or the list has been exhausted.

LNS facilities for table handling should be used wherever feasible within the system giving a consistency to table structure and supporting the notion of a uniform set of requests which can be applied to system descriptors.

2.2.3 JOBS

The job is the mechanism through which the batch or interactive user interfaces to the IPL system. One Job identifies one user to the system, owns and is responsible for resources. The Job environment provides a system supported accounting, operator communication, logging and recovery envelope for all work performed in the system. Every Job consists of one segmented address space which is initialized with several system supplied code and data segments.

The normal mode of operation of the IPL Operating System is for all jobs known to the system to be in some sense active. User validation and command language interpretation are not done until a job has been established and has executed for some period of time. Thus, very little is known about a job before it is executing. Job scheduling is designed to accomodate this mode of operation, where jobs must be preempted as they express requirements for resources and resumed as the resources become available.

A job may grant other jobs direct or indirect access to a resource it owns.

DIRECT: This method has one globally held descriptor which is pointed to by descriptors held locally to the users of the resource. A code segment is an example of a resource shared via this method.

INDIRECT: This method allows a user to access a procedure or procedure set which operates on behalf of the resource owner. A disk is an example of a resource shared via this method.

The Operating System contains a System Job which has both private resources and resources it shares with other jobs. Task Services is an example of a procedure set which can execute on behalf of its caller (User Job) or its owner (System Job). It is intended that the subsystem writer use some of the same resource management and control methods used by the System Job.

IPL Jobs may communicate (via LNS or Signals) and may share segments. These capabilities, together with the ability for a Job to submit other jobs, allow for the solution of complex problems using a simple, well understood construct (the Job). Job sequencing may be accomplished by using these standard job communication facilities.

2.2.4 TASKS

Every system defines and supports a fundamental unit of work which can be uniquely:

- o Identified
- o Accounted
- o Scheduled
- o Allocated/Dispatched
- o Created/Destroyed

The objective is to have a consistent work unit which the Operating System can effectively multiplex among the processors available within the configuration. For some problems this unit can be the "Job" (multiple dependent Jobs). For other problems the level of efficiency for communicating, creating/destroying, etc. must be much higher in order to use multiple processors effectively. One way to increase efficiency and intimacy between work units is to increase what is common between them (sharing). IPL/OS allows varying degrees of sharing between work units (CODE, COMMON, INTERNAL STATIC). This fundamental work unit within IPL/OS is the task.

2.2.5 FILES

A file may be thought of as a named repository for information external to the program that is using the file. File management procedures allow users to exercise selectable amounts of control over file characteristics, data organization and mapping onto elements of the hardware configuration. For example; one file may consist of a specified region of storage on a specified mass storage volume set and have a specified data organization, a second file may consist of a specified interactive terminal operating with user-selected data delimiters, while a third file may consist of a virtual memory segment that is directly addressed by the user's program.

File descriptions are initially constructed in LNS tables by a DEFINE_FILE request. Subsequent file management requests obtain the majority of their parameters from the LNS file descriptions. Although all file management requests may be issued from a running program, a more typical situation will be one in which the file definitions are supplied through command language requests and only the OPEN_FILE/CLOSE_FILE requests are issued from the user's program. This permits the program to remain considerably less dependent on hardware type or file organization than if it generated its own file descriptions.

Descriptions of permanent files are recorded in catalogs associated with the storage on which the files are contained. Mass storage volume sets contain a catalog of their files and available/assigned space in order to permit them to be easily transportable among IPL sites. Access control lists allow the owners of permanent files to selectively grant access to other users or groups of users. Depending on results of current design work, the owner of a permanent file may also be able to restrict all access requests to be issued through a specified program.

At the time a file is opened for processing, file management procedures establish linkage between the user's program and the file access procedure (FAP) appropriate for the file organization and hardware type.

2.2.6 FILE ACCESS PROCEDURES

File Access Procedures (FAP's) logically reside between the user program and a file and provide a standard set of requests for accessing data in a file. With one exception, the linkage between a user program and a FAP is established when a file is opened and severed when a file is closed. The single exception occurs when a file is opened for implicit access. In this case, the user is returned a segment identifier through which the file can be accessed directly with machine instructions - no FAP exists.

Standard file access procedures are defined for both block-level and record-level access to a file. Block-level access procedures are used primarily by record-level access procedures, although they are available to other programs which need access to all the bits within blocks of data.

Standard record-level FAPs are intended for use by the Operating System and other products and should be used to process any file that is a candidate to be processed by another program. While this will not guarantee that the contents of the records will be intelligible to other programs, it will guarantee that the records conform to standard delimiting rules and can be located within larger strings of bits.

Non-standard FAPs are the subject of current design work and are intended to assist IPL programs in processing non-standard data formats or file organizations. The general intent is that a non-standard FAP can be written to process a non-standard file (a Cyber or Century tape file, for example) using a standard set of IPL record-level access requests. The OPEN_FILE processor will establish linkage to the non-standard FAP by recognizing a non-standard LNS file description.

Notable characteristics of standard record-level File Access Procedures are listed below:

- they support the file organizations and access requests required by ANSI standard programming languages.
- where practical, they support additional file organizations and access requests needed by the Operating System and other products.
- within reasonable limits, they provide an interface for sequential record access that is independent of file organization or hardware type.
- they insulate the user program from the buffering technique applied to a file and support both explicit and implicit access to mass storage files.
- current design work is aimed at supporting concurrent access by multiple writers of a single file opened for shared update.

2.2.7 SECURITY

Interest in security is currently at a high point in the "presentation" circles (analogous to structured programming the last few years) but it is difficult for any implementation to separate what is fundamental in security from complete solutions to security. The Operating System objective is to supply a set of interfaces and conventions upon which various security problems can be solved.

Although protection/security concepts are not yet formulated within the Operating System, a report written by TRW on secure Operating Systems is being used to test what is required to support security.

The following security related topics are being explored in implementation:

- o Extending data access control beyond traditional Read, Write, Execute and Append categories. For example, Permit Write access but only through a specified procedure.
- o Constant review of absolute identification of requestor within the system (non-forgible id) to be sure this is possible.
- o Constant review of simplifying the user interface with the objective of making it certifiable. (This objective leads to reducing the asynchronous action within an address space).

2.3 CODE STRUCTURE

The Operating System supports three basic environments within which O.S. services may be implemented:

System Monitor (one per system)

Task Services (within each job)

System Tasks (within the System Job)

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

System Monitor - That portion of the Operating System that is most directly related to the hardware environment and provides:

- o Directing signals
- o CPU Dispatching
- o Basic CPU Scheduling
- o Changing Task Status
- o Interrupt Handling

System Monitor is interrupt driven, nonpageable, and will represent the most thoroughly debugged, least frequently changed code within the Operating System.

2.0 STRUCTURE
2.3 CODE STRUCTURE

Task Services - That portion of the Operating System that is most directly related to the requestor's environment and provides major portions of:

- o Record Management
- o File Management
- o Program Management
- o Storage Management
- o Request Recognition/Routing

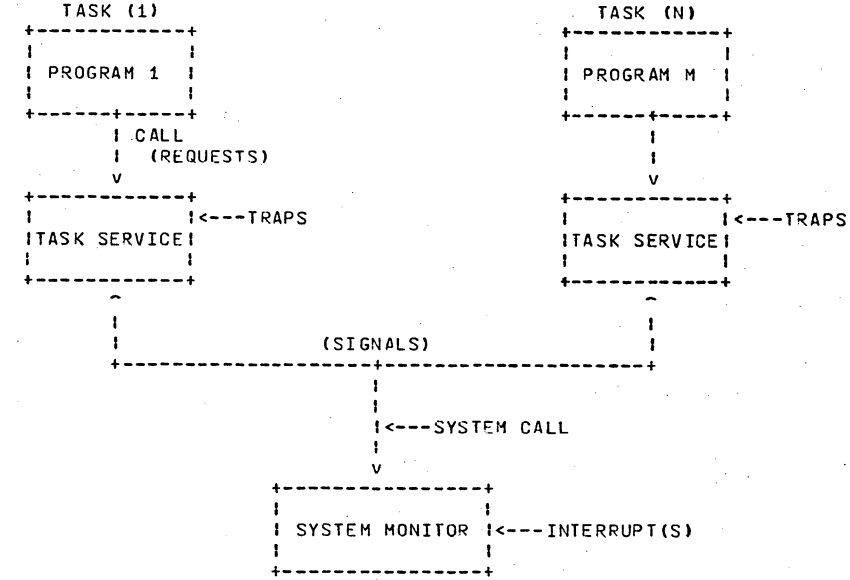
Task Services is always called via a standard call, is paged, and has the same (clock) accounting, scheduling and execution characteristics as the requestor.

System Tasks - That portion of the Operating System that is relatively independent of the requestor's environment, may be asynchronous to the requestor and provides major portions of:

- o Job Management
- o Operator Communications
- o Block Management
- o Storage Management
- o Scheduling

Each execution of a program is defined to be a task. The majority of these executions will be triggered by a signal. Each task has its own (clock) accounting, scheduling and execution characteristics. All Operating System tasks belong to the System Job.

2.0 STRUCTURE
2.3 CODE STRUCTURE



2.3.1 SYSTEM MONITOR

System Monitor has its own address space and is divided into two major components, the monitor mode Nucleus and the job mode Overseer.

Nucleus is the only system element that executes in monitor mode and cannot tolerate a page missing interrupt. It is entered via hardware interrupt (i.e., system call, external interrupt, access violation, monitor interval timer, etc). All the processors in the system share the Nucleus code and its one Segment Descriptor Table. However, each processor executing Nucleus has its own control point, signal buffer, stack, and state registers. When a system call is made to Nucleus, parameters are passed via the register image of the caller.

There are two control points executing Overseer code in job mode. One is signal driven and can tolerate page missing interrupts and wait conditions. This control point performs the following functions:

- Job scheduling
- Signal buffering
- Deadstart
- Recovery and tracing

The other control point is both signal driven and dispatch driven (i.e., it is periodically dispatched). It cannot tolerate page missing interrupts. This control point performs the following functions:

- Error monitoring
- Wired table management

Parameters are passed to each of these control points via signals.

2.3.2 TASK SERVICES

Task Services is a set of procedures which provide Operating System services. These procedures are directly callable by user code, no exchange jump or supervisor call is required. The call instruction can, however, cause a change in privilege for the called procedure, allowing it to execute with more or different privileges than the calling procedures. This type of structure allows much of the Operating System to execute within the user environment. All of Task Services has the same (clock) accounting, scheduling, and execution characteristics as the requestor. The only difference is access rights to data and code.

Task Services provide a central point for the handling of all requests and responses made/received by a Task. If the service requested is not supported by Task Services, the request is passed on to System Monitor or an Operating System Task.

Task Services occupies two rings (2,3) within each address space. The lower ring is used by Task Monitor procedures, which will have access to the signals for this address space. Task Monitor will move signals out of the signal buffer into an area accessable by Task Services. The entry to Task Monitor is callable from Task Services only.

In Version 1.0 there is one protected entry point (OS#GATE) into Task Services. This allows monitoring of system requests and responses. In later versions, based on monitoring results, additional protected entry points may be provided. Existing programs will still operate, but recompilation will be required to use the new protected entry points.

One gate into Task Services will be used exclusively for signal processing. A trap procedure will exist in every job at the user level (ring). This procedure will, based on trap type, call Task Services (signal, clock ... etc.) or a user supplied procedure (overflow ... etc.).

Since each address space (job) may contain a set of request processors which belong to a subsystem or System Job, a means of identifying on whose behalf a request is made must exist. Rings will be used for this purpose. Every request processor, when executing, will have a current validation level (ring) which must be correctly maintained by the request processor.

Task Services code may execute either as an extension of the requestor or the System. When a request is made, Task Services is immediately executing at its most privileged level, that of the System. Task Services may use the ring of validation of the requestor to access information in an area specified by the requestor, thus guaranteeing the requestor has legitimate access to that area.

Note that rings are a global resource such that a ring level in every user address space has the same privilege and belongs to either the System Job or a Subsystem Job.

Setting the validation level also sets which segment (LNS Job Segment) is to be used when searching for descriptors for the request made (i.e., System Local tables of the System Job will be used when validation level is that of Task Services).

2.3.2.1 Request Handling

Task Services is called via a standard call instruction and requires two parameters on entry; one includes a request code and a pointer to a request block (input), the other is an output parameter which reflects the request status (output). The request block is generated by a macro and may contain variables which point to additional parameters. The binding of parameter values may occur at Compile, Load or Execution time. Since the request block is moved into the registers before System Monitor is called, the Request Block size is limited.

All requests in the system (inside and outside the address space) are assigned a unique code which is used to obtain the desired request processor entry. Task Services has a private binding section which contains the entry points (code base pointer) for all request processors. The R3 field of each code base pointer will determine who can call the associated request processor. Task Services will fabricate an offset into this binding section using the request code and the callers ring of validation. This offset will be used in providing a check on the callers access to the desired request processor.

The decision to pass a request beyond Task Services (System Monitor) is not made by the routing mechanism but by the target request processor such that the same control over entry exists irrespective of where implemented.

2.3.2.2 Signal Handling

The Operating System provides a basic signal handling service. The signals have a fixed format, maximum size and are used by the Operating System primarily for communication between address spaces. Every subtask has a signal buffer (segment 0) which is used for signal queueing. System Monitor is responsible for placing signals into the proper signal buffer and for notifying the proper Task Monitor that a signal exists. Task Monitor is responsible for taking signals out of a signal buffer and passing it to a Task Services signal handler. Routing, based on signal type, to a signal processor within Task Services will be effected by the Signal Handler.

Task Services will provide a send request processor which will build a signal supplying the destination, validation level, signal type, and selection information. The send request processor will then do a system call to System Monitor.

2.3.2.3 Major Functions

The major functions of the Task Services procedures are as follows:

Record Management

- Providing access to individual logical records in previously opened files.
- Providing an interface which allows programs to be reasonably independent of file organization, buffering and device characteristics.
- Providing exit points to user-supplied application procedures.
- Performing automatic blocking, deblocking, buffering and index management.

File Management

- Supervising the formatting, labelling and cataloging of all peripheral devices.
- Managing the creation, deletion, labelling, allocation and cataloging of files held on mass storage devices.
- Performing open and close operations to logically attach and detach files to programs. These files may exist on any local or remote peripheral device.
- Verifying that access privilege and privacy rules specified by file owners are followed by all file users.

Operator Communications (Job Interface)

The Operator Communication requests provide for message communication between a Job and one of seven logical operators (tape operator, customer engineer, etc.). If the operator with which a job wishes to converse is not logged into the system, the information will be passed to/from an alternate.

Logical Name Space Management

- Maintaining the LNS search list via the ATTACH and DETACH requests.
- Declaring and removing simple and complex data types.
- Assigning values into the data types.

- o Defining new complex types (control blocks) to the system.

Program Communications

Program Communications is a set of procedures which provide various mechanisms to permit communication and synchronization among various parts of programs and among various tasks in the same job. A mechanism is also provided for communication among tasks in different jobs. These mechanisms are:

- o Events. Events permit synchronization and interrupt control (Attach Interrupt Procedure) for asychronous activities within a job. An event is represented by an event control block in storage and several requests to manipulate the control block.
- o Signals. Signals are short interjob messages. Signals may be sent between arbitrary User Jobs and are also the mechanism used to permit a User Job to communicate with the System Job.
- o Queues. The queuing mechanism provided allows the sending, storing, and retrieving of arbitrary data structures between asynchronous activities within a job. A queue is represented by a queue control block in storage and several requests to manipulate the queue.
- o Semaphores. Semaphores permit communication and synchronization among asynchronous activities within a job. A semaphore is represented by a semaphore control block in storage and two requests to manipulate the control block. Semaphores are the most primitive facility supported by the Operating System for serialization and synchronization of asynchronous activities. The traditional locking functions may be accomplished via semaphores.
- o Signature Locks. Signature Lock(s) provide an externalizion of the compare/swap hardware instruction and is intended for synchronization between Jobs.

Program Execution

Program execution procedures support the establishment loading of programs and procedures and their execution. The execution of a program is termed a task while the execution of a procedure is termed a subtask. The Loader is considered part of program execution.

- o Loading programs and initiating tasks to execute programs and subtasks to asynchronously execute procedures of those programs.
- o Controlling tasks and subtasks through declaring, initializing, maintaining and removing control point structures.
- o Terminating tasks and subtasks normally and abnormally in an orderly fashion.
- o Constructing an object module segment, a working storage segment and a binding segment.
- o Maintaining a symbol table (Loader Symbol Table) of all currently known entries and externals in the program.

The Loader is responsible for the integrity of the binding segments which are among the more important aspects of protection in the system.

Device Scheduling

- o Allocation of individual peripherals to jobs.
- o Resolution of contention for devices by multiple jobs.
- o Overcommitment of devices while not permitting deadlock between jobs.

The device scheduler is a set of procedures in Task Services which works in close cooperation with the File Management procedures in Task Services and the Configuration Manager in the System Job.

Segment Control

- o Manage the allocation/release of space on mass storage devices in conjunction with file management.
- o Manage the allocation/release of segment numbers.
- o Organize the movement of global segment/file tables between memory and mass storage according to usage.
- o Organize page working-set sizes and move private segment/file tables to/from mass storage (swapping).
- o Maintain storage system restart information.
- o Assist Job Management with the creation of new and deactivation of old address spaces.

Stream Control

- o Supporting the connection or disconnection of files to

streams.

- Performing automatic broadcasting to all files connected to a stream on output.
- Allowing one file to be connected to N streams reducing amount of file resources required.

These services are externalized for output only and will be used by Job Management in creating the logging environment (dayfile, errors, accounting,...) for a Job.

System Table Management

To be supplied.

2.3.2.4 Request Summary

Operator Communication (Job Interface) Requests

CONVERSE	send a message to an operator and solicit a reply.
INFORM	send an information message to an operator.
RECEIVE	accept operator input.
SELECT	associate an event control block with the arrival of operator input.

Data Management Requests

(Record Management)

GET	retrieve next record
PUT	store next record
GETP	retrieve next partial record
PUTP	store next partial record
GETD	retrieve record by file address
GETKEY	retrieve record by key or ordinal
PUTKEY	store record by key or ordinal
DELETE	delete previous record
DELKEY	delete record by key or ordinal
DELETED	delete record by file address
REPLACE	replace previous record
FINDF	position to first record
FINDP	position to previous record
FINDKEY	position to key or ordinal
FINDD	position to file address
SETLOCK	set a record lock
CLEARLOCK	clear a record lock

(File Management)

DEFINE_FILE	create a new file control block
CREATE_FILE	allocate mass storage space for a temporary or permanent file
EXPAND_FILE	allocate additional mass storage space for an existing temporary or permanent file
CONTRACT_FILE	release part or all the mass storage space allocated to temporary or permanent file
SAVE_FILE	convert a temporary mass storage file to a permanent file
DELETE_FILE	delete a temporary or permanent file from the system
ATTACH_FILE	attach permanent file to a job
DETACH_FILE	detach a permanent file from current job

2.0 STRUCTURE
2.3.2.4 Request Summary

REDEFINE_FILE	redefine some of the logical characteristics of an existing permanent mass storage file	1 2
OPEN_FILE	establish a logical connection between a file and the current program	3 4
CLOSE_FILE	sever the logical connection between a file and the current program	5 6
CLOSE_VOLUME	close current volume of a tape file and open new volume	7 8
PERMIT	gives or modifies access(x) to file(y) for user(z)	9 10
PROHIBIT	removes access(x) to file(y) for user(z)	11 12
		13
(Block Management)		14
READ	read next block	15
READ_DIRECT	read block direct	16
READ_STATUS	read external status	17
WRITE	write next block	18
WRITE_DIRECT	write block direct	19
POINT_FIRST	position to first data block	20
POINT_LAST	position to last data block	21
POINT_PRECEDING	position to preceding data block	22
POINT_NEXT	position to next data block	23
DEVICE_CONTROL	operate external device	24
CHECK	check for IO response	25
ASSIGN_BUFFER		26
RELEASE_BUFFER		27
		28
		29
Program Management Requests		30
		31
(Logical Name Space)		32
LNS_ATTACH	add a new LNS segment	33
LNS_DETACH	remove an LNS segment	34
LNS_DECLARE	create an LNS entry	35
LNS_REMOVE	delete an LNS entry	36
LNS_ENTRY	get descriptor of an LNS entry	37
LNS_NEXT	get descriptor of an LNS item or field	38
LNS_SLICE	get descriptor of an LNS array element	39
LNS_GROW	expand an LNS entry or item	40
LNS_LOCK	lock an LNS entry or item	41
LNS_UNLOCK	unlock an LNS entry or item	42
LNS_INSERT	insert a new LNS item	43
LNS_DELETE	delete an LNS item	44
LNS_GET	get the value of an LNS item	45
LNS_PUT	set the value of an LNS item	46
LNS_ATTRIBUTES	set extrinsic attributes of LNS entry or item	47
LNS_RECORD	define a new LNS complex type	48

2.0 STRUCTURE
2.3.2.4 Request Summary

LNS_FIELD	define a field of a complex type	1
LNS_LOCK_SEGMENT	lock an LNS segment	2
LNS_UNLOCK_SEGMENT	unlock an LNS segment	3 4
		5
(Program Communications)		6
ATTACH_PROCEDURE	associate interrupt procedure with an event	7
DETACH_PROCEDURE	remove interrupt procedure/event association	8
STATUS_EVENT	return the status of an event	9
CAUSE_EVENT	set event to caused and initiate event action	10
CAUSE_CLEAR_EVENT	initiate event action and leave event cleared	11
CLEAR_EVENT	set event to cleared	12
DISABLE_EVENT	prohibit indicated event action	13
ENABLE_EVENT	allow indicated event action	14
WAIT_EVENT	suspend control point until one or all events are caused	15 16
WAIT_CLEAR_EVENT	suspend control point until one or all events are caused and set events to cleared	17 18
SEND_SIGNAL	send a signal to a job	19
SELECT_SIGNAL	prepare to receive indicated signal	20
DESELECT_SIGNAL	discontinue reception of indicated signals	21
STATUS_SIGNAL	determine if indicated signal has arrived	22
DISABLE_SIGNALS	disable signal processing	23
ENABLE_SIGNALS	enable signal processing	24
IDENTITY	returns execution identity of requestor.	25
ENQUEUE	add item to a queue	26
DEQUEUE	remove the first item on the queue	27
STATUS_QUEUE	determine if any items are on a queue	28
SIGNAL_SEMAPHORE	increment semaphore value and allow one waiting control point to continue	29 30
WAIT_SEMAPHORE	decrement semaphore value and suspend control point if indicated	31 32
SIGN_LOCK	sign a signature lock with the control point id of the requestor	33 34
UNSIGN_LOCK	unsign signature lock by writing with zeroes	35 36
		37
(Program Execution)		38
EXECUTE	Load and asynchronously execute on program	39
EXIT	Indicates task completion	40
TERMINATE	used by a task to terminate another task	41
SPAWN	start an asynchronous execution of a Subtask	42
LOAD	load procedure not yet referenced in program	43
ENTRY	retrieve an existing entry point value	44
REINITIALIZE	to support Cobol cancel	45
ESTABLISH	establish a program (subsystem services) within a job	46 47
DISESTABLISH	remove an established program	48

2.0 STRUCTURE
2.3.2.4 Request Summary

Storage Management Requests		
(Segment Control)		
INITIATE_SEGMENT	define a segment and assign segment number	
TERMINATE_SEGMENT	remove a file/segment definition	
MAP_IN	allows indirect updating of a file/segment	
MAP_OUT	moves updated pages to primary file/segment	
SET_MAX	set maximum segment length	
RELEASE_UNUSED	release unused portion of a segment	
EXPAND_SEGMENT	expand segment length	
TRUNCATE_SEGMENT	truncate segment length	
(Page Control)		
ADVISE_IN	transfer N consecutive pages to memory	
ADVISE_OUT	transfer N consecutive pages to mass storage	
SET_USAGE_LEVEL	give virtual memory usage advice to system	
LOCK_PVA	suspend paging on N consecutive pages	
UNLOCK_PVA	restart paging on N consecutive pages	
FIX_PAGE	allocate a contiguous section of real memory and associate a virtual address range	
RELEASE_PAGE	return a contiguous real memory section allocated by a previous FIX_PAGE	
Miscellaneous Requests		
STREAM_CONNECT	connect a file to a stream	
STREAM_DISCONNECT	disconnect a file from a stream	
STREAM_GET	input a record from a stream	
STREAM_PUT	output a record to a stream	
Job Management Requests		
SUBMIT	initiates the establishment of another job	
DIRECT	establishes destination for file to be routed	
ROUTE	initiates the transfer of a file to some specified destination	
SYSLOG	transfer information (accounting, system error and dayfile) to the system log file	
CLAIM	set maximum usage of a class of devices for a job	
CHANGE_CLAIM	increase/decrease number of units of a class	
RESERVE	register the requirement for a non-preemptible resource (file, volume set, unit set)	
CANCEL_RESERVE	cancel all previous reserve requests	
ACQUIRE	satisfy all previously executed reserve requests	

2.0 STRUCTURE
2.3.2.4 Request Summary

RETURN	return a file, volume set or unit set previously allocated to a job via reserve/acquire
SET_PRIORITY	establish/change base priority of a job
SET_CLASS	change the class of a job
GET_CAT	get a copy of a record of the class attribute table
REPLACE_CAT	replace the cat attribute table record
GET_CPLT	get copy of class priority level table record
REPLACE_CPLT	replace a class priority level table record
GET_CTT	get a copy of a class transition table record
REPLACE_CTT	replace a class transition table record
GET_SCT	get a copy of the scheduler control table
REPLACE_SCT	replace the current scheduler control table
System Table Requests	
	To be supplied.
System Monitor Requests	
CREATE_ADDRESS_SPACE	create and initialize a new address space
REMOVE_ADDRESS_SPACE	remove an existing address space
DEFER_ADDRESS_SPACE	transfer a job's working set and tables to the jobs swap segment
RUN_ADDRESS_SPACE	retrieve a Job's working set and tables from the Job's swap segment
GET_RJOT_ENTRY	retrieve a running job ordinal entry
PUT_RJOT_ENTRY	update the fields of a running job ordinal entry
CREATE_CP	reserve memory space and initialize a new control point
REMOVE_CP	terminate outstanding signals deallocate stack segements and free control point occupied memory
CHANGE_EXCH_PACK	set specified field within the exchange package of a control point
STATUS_CP	retrieve a control point's current status
CHANGE_CP_STATUS	change a control point's dispatch state
DEACTIVATE_CP	force a running or ready control point into halted state
ACTIVATE_CP	remove halt state to make control point ready
ENQUEUE_CP	place one control point into another control point's action queue
DEQUEUE_CP	remove one or more control points from an action queue
SELECT_INTERNAL_INT	specify the internal interrupts whose

	occurrence should be signalled
CANCEL_INTERNAL_INT	remove a previous selection
SELECT_REAL_TIME	signal control point after the elapse of a real time interval
CANCEL_REAL_TIME	remove a real time selection
MONITOR_CP	permit one control point to monitor a second control point

2.3.3 SYSTEM TASKS

Many Operating System facilities are provided by programs which operate relatively independently of the user. These programs are executed as a set of tasks. All but one of these tasks exist in the address space of a System Job. The exception is the Sequence Monitor Task. There is an instance of this task in every job. The System Job is very much like any User Job with one additional characteristic: Task Services/Task Monitor is a program in every job which belongs to the System Job and has the same privileges and shares the responsibilities of the System Job.

2.3.3.1 Tasks in Every Job

2.3.3.1.1 SEQUENCE MONITOR TASK

Sequence Monitor gains control as a result of a job being placed into execution for the first time. Main control of Sequence Monitor issues internal procedure calls to carry out the functions of job initiation, command language statement processing, user validation, job termination and some parts of operator communication.

Job Initiation Procedures

The job initiation procedures define standard job streams, create standard job files, perform default file to stream connections and additional functions as specifications of job structure and environment evolve.

Command Language Interpreter Procedures

The command language interpreter procedures perform the functions of login processing, user validation, and command language statement processing.

Job Termination Procedures

The job termination procedures perform the following functions:

- Close all files in the job.
- Release all temporary files in the job.
- Route all files in the job which have been directed

previously via DIRECT_FILE requests.
- o Remove LNS variables declared for the job in System Global LNS.
- o Format job termination accounting record and request its transfer to the accounting stream.
- o Others to be defined.

Three job termination procedures may be invoked from procedures other than termination procedures:

- o One procedure is called from the main control procedure of Sequence Monitor as the results of Command Language Interpreter processing a LOGOUT.
- o One procedure is called from the Command Language Interpreter as the result of Command Language Interpreter encountering a second LOGIN.
- o One procedure is invoked as a result of a signal and event occurrence indicating an abnormal situation which is to result in job termination.

2.3.3.2 Tasks in the System Job

2.3.3.2.1 SYSTEM ACCESS MANAGER TASK

The System Access Manager performs the following functions:

- o Validate the newly active terminal.
- o If the terminal is a batch input device, the Stager is invoked to service the terminal.
- o If the terminal is an interactive device, the System Access Manager does the following:
  - Declares a Job Control Block in System Global LNS.
  - Sets the appropriate values into various fields of the JCB.
  - Declares a File Control Block for the interactive device in System Global LNS.
  - Invokes job establishment.
  - Continues to poll the System Input Device List for another terminal becoming active.

The System Access Manager Task is present at deadstart and continues to exist until shutdown.

2.3.3.2.2 STAGER SUBTASKS

The Stager performs the following functions:

- o Opens the batch input device.
- o Declares a File Control Block in System Global LNS for a disk resident file.
- o Creates this disk resident file as a permanent file.
- o Copies a logical input file from the batch input device to the disk file;
  - A logical input file consists of data which occurs between FENCE (system defined delimiter) records.
  - Stager must also recognize and treat HEDGE (system defined delimiter) records which may occur in the logical input file to delineate points for restart of staging.
- o Declares a Job Control Block in System Global LNS.
- o Sets appropriate values into various fields of the JCB.
- o Closes the disk resident file
- o Invokes job establishment
- o Formats a job staging report.
- o Continues declaring FCBs if additional logical input files exist on the batch input device.
- o If no additional logical input files exist, the Stager:
  - Closes the batch input device.
  - Outputs the accumulated job staging reports.
  - Terminates execution.

There is a Stager Subtask for each active batch input device. The number of Stager Subtasks will vary during the processing day. These executions are subtasks of the System Access Manager Task.

2.3.3.2.3 QUEUED JOB MONITOR TASK

The Queued Job Monitor performs the following functions:

- o Selects highest priority queued job from the Known Job List.
- o Invokes job establishment

It may be possible to eliminate the Queued Job Monitor. If comprehensive job swapping is provided, job establishment could unconditionally establish a batch job by creating a swap file image and then linking the swap file into the System Scheduler's Deferred Job List.

2.3.3.2.4 OPERATOR COMMUNICATIONS

Each physical operator for a site will be represented by an instance of the Sequence Monitor (includes the System Command Language interpreter). This will be accomplished by having a Job for each active physical operator. In the minimum case (one physical operator) the Sequence Monitor of the system job will be used.

2.3.3.2.5 JOB ESTABLISHER TASK

The Job Establisher performs the following functions:

- o Execution is activated by the reception of signals which originate from within the SUBMIT request processors of the System Job and User Jobs.
- o Determines whether current system imposed thresholds permit immediate establishment of another job.
  - . If immediate establishment is prohibited and the job currently being considered is either interactive or batch with no queue permission specified:
    - Issues a reject to the SUBMIT request processor from which the establishment request (i.e., signal) originated.
  - . If immediate establishment is prohibited and the job being considered is batch and has queue permission specified:
    - Constructs an entry for the job in the Known Job List and marks that entry to indicate it is 'not established'.
  - . If immediate establishment is permitted:
    - Constructs a swap file image for the job. This image will be provided by a system template which contains a Segment Descriptor Table image with standard system segments, and images of Control Points and tables which will be used to control initial and subsequent execution of Sequence Monitor in the job once it has been swapped into memory.
    - Constructs an entry for the job in the Known Job List and marks that entry to indicate 'established and swapped-out'. (Note: If the job is one for which a Known Job List entry marked as 'not established' already exists, that entry is remarked as 'established and swapped-out'). A link to the relevant swap file is placed into the Known Job List entry for the job.
    - Assures the activation of the System Scheduler by causing an event upon which this scheduler waits when

there are no requirements for its execution.

- o Obtains another establishment request, if one exists, from its signal queue, then begins the process again. If no request exists, Job Establishment waits for the reception of the next signal from a SUBMIT request processor.

The Job Establisher Task is present at system deadstart.

2.3.3.2.6 JOB COLLAPSER TASK

The Job Collapser performs the following functions:

- o Execution is activated by the arrival of signals which originate from within Job Termination procedures. These signals contain information which identifies jobs which are to be removed from the system.
- o Performs deallocation of job elements which are essential for the system's identification and control of a job. These elements, therefore, cannot be deallocated from within the job itself.
  - . Deallocates system segments assigned to the job.
  - . Releases job related tables.
  - . Rethreads tables where appropriate.
  - . Releases the swap file associated with the job.
  - . Removes the Known Job List entry associated with the job.
- o Causes an event to assure activation of System Scheduler.
- o Causes an event to assure activation of Queued Job Monitor.
- o Obtains another COLLAPSE request, if one exists, from its signal queue, then begins the process again. If no request exists, Job Collapser waits for the reception of the next signal from Job Termination procedures.

The Job Collapser Task is present at system deadstart.

2.3.3.2.7 FILE ROUTER TASK

The File Router performs the following functions:

- o Awakened by signals from the ROUTE request processor in this job and in User Jobs.
- o Determines identity of file to be routed and the desired routing destination from information accompanying the signal.
- o Determines whether output to the desired destination is currently active.
  - . If so, places name of file to be routed in a queue for

2.0 STRUCTURE
2.3.3.2.7 FILE ROUTER TASK

the relevant task of the Output Distributor.
. If not, invokes the execution of an Output Distributor task and specifies the identity of the destination and the identity of the file to be routed.

The File Router is present at system deadstart.

2.3.3.2.8 OUTPUT DISTRIBUTOR SUBTASKS

The Output Distributor performs the following functions:

- o Opens a System Output Device.
- o Transfers a file from disk storage to the System Output Device.
- o Closes the output device when no additional files are queued for transmission to the System Output Device currently being serviced.
- o Terminates execution.

There is an Output Distributor Subtask for each active System Output Device. These executions are subtasks of the File Router Task.

2.3.3.2.9 CONFIGURATION MANAGER

The Active Device Detector is a procedure of Configuration Manager and performs the following functions:

- o Detects hardware level signals which indicate that a previously inactive device has become active.
- o Associates a hardware signal with a device and places the device type and device identifier in the System Input Device List.
- o Causes an event or sends a signal to awaken System Access Manager.

The Configuration Manager operation environment will be supplied.

2.3.3.2.10 RUNNING JOB MONITOR

The Running Job Monitor performs the following functions:

- o Determines which active jobs should be moved to ready status based on time slice and whether the job is waiting or not.
- o Determines which ready jobs should be moved to active status based on time slice, whether the job is waiting or

2.0 STRUCTURE
2.3.3.2.10 RUNNING JOB MONITOR

not, availability of enough real memory for the working set of the job, and the number of already active jobs.
- o Performs the state changes for the appropriate jobs.

The Running Job Monitor Task is present at system deadstart.

2.3.3.2.11 DEFERRED JOB MONITOR

The Deferred Job Monitor performs the following functions:

- o Determines which jobs should be swapped out based on relative priorities, time slices and job status.
- o Swaps out appropriate jobs. Swapping out involves collecting together all system tables about the job and writing them to the swap file, deallocating all nonshared active segments, and changing the job status in the Known Job List.
- o Determines which jobs that are swapped out should be swapped in based on relative priorities, job status, and availability of system resources.
- o Swaps appropriate jobs which involves reallocating system tables and active segments.

The Deferred Job Monitor Task is present at system deadstart.

2.3.3.2.12 BLOCK MANAGER TASK

Page Control

Page control is responsible for the management of real memory according to usage and scheduling requirements and performs the following functions:

- o Maintains the status of all pages in memory and performs requests involving the allocation, locking and fixing of memory page frames.
- o Performs all requests for the movement of pages from immediate access memory to external mass storage and back in segment dependent block size, which must be multiple of page frame size.
- o Maintains job working set size for scheduling and swap control.
- o Controls memory occupancy based on usage and scheduling requirements.
- o Performs second level error processing in conjunction with Block Management.

2.0 STRUCTURE
2.3.3.2.12 BLOCK MANAGER TASK

The Page Control procedures are a part of the Block Management program.

Block Management

- o Controlling the movement of data blocks between buffers or segments and previously opened files on peripherals.
- o Performing second level error recovery for peripheral errors that cannot be handled by controllers or channels.
- o Managing buffer assignment within virtual memory for explicitly referenced files.

2.3.3.2.13 INPUT OUTPUT DRIVERS

To be supplied.

2.0 STRUCTURE
2.3.4 SYSTEM EXTENSIONS

2.3.4 SYSTEM EXTENSIONS

System Job/Task Services

System extension techniques analogous to today's systems will be available via system generation. System generation will be used when extending task services and/or the system job. The formal interfaces and conventions (OS#GATE, signals, LNS, SWL, standard file formats...) defined and used by the intitial operating system will make extension easier.

Subsystems

A fundamental concept of IPLOS is the separation of operating system code into two parts: the part that runs outside the user address space (System Job) and the part that runs inside the user address space (Task Services). Task Services are protected from the user by being in a lower, more privileged ring of protection. This allows task services to be directly called by the user, to pass parameters by reference, and in general, accrue all the benefits of a common addressing context, while retaining protection and integrity. The operating system is not the only code for which such a separation is desirable.

Subsystem Services

Subsystem services also have the need to be rapidly and conveniently accessed by their users as well as being protected from them. They also share the users need of utilizing task services so in that regard, they appear to task services as being just another user from which it must be protected and to which it must be readily accessible. Subsystem Services then may be viewed as a generalization of the address space relationship between task services and the user. For this reason, the hierarchical aspect of rings of protection applies directly, since there is a natural hierarchy between the sytem, subsystem and user.

The file system in supporting FAP's will be the initial user of subsystem techniques and will provide the initial example of the creation, establishment, calling, execution environment, and termination of subsystem code. As the operating system and data base management projects progress, the requirements for support of subsystems will become better defined.

2.4 DATA STRUCTURES

2.4.1 INTRODUCTION

SYSTEM TABLES

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

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

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

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

- Addressing - Should the information be symbolically or directly addressable? When the user wishes to perform program control and data modification functions, he must address the system symbolically (Command Language).

Therefore, any features which are externalized via the Command Language must be supported by symbolically addressed tables. Throughout the system, this group of tables is referred to as the Logical Name Space (LNS).

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

- Crash Resistance - When the System crashes, how will the tables be reconstructed? What impact will there be on recovery if the tables cannot be reconstructed? Will the corrupting of the tables cause a System crash? What protection will be provided to detect corruption?

2.0 STRUCTURE
2.4.1 INTRODUCTION

The Operating System table structures are contained in several segments which have differing combinations of the attributes of protection, scope, residence, addressing and life cycle mentioned before. Every request made by a user will result in action against structures contained in the supported environments.

The attributes listed are available in some segments as detailed later. Not all combinations of attributes are necessary or provided.

- o Protection attributes are:
  - directly readable (DR)
  - directly writable (DW)
  - interpretively readable through LNS (IR)
  - interpretively writable through LNS (IW)
- o Scope attributes are:
  - by the OS in the System Job (SJ)
  - by the OS in Task Services in User Job (TS)
  - by the user program (USER)
- o Residence attributes are:
  - pageable (P)
  - fixed, which implies that some but not necessarily all of the segment may be fixed (F)
  - swappable (SWAP)
- o Addressing attributes are:
  - direct (DA)
  - symbolically through LNS (SYM)
- o Lifecycle attributes are:
  - for the duration of the system day (SYS)
  - for the duration of the job (JOB)
  - for the duration of the task (TASK)

There are several segments containing system tables which are always present and have known attributes. These are described below. User and System Tasks may have LNS, working storage, and stack segments with varying attributes which may contain some system tables. This will be noted in the descriptions of the individual table structures. The segments shown in the following table are always supported and have the indicated attributes.

	SEGMENT	PROTECTION FOR			RESIDENCE	ADDRESSING BY			LIFE CYCLE
		SJ	TS	USER		SJ	TS	USER	
0	SIGNAL BUFFERS	DR DW	DR DW		F SWAP		DA		SYS
1									
2									
3	SEGMENT DESCRIPTOR TABLES	DR	DR	DR	F SWAP		DA	DA	SYS
4	CONTROL POINT BUFFERS				F SWAP				SYS
5	LNS SYSTEM GLOBAL	DR DW IR IW	DR DW IR IW	IR	F	DA SYM	DA SYM	SYM	SYS
6	LNS USER LOCAL		DR DW IR IW	DR DW IR IW	P SWAP		DA SYM	DA SYM	TASK
7	LNS SYSTEM LOCAL		DR DW IR IW	IR IW	P SWAP		DA SYM	SYM	JOB
8	LNS USER GLOBAL	DR DW IR IW	DR DW IR IW	IR IW	P	DA SYM	DA SYM	SYM	SYS
9	LNS SYS LOCAL OF SYS JOB	DR DW IR IW	IR IW		F	DA SYM	SYM		SYS
10									
11	TASK MONITOR WORKING STORE		DR DW		P SWAP		DA		JOB
12									
13	TASK SERVICES WORKING STORE		DR DW		P SWAP		DA		JOB
	ADDITIONAL USER DATA SEGS, INCLUDING STACK AND WORKING STORAGE		DR DW	DR DW	P SWAP		DA	DA	JOB TASK
	ADDITIONAL LNS SEGMENTS		DR DW IR IW	DR DW IR IW	P SWAP		DA SYM	DA SYM	JOB TASK

Figure 2.4.1-1
SEGMENT ATTRIBUTES

NCR/CDC PRIVATE REV 30 MAY 75

2.0 STRUCTURE
2.4.1 INTRODUCTION

The Operating System table structures are contained in several segments which have differing combinations of the attributes of protection, scope, residence, addressing and life cycle mentioned before. Every request made by a user will result in action against structures contained in the supported environments.

The attributes listed are available in some segments as detailed later. Not all combinations of attributes are necessary or provided.

- Protection attributes are:
  - directly readable (DR)
  - directly writable (DW)
  - interpretively readable through LNS (IR)
  - interpretively writable through LNS (IW)
- Scope attributes are:
  - by the OS in the System Job (SJ)
  - by the OS in Task Services in User Job (TS)
  - by the user program (USER)
- Residence attributes are:
  - pageable (P)
  - fixed, which implies that some but not necessarily all of the segment may be fixed (F)
  - swappable (SWAP)
- Addressing attributes are:
  - direct (DA)
  - symbolically through LNS (SYM)
- Lifecycle attributes are:
  - for the duration of the system day (SYS)
  - for the duration of the job (JOB)
  - for the duration of the task (TASK)

There are several segments containing system tables which are always present and have known attributes. These are described below. User and System Tasks may have LNS, working storage, and stack segments with varying attributes which may contain some system tables. This will be noted in the descriptions of the individual table structures. The segments shown in the following table are always supported and have the indicated attributes.

2.0 STRUCTURE
2.4.1 INTRODUCTION

SEGMENT	PROTECTION FOR			RESIDENCE	ADDRESSING BY			LIFE CYCLE
	SJ	TS	USER		SJ	TS	USER	
0 SIGNAL BUFFERS	DR DW	DR DW		F SWAP		DA		SYS
1								
2								
3 SEGMENT DESCRIPTOR TABLES	DR	DR	DR	F SWAP		DA	DA	SYS
4 CONTROL POINT BUFFERS				F SWAP				SYS
5 LNS SYSTEM GLOBAL	DR DW IR IW	DR DW IR IW	IR	F	DA SYM	DA SYM	SYM	SYS
6 LNS USER LOCAL		DR DW IR IW	DR DW IR IW	P SWAP		DA SYM	DA SYM	TASK
7 LNS SYSTEM LOCAL		DR DW IR IW	IR IW	P SWAP		DA SYM	SYM	JOB
8 LNS USER GLOBAL	DR DW IR IW	DR DW IR IW	IR IW	P	DA SYM	DA SYM	SYM	SYS
9 LNS SYS LOCAL OF SYS JOB	DR DW IR IW	IR IW		F	DA SYM	SYM		SYS
10								
11 TASK MONITOR WORKING STORE		DR DW		P SWAP		DA		JOB
12								
13 TASK SERVICES WORKING STORE		DR DW		P SWAP		DA		JOB
ADDITIONAL USER DATA SEGS, INCLUDING STACK AND WORKING STORAGE		DR DW	DR DW	P SWAP		DA	DA	JOB TASK
ADDITIONAL LNS SEGMENTS		DR DW IR IW	DR DW IR IW	P SWAP		DA SYM	DA SYM	JOB TASK

Figure 2.4.1-1
SEGMENT ATTRIBUTES

2.4.3 ADDRESS SPACE OF SYSTEM JOB

DEDICATED SEGMENTS:

0 Signal Buffers for all Running Jobs
1 Invalid
2 Invalid
3 Segment Descriptor Tables for all Running Jobs
4 Control Point Buffers

LNS Segments:
5 System Global
6 User Local for this Job
7 System Local for this Job
8 User Global
9 System Local for the System Job

Task Monitor Program Segments:
10 Code
11 Working Storage

Task Services Program Segments:
12 Code
13 Working Storage

Sequence Monitor Program Segments:
14 Code
15 Working Storage

Sequence Monitor Task Segments:
16 Stack for Task Monitor execution
17 Stack for Task Services execution
18 Stack for Sequence Monitor execution
19 Binding for TM, TS, SQM

20 Job Buffer Pool
21 Binding for this Job

NONDEDICATED SEGMENTS:

System Access Manager Segments:
Task
Stack for Task Monitor execution
Stack for Task Services execution
Stack for System Access Manager program execution
Program

Code
Working Storage
Stager Subtasks (3 stacks per subtask)
Stack for Task Monitor execution
Stack for Task Services execution
Stack for Stager procedure execution

Job Establisher Segments:
Task
Stack for Task Monitor execution
Stack for Task Services execution
Stack for Job Establisher program execution
Program
Code
Working Storage

Queued Job Monitor Segments:
Task
Stack for Task Monitor execution
Stack for Task Services execution
Stack for Queued Job Monitor program execution
Program
Code
Working Storage

Deferred Job Monitor Segments:
Task
Stack for Task Monitor execution
Stack for Task Services execution
Stack for Deferred Job Monitor program execution
Program
Code
Working Storage

Running Job Monitor Segments:
Task
Stack for Task Monitor execution
Stack for Task Services execution
Stack for Running Job Monitor program execution
Program
Code
Working Storage

Job Collapser Segments:
Task
Stack for Task Monitor execution
Stack for Task Services execution
Stack for Job Collapser program execution

2.0 STRUCTURE
2.4.3 ADDRESS SPACE OF SYSTEM JOB

```
 Program
  Code
  Working Storage

File Router Segments:
 Task
  Stack for Task Monitor execution
  Stack for Task Services execution
  Stack for File Router program execution
 Program
  Code
  Working Storage
 Output Distributor Subtasks (3 stacks per subtask)
  Stack for Task Monitor execution
  Stack for Task Services execution
  Stack for Output Distributor procedure execution

Block Manager Segments:
 Task
  Stack for Task Monitor execution
  Stack for Task Services execution
  Stack for Block Manager program execution
 Program
  Code
  Working Storage
```


2.0 STRUCTURE
2.4.4 ADDRESS SPACE OF JOB USING A SUBSYSTEM

```
2.4.4 ADDRESS SPACE OF JOB USING A SUBSYSTEM


DEDICATED SEGMENTS:

0   Signal Buffers for all Running Jobs
1   Invalid
2   Invalid
3   Segment Descriptor Tables for all Running Jobs
4   Control Point Buffers

   LNS Segments:
5   System Global
6   User Local for this Job
7   System Local for this Job
8   User Global
9   System Local for the System Job

   Task Monitor Program Segments:
10  Code
11  Working Storage

   Task Services Program Segments:
12  Code
13  Working Storage

   Sequence Monitor Program Segments:
14  Code
15  Working Storage

   Sequence Monitor Task Segments:
16  Stack for Task Monitor execution
17  Stack for Task Services execution
18  Stack for Sequence Monitor execution
19  Binding for TM, TS, SQM

20  Job Buffer Pool
21  Binding for this Job


NONDEDICATED SEGMENTS:

   Subsystem Services Program Segments:
    Code
    Working Storage

   Subsystem LNS Segments:
    Subsystem Local for this Job
```

2.0 STRUCTURE
2.4.4 ADDRESS SPACE OF JOB USING A SUBSYSTEM

Subsystem Local for the Subsystem Job

User Task Segments:
Stack for Task Monitor execution
Stack for Task Services execution
Stack for Subsystem Services execution
Stack for User Program execution

User Program Segments:
Code
Working Storage
Files
Libraries

2.0 STRUCTURE
2.4.5 ADDRESS SPACE OF SUBSYSTEM SUPERVISOR JOB

2.4.5 ADDRESS SPACE OF SUBSYSTEM SUPERVISOR JOB

DEDICATED SEGMENTS:

0 Signal Buffers for all Running Jobs
1 Invalid
2 Invalid
3 Segment Descriptor Tables for all Running Jobs
4 Control Point Buffers

LNS Segments:
5 System Global
6 User Local for this Job
7 System Local for this Job
8 User Global
9 System Local for the System Job

Task Monitor Program Segments:
10 Code
11 Working Storage

Task Services Program Segments:
12 Code
13 Working Storage

Sequence Monitor Program Segments:
14 Code
15 Working Storage

Sequence Monitor Task Segments:
16 Stack for Task Monitor execution
17 Stack for Task Services execution
18 Stack for Sequence Monitor execution
19 Binding for TM, TS, SQM

20 Job Buffer Pool
21 Binding for this Job

NONDEDICATED SEGMENTS:

Subsystem Services Program Segments:
Code
Working Storage

Subsystem LNS Segments:
Subsystem Local for this Job

2.0 STRUCTURE
2.4.5 ADDRESS SPACE OF SUBSYSTEM SUPERVISOR JOB

Subsystem Local for the Subsystem Job

Subsystem Supervisor Task Segments:
- Stack for Task Monitor execution
- Stack for Task Services execution
- Stack for Subsystem Services execution
- Stack for Subsystem Supervisor Program execution

Subsystem Supervisor Program Segments:
- Code
- Working Storage
- Files
- Libraries

2.0 STRUCTURE
2.4.5 ADDRESS SPACE OF SUBSYSTEM SUPERVISOR JOB

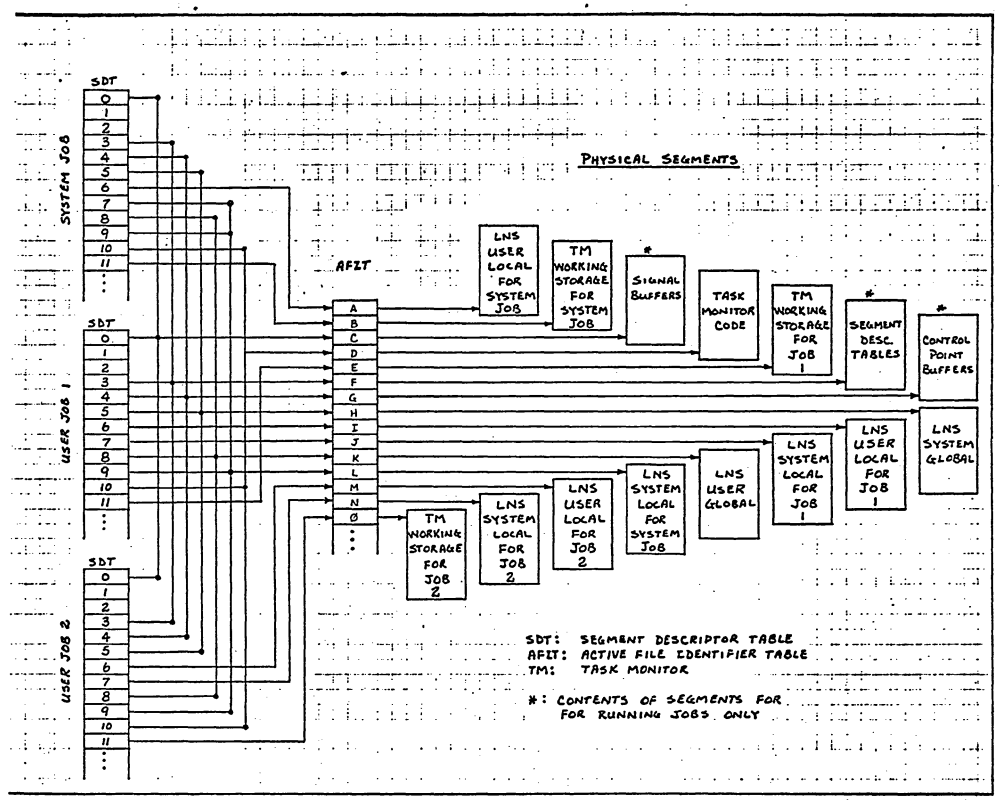


Figure 2.4.5-1
SEGMENT USAGE EXAMPLE

2.4.6 BASIC SYSTEM OBJECTS

2.4.6.1 Job

A job is an entity directly related to a user and has the following characteristics:

- there is one address space per job
- it is the identifiable user of resources (files, devices, memory, ...) which it can create, control access to, and destroy
- it is the basis of system supported accounting
- it is the swappable entity
- there is one LNS search list per job

The system maintains several lists that reflect the current state of a job as shown in Figure 2.4.6-1 and are:

- Known Job List (KJL)
- Deferred Job List (DJL)
- Running Job List (RJL)
- Active Job List (AJL)

The Known Job List contains all jobs known to the system

- Not yet established jobs
- Swapped out jobs with system segments deallocated
- Running jobs

The Deferred Job List contains all those job swapped out.

- Just established jobs that have not yet been swapped in
- Jobs that were running but are now swapped out

The Running Job List contains those known jobs that are established and swapped in.

- System segments are assigned
- Active jobs
- Inactive jobs

The Active Job List contains those Running jobs that have their working set available in memory. Each control point in an active job is on the dispatch chain with a status of either Ready or Not-ready.

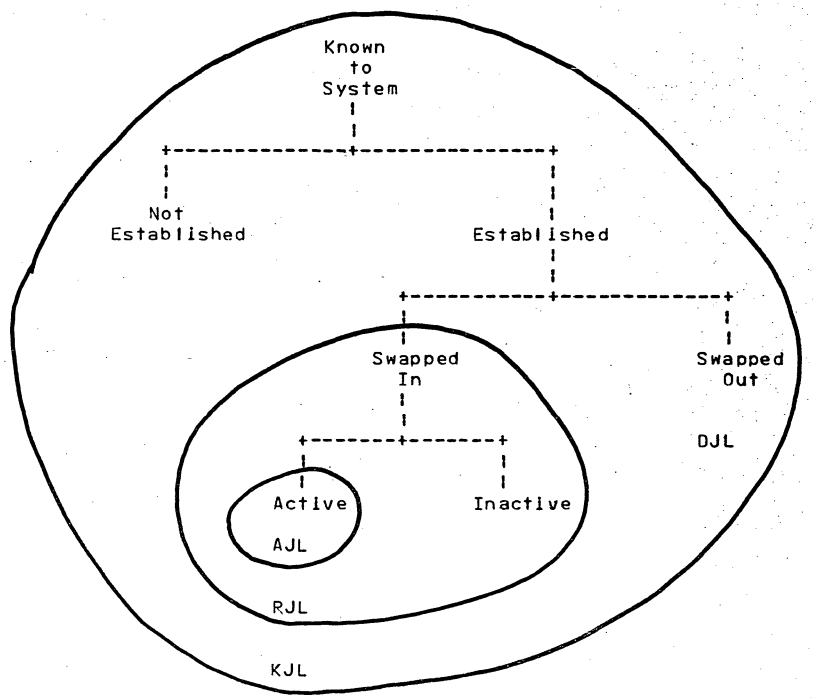


Figure 2.4.6-1
JOB LISTS

2.4.6.2 Task

A task is the execution of a program. A standard job has two tasks, the execution of the Sequence Monitor program and the execution of product set programs or user programs.

- a task shares an address space with other tasks in the job.
- a task does not own resources.
- all tasks in a job are swapped together.
- there is one signal buffer and signal selection list per task.
- a task may have within it several asynchronous executions of procedures, referred to as subtasks.
- all the subtasks of a task share the same
  - Loader Symbol Table
  - Binding Section
  - Signal Buffer
  - Signal Selection List
  - Common
  - Object Segment List
  - Library List
  - Working Storage
- each subtask of a task is separately dispatchable and has its own
  - Control Point
  - Stacks
- the System Monitor maintains a Dispatch Control Table that contains control point information
  - Status
  - Kind
  - Priority

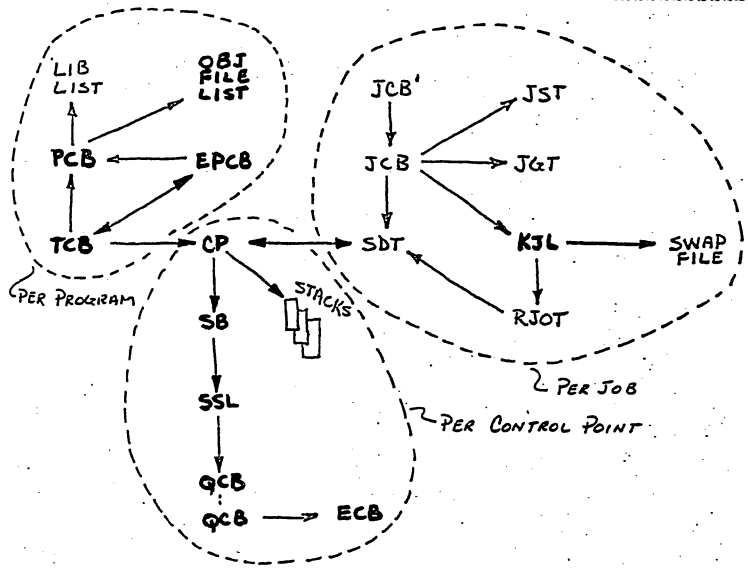


Figure 2.4.6-2
EXECUTION CONSTRUCTS

PCB Program Control Block
LNS structure. Can be in any LNS segment.

TCB Task Control Block
LNS structure. Can be in any LNS segment local to this job.

CP Control Point
System table structure. In segment #4.

SB Signal Buffer
System table structure. Is in segment #0.

SSL Signal Selection List
System table structure.

QCB Queue Control Block
ECB Event Control Block
Referenced by address by system code. Can be LNS structure

in any LNS segment local to this job. Can also be at any RW address in address space of this job (Working storage, Stack, Record, etc).

SDT Segment Descriptor Table
Hardware table structure. Is in segment #3 for this job.

JCB Job Control Block
LNS structure. Is in System Global LNS segment.

JCB' Alias Job Control Block
LNS structure. Is in System Local LNS segment for this job.

EPCB Established Program Control Block
JST Job Stack Table
JGT Job Gate Table
KJL Known Job List
RJOT Running Job Ordinal Table
System table structures.

2.4.6.3 File

Depending on the users level of interface to the Data Management system, a file may be viewed in the following ways:

- o File Management Level - A file consists of a peripheral device or a region of storage on a volume.
- o Record Management Level - A file consists of a set of records addressable by key, by ordinal, or by file address.
- o Block Management Level - A file consists of a set of blocks addressable by block numbers.
- o Segment Level - A file consists of a segment of virtual memory addressable by segment number and byte offset.

The File Control Block Provides the primary interface through which a user supplies the definition of a file.

- o A File Control Block must exist before a file can be referenced.
- o Cataloged values can replace any existing File Control Block values.
- o Values supplied through LNS requests can replace any existing File Control Block values.
- o Values supplied through the FM#DEFINE_FILE request can replace only null values.
- o No File Control Block fields can be directly modified by the user after the file is created (new mass storage files), attached (existing mass storage files) or opened (non-mass storage files).

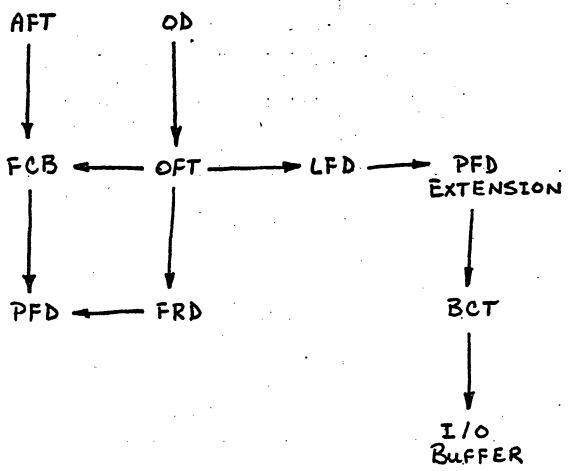


Figure 2.4.6-3
FILE CONSTRUCTS

FCB File Control Block
LNS structure. Can be in either System Global LNS segment or System Local LNS segment for this job.

LFD Logical File Descriptor
BCT Buffer Control Table
OFT Open File Table
FRD File Request Descriptor
AFT Attached File Table
Data Management structures. In System Local segment for this job.

PFD Physical File Descriptor
System structure. In system global segment.

OD Open Descriptor
Data Management structure. In user memory.

2.4.6.4 Segment

A segment is defined by a segment number (unique id), a segment descriptor entry (segment attributes), and a body which is an array of consecutive elements (target data). Since a segment cannot wholly reside in memory, a mapping between memory and mass storage addresses is always maintained by the Operating System. Due to varying performance and processing requirements this mapping is supported in multiple ways:

- Direct Segment- a direct mapping between a mass storage file and a segment exists, and any modifications made to the segment will be reflected automatically in the file. Examples would include:
  - Output files
  - Read-only data
  - Program libraries

- Temporary Segment- a segment which cannot survive beyond the life of the creating job. Several of these segments will be mapped into one mass storage paging file. Temporary Segments provide a low overhead mechanism for allocating and managing temporary structures. Examples would include:
  - Stacks
  - Working storage
  - Heaps
  - Binding sections
  - Local LNS

- Indirect Segment- an indirect mapping between a mass storage file and a segment exists. Any modifications made to the segment are reflected in a paging file, which exists for the life of the job and may also contain modifications to other such segments. A program must explicitly ask the system to reflect the changes back into the original file. Examples would include:
  - Files being edited
  - Periodic updating
  - Batching updates

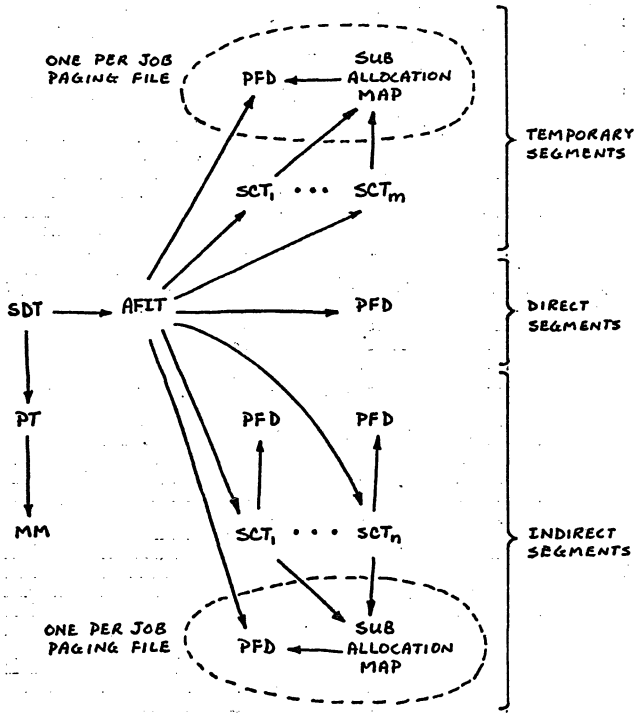


Figure 2.4.6-4
SEGMENT CONSTRUCTS

SDT Segment Descriptor Table
Hardware structure. In segment #3 for this job.

PT Page Table
Hardware structure. In real memory.

PFD Physical File Descriptor
System structure. In system global segment.

AFIT Active File Index Table
SCT Segment Control Table
MM Memory Map
Storage Management structures. In system space.

3.0 CODING/DOCUMENTING CONVENTIONS

3.1 INTRODUCTION

The intent of this section is to record information pertinent to the development and implementation of the Operating System. It will provide a basis for discussion and improvements, and will evolve towards implementation conventions within the project.

3.2 IPLOS NAMING CONVENTIONS

- The maximum length of all system and user names and symbols will be limited to 31 characters.
- All IPLOS modules that deal with 1 - 31 character names must be ultimately concerned with the space required to support the long names. Wherever feasible, all such modules should use space/table allocation techniques that optimize at 8 character names with minor performance penalties for longer names (such techniques must be invisible to users).
- Within IPLOS itself, three levels of naming environments will exist:

1) User Visible Names

   - 1 to 31 characters long
   - Maximum mnemonic value

   ex. CAUSE_EVENT
       SUBMIT_JOB

2) System Global Names

   - 1 to 31 characters long
   - format
     XX#S..S OR XXX#S..S

   where

XX = OS section code string
XXX = OS section acronym
# = literal string
S..S = mnemonic character string

ex. LNS#ATTACH

3) Module Internal Name

   - 1 to 8 characters long
   - Mnemonic acronym

3.2.1 IPLOS SECTION CODE STRINGS

OS Operating System

UI	User Interface	
	IC	Input/Output Control
	CL	Command Language
	OC	Operator Communications
	SA	System Access Manager
	FR	File Router
	MG	Message Generator
JM	Job Management	
	IT	Initiator/Terminator
	RA	Resource Allocator
	JS	Job Scheduler
	AL	Accounting/Logging
	CR	Checkpoint/Restart
PM	Program Management	
	PE	Program Execution
	PC	Program Communication
	LL	Loader/Linker
	LN	Logical Name Space
DM	Data Management	
	FM	File Management
	AP	Access Procedures
	BM	Block Manager

DD Device Drivers

MM Storage Management (Memory)

MC Memory Control
SC Segment Control

SM System Management

SS System Scheduler
CM Configuration Manager
DS Deadstart
SG System Generator

SS System Structure

SY System Monitor
TM Task Monitor
TS Task Services
SB Subsystem Supervisors

UT Utilities

LG Library Generator
DR Dump/Restore
MA Measuring/Analysis

3.2.2 SOURCE LIBRARY NAMES AND CONVENTIONS

3.2.2.1 Documentation Files

o Working Document

A working copy of each GDS Chapter is under the user id MAD with file names CH01...CH12. These are TEXTFORM files to which all Operating System personnel have read permission.

o Version 4 GDS

A file named OSGDS04 has been created via SCM under user

id DTC and contains a TEXTFORM copy of the 30 MAY 75 release of the GDS. The deck names are CHP01...CHP12 and APDXA...APDXE. Use SCM to retrieve any of this data.

o Conventions

- MODIFY deck names will be equal to document sections (in the TEXTFORM context)
- All documentation source will be in TEXTFRM input format
- The user interface to MODIFY and TEXTFRM will be processed thru the standard IPLOS source maintenance commands.
- The major use of the Global Libraries (i.e., OSOPL, JMOPL, ...) will be to contain all system and section global table and symbol definitions and declarations.

  ex. SWL CONST definitions
  SWL TYPE definitions
  table declarations

3.2.2.2 Source Code Files

o Library Names

FORMAT
  XXOPLNN
where
  XX - 2 letter IPLOS section code
  OPL = Literal
  NN = Version Number of library

(blanks inserted for clarity, actual name without blank)

OS	OPL	01	Operating System Global Library
UI	OPL	01	User Interface Global Library
IC	OPL	01	Input/Output Control Library
CL	OPL	01	Command Language Library
OC	OPL	01	Operator Communications Library
SA	OPL	01	System Access Manager Library
FR	OPL	01	File Router Library
MG	OPL	01	Message Generator Library
JM	OPL	01	Job Management Global Library
IT	OPL	01	Initiator/Terminator Library
RA	OPL	01	Resource Allocator Library
JS	OPL	01	Job Scheduler Library
AL	OPL	01	Accounting/Logging Library
CR	OPL	01	Checkpoint/Restart Library
PM	OPL	01	Program Management Global Library
PE	OPL	01	Program Execution Library
PL	OPL	01	Program Communication Library
LL	OPL	01	Loader/Linker Library
LN	OPL	01	Logical Name Space Library
DM	OPL	01	Data Management Global Library
FM	OPL	01	File Management Library
AP	OPL	01	Access Procedures Library
BM	OPL	01	Block Manager Library
DD	OPL	01	Device Driver Library

MM	OPL	01	Memory Management Global Library
MC	OPL	01	Memory Control Library
SC	OPL	01	Segment Control Library
SM	OPL	01	System Management Global Library
SS	OPL	01	System Scheduler Library
CM	OPL	01	Configuration Manager Library
DS	OPL	01	Deadstart Library
SG	OPL	01	System Generator Library
SS	OPL	01	System Structure Global Library
SY	OPL	01	System Monitor Library
TM	OPL	01	Task Monitor Library
TS	OPL	01	Task Services Library
SB	OPL	01	Subsystem Supervisors Library
UT	OPL	01	Utilities Global Library
LG	OPL	01	Library Generator Library
DR	OPL	01	Dump/Restore Library
MA	OPL	01	Measuring/Analysis Library
SC	OPL	01	Source Code Maintenance Library

o Conventions

- the source libraries will all be maintained in the MODIFY program library format
- the use of MODIFY COMMON decks techniques must be maximized
- all O.S. table declarations must be stored as COMMON decks.
- the library structure should be viewed as a three level structure
  - o OS global highest
  - o Major division global lower
  - o Discrete libraries lowest
- All code modules and COMMON decks must be placed at the lowest level in the structure possible

- All source coding will be done in ISWL/SWL

- Each MODIFY DECK will represent one and only one ISWL/SWL module

Until such time as ASL/IPL wide source maintenance conventions are developed, a special set of terminal commands will be developed and utilized for the IPLOS project.

3.3 CODING CONVENTIONS

This section will attempt to suggest a few practical and probably obvious conventions, with the hope that the list will be modified as suggestions from project personnel are forthcoming and evolve into a prototype standard.

3.3.1 DECLARATIONS

- Arrange declarations into some logical grouping and identify with headings

  CONST

  TYPE

  COMMON FILES

  EXTERNALS

  LOCAL DATA

- Indent level numbers

- Identifiers should tend toward self description (within length constraints)

- Be consistent with margins and starting columns for elements within declaration statements

- Position compiler control toggles to the left side of listing

3.3.2 PROCEDURES

- Keep procedures small as possible, one or two pages average
- Provide comments for groups rather than individual statements in order to maintain continuity of code - do not over kill on commenting.
- One procedure statement per line
- Use indentation for both code and comments
- Avoid use of GO TO statement

3.4 SOURCE DOCUMENTATION CONVENTIONS

IPLOS will use the SES defined Documentation Prompter available under the subsystem for source module documentation. An example is as follows:

```
/batch
$RFL,20000.
/get,runses/un=all
/get,rundp/un=soe
/-runses
**MSG 1; SES V1.0 75/05/30. 09.10.22. -PLEASE LOGIN
? login mad,profile=rundp
**MSG 81; PROFILE PROCESSING INITIATED
0001030=*PROCEDURE:
? tentatively_allocate
0001040=
? =
0001040=*PROGRAMMER:
? MAD
0001050=
? =
0001050=*PURPOSE:
? This procedure determines if an allocation of a
0001060=
? particular peripheral to a job can be permitted
0001070=
? based on the peripherals current usage, the
0001080=
? usage requested, and the possibility of deadlock
0001090=
? if the allocation were made.
0001100=
? =
0001100=*INPUT:
? unit:: the unit table entry of the peripheral to be
0001110=
?       allocated.
0001120=
? req_shareable:: boolean set to true if the unit is
0001130=
?       requested as shareable, false if the unit is
0001140=
?       requested for exclusive use.
0001150=
? rsac:: the current shared, assigned, claimed for the
```

```
0001160=
?       job reflecting any previous tentative allocations.
0001170=
? =
0001170=*OUTPUT:
? allocation_ok:: boolean set to true if the allocation
0001180=
?       can be permitted, false if it cannot.
0001190=
? =
0001190=*METHOD:
? =
0001200=*DATA_USAGE:
? pseudo_available:: arrary of available peripherals
0001210=
?       reflecting any previous tentative allocations.
0001220=
? =
0001220=*COMPILE_OPTIONS:
? =
0001230=*NOTES:
? =
0001240=*MESSAGES:
? =
0001250=*RESTRICTIONS:
? =
 DOCUMENTATION NOW ON DOCE - SAVE IT
**MSG 67; RUN COMPLETED
? logout
**MSG 84; CONNECT TIME =  00.09.56.
* END SS 1.0
```



```
/edit,doce
 BEGIN TEXT EDITING.
? 1;*
"
*BEGIN_DOCUMENTATION:
*PROCEDURE:          tentatively_allocate
*PROGRAMMER:         MAD
*PURPOSE:            This procedure determines if an allocation of a
*                    particular peripheral to a job can be permitted
*                    based on the peripherals current usage, the
*                    usage requested, and the possibility of deadlock
*                    if the allocation were made.
*INPUT:              unit:: the unit table entry of the peripheral to be
*                         allocated.
*                    req_shareable::  boolean set to true if the unit is
*                         requested as shareable, false if the unit is
*                         requested for exclusive use.
*                    rsac:: the current shared, assigned, claimed for the
*                         job reflecting any previous tentative allocations.
*OUTPUT:             allocation_ok:: boolean set to true if the allocation
*                         can be permitted, false if it cannot.
*METHOD:             NONE
*DATA_USAGE:         pseudo_available:: array of available peripherals
*                         reflecting any previous tentative allocations.
*COMPILE_OPTIONS:    NONE
*NOTES:              NONE
*MESSAGES:           NONE
*RESTRICTIONS:       NONE
*END_DOCUMENTATION:
"
```

3.4.1 TABLE SPECIFICATION

TABLE NAME: "system name of table"

Purpose: "one line description"

Usage: "reason for existence"

Creator: "system module building table"

Readers: "system modules"

Writers: "system modules"

Reference: "other tables/structures pointed at"

Declaration: "SWL declarations/definitions"


```
011500  "$s"
011600"-------------------------------------------------------
011700table name: logical file descriptor (lfd)
011800purpose:    an lfd is the primary control block for an instance
011900            of open of a file.
012000usage:      the lfd contains all information necessary to describe
012100            an instance of open of a file. the lfd contains
012200            the working storage and current file status for the
012300            file access procedure that processes user requests
012400            against the file.
012500creator:    the lfd is created at file open time by the file access
012600            procedure that processes requests against the file.
012700readers:    file access procedure
012800writers:    file access procedure
012900references: PFDX for file
013000"
013100
013200
013300"define temporary types for temporary buffer manager"
013400
013500
013600  CONST
013700    bufcount = 50; "max number of blocks"
013800  TYPE
013900    bct_entry = RECORD
014000      blknum: integer,
014100      blk_allocated: boolean,
014200      blkptr: ^array[ * ] OF char,
014300    RECEND;
014400
014500"define logical_file_description_table"
014600
014700  TYPE
014800    rel_fap_lfd = RECORD
014900      job_id: os#kjl_ordinal,
015000      cur_rec_otr: rel_fileaddress,
015100      auth_func_codes: rm#requests_set,
015200      rn_inc: boolean,
015300      locking: boolean,
015350      lockopt: rm#reserve_option,
015400      cur_blk_num: integer,
015500      timeout: integer,
015600      blkptr: ^array[ * ] OF char,
015700      pfdxp: ^rel_fap_pfdx,
015800    RECEND;
```

3.5 OPERATING SYSTEM REQUEST/RESPONSE STATUS FORMAT

All IPLOS Request processors that return status information will use a system standard status record format. The system message generator will also use the same record format as input.

SWL Record format:

```
TYPE
  OS#STATUS = RECORD
    LEVEL: 0..OFF(16), "general level indicator"
    FROM: STRING (2) OF CHAR, "issuing OS section"
    ST_CODE: 0..OFFFF(16) _specific status code"
    MESG: STRING (32) of CHAR, "message mask"
    RECEND:
```

WHERE:

LEVEL - Indicates the general status, the values of which are shown in the following table.

FROM - Indicates the Operating System category that issued the status

ST_CODE - Indicates the specific code issued by convention threat the upper digit as a category and the remaining digits as specific errors within a category.

```
1xx - parameter errors
2xx - access errors
3xx - functional errors
 .
 .    (to be supplied)
9xx - internal condition rejects
Axx - internal error rejects
Fxx - unidentifiable problem
```

MESG - One or more character insertion strings with asterisk (*) separators (first character of the message will be used as separator). The total length of the text string including separators is 32 characters. The message generator references a message dictionary with a key based on the request status code and message text with asterisk substitution indicators.

S=	0 1 2 3	4 5 6 7	8 9 A B	C D E F
STATUS				
Accepted	x	x		
Completed	x			
Not completed		x		
Rejected			x	x
User problem			x	
System problem				x

EXAMPLE

Status "8AP701" '*MYFILE*47'

8 - rejected, user problem

AP - issued by Access Procedures

701 - recorded data boundary encountered

Message Dictionary

key - 8AP701

text - 'END OF DATA ON * AT RECORD *'

Diagnostic Generated

'END OF DATA ON MYFILE AT RECORD 47'

4.0 PRODUCT SET INTERFACES

4.0 PRODUCT SET INTERFACES

This area will contain information about the executed compiler environment (Parameters, errors, IN, OUT, OBJECT, DEBUG, line numbers, LGO example and general loader facilities.

5.0 SAMPLE REQUESTS

5.0 SAMPLE REQUESTS

This area will contain some sample requests (get/put, read, open, access interrupt) and how control will flow upon their execution.

6.0 STRATEGY

This area will describe the O.S. position and understanding of selected general topics.

6.1 MULTIPROCESSOR(S)

- Multiprocessing is primarily a throughput facility.
- The O.S. will externalize two levels of multiprocessing to the end user.
- The O.S. itself should be constructed in a manner which allows multiproccessing to occur. (Splits at natural asynchronous boundaries. i.e., I/O scheduling)
- The O.S. will support processors of differing types (e.g., P0 - P1, P2 - P4, Emulators, ...)

6.2 COMPATIBILITY

- The O.S. will support emulation services as requested by the emulator projects only.
- Emulation support will be introduced into the system via the subsystem conventions.
- Compatability at the operator console will not be supported.
- The O.S. will support conversion from one code to another on the fly. Mixed data files must be processed by a program which understands the static form of the data.
- The general model for work supported by the O.S. is described in the System Command Language document.

6.3 VIRTUAL MEMORY AND PROTECTION

- The Virtual Memory and Protection mechanisms in the hardware offer more capability than the Operating System will use, initially.
- Virtual memory will aid the O.S. in managing large physical memory and reducing the impact of large changes in physical memory sizes and types.
- The O.S. is not relying on a paging device.
- The O.S. will not externalize ring(s) and keys to the end user in V1.0.
- The O.S. will use rings to implement the multiple monitor concept. Use of keys and locks will not be in V 1.0.
- Rings and rules for their use must be externalized to the subsystem writer.
- The O.S. must allow for multiple protection levels to be introduced. For example it should be possible for an installation to force all file requests through an installation - supplied subsystem and guarantee that no other path can be taken.

6.4 SHARING

- The OS will support sharing of code, which implies that compilers must generate pure procedures.
- The O.S. will allow shared access to data (via virtual memory and explicit input and output) and will provide service routines to assist the user in controlling shared access.

6.5 IOSS

Requirements Draft:
Dated November 5, 1974

6.6 NETWORKS

- For purposes of IPLOS implementation it will be assumed that host-to-host networks will not be required in V 1.0.
- IPLOS design will track the design plans for general networks as defined by CDC and NCR.
- A general mode of design and implementation of IPLOS will be to provide the basic routines and services to allow general computer-to-computer communications and to rely on library routines and user code to actually control the use of such services (i.e., networking will not be an automatic, transparent function of IPLOS V 1.0).

6.7 RAS

- IPLOS will be designed to operate in an unattended manner.
- A primary goal of IPLOS will be to detect and isolate all errors in system operation and use. User handling of errors will be supported wherever feasible.
- The IPLOS will be designed to function in degraded configurations. Automatic reconfiguration will occur where feasible.
- An accurate history file of all relevant data pertaining to any error in system operation will be captured and saved.
- Support for allowing use of diagnostic services in a production environment will be provided to the maximum degree feasible.
- Multiple levels of job and system checkpoint and recovery will be provided in the design of IPLOS. Version 1.0 will be mostly concerned with system recovery support.
- A major goal of the IPLOS design and implementation will be to allow the modification and/or replacement of any system module in a controlled manner in a production environment.

6.8 TRANSACTION

- ASL will initiate a project to study implementation of TOX/RSX (NCR) on IPL.
- A form of transaction processing will coexist with the O.S. and be a subset performance and feature wise of TOX/RSX. (Use Subsystem interface)
- O.S. will externalize physical I/O interfaces which a TOX/RSX type system will use.

6.9 CONFIGURATION

O.S. Design Target 3 MBYTE - P2

1 MBYTE - P1

Peripherals unknown.