UNISYS

OS/3 Supervisor Technical Overview

Previous Title: OS/3 Supervisor Concepts and Facilities

Relative to Release Level 11.0

Priced Item

August 1987

Printed in U S America
UP-8831 Rev. 3

PAGE STATUS SUMMARY

ISSUE: UP-8831 Rev. 3
RELEASE LEVEL: 11.0 Forward

Part/Section	Page Number	Update Level
Cover		
Title Page/Disclaimer		
PSS	1	
Preface	1 thru 3	
Contents	1, 2	
1	1 thru 5	
2	1 thru 12	
3	1 thru 9	
4	1	
User Comment Form		

Part/Section	Page Number	Update Level

Part/Section	Page Number	Update Level

Preface

This manual describes the concepts behind the Operating System/3 (OS/3) Supervisor and the facilities the supervisor makes available to OS/3 users. It presents an overview of the supervisor that is intended for two audiences: high-level language programmers and site administrators. The manual is organized as follows:

- Section 1. Basic Supervisor Concepts

  For both audiences. Introduces the supervisor, describes its functions in terms of tasks and multitasking, and outlines its interfaces with user software.

- Section 2. Supervisor Features and Functions

  For both audiences. Describes supervisor features by functional categories.

- Section 3. Supervisor Features and High-Level Languages

  For high-level language programmers. Builds upon the list of features in Section 2 by listing the languages that can call upon each feature: FORTRAN, COBOL, RPG II, BASIC, ESCORT programming language, and job control language (JCL).

- Section 4. Choosing an OS/3 Supervisor – Ours or Yours?

  For site administrators. Discusses reasons for choosing the ready-to-use supervisor supplied with the OS/3 system or generating your own supervisor in the SUPGEN phase of system installation.

This manual is intended as a general discussion of those supervisor features you directly or indirectly call on, rather than as a detailed guide to all supervisor functions. (Some parts of the OS/3 supervisor, such as spooling and job accounting, are priced separately and therefore may not necessarily be installed in your system.) After reading this manual, you should have a good idea of:

- what supervisor features are available through the OS/3 high-level languages; and
- the supervisor configuration, specified at system installation time, that best suits your needs.

For a detailed guide in using supervisor services, refer to the *Supervisor Macroinstructions User Guide/Programmer Reference,* UP-8832 (current version).

Other current OS/3 publications referenced in this manual that are helpful when using the supervisor are:

- *System Service Programs (SSP) User Guide,* UP-8841

  Describes various system utilities (e.g., librarian, linkage editor).

- *Consolidated Data Management Concepts and Facilities,* UP-9978

  Describes the organization and record formats of various file types.

- *Basic Data Management User Guide,* UP-8068

  Describes the effective use of OS/3 basic data management.

- *Interactive Services Concepts and Facilities User Guide/Programmer Reference,* UP-9972

  Describes the commands and operating procedures for workstation terminals.

- *Spooling and Job Accounting Concepts and Facilities,* UP-9975

  Describes basic spooling and job accounting concepts and options available to control spooling systems.

- *System Installation User Guide/Programmer Reference,* UP-8839

  Describes the procedures necessary to install, tailor, and maintain OS/3 software in a System 80 environment.

- *Security Maintenance Utility User Guide,* UP-12028

  Describes OS/3 system security to security administrators working in a System 80 environment.

- *Operations Handbook Operator Reference,* UP-8859

  Describes system operator procedures.

- *System Activity Monitor User Guide/Programmer Reference,* UP-9983

  Describes the use of the system activity monitor (SAM) for evaluating system performance.

- *1974 American National Standard COBOL Programmer Reference,* UP-8613

  Describes 1974 ANS COBOL for the applications programmer.

- *FORTRAN IV Programmer Reference,* UP-8814

  Describes FORTRAN IV for the applications programmer.

- *Report Program Generator II (RPG II) User Guide,* UP-8067

  Describes RPG II for both novice and experienced applications programmers.

- *BASIC Programmer Reference,* UP-9168

  Describes BASIC for the applications programmer.

- *ESCORT Programming Language User Guide,* UP-8855

  Describes ESCORT programming language for applications programmers.

- *Job Control User Guide,* UP-9986

  Describes the job control language used under OS/3.

Contents

PAGE STATUS SUMMARY

PREFACE

CONTENTS

1. BASIC SUPERVISOR CONCEPTS

1.1. GENERAL 1–1

1.2. TASKS AND MULTITASKING 1–1

1.3. CONTROLLING MULTIPLE TASKS 1–2

1.4. USER PROGRAM INTERACTION 1–3

1.5. EXTERNAL EVENT INTERACTION 1–4

1.6. OVERVIEW OF SUPERVISOR FEATURES 1–5

2. SUPERVISOR FEATURES AND FUNCTIONS

2.1. GENERAL 2–1

2.2. PROGRAM INITIATION AND LOADING 2–1

2.3. PROGRAM TERMINATION 2–2

2.4. TIMER SERVICES 2–2

2.5. PROGRAM LINKAGE 2–3

2.6. ISLAND CODE LINKAGE 2–3

2.7. SYSTEM INFORMATION CONTROL 2–4

2.8. CONTROL STREAM READER 2–4

2.9. DISK AND DISKETTE SPACE MANAGEMENT 2–5

2.10. SYSTEM ACCESS TECHNIQUE 2–5

2.11. MULTIPLE-INDEXED RANDOM ACCESS METHOD 2–6

2.12. MULTIJOBBING 2–6

2.13. MESSAGE DISPLAY AND LOGGING 2–6

2.14. INTERACTIVE SERVICES 2–7

2.15. SPOOLING 2–7

2.16. PRINTERLESS SYSTEMS 2–8

2.17. JOB ACCOUNTING 2–9

2.18. DIAGNOSTIC AND DEBUGGING FACILITIES 2–9

2.19. RESOURCE MANAGEMENT 2–10

2.20. SECURITY 2–11

2.21. SHARED CODE MANAGEMENT 2–11

2.22. DYNAMIC BUFFER MANAGEMENT 2–12

2.23. SYSTEM MONITORING FACILITIES 2–12

3. SUPERVISOR FEATURES AND HIGH-LEVEL LANGUAGES

3.1. GENERAL 3–1

3.2. SUPERVISOR FEATURES AND LANGUAGE USE 3–2
3.2.1. Program Initiation and Loading 3–2
3.2.2. Program Termination 3–2
3.2.3. Timer Services 3–3
3.2.4. Program Linkage 3–4
3.2.5. Island Code Linkage 3–4
3.2.6. System Information Control 3–4
3.2.7. Control Stream Reader 3–5
3.2.8. Disk and Diskette Space Management 3–5
3.2.9. System Access Technique 3–6
3.2.10. MIRAM 3–6
3.2.11. Message Display and Logging 3–6
3.2.12. Spooling and Job Accounting 3–7
3.2.13. Diagnostic and Debugging Aids 3–8

4. CHOOSING AN OS/3 SUPERVISOR – OURS OR YOURS?

USER COMMENT FORM

1. Basic Supervisor Concepts

1.1. GENERAL

The Operating System/3 (OS/3) Supervisor is a package of routines that form the heart of OS/3. It is the supervisor that allows other parts of OS/3 to work together and makes possible such useful OS/3 features as multijobbing and spooling.

As far as your user programs are concerned, the supervisor has two main functions:

- It interacts with user programs and symbionts to provide the services and control they need.
- It acts when necessary to handle randomly occurring external events, such as errors, that might otherwise disrupt a user program or even the entire system. Thus, an error occurring in one job may cause only that job to be terminated, leaving other jobs unaffected.

The supervisor is built around executable modules, or routines, each of which has a specialized function. Those routines commonly used by the supervisor always reside in main storage. Other less often used routines, called transients, are stored on the SYSRES volume and are loaded in main storage only when the supervisor needs them. This arrangement promotes supervisor efficiency: it minimizes the amount of main storage the supervisor uses by overlaying unneeded transients with newly loaded transients, and it eliminates the input/output time needed to load the most commonly used routines by keeping them resident.

1.2. TASKS AND MULTITASKING

The OS/3 supervisor does much of its work through the use of tasks. A task is the basic unit of work that can compete with other tasks for control of the central processor. By *processor control* we don't mean that the task executes machine code by using the processor but, rather, that it acts as a logical point of control for a physical sequence of executable machine code – a program – that does use the processor.

To grasp the difference between a task and a program, picture a system in which only one program at a time were allowed in main storage. In this case, no task control would be needed because the program would execute and terminate without interruption. But most programs do not continuously use the processor: you may, for example, request input or output, in effect leaving the processor idle until the I/O channel signals that the I/O operation has finished, at which time the program could resume use of the processor.

If more than one program were allowed in main storage, however, processor time could go from an idle program to one that isn't idle and later return to the first program when it is ready to resume processing. In OS/3, up to 256 such programs can exist concurrently in any one job; to prevent confusion and ambiguity, each program is controlled by a task. This ability to juggle processor control among tasks, maximizing processor use, is called multitasking.

Programs run under tasks as follows:

- Each user job step and symbiont runs under control of one task, called the primary task.
- Each transient runs under control of a task.
- Other system facilities, such as spooling and the integrated communications access method (ICAM), run as tasks.

The only OS/3 routines that do not run under task control are critical supervisor routines, such as error handling, and those routines that themselves manage tasks.

Since only one task at a time can control the processor (meaning the processor executes the code that task defines), some mechanism is necessary to take processor control away from one task and give it to another. In OS/3, that mechanism is the switcher, a critical supervisor routine that maintains a list, called the switch list, of all the tasks currently existing in the system. The switcher can take processor control away from a task and give it back later without disrupting the task's program and can do this as many times or as often as necessary.

So far, we have outlined the basic requirements for multitasking: the ability to create and maintain multiple tasks and a means of allocating processor time among available tasks. But we need more, as the next subsection explains.

1.3. CONTROLLING MULTIPLE TASKS

If all tasks in an OS/3 system were completely independent, all would compete for processor time on an equal basis. The system would quickly become unusable, though, because OS/3 tasks are not independent. For example, task A requests that task B do some work for it before it can continue. If both tasks continue to compete for processor time, what prevents task A from resuming processor control prematurely, before task B finishes? OS/3 solves this and related problems by using the task control block (TCB).

A TCB exists for each task in the system. It contains the physical, nonexecutable data that corresponds to a task's logical control of a program. It defines a task and acts as the task's interface with other tasks and the supervisor. By defining the task and its limits, the TCB prevents it from being disrupted by other tasks in the system, helping to make multitasking possible. At least one TCB exists for each user job (in the job prologue), and a TCB exists for each active transient routine.

When the switcher transfers processor control from one task to another, it uses information contained in the two TCBs involved. The switcher stores the problem registers and program status word (PSW) associated with the first task in its TCB, then loads the registers and current PSW with the same type of information contained in the second TCB, effectively allowing the second task to pick up execution exactly where it left off earlier.

Other control information contained in a TCB can be set to cause the switcher to ignore its task, in effect suspending that task or making it ineligible for processor control. A task remains in this suspended condition until the same control information is reset, causing the task to become active or eligible once again for processor control. While a task is suspended, all other active tasks continue to compete for processor time through the switcher. Often a task is suspended until some needed action is taken; only then is the task's control information reset, and only when that task is ready to resume processor control. It is the concept of task suspension that allows the supervisor to control task access to the switcher and thus to keep task competition from going out of control.

The supervisor has one other mechanism for controlling tasks, that of priority levels. When created, a task is assigned a priority. The switcher, when looking for a task to which it can transfer processor control, scans active tasks from those with the highest priority on down, selecting the active task with the highest priority to get processor control. System tasks (like transients) run at a higher priority than user jobs. Running systems tasks at a higher priority does not adversely affect user jobs, which, in any case, must call on them for various services. Also, system tasks running at a higher priority can begin and finish more rapidly, thus actually improving overall system efficiency.

1.4. USER PROGRAM INTERACTION

The supervisor's ability to maintain multiple tasks serves it well in interfacing with user programs. Since most supervisor services run as tasks themselves, the most common sequence of events is as follows:

1. A user program running under control of a primary task called, for example, task A, calls a transient routine to perform some system service. The supervisor responds by loading the transient in a portion of main storage set aside for transients and establishing the transient as a task with its own TCB called, say, task B.

2. The supervisor suspends task A and sets control information in the task's TCB that marks the task as suspended and unavailable.

3. After task A gives up processor control, the switcher scans the switch list for the highest priority task awaiting control. It may give control to task B or to some other active task but, task A being marked unavailable for the moment, the switcher will not return control to that task.

4. Eventually, the transient routine controlled by task B finishes. Only at this point does the supervisor reset the control information in task A and restore it to active status, task B having finished its work.

1.5. EXTERNAL EVENT INTERACTION

Another use of multitasking is as a response to external events. These events are called interrupts because they interrupt normal processor flow and must be handled in some way before processing can continue. OS/3 recognizes eight types of interrupts:

- Supervisor call – occurs in response to the SUPERVISOR CALL (SVC) machine instruction. Though it is handled as an interrupt, your programs routinely use the supervisor call to request supervisor services, as described in 1.4.

- Exigent machine check – indicates a malfunction in or around the processor. If the malfunction is in the supervisor area, the system is halted. If the exigent machine check indicates an error in the memory area of a job region, the system continues to run, though that memory area is marked as not usable and the job is terminated.

- Repressible machine check – indicates a malfunction in or around the processor from which recovery is possible.

- External interrupt – generated either by the processor interval timer or the system console interrupt key.

- Program check – occurs when the processor attempts to execute a nonexistent instruction or to execute an existing instruction in an illegal manner.

- Program event recording (PER) – provides dynamic monitoring of executing programs by storing information about the current instruction whenever a specified event occurs.

- Input/output – occurs in response to signals from I/O channels.

- Restart – occurs when the restart key on the system console is pressed and can be used to put a stopped processor in the operating state.

Some interrupts, like the supervisor call or input/output, are routinely encountered; others, like program or machine checks, represent serious errors that the supervisor must handle with minimal system interruption. As you will see in the following sections, the supervisor can handle a wide variety of interrupts and program requests.

1.6. OVERVIEW OF SUPERVISOR FEATURES

The services provided to your programs by the supervisor fall into the following categories:

- Task management
- Program management
- Input/output related services
- Spooling
- Logging and accounting
- Diagnostic services

2. Supervisor Features and Functions

2.1. GENERAL

This section presents an overview of the supervisor facilities available with your OS/3 system. Those features described here can either be used through OS/3 high-level programs or be specified at system installation time. For a detailed discussion of how high-level languages use the supervisor, see Section 3.

2.2. PROGRAM INITIATION AND LOADING

Before any of your programs can run, the supervisor has to load it in the portion of main storage allocated to your job by job control. The supervisor initiation and loading facility takes information from the job prologue (tables located in the low-order portion of your job region) that helps it determine where to load your job. The facility then locates your program as a load module on disk, loads it in main storage at the proper location, links the user job step – now a primary task – at the proper execution priority, and passes control to the task switcher.

In some high-level languages, your program/load module can load other load modules whenever needed, often as overlays. In these cases, it is your program that directs the loader how and where to load the new module.

Within a loaded program, the supervisor relocates, if necessary, every address constant the program contains. This lets job control load your program anywhere in your job region while ensuring that it will run correctly. Relocation is not limited to job control; where the facility is available to high-level user programs, it also performs relocation.

The program initiation and loading facility is always included in your system.

2.3. PROGRAM TERMINATION

The supervisor has facilities to end a job step in an orderly way even if the system detects errors in the step. With these facilities, the supervisor causes the system facilities assigned to a job or to a task to be relinquished for assignment to other jobs or to other tasks. Program termination falls into two categories, normal and abnormal termination:

- Normal termination implies normal completion of the job step and continuation of the job. The supervisor allows all task and I/O functions to idle down prior to terminating the program. Normal termination usually occurs when the primary task controlling the job step detaches itself from the switch list after completion of the step. Under some high-level languages, though, you can code a program to terminate the job step under which it runs at any point in its logic. In some cases, too, you can cause the supervisor to generate a main storage dump of your job region upon normal termination of a job step.

- Abnormal termination implies that a system-detected error has occurred from which the program cannot recover. In this case, the supervisor detaches the primary task, delinks all outstanding I/O, and waits for all outstanding system functions to be completed. It provides a printout of the contents of the job region if one of the DUMP options was specified on the OPTION statement and if there is a printer assigned to the job or there is a printer available.

High-level programs usually generate the proper machine language to handle most of the errors your program may encounter (see 2.6). You do have some options available to let you handle errors in other ways, depending on the path your program logic takes.

Normal program termination is always included in your system. With the exception of SYSDUMP (see 2.18), all abnormal termination facilities are always included, too.

2.4. TIMER SERVICES

Supervisor timer services fall into two types: the day clock and the interval timer. The day clock facility maintains the current time and date in a simulated day clock, which is contained in the resident supervisor. The interval timer generates a timer interrupt after a certain amount of time has passed. If interrupt timer island code (see 2.6) is active at the time, control then passes to it.

Most high-level languages get the current system date and time of day from the day clock, although only the system operator can change them. No interval timer service is available to high-level language programmers.

The full range of timer services is always included in your system.

2.5. PROGRAM LINKAGE

A program may consist of several phases or routines produced by an assembler, compiler, or other language translator and then combined by the linkage editor. Control can be passed from one routine to another within the program. This is referred to as direct linkage. Linkage can proceed through as many levels as necessary. During the execution of a job step, a routine (referred to as the calling program) passes control to another routine (the called program), which can in turn become the calling program passing control to a third routine (the called program), etc. This branch and linking process requires that the contents of certain registers be saved, then restored, so that control can be returned to the calling program.

Program linkage is available from all high-level languages, in conjunction with the linkage editor (see the *System Service Programs User Guide,* UP-8841 (current version)).

2.6. ISLAND CODE LINKAGE

As you know, there are eight levels of interrupts in OS/3. Some of these interrupts are handled by system routines; however, there are four interrupts that some user programs handle themselves. These interrupts are:

1. Program Check – An operation in your program causes a program check interrupt, such as an addressing error, arithmetic overflow, or operation exception.

2. Interval Timer – A time interval elapses.

3. Abnormal Termination – An error occurs that makes continuation of your program impossible.

4. Operator Communication – The operator entered an unsolicited message at the system console or the workstation (the contents of register 1 at the time the island code gains control indicates whether the operator entered the message from the console or a workstation).

To handle these interrupts, all high-level languages generate closed routines called *island code,* which are linked to your program at execution time. When one of these interrupts occurs, the supervisor stores the contents of the program status word (PSW) and general registers and then transfers control to the appropriate island code routine. If the island code returns control to your program, the supervisor uses this stored information to return control at the point the interrupt occurred.

The purpose of the program check, interval timer, and operator communication island code routines is to handle program contingencies or to notify your program that the interrupt has occurred. In the case of abnormal termination, the function of the island code routine is to terminate either a task or a job step rather than the entire job (normal procedure for abnormal termination if there is no abnormal termination island code routine).

You cannot, generally, write your own island code for high-level language programs. There is little need to; the island code that high-level language processors themselves generate is capable of handling most interrupts in an orderly way.

Island code linkage is always included in your system.

2.7. SYSTEM INFORMATION CONTROL

Each user program is assigned a variable-length storage area within the job region; this area is called the job prologue. This area contains the primary task control block (TCB) and other information so critical that, to preserve system integrity, application programs can only read from or write to one 12-byte prologue field, called the communication region, used to pass information from one job step to the next. Within the communication region, high-level languages let you access only its last byte, called the user program switch indicator (UPSI) byte. In most high-level languages, you can access either the entire UPSI byte or individual bits within it. For more information on the UPSI byte, refer to the *Job Control User Guide,* UP-9986 (current version).

2.8. CONTROL STREAM READER

The control stream reader allows you to access data that was entered into the system with the job control stream. This provides a convenient method to handle small quantities of input that would normally have been handled as a card or diskette file. Because the data is embedded within the job control stream, there is no need to define a card file, nor is a device assignment set required for the card reader.

This embedded data might consist of transactions or changes to be processed against a master file, source code, or control statements to be processed by a utility routine; or it might consist of PARAM job control statements to introduce parameters that can be used during program execution. Refer to the job control user guide, UP-9986 (current version) for a description of statements within embedded data.

When job control reads the job stream, it stores the embedded data in compressed form in the job's run library file: YRUN. During the execution of the job step, YRUN is read into main storage and may be accessed by instructions available in all high-level languages. Each requested record is expanded to its original form and stored in an input area you specify.

Except for PARAM statements, each retrieved record is an exact image of the source statement, which may be from 1 to 128 bytes. Thus, you can read 80-byte images from punched cards or 128-byte images from diskette.

NOTE:

Although PARAM and other job control statements may be handled as part of an embedded data set, they must still observe the job control statement conventions. Remember that job control statement information cannot extend past character position 71, and that position 72 is used to indicate continuation of a statement.

The control stream reader is always included in your system.

2.9. DISK AND DISKETTE SPACE MANAGEMENT

Space management comprises a group of routines that provide an efficient and completely automatic disk and diskette space accounting capability. These routines relieve your programs of the responsibility of knowing the precise contents of disk and diskette volumes. These routines also resolve competing demands for space allocation and establish standard interfaces with job control, utility, and service programs.

Space management routines perform the following functions:

- Allocate files
- Extend files already allocated (disk only)
- Scratch files that are no longer needed
- Rename files (disk only)
- Obtain label and extent information

Space management is strictly a supervisor function and so is not available to your high-level language programs. It is always included in your system for every disk or diskette you select at system installation time.

2.10. SYSTEM ACCESS TECHNIQUE

The system access technique (SAT) is a specialized block level device handler provided for both disk and magnetic tape files. Any interface between your high-level language programs and SAT is handled automatically, relieving you of the responsibility of managing SAT files. SAT is always included in your system.

2.11. MULTIPLE-INDEXED RANDOM ACCESS METHOD

The multiple-indexed random access method (MIRAM) is a disk access technique that lets you process a single disk file in several different ways. All disk files created by your high-level language programs are MIRAM files, and all language processors generate code that automatically handles file organization, relieving you of that chore. Since MIRAM is a data management function, refer to the *Consolidated Data Management Concepts and Facilities Manual,* UP-9978 (current version), for more information.

2.12. MULTIJOBBING

With its multijobbing features, the OS/3 supervisor can process concurrently up to 14 jobs for models 3, 4, 5, and 6 or 48 jobs for models 8, 10, and 20, each job consisting of one or more job steps that are executed serially. As mentioned in 1.2, a switch list allocates processor time to these jobs based on task priorities (each step of each job representing a primary task), synchronization, and I/O usage. While one task is awaiting the completion of an external event (such as completion of an I/O request), the supervisor activates another task that is ready, thus making maximum use of the processor. Since most programs require support other than merely processing instructions, multijobbing provides you with an effective method to reduce processor idle time and increase system productivity.

Every job step submitted to OS/3 is established as a primary task. The switch list has the capacity to allow you to specify up to 60 levels of processing priority for tasks. The maximum number of task priority levels that the supervisor will recognize is established at system generation time. The technical limit is 60; however, a more practical number of 3 to 15 is sufficient to achieve a high degree of processor utilization. When a task is interrupted to perform external processing (external to the instruction processor), it frees the processor, and OS/3 searches the switch list for the highest priority task that is not waiting for an external event to be completed. This task could be in the same job or it could be from any other job currently being processed.

Multijobbing is always included in your system, and system installation options let you control the relative priorities of user and some system tasks.

2.13. MESSAGE DISPLAY AND LOGGING

Successful operation of a computer system requires frequent communication. You use job control statements and assembler instructions to tell the CPU what to do, and how and when to do it. The operating system tells the operator what to do and tells you what was done and when. The operator gets a message from the supervisor (or from you) and answers a question or performs an action.

OS/3 provides several methods by which you can communicate with the operating system and with the console operator. These consist of a system log, display to the operator, and display to the workstation, which can be used singly or in combination.

A system log file is maintained by the supervisor spooling function. Job logs are subfiles of the system log file and receive all log and accounting information for the job, including messages you write to the log by using instructions in your program. Other logs maintained by spooling include a console log for the console workstation and additional logs, one for each active workstation.

You can display a message to the operator at the system console. The message may be for information only, or you may request a reply by the operator. Also, you can combine a log entry and a display. In this case, the message displayed and any reply from the operator are written to the system log and also to a console log if one is configured at system generation. In addition, you can display data to or receive data from a workstation.

Not all of these facilities are available to all high-level language programmers; see 3.2.11 for details.

2.14. INTERACTIVE SERVICES

In addition to the message display facilities discussed in 2.13, OS/3 provides you with the means to make even more extensive use of system workstations through interactive services. These services are discussed in the *Interactive Services Concepts and Facilities User Guide/Programmer Reference,* UP-9972 (current version). Briefly, these services include:

- Interactive job control
- Interactive data utilities
- General editor
- Screen format services
- Interactive command set

2.15. SPOOLING

Spooling is the technique of buffering data files for low-speed input and output devices to a high-speed storage device independently of the program that uses the input data or generates the output data. Data from card readers or from remote sites is stored on disk for subsequent use by the intended program. Data output by the program is stored on disk for subsequent punching or printing. The spooling function also handles diskette files. It treats input from diskette as though it were from a card reader and output to a diskette as though it were a card punch. In this description of spooling, any reference to a card reader, card input, or card file also includes diskette input; any reference to a card punch, card output, or card file also includes diskette output.

Spooling enhances system performance:

- by releasing large production programs and system software from the constraint of the slower speed devices, thereby freeing the main storage occupied by these programs sooner; and

- by driving the slower speed devices at their rated speed on a continuous basis, thereby making full use of the devices during the time that is normally lost to systems overhead or to job steps not using printers.

In addition to buffering data files, spooling maintains, for each job in the system, a job log that contains log and accounting information for the job. Spooling also maintains individual logs for the console workstation and each additional active workstation in the system; each of these logs records the system mode communications between the system and a console or workstation during a work session.

The spooler is the hub of the spooling facility and is a part of the supervisor. It provides record level input and output to and from the spool file for each element in the system needing access to that file. It intercepts all input/output commands to the virtual printer, punch, and card reader (which, to your programs, look no different from their real counterpart devices) and accesses the high-speed storage device, the disk, when necessary. Besides handling program input and output spooling, it also holds the system log, which contains job and accounting records for each job in the system.

Besides the spooler, the spooling facility also includes:

- the spool file, on which the spooler stores all spooled input and output;

- the input reader, the spooling element that reads input to be spooled from a card reader; and

- the output writer, the spooling element that writes spooled output from the spool file to a printer, or card punch, and writes print or punch output to a tape, disk, or diskette for temporary storage (redirected output).

Spooling is a supervisor option, and at system installation time you may specify options associated with it. For more information on spooling, refer to the *Spooling and Job Accounting Concepts and Facilities Manual,* UP-9975 (current version).

2.16. PRINTERLESS SYSTEMS

When the OS/3 system has a spooling facility, it can be configured and used without a physical printer. To achieve this, supervisor initialization is modified to recognize an "indirect" printer. Rather than producing output to a printer, such a system generates printer files. These printer files can be used for printing on an OS/3 system that does have a physical printer. For more information, refer to the *System Installation User Guide/Programmer Reference,* UP-8839 (current version).

2.17. JOB ACCOUNTING

The job accounting package consists of resident routines that are linked with the supervisor and elements of the job step processor at system installation time. These routines provide a count of the facilities utilized by each job step during its execution within the system. The message logging facility of the spooling function transfers this data from main storage to disk as part of the output spoolfile. The output writer prints the job step and job values as part of the normal message log output for each job. Optionally, the output writer can write the accounting information to a standard magnetic tape file or SAT disk file for offline processing by user-developed accounting routines or by OS/3 data utility routines. You can assign an account number by using the JOB job control statement that is carried along with the accounting records. This enables you to accumulate statistics from the disk or tape file for computer time and resources charged against an account number, which could represent a project, department, cost center, etc.

Data collected by job accounting for each job step and for the entire job includes the time of usage for the central processing unit, the wall clock duration of the job steps, the number of SVC calls, the amount of main storage allocated and actually used, the number of transient calls, and the number of device calls. Job accounting is a part of spooling, and so is available only when spooling is configured.

2.18. DIAGNOSTIC AND DEBUGGING FACILITIES

The supervisor has several facilities to help you find and remove errors from your programs. These occur in the following categories:

- Main Storage Displays

  These include snapshot dumps of selected main storage locations as well as dumps of whole job regions (EOJ dumps and JOBDUMPs) and even the whole of main storage (SYSDUMP). The snapshot dumps are available to some high-level language users; the job and system dumps, to all users. System dumps are an option you can include at system installation time, while job region and snapshot dumps are always included.

- Checkpoint/Restart

  Hardware and software malfunctions can cause your job to terminate before its normal completion. Another reason for termination could be that the operator cancelled your job because a high-priority job required all the facilities of the computer. If the job is small, you can rerun it without any really great loss. But, what if it is a long or complex job, where rerunning the job could increase both processing time and cost, thereby reducing productivity? OS/3 has provided the checkpoint facility, which allows you to periodically generate records containing the operational status of your job. Each such record is called a *checkpoint* and is written to a checkpoint file on disk, format label diskette, or magnetic tape. Should a job step fail, you can use the last checkpoint written before the failure to restart the job at the point where the checkpoint was taken rather than starting the entire job all over again.

You might want to create a checkpoint record at some specific occurrence, such as the end of a magnetic tape reel in a multivolume input file, or after processing a specific number of records.

The capability to generate checkpoint records is a function of the supervisor, and the capability to use these checkpoint records to restart a job is a function of job control (through the RST job control statement).

The checkpoint facility is available only to COBOL high-level language programmers. It is always included in your system.

- Monitor/Trace

  One means of debugging a program is the monitor/trace supervisor facility. This routine monitors each instruction in a program before and after it is executed and then signals if an event has occurred, such as:

  – a specified main storage location has been accessed; or

  – a specified location has been reached.

  How the supervisor reacts to these events is determined by the high-level language that calls the monitor. You may get a printout of the locations you specify and the option of continuing the program or terminating it. Not all the monitor's capabilities are available to all programs; see 3.2.13 for details. The monitor/trace facility is always included in your system.

- System Debugging Aids

  The supervisor has several debugging aids that can help identify system problems a system dump by itself cannot uncover. These include a supervisor debug option that monitors the entire operating system; a console debug option that halts processing on an I/O, transient, or loader error; transient, symbiont, or shared code halt routines that halt whenever a specified transient, symbiont, or shared code module is loaded; and a soft-patch symbiont that applies temporary patches to transients, shared code modules, object modules, load modules, and the resident supervisor. All these debugging aids require some knowledge of how the supervisor operates internally. All of them are always included in your system.

2.19. RESOURCE MANAGEMENT

The resource management facility is a supervisor feature that dynamically tunes the system environment. This is a versatile facility that provides a way to balance the system use among various batch and interactive activities.

Resource management permits the console operator to control system use (e.g., memory assignment, maximum number of interactive users, and the maximum number of batch jobs) through various resource parameters; these parameters are specified during the system generation process. Commands are available that allow the console operator to change these values during the operation of the system.

2.20. SECURITY

Security is a supervisor feature that includes:

- Security Maintenance Utility

  Enables the security administrator to create, display, modify, and delete user execution and command profiles. This ability allows the security administrator to define and limit the access of interactive users to the system.

- Security Logon Service

  Used to identify your attempt to gain access to the system. Logon denies system access to those interactive and batch users not authorized by a process that checks the system dictionary for valid user-ids, passwords, and account numbers. (Both password and account number input are system generated.) Once you have access to the system, logoff will end your session.

Refer to *Security Maintenance Utility User Guide,* UP-12028 (current version) and to *Interactive Services Commands and Facilities User Guide,* UP-9972 (current version).

2.21. SHARED CODE MANAGEMENT

Many run-time modules run as shared code, which means that one module can be used by several jobs simultaneously. This feature not only saves main storage space but also saves the time it would otherwise take to load a separate copy of the module for each job that needed it. All shared code modules are provided by Unisys and are included in the library file YSCLOD on your system resident volume. These modules include screen format services, interactive services, data management, the general editor, the RPG editor, BASIC, and other system programs.

Shared code management is available for system programs only, not for application programs. The system administrator can, at his discretion, make certain shared code modules, or groups of modules, resident. A list of resident modules can be specified at system generation time, and this list can be modified at each subsequent IPL (initial program load). At IPL time, the supervisor allocates main storage for the modules in the modified list, loads them, and makes them resident. Once a shared code module becomes resident, it remains resident until the next IPL.

Shared code management is always available in your system. System installation parameters are available to modify shared code for improved performance; for more information on specifying resident shared code modules at IPL time, see the *Operations Handbook Operator Reference,* UP-8859 (current version).

2.22. DYNAMIC BUFFER MANAGEMENT

The dynamic buffer management facility is a supervisor feature that can dynamically allocate extra main storage to system routines needing it. With this facility, system routines need not wait to obtain buffers that they need to continue processing. This in turn enhances the performance of your user programs, especially those using interactive services.

Because dynamic buffer management is limited to system routines, user programs have no direct control over it. You can, however, specify parameters at system installation or IPL time for its most efficient operation.

2.23. SYSTEM MONITORING FACILITIES

Unisys makes available to you facilities for recording hardware malfunctions and monitoring your system's normal activity. These facilities are the error log and the system activity monitor.

The error log is a file on the system resident volume (identifier YELOG) that contains records of hardware errors kept for the statistical and historical use of Unisys customer engineers. When a hardware error occurs, the error logging function, which is a part of the supervisor, stores pertinent information in the error log. (Models 3, 4, 5, and 6 require at least three resident error log buffers, while models 8, 10, and 20 require at least six resident error log buffers.) Unisys customer engineers can read these records from the file into main storage for processing and storage as permanent records, using these records in maintaining customer equipment. Error logging is always included in your OS/3 system.

Another useful system monitoring facility is the system activity monitor (SAM), a system symbiont that monitors and records your system's activity. It is intended for use by the system administrator and installation manager to aid in the detection of production bottlenecks, to optimize production job mixes, and to identify and change system variables that influence system performance.

For more information on the system activity monitor, see the *System Activity Monitor User Guide/Programmer Reference,* UP-9983 (current version). You may optionally include the system activity monitor in your system at system installation time.

3. Supervisor Features and High-Level Languages

3.1. GENERAL

This section outlines those supervisor features that you can take advantage of through the high-level languages OS/3 provides. In the following discussion, each feature has a list of its availability in the following high-level languages:

- COBOL
- FORTRAN
- RPG II
- BASIC
- ESCORT
- JCL

For any feature, only those languages are listed that make use of that feature along with the specific instructions or directives needed. Refer to the Preface for a complete list of appropriate manuals with more information about a feature.

The COBOL compiler conforms to the specifications of the *American National Standard COBOL, X3.23–1974* and contains extensions, many based on supervisor facilities, that enhance the capabilities of COBOL beyond the basic requirements of the standard. The FORTRAN compiler accepts FORTRAN IV, which includes the *American National Standard FORTRAN X3.9–1966* and the IBM System 360/370 DOS FORTRAN IV languages as subsets. JCL is included in this discussion together with the languages because it, too, takes advantage of many supervisor features.

3.2. SUPERVISOR FEATURES AND LANGUAGE USE

3.2.1. Program Initiation and Loading

- COBOL

  Available through the CALL instruction.

- FORTRAN

  Available through the CALL FETCH and CALL LOAD instructions.

- BASIC

  Programs residing in an OS/3 library file may be loaded into your BASIC work space prior to execution by means of the OLD command, or dynamically loaded at run time using the CHAIN statement. A library file may be added to the contents of the work space with the MERGE command.

- JCL

  Used with the // EXEC statement

3.2.2. Program Termination

- COBOL

  Normal termination of dynamically loaded programs is available through the CANCEL instruction. Abnormal termination by the user is not available.

- FORTRAN

  Normal termination is available through the CALL EXIT instruction (without a dump) or the CALL DUMP instruction (with a dump). Abnormal termination by the user is not available.

- ESCORT programming language

  Normal termination from ESCORT language is available by use of the EXIT selection on the menu; this returns control to interactive services. Abnormal termination by use of the EXIT selection is also available if ESCORT language is run in supervisor debug mode. In this case, you are given the option of obtaining a system dump.

- JCL

  Program termination is not a JCL function, but JCL allows you to generate one of several types of dumps if abnormal termination occurs, no matter what language the program was written in. For the dumps, you can use the // OPTION DUMP, // OPTION JOBDUMP, // OPTION SYSDUMP, // OPTION GDUMP, // OPTION GJOBDUMP, and // OPTION GSYSDUMP statements. In addition, you can specify // OPTION statements that generate a program check if binary overflow (BOF), decimal overflow (DOF), significance (SIG), or exponent underflow (XUF) errors should occur.

3.2.3. Timer Services

- COBOL

  You can access the system's calendar date, Julian date, and time by using the ACCEPT statement.

- RPG II

  You can access the system's calendar date and time by using the TIME operation.

- BASIC

  You can set a processor time limit for your currently running program with the TIME statement, and get the elapsed running time for a program with the TIME function. In addition, you can access the system's calendar date with the DAT$ function and the current time of day with CLK$ function.

- ESCORT programming language

  You can access the system's calendar date using the DATE$, DAY$, MONTH$, and YEAR$ utility fields. The current time is available by use of the TIME$ utility field.

- JCL

  You can set a maximum execution time for your job by using the max-time parameter in the // JOB statement. Within the WRTBIG job procedure, you can print the date and time your job was run. And, to temporarily change the system's calendar date until the end of a job, you can use the // SET DATE statement.

3.2.4. Program Linkage

- COBOL

  You can call a subroutine that is linked to your program and pass data to and from it by using the CALL statement.

- FORTRAN

  You can call a subroutine or function linked to your program and pass data to and from it. For a subroutine, you use CALL and SUBROUTINE statements in the calling and called programs, respectively. For a function, you use the function name and FUNCTION statements, respectively.

- BASIC

  You can call a subroutine with the CALL statement and define parameters to be passed with a SUB statement entered as the first statement of the subroutine. The subroutine can reside in the calling program file or it can reside in an OS/3 library file named with the LIBRARY statement.

- RPG II

  You can call an internal subroutine, one associated with the calling object module, by using the EXSR, BEGSR, and ENDSR operations. You can call an external subroutine, one compiled separately from, but linked to, the calling program, by using the EXIT, BEGSR, and ENDSR operations. To pass data to and from an external subroutine, you use the ULABL and RLABL operations.

3.2.5. Island Code Linkage

- FORTRAN

  You can write routines to which program control passes if two types of program exceptions are found. These are CALL OVERFL for an arithmetic overflow or underflow and CALL DVCHK for a divide check. Under no other conditions can you write or access any island code.

3.2.6. System Information Control

- COBOL

  You can change or test the UPSI byte either as a whole or bit by bit by using the SYSSWCH clause. You can also change or test the entire communication region by using the SYSCOM clause.

- FORTRAN

  You can test portions of the UPSI byte by using CALL SSWTCH. The remainder of the communications region is inaccessible to your program.

- RPG II

  You can test or change the UPSI byte by using indicators U1 through U8 in your program. You cannot change or test the remainder of the communications region.

- JCL

  You can change the UPSI byte with // SET UPSI and the communication region with // SET COMREG. In addition, you can test the UPSI byte by using // SKIP.

3.2.7. Control Stream Reader

- COBOL

  You can access the control stream by using the ACCEPT statement and the SYSIN clause.

- FORTRAN

  Most card input to your programs comes through the control stream reader. To access it, you use the READ statement for I/O units 1 or 5.

- RPG II

  You can access the control stream reader with a file description specification statement specifying device type CTLRDR.

3.2.8. Disk and Diskette Space Management

- COBOL, FORTRAN, RPG II

  You cannot directly use the space management facility in a user program; this facility is a data management function. See the *Consolidated Data Management Concepts and Facilities Manual,* UP-9978 (current version), for more information.

- JCL

  You allocate and extend space for a disk or diskette file by using the // EXT statement.

3.2.9. System Access Technique

- JCL

  You can allocate space for a SAT file by using the // EXT statement. You will usually use this statement to allocate files that are solely managed by system programs. Examples of these are user library files handled by the SAT librarian and checkpoint files handled by the checkpoint/restart function (3.2.13).

3.2.10. MIRAM

You can access MIRAM files from your programs. For more information, refer to the *Consolidated Data Management Concepts and Facilities Manual,* UP-9978 (current version). BASIC can read a MIRAM file written in any format but will write only unkeyed records; see the current version of the *BASIC Programmer Reference,* UP-9168, for more information.

- JCL

  You can allocate space for MIRAM files by using the // EXT statement.

3.2.11. Message Display and Logging

- COBOL

  You can send messages to the system console by using the SYSLOG clause and send messages with operator replies by using the SYSCONSOLE clause. In addition to the above facilities, you can display messages on workstations by using one of the following clauses:

  - SYSTERMINAL – for the master workstation
  - SYSWORK – for the workstation assigned to your program through JCL
  - SYSFORMAT – for the workstation associated with screen format services

- FORTRAN

  For a STOP or PAUSE instruction, you can specify a decimal integer that is displayed on the system console when the instruction is executed. PAUSE displays the integer and stops program execution until the operator continues it; STOP displays the integer and terminates the program.

  FORTRAN allows you to treat a workstation as an I/O device; consequently you can move data into and from it with the usual FORTRAN statements.

- RPG II

  You can send information to or get information from the system console by using the DSPLY operation and a file description specification statement with the device name CONSOLE. With DSPLY, you can either display a message and continue processing or display a message and cause a halt until the operator resumes processing. During the halt, the operator may enter data that the program accepts when it resumes processing.

  You can send data to or receive data from a workstation by using screen format services and a file description specification statement with device name WORKSTN.

- JCL

  Your job can send messages to the system console or to any workstation. If you want an operator reply, you use // PAUSE; if not, you use // OPR.

3.2.12. Spooling and Job Accounting

- COBOL

  No program modifications are needed to use input and output files with spooling configured in your system. With spooling, you have the additional capability of copying SYSLOG and SYSCONSOLE messages (see 3.2.11) to your spool file, from which they can later be printed or even stored on disk or magnetic tape. In addition, you can output your own data, independent of any console display, to the job log for your job by using the SYSLST clause.

- FORTRAN

  With spooling, console messages displayed through the STOP or PAUSE instruction (see 3.2.11) may be copied to your spool file for later output to printer, disk, or magnetic tape. As for input or output files used by your programs, no modifications are needed to use spooling.

- RPG II

  With spooling, console messages and responses displayed through the DSPLY operation (see 3.2.11) may be copied to the spool file for later output to printer, disk, or magnetic tape. As for input or output files used by your programs, no modifications are needed to use spooling.

- BASIC

  With spooling, you can run BASIC in batch mode by spooling in a card file containing exactly the statements you would use in an interactive BASIC session.

- ESCORT programming language

  Spooling with ESCORT language involves only printer output. No modifications to your ESCORT program are needed to take advantage of spooling with printer output files.

- JCL

  The spooling facility stands at the center of several powerful JCL statements. You can, for example, use // SPL to output program data to a spool file on disk or diskette, // DST to send spool output to a remote device, or // DATA to load card images to the spool file for later input to a program. In addition, the parameters in the // JOB statement help you control spool output.

3.2.13. Diagnostic and Debugging Aids

- COBOL

  You can use the following supervisor features in your COBOL program:

  – You can get an EOJ dump, JOBDUMP, or SYSDUMP if your program goes through abnormal termination (see the JCL description).

  – You can use checkpoint/restart with the RERUN clause.

  – You can use debugging statements to monitor data items or procedures while your program is running.

- FORTRAN

  You can get an EOJ dump, JOBDUMP, or SYSDUMP if your program goes through an abnormal termination (see the JCL description). In addition, FORTRAN lets you include a variation of the PAUSE instruction in which an operator response of DUMP terminates the program with a dump.

- RPG II

  You can use the DEBUG operation to tell you what specified indicators and fields contain at various points in your program. You can get an EOJ dump, JOBDUMP, or SYSDUMP if your program terminates abnormally. In addition, you can tell RPG II to output a formatted error analysis dump if an error occurs in your program. Run-time facilities let you display or change data while the program is executing. The operator control feature lets the system operator take action if your program sets any halt indicators; the operator can continue program execution, bypass the remainder of the program cycle, or terminate the program altogether.

- BASIC

  BASIC facilities like the syntax checker handle program and run-time errors dynamically during the BASIC interactive session rather than through the supervisor.

- ESCORT programming language

  ESCORT facilities handle program errors dynamically during the ESCORT interactive session rather than through the supervisor. These facilities display error messages prior to run time, warning of illogical conditions in the program being coded.

- JCL

  You use the JCL options DUMP, JOBDUMP, or SYSDUMP to specify the type of dump to be taken should your program terminate. In addition, you use the // RST job control statement to rerun a COBOL program from a checkpoint.

4. Choosing an OS/3 Supervisor — Ours or Yours?

So far we've shown you what makes up a supervisor and how you can use its features in your own programs. Before you can use a supervisor, though, you have to have one to use. And you've got a choice:

- Your OS/3 system comes with its own ready-to-use supervisor, named SY@BAS (models 3, 4, 5, and 6), SY#BAS (model 8), or SY$BAS (models 10 and 20). With it you can usually begin normal system operations immediately. In fact, the IPL (initial program load) procedure that you use to start up your system calls this supervisor unless you specify another.

- You can generate your own supervisor embodying some or all of the features we've described. Supervisor generation is done by the SUPGEN phase of system installation, under the control of parameters you specify.

You may be wondering whether to use our supervisor or generate your own (or more than one; that's possible too with system installation). SY@BAS, SY#BAS, and SY$BAS have been designed to meet most processing requirements at an average OS/3 site. The ready-to-use supervisor, however, may be too big or too small for your needs: too big in having features you don't ordinarily use and can do without; too small to satisfy special requirements you may have.

In either case, you may want to generate your own supervisor. SUPGEN parameters tailor a supervisor to your needs. Some parameters add modules that increase the main storage requirements of your supervisor, while other parameters merely set defaults for the supervisor to follow when it is in main storage and functioning.

The *System Installation User Guide,* UP-8839 (current version) provides details on the three ready-to-use supervisors, all of the SUPGEN parameters, and the entire system installation process.

UNISYS

USER COMMENTS

We will use your comments to improve subsequent editions.

NOTE: Please do not use this form as an order blank.

_____

(Document Title)

_____ _____ _____

(Document No.) *(Revision No.)* *(Update Level)*

Comments:

From:

_____

(Name of User)

_____

(Business Address)

Fold on dotted lines, and mail. (No postage is necessary if mailed in the U.S.A.)
Thank you for your cooperation

CUT

FOLD

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

Unisys Corporation
E/MSG Product Information Development
PO Box 500 C1-NE6
Blue Bell, PA 19422-9990

FOLD