IBM

Washington
Systems
Center

Technical
Bulletin

JES3 A Primer

By J. Brown

Edited by The Washington Systems Center

DAPS Code 0894
GG22-9200-00
June 1980

JES3 A Primer

By J. Brown

Edited by The Washington Systems Center

DAPS Code 0894
GG22–9200–00
June 1980

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either express or implied. The use of this information is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment.

Requests for copies of this and other IBM publications should be made to your IBM representative or to the branch office serving your locality.

A form is provided in the back for any comments or criticisms you may have. If the form has been removed, comments may be addressed to IBM Washington Systems Center, Field Support, 18100 Frederick Pike, Gaithersburg, MD 20760. Comments become the property of IBM.

Washington Systems Center
Gaithersburg, Maryland
Technical Bulletin

JES3 - A PRIMER

Written by J. Brown

Edited by Washington Systems Center

DAPS Code 0894
GG22-9200-00
June, 1980

This publication is designed to serve as a learning aid for those who have a need to expand their knowledge of JES3 and its relationships to MVS.

The author assumes that the reader has, at a minimum, an introductory level background in MVS. It is not the intent of this publication to provide a detailed description of JES3 or its components. The most valuable use of this material will be as a supplementary text for the standard JES3 courses offered by the IBM Advanced Education Centers.

Chapter 1, a short history of JES3, was extracted from other IBM publications and is included only to give the reader an appreciation of the many evolutionary changes through which multiprocessor systems have come.

Many readers will find Chapter 2 to be the most informational of the chapters, since its purpose is to introduce many of the JES3 features. For some, whose involvement with JES3 will be at a minimum, reading beyond Chapter 2 will be unnecessary. Readers who require more detailed information are referred to OS/VS2 MVS System Programming Library: JES3 (GC28-0608). Coding details for JES3 control statements can be found in OS/VS2 JCL (GC28-0962).

Comments and constructive criticisms which may be used to improve the publication are welcomed by the author.

TABLE OF CONTENTS

Chapter 1

Development of Job Entry Subsystems

Introduction

The scope of data processing and its degree of involvement in business and industry have changed dramatically during the past two decades. Very large computer complexes, such as we know today, utilizing complicated software have grown through many evolutionary stages.

During these twenty years of rapid growth and expansion, many installations discovered that single processors, even very large ones, simply were not capable of handling their ever-increasing workload. An obvious approach, then, was to install second, third, and subsequent processors. The addition of processors, with the attendant operational and work scheduling problems, sometimes presented real challenges. Since many of the computing systems being added were not symmetric to those already installed, requirements for new approaches to managing large installations began to surface.

Rapidly changing hardware and software facilities dictated the need for some type of "external," reasonably consistent interface to the often radical "internal" differences. Any viable new approach to effective resource utilization across computing systems would ideally provide a single job scheduling process built into the operating systems of the day. Additional design requirements were such things as:

- the ability to support the intermixing of different operating systems.
- semiautomatic (at minimum) centralized operations with external control when necessary.
- a single job queue to be shared among the multiple processors via some sort of scheduling interface.

Let's follow some of the highlights in the development of the subsystem we know today as JES3.

Tape Oriented Systems

Job Entry Subsystems became known during the late 1950's and early 1960's. The larger installations recognized the economics of using a relatively small (and less expensive) computer to prepare work for their large processors. Jobs

were batched onto input tapes, often by IBM 1400 systems, and then processed by the larger systems. At completion of a batch of work, output tapes from the large system were returned to the smaller system for printing. One fairly significant innovation was the sharing of tape drives, which were then dial-addressable, between the two processors. This implementation removed the need for moving the tapes from one machine to the other. These tape oriented systems, while relieving the larger computers of the unit record processing, generated new scheduling problems. With the input and output on tape, and the job queue external to the processing system, operations personnel had little control over the order of execution of jobs in the stream.

Direct Coupled System

Loosely coupled processors were seen in 1962 as a possible means of overcoming many scheduling difficulties as well as retaining the ability to insure that large, expensive computers were more fully utilized. The Direct Coupled System (DCS) included an IBM 7040/7044 (to prepare and manage the job queue) and an IBM 7090/7094 (for job processing). The two machines were coupled through selector channel and memory bus connection. Many DCS configurations even attached application I/O devices to the 704X, due to lesser expense than having them on the 709X. Requests for I/O were routed from the 709X to the 704X which then wrote or read directly into or out of the 709X memory. The 709X software, known as IBSYS, did not support multiprogramming as we know it today. Consequently, tape mounts idled the entire system until the mount was completed. Pre-execution setup (the mounting of required volumes prior to job selection and selecting jobs only after completion of the mounts) greatly reduced this idle time. The front-end processor, the 704X, also managed the input and output unit record equipment, performed prioritized job selection, and essentially automated operation of the back-end processor, the 709X. Now that the job queue was an internal one, operations personnel could control or alter the sequence of job processing.

As time passed and data processing needs increased, other features were added to DCS. A remote job entry facility, using the Synchronous Transit/Receive Access Method, allowed remote users to feed job streams to the front end processor and receive output of their jobs. The advent of simultaneous peripherial output online (SPOOL) permitted additional scheduling flexibility for output since several printers could be attached to the 704X.

HASP/ASP Systems

1964 brought new hardware, the IBM System/360, and new software, Operating System/360. A study, conducted to determine the need for a follow-up system to DCS, concluded that though some of the factors which had led to the development of DCS no longer applied, a follow-up system was still required. During these evaluations, two independent groups designed new approaches to the solutions offered by DCS. Two different systems resulted; the Houston Automatic Spooling Processor (HASP) and the Attached Support Processor (ASP). HASP initially operated on a single processor with the purpose of eliminating a great deal of the job scheduling overhead which was inherent in the early releases of OS/360. More closely resembling DCS, ASP was to support multiple processor installations by providing the features of DCS, augumented by the new hardware and software facilities.

Development of ASP began in 1964, and the first version was released in 1966. Like DCS, ASP was implemented on a combination of front-end and back-end processors using, however, a channel-to-channel adapter (CTC) as the means of connection. Execution devices were attached only to the back-end processor. Front-end processing was provided by a System/360 Model 40 or larger, and a Model 50 or larger served as the back-end processor. The front-end and back-end processors became known as the support and main processors, respectively. The following facilities were managed by the support processor:

- primary inputting and outputting of jobs and management of associated tape and unit record equipment.
- pre-execution setup of volumes required for job execution with the devices now attached to the main processor.
- job selection and scheduling.
- operator communications for both systems.

The first release of ASP had both processors operating under control of OS PCP, a non-multitasking system. Therefore, to provide multiple synchronous facilities, a pseudo multitasking structure was built into the support processor's programming scheme. Each component routine of ASP was called a Dynamic Support Program (DSP), with the executions of DSPs being defined as functions. Each active

function was represented by a Function Control Table Entry (FCT), roughly analogous to OS's task control blocks. The ASP dispatcher, known as the multifunction monitor (MFM), used the queue of FCTs to represent potentially dispatchable units of work. MFM defined processing events in such a manner that they were not necessarily dependent upon hardware interrupts. Functions, then, could be dispatched, in sequence, as long as any function could perform useful work. An OS WAIT was only involved when no function was found by MFM to be dispatchable. Hardware interrupts were "filtered" by ASP code to determine whether ASP's services were required. This filtering was performed by code entered via modifications to the I/O, SVC, external, and program check PSWs. When ASP's services were required, as the result of an interrupt, the filter code OS POSTed ASP, causing entry to the MFM, and subsequent dispatching of some ASP function.

In the period of 1967-1973, ASP, not unlike its predecessor DCS, was designed for those installations generally classified as "scientific" shops. ASP Version 2 provided support for up to three main processors in 1968 and included facilities to attempt balancing of workload across the complex. A device management scheme, named Main Device Scheduling or MDS, maintained awareness of use of devices shared among the main processors. Six releases of ASP Version 2 added capability, but not until Version 3, in 1973, was the system architecture substantially changed.

With Version 3 of ASP came support for as many as thirty one main processors, and ASP began to take on the characteristics of a commercial system. A facility for maintaining the disposition status of volumes was added to MDS. This facility enchanced the scheduling of jobs, with compatible data set dispositions, across the complex.

Job Entry Subsystems

IBM announced OS/VS2 Release 2 (MVS) for System/370 in January, 1973. The design of MVS, implementing a multiple address space environment, enhanced the virtual storage concepts introduced with SVS (VS2 Release 1). Because HASP and ASP had been widely accepted as add-ons for MVT and SVS, it was decided to create similar systems for MVS. The follow-ons to HASP and ASP were, however, to become integrated components of MVS.

A number of components in MVT and SVS had proven to be system bottlenecks, and one of the major bottlenecks was, for both HASP and ASP, the addition of another job queue to

the operating system environment. Since MVS involved a major reevaluation of control program design, the decision was made to improve intercomponent communications and system performance. MVS was structured around the concept of subsystems to perform services for address spaces which required those services; the mechanism to provide this communication was formalized into the subsystem interface (SSI).

One of the several subsystems required for MVS came to be known as a JOB Entry Subsystem (JES) and was to be responsible for inputting and outputting work to and from the system. Studies of possibilities for JESs resulted in the conclusion that requirements for users of HASP and ASP were different enough to necessitate replacement of each of the two systems. The JES which provided HASP-like function was named JES2, and ASP-like processing was called JES3. Both JES2 and JES3 would provide the basic functions of a subsystem, but, in addition, each would be designed with additional facilities to specifically address the requirements of its respective user community. JES2 provided a shared, multiaccess spool and a common job queue. Each processor, however, continued to perform its own job scheduling and resource management independently of the other processors. JES3, on the other hand, was to have a design philosophy much like that of ASP - a loosely coupled multiprocessor system with centralized job management, resouce management, a shared spool, and awareness of work in all the processors which constitute the complex. Also, in order to provide a migration path for ASP users moving to MVS/JES3, JES3 provided continued support for processors running as ASP mains, using MVT or SVS.

Summary of JES3 Features

As stated earlier, MVS requires a job entry subsystem to perform any useful work. IBM provides two products, JES2 and JES3; in addition, the installation may provide its own. MVS has a formally defined subsystem interface which it uses to communicate with JES. What the JES does in response to requests made by MVS across the interface depends upon the design of the JES. JES3 has many functions and, therefore, uses most of the job entry subsystem interface points supported by MVS. Some of the major functions provided by JES3 to manage a multiprocessor complex are:

- introduction of jobs to the system from any of several sources, both external and internal.

- job selection based upon any of several installation provided algorithms.

- maintenance, complex-wide, of the integrity of devices, data sets and volumes.

- operational control of the complex through functionalized consoles - local, remote, and TSO.

- passing spooled - in-stream data to a running program.

- accepting and spooling SYSOUT data sets to local devices, remote workstations, or TSO users.

- user exits from the JES3 code to provide the ability for the installation to include additional features it desires.

- installation or user defined enhancements to job scheduling in the form of Deadline Scheduling and Dependent Job Control.

- operator controlled utility functions, including many debugging aids.

- collection of accounting data.

- maintenance of system log and journal data sets.

- provision of system recovery facilities at the system or DSP levels.

NOTE:

Throughout the balance of the text, the right margin contains important words and phrases with which the reader should become very familiar.

Also noted in the right margin are the references to diagrams located in the Appendix.

Chapter 2

Overview of JES3 Job Flow

In this chapter, we want to learn about the major activities which take place as a job flows through the JES3 environment. Many terms will be defined, features introduced, and concepts discussed. Our thrust will be toward JES3 and the work it performs for MVS/JES3 processors, rather than emphasizing the support for ASP main processors.

Definition of Terms

Our first chore, as we begin our study of JES3, necessarily becomes that of defining many of the terms we will be using. JES3 executes in an MVS address space, and in many respects, is somewhat similar to problem programs. The machine configurations we find in JES3 installations vary from one processor to, perhaps, four, five, or more. Usually associated with these processors are many I/O devices - it is not unusual to find installations with ten or more printers, perhaps sixty-four or more tape drives, and as many as one hundred spindles of DASD, with many or all of these devices shared between processors. One of the processors is designated as the controller of these complexes (even if there is only one processor). The "controller" has the responsibility of managing the system resources, scheduling of jobs that enter the complex, and presenting to the installation's user community a "single system image."

GLOBAL

LOCAL

MAIN

We call this controller the <u>global</u> system to distinguish it from the other MVS/JES3 processors which have lesser responsibility from a JES3 viewpoint. The other MVS processors are known as <u>locals</u>. Work is scheduled by global JES3 onto the same processor, onto the local processors(s), and in some installations onto ASP processors. We will use the term <u>main</u> to generically define all these processors-global, locals, and ASP mains. You will probably want to associate the execution of jobs with "main" since "global" and "local" are JES3-related terms. Local processors and ASP processors are attached to the global processor by channel-to-channel adapters which serve as communication paths. Communication between global

JES3 and the main which is physically housed in the same processor is carried out via MVS SRB scheduling. Now that we have defined the three major terms which refer to our environment, let's begin to look at several other important definitions. Some of the terms we use are "old" terms with new meanings, while others are completely new for anyone with no prior experience with ASP or JES3.

Most of us are already acquainted with the terms "job." This word has been traditionally used to define a set of work, identified by the presence of a JCL - JOB statement. Also, jobs have consisted of one or more steps, defined by EXEC statements. Though we will not change that definition, we are going to add to the list of items we call jobs in the JES3 environment.

There are several special cases for JES3 in which a job can come into existence without there being any JCL involved. This can happen as the result of a JES3 operator command, which we will see later. For now, we are going to recognize that JES3, like MVS, defines a job with a control block called a Job Control Table entry (JCT). Then, to be technically correct, we will define a job as "whatever JES3 has defined with a JCT entry." The work that JES3 performs on behalf of jobs is described quite differently from the way it was in earlier versions of the operating system.

JCT
JOB

JES3 does not break a job into steps (though we will continue to see job steps) as does MVS. Rather, JES3 stages its services on behalf of a job through something we call job segments or simply segments. A segment is defined as a logical portion of the work JES3 performs on behalf of a job. Let's try to put that definition into a meaningful perspective by discussing processing for a traditional type of job, i.e., represented by JCL.

SEGMENT

In order for a job of this nature to exist, JES3 must become aware of it. First, the job must somehow (and there are many means of doing this) be entered into the system. One set of work, therefore, simply consists of "reading" the job, and constructing a concise definition of that job to JES3, in the form of a JCT. Included in the

definition of a job are also the defintions of its JES3 segments. They are, for the typical job:

- an interpreter segment to cause the job's JCL to be interpreted in what appears to be an almost normal manner.

- a summary segment, called MAIN, which allows JES3 to do a number of things on behalf of the job:

  - acquire I/O resources (data sets, volumes and devices) needed by the job.

  - schedule the job to MVS so that the job can execute.

  - free up the I/O resources (when they become available), making them available for other jobs.

- an outputting segment, the processing for which is the handling of SYSOUT data.

- a cleanup segment to cause graceful removal of the job from the system.

Now, imagine a control block which defines the existence of a job (the JCT), having appended to it a set of small extensions, each one being the definition of a JES3 segment of work. These extensions are called scheduler elements. I 2.1

SCHEDULER ELEMENTS

It is very important to further discussion that the next statement be fully grasped. "JES3 does not really schedule jobs (in the MVS sense of scheduling jobs), it instead schedules JES3 job segments." It should be obvious that code is requred to perform the work defined by a job segment; hence, our next definition set.

The programmed routines which perform the work defined by job segments are called Dynamic Support Programs or DSPs. You may remember encountering that acronym in Chapter 1. So, when JES3 schedules a job or segment, the result is execution of one or more DSPs. The execution of a DSP is defined as a function. A function, for

DSP

FUNCTION

purposes of an analogy, is very much like an MVS task, but we do not have "mother" functions or "daughter" functions. The function is JES3's dispatchable unit of work defined by a Function Control Table entry. Those functions (and FCTs) which are not resident in the JES3 nucleus are created by the Job Segment Scheduler. JSS is responsible for all new work entering the system; its primary responsibility is the scheduling of the work defined by jobs' segments. The JES3 function "dispatcher" is named the Multifunction Monitor, as in ASP. You should now be familiar with

FCT

JSS

MFM

- global and local
- job and JCT
- segment and scheduler element
- DSP and function
- FCT and JSS
- Multifunction monitor and MFM

There are several classifications of jobs in JES3. One term you will hear often is standard job. A standard job is one defined in JES3's usual manner) completely by JES3 that is, with no help from the user who created the job's JCL. Standard jobs, as defined in the distributed JES3 system, consist of four segments: 1)Converter/Interpreter service, 2) Main Service, 3) Output Service, and 4) Purge. Note that should your installation include ASP mains, a job eligible to run on both ASP main(s) and JES3 main(s) would also contain a segment for scheduling the OS Reader/Interpreter.

STANDARD JOB

We implied, in the previous paragraph, that a user could help JES3 define the job. By coding special JES3 control statements in the JCL, the user can define the number and the order of the job segments for the job. Jobs for which this is done are classified as non-standard jobs.

NON-STANDARD JOB

There remain a few more terms to define, and their context relates to the several sources of jobs (referred to earlier). Normal jobs, which may be

NORMAL JOBS

either standard or non-standard, are those entered into JES3 from:

- locally attached devices - card readers, tapes, or members of a special PDS.
- remotely attached devices through the Remote Job Processing facility.
- Data sets containing job streams SUBMITted by TSO users.
- SYSOUT data sets containing job streams passed to JES3 via the MVS internal reader. **INTRDR**
- another global JES3, using a facility called Network Job Processing. **NJP**
- another node in the network using the JES3 networking PRPQ.

Much or most of the work in JES3 installations will consist or normal jobs of one form or other.

Another source of jobs, however, is the result of an operator command - these jobs we classify as called jobs, since the operator enters a CALL command to invoke the job. Called jobs are unique in that they are not defined by JCL; there is, in fact, no JCL involved at all. These jobs are actually internally generated by JES3 in response to the CALL command, and their JCTs always contain two scheduler elements - one to represent the DSP invoked by the operator via the call command and one for PURGE. **CALLED JOBS**

Our third and final source of jobs is that of the started tasks or tso LOGONs. In this case, the resulting job is termed a demand select job, because the request for the job is in the form of a specific job being demanded by initiator code (more about this later). **DEMAND SELECT**

Read a Jobsteam

In order for a normal job to be executed, it must be first read into the system. This is accomplished by functions that we know as readers. Readers exist in three forms (card reader, tape **READERS**

reader and disk reader), the only difference between them being the device dependent code necessary to read from different device types. In our example, we will have the operator CALL the Card Reader DSP to read a jobstream in the form of a deck of cards. The operator enters:

*X CR [B=3 .... optional paratmeters]

Since this is the first JES3 operator command we have seen, let's examine its parts. The first component of the command, the *, serves as a signal to distinguish between JES3 commands and MVS commands. This signal is necessary (though it may be an installation-defined character other than *) on commands entered from consoles at which both MVS and JES3 commands may be entered. Were the console a JES3-only console, the * would be unnecessary. The second part of the command we call the "verb" due to its characteristic of causing action to take place. "X" is the abbreviated form of CALL. We use the CALL command to cause JES3 to invoke a function when the operator requires that function. Following the verb in the command, we see a "noun," CR. Three forms of the noun are meaningful to JES3, depending upon the situation involved:

- name of a DSP (as is true in the examples).
- symbolic name of a device such as PR4 for printer four.
- device address such as 00E.

Optionally, most commands allow entry of one or more parameters, serving as control information to the DSP. Our example illustrates a B=3 parameter, telling the CR DSP to break the jobstream into batches of 3 jobs each. The last batch may contain 3 or fewer jobs, of course. There will be other sample commands used from time to time in the text.

Processing of the CALL command results in the use of several of JES3's functions. First, the command is read by a function named "CONSOLES," responsible for traffic management for locally attached consoles. CONSOLES, not wanting to

CONSOLES

process the CALL command since a JCT must be constructed, passes the command to another JES3 function, the Work-To-Do-Driver.

WTDDRVR

Construction of a Reader Job

The construction of a job, particularly a called job, is a relatively simple procedure. It consists of "filling in the blanks" in a JCT with information about the job, building the scheduler elements for the called DSP and PURGE, and enqueueing the resulting JCT into the JES3 JCT data set. Called jobs are enqueued at job priority 15 to insure faster reaction to operator commands. A significant part of the enqueuing process is the posting (or waking up) of the JSS function.

Before we proceed further with our jobstream, let's examine functions, in general, and how they are defined and handled.

Resident and Non-Resident Functions

I 2.2

The referenced illustration shows the structure of the JES3 Function Control Table. It is basically a queue of entries (FCTs) chained in DSP priority sequence. The priority used to determine the order of chaining is the one specified when the DSP was defined to JES3. Note that the FCTs shown only represent those functins which are "resident." Invocation of other functions, such as CR, will cause new FCTs to be added, in priority sequence, to the queue of FCTs. At completion of one of these "transient" functions, its corresponding FCT will be removed from the chain. Each FCT on the chain contains flags which indicate the execution status of the related function. The MFM (and perhaps this is obvious) uses the FCT chain as input to its dispatching decision, with the status flags indicating whether or not a function is currently dispatchable.

RESIDENT FUNCTION

TRANSIENT FUNCTION

Scheduling a Reader Job

Now that WTDDRVR has constructed and enqueued a job, and in the process caused JSS to be posted, WTDDRVR "waits." The JES3 "wait" is really nothing more than branch entry to MFM. MFM examines FCTs from the top of the chain downward,

testing each FCT, in turn, to determine if that function is dispatchable.

In our case, JSS is now ready to be dispatched. Upon being dispatched, JSS looks into the queue of JCTs at the appropriate level of priority to attempt scheduling of job segment. What do we mean by "appropriate level of priority?"

The JES3 JCT is managed by level of priority - in other words, the queue logically consists of 16 subqueues, one for each level of job priority. At the time a job's JCT is enqueued, flags are set to indicate to JSS which specific level of priority it is to examine. So, if a job is enqueued at priority 4, for example, JSS doesn't waste time looking at priority levels 15-5. Those subqueues would have been examined during an earlier scan, anyway.

Back to our job again - JSS finds the JCT for the reader job at priority level 15 and attempts to schedule the first scheduler element. There are many reasons why, in almost any given case, JSS would not be able to schedule a job segment...but assume for now that the CR DSP can be scheduled. In order to activate a function, JSS creates an FCT and chains it into the FCT chain (for CR, at priority 4). JSS then "waits," causing entry to MFM. Later, since functions of higher priority may require time in the CPU by now, when the CR FCT's turn comes, it will be dispatched by MFM, its load modules brought into memory, and its primary module (called a "driver") branch entered. Incidentally, since module loading involves I/O, other functions are potentially dispatched by MFM during the loading process.

I 2.3

DRIVER

When the reader DSP becomes dispatchable, it performs the following processing:

- retrieval of the parameters entered on the CALL command and sets itself up to process in the manner defined by those parameters. In our example, CR knows its batch size to be 3.

- obtains, by JES3 allocation, its input device and begins to read the job stream, counting OS job statements up to the

batch size indicated (3) and recording each batch as a collection of files in JES3 spool. BATCHING

- for each batch, CR constructs a JCT at priority 15. This JCT has two scheduler elements, Input Service driver and PURGE. The new JCT is then enqueued, and it can usually be scheduled by JSS during the I/O times inherent in CR processing. INPUT SERVICE JOBS

- continues to build up batches of jobs and construct Input Service jobs until the job stream is exhausted. At that time, assuming we are not using a HOT reader, CR would return contol to JSS, which would schedule the PURGE scheduler element (yes, another transient FCT) for the job. After PURGE processing is complete, JSS removes the CR JCT from the JCT queue.

Input Service Processing

Meanwhile, the Input Service job is being processed by JSS in the manner described for processing of the CR job. Once again, JSS will construct an FCT - this time for Input Service - and add it to the FCT chain at priority 4. The primary purpose of Input Service is the construction of individual jobs originally in the job stream. Like CR, the processing is done in multiple stages:

- first, Input Service retrieves the batch which is its input from the JES3 spool. I 2.4

- examines the job's JCL for JES3 control statements. You may hear this phase of Input Service referred to as the "Control Card Processor." The presence of JES3 control statements will influence the construction of the job - an example, perhaps the user has defined a non-standard job.

- for each job in the batch, Input Service will gather the information necessary to construct a JCT. This will include such things as:

- a JES3 job number.
- the jobname from the job statement.
- the job's priority.
- its job class.
- the symbolic origin for the job (used ultimately for the purpose of returning the job's output to its origin which becomes the default output destination).
- specific information from JES3 control statements, etc.

The new JCT, containing the items listed above, is enqueued (at the proper level of priority) in the JES3 JCT queue. Now, for the first time, JES3 is aware of the existence of a job from the job stream. Job construction is repeated for each job in the batch, and at batch exhaustion, Input Service returns to JSS, has its PURGE scheduler element scheduled, and its JCT removed from the queue.

Remember that we would probably have had several of the Input Service jobs to be scheduled, due to there being a number of jobs in the job stream. For 10 jobs in the stream, our example case would have caused construction of 4 batchs (3 jobs, 3 jobs, 3 jobs, 1 job) and four Input Service jobs.

Standard Job Processing

As a result of Input Service processing, we now have at least one normal job in the job queue. If we assume a standard job, the JCT would contain scheduler elements for Converter/Interpreter Service, Main Service, Output Service, and PURGE. Let's examine the processing for each of these job segments in turn.

Converter/Interpreter Processing

JSS, posted when a job is added to the JCT queue, selects a job based upon its priority and attempts

to schedule a job segment - either the first scheduler element or the first one not yet marked complete. It is still true that there could be many reasons to prevent JSS's scheduling a given segment for a given job - again, assume that everything required is available and the Converter/Interpreter (C/I) scheduler element can be scheduled. JSS builds an FCT, chains it, and C/I processing is underway.

CONVERTER INTERPRETER SUBTASKS
C/I DSP

During JES3 initialization, an installation-defined number of MVS converter/interpreter subtasks are attached. These tasks are subtasks to the JES3 task which is primarily responsible for the JES3 address space. The C/I DSP, whose execution is prescribed by the first scheduler element of a standard job, serves as an interface between JES3 and the MVS C/I subtasks. We have a C/I DSP for one basic reason - to gather the information concerning the job's I/O requirements prior to MVS execution of the job. This information is necessary to be able to carry out the process of preexecution setup, one of the major facilities of JES3. The information we need is, of course, specified by DD statements in the job's JCL. JCL would ultimately be passed to the MVS converter/interpreter anyway; we simply perform the intrepretation process ahead of its normal time. The responsibilities of the C/I DSP are as follows:

- to serve as a programmed interface to the MVS converter/interpreter. This is done by communicating the job's JCL to the MVS converter/interpreter subtask of JES3. The JCL was recorded onto spool at input service "time" (while the job was being processed by Input Service).

- failing, from a JES3 standpoint, the job if JCL errors are detected by the MVS converter/interpreter. This "express cancel" is a side benefit of processing the JCL prior to initiation of the job.

- to collect, and save on spool, the SWA control blocks being generated by the interpretation phase. The MVS interpreter "thinks" it is mapping these

SWA control blocks into a user address space.

- to extract the information which defines the I/O requirements for the job. This extraction process will include issuance of MVS LOCATEs for catalog references. The items which concern us relate to:

  - device requirements.
  - volume requirements.
  - data set requirements.

  The I/O requirement data is stored into job-related control blocks called a Job Summary Table (JST) and a Job Volume Table (JVT), which are recorded on spool along with the other control blocks relating to a job. In addition, the volume and data set information is later used to either add new entries or cause updating of use counts in the tables JES3 keeps in memory to maintain the system-wide integrity of volumes and data sets.

  JST and JVT

- to return to JSS when the job's JCL processing has been completed.

Upon return from the C/I DSP, JSS performs its normal processing steps to remove the C/I FCT, mark complete the C/I scheduler element for the job, and attempt the scheduling of the next scheduler element belonging to the job.

Before proceeding further with our job flow, let's investigate the concept of scheduling "times." You should be aware , by now, of the cyclical process of scheduling one scheduler element after another for a job. We have already referred to Input Service time...additional times we will see are C/I time, Main Service time, Output Service time, and PURGE time. This phraseology should be easily understood, as well as allowing fewer descriptive words as we discuss the job's processing segments.

Now, back to our job and the processing on its behalf.

Main Service Processing

The second standard scheduler element, Main Service, represents the execution of more than one DSP e.g., more than one JES3 DSP is involved. In addition, we will be working with resident functions rather than transient ones. The idea of resident functions being used nets out to the fact that, when scheduling a job onto Main Service, JSS need not construct an FCT...one already exists.

Main Service, as it relates to a typical batch job, normally consists of many processing stages. Two DSPs, Main Device Scheduling and Generalized Main Scheduling, share the responsibilities for processing the Main Service scheduler element. The work performed by MDS and GMS represent the real "heart" of JES3...major reasons for existance of a system like JES3. Those reasons fall into the categories of effective resource utilization and maximum job throughput.

MDS and GMS

Main Device Scheduling

MDS, unlike many of the JES3 functions, is optional: e.g., an installation chooses whether or not to include MDS in its JES3 subsystem. Then, assuming we have chosen to include MDS, the installation has several optional features of MDS from which to choose. Those choices include such things as:

- the volume fetch option, which allows MDS to issue messages to library consoles, thus instructing librarians to retrieve the volumes required for a job and send them to the machine room. This option may not be as useful for those installations which retain volumes in their machine rooms.

VOLUME FETCH

- job or high watermark setup. The objective of preexecution setup is the allocation and mounting of devices and volumes prior to the beginning of MVS execution of the job. What we gain by this technique is, of course, the time otherwise expended to mount volumes (the operator intervention) as the job enters its execution. In the JES3 environment,

jobs requiring volume mounts do not even become eligible for MVS processing until the necessary volumes are ready. If we choose the "job setup" option, all devices and volumes required for execution of a job are preallocated and premounted (normally, only the first volume of multivolume data sets is premounted). This option tends to be somewhat extravagant in terms of the number of devices allocated to a single job (example: 3 step job, each step requiring 4 tape data sets with no backward references, equals 12 tape drives), but little or no operator intervention is necessary between job steps. Job throughput is the favored objective.

JOB SETUP

High watermark setup (HWS) is available for either tape or DASD or all JES3 setup devices, and is a compromise type of allocation. Using this option, fewer devices are allocated at the expense of operator intervention. Here, on a device-type basis, we allocate the number of devices required (generally speaking) by the job step having the maximum device requirements. That subset of devices would then be used for all steps of the job, with intervening mounts, as long as the devices are necessary for the job.

HWS

- As we speak about device allocation, it must be noted that MDS has a built-in early resource release facility. This feature allows the deallocation of devices, data sets or volumes at the end of a job step which represents the last requirement of the resource during the job. Did you notice the listing of resources for which MDS is responsible? They are, again, devices, data sets, and volumes. Device types which may be defined to MDS are tape, DASD, unit record, and graphic.

MDS processing consists of several potential phases, infuenced by the choice of MDS options:

- volume fetch phase

  Messages are sent to library consoles to indicate whether a volume is to be fetched or not. A USES message, rather than a GET message, is issued when MDS "knows" that a volume has been previously fetched for a prior job and is still accessible.

- allocation phase

  In this phase, either MANUAL or AUTOmatic allocation may be used. Manual allocation means that MDS will begin allocation of devices for a job when an operator notifies MDS that the required volumes have been successfully "fetched" to the work area. This notification is done via a command:

  *START S nn where nn is the job number.

  If volumes used by the installation's jobs are always in the machine room (easily accessible tape volumes, resident DASD, etc), automatic allocation may be more meaningful. No operation action is necessary to cause allocation to begin.

  During the allocation phase of MDS processing, the decisions made by MDS as to which set of devices may be allocated to a job are influenced heavily by many factors. For now, let's move our job (after mount messages have been issued instructing operators as to which volumes to mount where, and JST and JVT updates have been made) to the next phase of MDS processing.

- verify processing

  The verification process performed by MDS represents the work necessary to insure, as volumes are being mounted, that the proper volumes are being mounted on the proper devices. This work is carried on asynchronously as devices become ready.

JES3 is made aware of mounts being performed on devices it is managing. MDS is then responsible for maintaining the verify counts relative to individual job. When all the necessary mounts for a job have been accomplished, the job is passed from MDS to GMS - the job is now eligible to be passed to MVS for execution.

Generalized Main Service Processing

GMS JOB SELECTION

Our job is still active on its Main Service scheduler element. The "initial" phases of MDS have been accomplished, and the job will reenter MDS processing after execution for a final phase. It is now time, however, to have our job processed by GMS. This processing of jobs waiting to be selected is controlled in a manner determined by the installation's tailoring of JES3. Triggered by an MVS initiator's request for a job, GMS is primarily responsible for picking or choosing one of several potentially schedulable jobs to give to the initiator. The selection of one job, in preference to some other job, is influenced by a multitude of variables such as:

- type of work being processed during a shift (test vs. production, on-line vs. batch, etc.).
- eligibility relationships between jobs and processors based upon such things as:
  - job classes involved.
  - number of active initiators which can service the jobs.
  - processors being online or offline.
  - I/O rates of jobs in execution.
  - virtual memory requirements as related to working set size.
- job priorities, to the extent that the installation wishes to honor priority.

After considering these many factors, GMS will pick a job, send information about the job to the

requesting initiator, and indicate that the selected job is now "on main." It may be valid, due to combinations of eligible job characteristics and scheduling algorithm decisions, not to pick a job on a given selection pass. Having been given a job, the initiator schedules it through all its steps, with JES3 only being involved for such items as:

- notification of step to step transition.
- SYSIN/SYSOUT data sets being OPENed.
- dynamic allocation/unallocation of data sets on JES3-managed devices.
- requests for spool space.

From the viewpoint of job execution, there is so little dependency by typical jobs upon JES3 that jobs can perform much or most of their execution even if JES3 has teminated, if the termination occurred after job initiation. Yet, since dependencies do exist, at some point when a JES3 service is required, the job must MVS WAIT until that service request has been satisfied.

MDS Final Processing

BREAKDOWN PROCESSING

When the job has completed execution under MVS, it is returned to JES3 (more specifically, to MDS) for device breakdown processing. Remember that at the end of a job step which represents the last use of a device, volume, or data set, these resources are being returned to MDS for early resource release. Many, if not all, of a job's resources may have been returned, but in most cases, the devices required for execution of the last step of the job must be returned to JES3 (along with data sets and volumes). Breakdown processing consists of updating MDS's control blocks by removing entries or reducing use counts and issuing appropriate KEEP or RETAIN messages. Our purpose, of course, is to make the resources available for use by other jobs which may require them.

Our next activity, now that the job has executed, is to have JSS mark the Main Service scheduler

element complete and schedule the job onto its Output Service scheduler element.

Output Service Processing

Output Service, the third scheduler element for a standard job, is responsible for the management of SYSOUT data sets generated by jobs during their execution. The data sets we are concerned with were written directly onto spool by problem programs or by JES3 DSPs running on behalf of the job. In order to handle these data sets, a single, resident, FCT, known as OUTSERV, performs preliminary work necessary to schedule the transcription of data sets to output devices. This work essentially consists of generation of control blocks which represent work to be done by a transient writer function. So, OUTSERV differs from many DSPs in that it is a scheduler of other functions...writers of various types. The actual scheduling process is designed by an installation's JES3 System Programmers in the form of an algorithm driven by a set of variables. These variables are defined by the characteristics of data sets and output devices. Let's examine the characteristics with which we are concerned first, and then we will see how they are used.

- Priority

  Data sets to be managed by output service may have an assigned level of priority. Priorities may be assigned by either a JES3 DSP that generates a data set or by a JES3 control statement, or by default from SYSOUT class definitions. The final result is the same in any case...output service has a priority at which to enqueue a data set for subsequent handling (printing, punching, sending to an external writer, etc.).

OUTPUT SERVICE VARIABLES

- Destination

  The JES3 initialization process allows the allocation of symbolic names with the devices used by DSPs. These devices may be either local or remote. We might assign the same device group name to multiple devices. An example would be a

DEVICE GROUP NAMES

card reader and a printer located in a room near our applicaton programmer staff area. Perhaps we've assigned the name "TESTGRP" to these two devices. Our desire is that output service will send back to the TESTGRP printer the output of any jobs which entered the system from the TESTGRP card reader. Here we see that, by default, an "originname has become a "destination" name. For remote devices, both input devices and output devices take on a group name which is the name of the remote workstation. Destination names may be overridden by users with specifically assigned destinations in JES3 control statements associated with their jobs.

- Device Type

This characteristic is probably self explanatory. Data sets which were defined as "print type" will be sent to printers, "punch type" will be routed to punches.

- Forms, FCB/Carriage Tape, and Train

We have grouped these three characteristics together since they have a like effect on our output. Output service will route output (to be printed) to a printer which either has the proper "setup" characteristics or to a printer whose characteristics may be changed to match the requirements of a data set. Notice that we said "a printer whose characteristics may be changed." This statement implies that we can define setup characteristics with instructions that the printer characteristics are not to be changed. It follows, then, that many installations will have some printers with specified, unchangeable characteristics and other printers whose characteristics are changeable as required.

- Line Limit

  Another characteristic assigned to output devices is that of line limit. We use this limit to cause data sets of less than 10,000 lines, for example, to print on one printer and data sets with more than 12,000 lines to print on another. In this manner, we can cause routing of low volume output to a relatively slow printer and high volume output to a faster printer, should we so choose.

- Character Set Image and Forms Flash Cartridge

  These two characteristics are associated with the IBM 3800 printers, giving us the capability of changing the print character set and the forms to be "flashed."

The characteristics described above may be assigned to both devices and data sets. How does output service use them? The scheduling process essentially involves a matching of data set characteristics to device characteristics and routing data sets accordingly. You should remember that we described output service, a JES3 function, as a scheduler of other functions we call writers.

WRITERS

Writers may be categorized in several ways, but principally fall into two distinct groupings designated as DYNAMIC and HOT writers. Writers are coded routines which become JES3 functions when scheduled by OUTSERV. A dynamic writer is a type which is automatically scheduled when two things are true:

- there is work (one or more data sets) in the output service queue.

- there is an available device upon which to do the writing.

DYNAMIC WRITERS

Dynamic writers give operations personnel little or no control over when and how writing is to be done. An exception, of course, is that dynamic writers allow changing of setup characteristics of

devices. We might want dynamic writers to handle our volume printing on stock paper. Hot writers, on the other hand, give operations people total control of output handling. Operators enter commands to CALL and control hot writers. We might use a hot writer to direct classes of output which require special forms to a particular printer, thereby concentrating operator intervention on a single printer. To further differentiate between dynamic and hot writers, we need to become aware that the characteristics assigned to dynamic writers are specified in the JES3 initialization statements and become "fixed" at that time. An operator may allow a hot writer to assume some or all of the installation-default set of characteristics or directly specify an override set, thereby gaining total control over hot writers.

HOT WRITERS

The management of data sets, directed by class to TSO users, external writers, or to the internal reader, is most interesting. In these cases, we are generally speaking of transcribing data sets to programmed routines rather than to hardware devices. The external writer, a STARTed task in MVS, is used to transcribe special purpose data sets to devices such as magnetic tape, DASD, or plotters...these devices are not supported by output service. Data sets destined to the internal reader are routed by output service, not to a printer, but to a JES3 function we have already discussed - Input Service - since internal reader data sets contain job streams. SYSOUT data sets held for TSO processing are simply routed by output service to the TSO OUTPUT command processor upon demand.

Scheduling Output

We left our job at the point where JSS scheduled its output service scheduler element. When this occurs, the OUTSERV function goes to work on our job's behalf. OUTSERV accesses the control blocks necessary to locate the data set it is to process. These control blocks also contain the characteristics of each individual data set. From this information OUTSERV constructs other control blocks which represent work to be performed by writers and schedules the writers by invoking them as functions. (This type of scheduling assumes

that the data sets involved are not of the "held" nature.)

Writing Output

A writer, when scheduled by OUTSERV, becomes a transient JES3 function. It accesses the control information enqueued by OUTSERV and attempts matching the characteristics of a data set to those of the device obtained by the writer. Data sets are transcribed on a "best-match" basis, allowing for the fact that a given writer function may or may not be able to process all of a job's data sets. Scheduling of several writers may be necessary to complete the handling of a job's SYSOUT. Many dependencies, which determine where, when, and how data sets are managed, exist...remember those data set and device characteristics? As each data set is written, Type 6 SMF records are recorded for later use in job accounting. Our job remains active on its output service scheduler element until all its data sets have been properly transcribed to the proper destinations. When all the data sets have been written, we again return to JSS who marks the scheduler complete and schedules the job onto PURGE.

Purge Processing

We are finally about to see our job reach completion, since PURGE is its last scheduler element. There are several items for which PURGE is responsible:

PURGE

- recording of SMF record Type 25 which was generated during MDS processing to account for the setup requirements of the job.

- recording the SMF record Type 26, the normal job management record.

- most importantly, PURGE must "free" all the spool space which still belongs to our job. With the exception of a special case of data set SPINOFF, all the spool space acquired during the life of our job remains assigned until PURGE time. This space contains:

- the original JCL for the job.
- message data sets.
- SYSIN/SYSOUT data sets.
- JES3 control blocks which define our job and its processing characteristics.
- the MVS SWA control block generated at C/I time.

When PURGE completes its processing it returns control to JSS (like any function), but JSS recognizes that return is being made by PURGE. Now it is time for JSS to remove our job's JCT from the job queue, causing our job to leave the system.

Chapter 3

Subsystem Interface

Introduction

We will attempt in this chapter to describe the general structure of the communications between:

- MVS system components and JES3.
- JES3 and JES3.

To set the stage, let's quickly review the basic structure of MVS. The usual pictorial representation of an MVS environment consists of a large box representing the memory layout of the system. Areas of memory usually shown include (from high address to low): | 3.1

- System Queue Area.
- Pageable Link Pack Area.
- Common Storage Area.
- Memory in which virtual address spaces are established for.
  - Master Scheduler.
  - JES.
  - Users - batch, TSO, STARTed tasks.
- Nucleus.

By simple acceptance of the multiple address space structure and the requirements of address spaces for services, we recognize the necessity of communication between address spaces. A communication mechanism is established and has been defined as the Subsystem Interface (SSI). SSI Like most system components, SSI consists of several elements, namely:

- a series of control blocks for passing requests for service from sender to receiver and back.
- a set of macro instructions used by requestors of services and system components which respond.
- Programmed routines to perform the work necessary to respond to requests.

Let's quickly review the roles played by global JES3 and local JES3. You should remember that global JES3:

- introduces all jobs into the system, no matter what the source.
- converts and interprets JCL.
- performs preexecution setup of devices.
- schedules MVS jobs to all MVS processors in the complex.
- maintains awareness of all jobs in execution.
- handles all SYSOUT data sets.
- manages, and is responsible for, the allocation/unallocation of space on the shared spool devices.

When carrying out several of the responsibilities listed above, global JES3 needs the "assistance" of local JES3. This is, of course, only true if we are discussing the scheduling of work onto a local processor. (If global JES3 is scheduling work onto the global processor, we would not really speak of "assistance".) Local JES3 also has responsibilities:

- returning information to global JES3, as necessary, to satisfy:
  . requests for catalog LOCATEs.
  . verification of device mounts.
- routing of console messages to global JES3.
- generally keeping global aware of what is happening in the local.

So, we see that JES3 is deeply involved with, and on behalf of, MVS; there are many types of communications.

What exactly are the communication paths we are concerned about? Beginning with the global processor, we see that 1) user address spaces will | 3.2

make requests of global JES3 and responses will be returned. For example, imagine an initiator in a user address space requesting a job and having returned to it a set of descriptive data defining the job to be initiated. An extension of this form of request/response would be 2) a local user address space requesting a job from global JES3. There are also a few cases in which 3) global JES3 and local JES3 must communicate; either JES3 can initiate the communication, and, in many cases, responses are not required. It would appear from the illustration that a fourth type of information flow might be used - local user address space to local JES3. This last type of communication, however, is not used in today's JES3 environment. How are all these communication paths implemented?

MVS Definition of SSI

MVS has defined thirty seven (this number will vary depending upon which selectable units are installed) cases in which communications with JES is involved. Each of these cases is assigned a number, called a function code. These SSI function codes represent a different type of information:

I 3.3

FUNCTION CODES

- a request for a service or
- control information

passed to JES3. Currently, JES2 supports nineteen of the codes; JES3 supports all but code 15, which is handled by the Master Subsystem. Several of the request types are not passed to JES3. These are, instead, processed directly by special SSI function routines in the processor that originated the request. SSI function codes may be categorized by the type of processing to which they relate.

To implement the support required when JES3 communicates with JES3, a set of functions was designed for specific use by JES3. We call these codes JES3 destination codes. As we will see in the following discussion, both JES3 destination codes and MVS function codes are handled in the same general manner.

JES DESTINATION CODES

SSI Communications Overview

We mentioned earlier in this chapter that the SSI was made up of control blocks, macros, and routines. Prior to our following a request through the SSI, the control blocks involved should be described and understood. The sequence of control information for SSI really begins in the MVS nucleus with a control block named the JES Communications Table (JESCT). Our major concern in this block is the pointer to the primary subsystem (JES2 or JES3) communications vector table, the SSCVT, which is located in CSA. There will always be at least two SSCVTs, including one for the Master Subsystem. From the JESCT, we can locate the primary subsystem SSCVT, in which three items interest us:

I 3.4

JESCT

SSCVT

- the ID of the subsystem (i.e., JES3).

- a pointer to the Master Subsystem SSCVT ... SSCVTs are constructed during MVS IPL. They are used for two purposes: validation of the subsystem's being active when needed and to house the address of an SSVT.

- the address of the subsystem vector table, the SSVT.

  SSVT

  The SSVT is built by the subsystem itself during its initialization. A matrix is contained within the SSVT to describe (in a very simple way) which function codes are supported by the subsystem. In addition to the function matrix, the SSVT contains addresses of the routines to support the various function codes.

Then, chained from the Master Subsystem SSCVT are SSCVTs for any active secondary subsystems. The user memory, at the time of an SSI request will also use two control blocks which are described in the discussion below.

Let's assume, for our discussion, that we are an MVS initiator in a user address space. At that point in our processing at which we want to ask JES3 for a job, we issue an IEFSSREQ macro (not the run-of-the-mill macro since only system

IEFSSREQ MACRO

routines can use it). When we issue the macro, we point (via register 1) to a Subsystem Options Block, which contains a function code to define the request we are making. For our job select request, the code would be 5. The SSOB also contains the address of the Subsystem Identification Block which identifies the subsystem to which we wish the request to go. The expansion of the IEFSSREQ macro loads the address of the subsystem interface request routine from the JESCT into register 15 and branches to the routine, IEFJSREQ.

SSOB

SSIB

IEFJSREQ

SSI Request Routine

This routine, residing in PLPA, performs several activities relating to an SSI request. After certifying the validity of the SSOB and SSIB, the request routine determines that the subsystem requested is valid and started. This is done by checking subsystem IDs in the chain of SSCVTs. The function code (placed into the SSOB at IEFSSREQ time) now becomes very important, because we use it to determine if the subsystem supports the requested function. IEFJSREQ uses the function code as a displacement (after adding 4) within the SSVT matrix. You have probably guessed by now that the SSVT matrix is 256 bytes in length and that only part of the matrix is currently used. The resulting address (function code + address of SSVT + 4) contains zero for non-supported codes and another displacement value for the supported codes. This second displacement, when multiplied by 4 and added to the address of a vector list on the bottom of the SSVT, gives us the address of a routine to which the request is to be passed. During this process of getting to that routine, should any of the tests made by IEFJSREQ result in failure, an error return code is passed back to the issuer of IEFSSREQ.

I 3.5

The Function Routines

Though there are many (about 36) SSI functions supported by JES3, there is not an equal number of function routines. Several of the routines perform more than one function. We cannot, in this text, describe all the function routines, but you should be aware of their general logic

after we have described our example case, the job select request.

When a function routine is branch-entered by IEFJSRE2, we are executing code which also resides in PLPA. This code has access to the SSOB/SSIB pair, passed along by IEFJSRE2. One of the duties of a function routine is to construct a Service Entrance List which is, in essence, a communication control block. The SEL built by function routines contains such items as:

SEL

- a data area address.
- address of an ECB.
- optionally, an exit routine address.
- a response buffer address.
- address of a staging area (which we have not yet described).

So, the function routine (our case is IATSIJS, the job select routine) gathers data, places that data into an SEL, and sends the request on its way by issuing another macro, SSISERV. This macro provides entry into a routine we call Subsystem Common Services - IATSSCM is its module name.

SSISERV MACRO

IATSSCM

SSI Common Services - Router

IATSSCM consists of two distinct parts; the first part is entered as a result of the SSISERV macro. You may hear the routine referred to as the "common services router" or more simply "the top half of common services". The former name, router, is the more descriptive of the routine's services - the routing of a request to the appropriate address space. We will examine that routing process after describing the seven types of requests which can be made with SSISERV:

ROUTER

- wait type - a communication of data which requires a response. The requestor is MVS WAITed until the response is received.
- reply type - almost the same as wait type except the requestor is not MVS WAITed;

usually involves a user exit to process the response.

- ack type - communication of data with no required response other than an acknowledgement of receipt.

- comm type - sending of data only, no response communication is necessary.

- resp type - a response to an earlier wait, reply, or ack type of request.

- EOMT type - used as a memory is ending to cause special cleanup of SSI communication areas which might still be outstanding.

- purge type - used by JES3 DSPs to free the areas containing type=comm requests which the DSP has processed.

A short review of how we came to be processing in the router routine is probably necessary by now ...

- a system routine has issued a request for some service; the IEFSSREQ macro was used.

- the subsystem interface request routine has verified the request and passed it to an SSI function routine.

- the function routine assimilated the request into an SEL and entered the common services router via SSISERV.

STAGING AREA

The vehicle used to communicate between address spaces is called a staging area. Since multiple address spaces are involved, the staging area must be located in CSA. A preallocated pool of staging areas was built during JES3 initialization. After acquiring a staging area, the router fills it in with information from the service entrance list. Then, very importantly, the router must determine whether or not the sending address space and the address space of the receiver are in the same machine. This determination is necessary because if the sender/receiver are in the same processor, the request is MVS SRB scheduled on its way; if the

two address spaces are in different machines, we must issue a STARTIO macro to write staging areas on the CTC device. Following the SRB SCHEDULE or STARTIO macro, the action performed by the router depends upon the type of communication request being made:

- for type=wait, an MVS WAIT is issued.

- for type=comm or ack, the router returns control to the user address space.

We must note, at this point in our processing, that all the code we have executed in the subsystem interface has been located in PLPA and is run under control of the user address space.

SSI Common Services - Poster

The request, in the form of a staging area, now enters the second part of the common services routine called the "poster" or "the bottom half of common services". Its formal name is the Read End Subroutine. The poster routine is executed on the processor which holds the receiver of the request, under control of the receiver's address space. Processing by the poster may be summarized as getting the request to its receiver. Two things are required to do this:

POSTER

- queueing the request (staging area) on a queue serviced by the receiver.

- posting the receiver to notify it that a request has arrived.

Did you notice that we said that the poster runs under control of the receiver's address space ... how did we obtain addressability? As it turns out, the poster is always SRB scheduled, allowing the equivalent of a "cross memory post".

We have, in CSA, a control block to facilitate the accomplishments listed above ... queueing and posting. The control block name is the Destination Routing Table, sometimes called the Destination Queue (DSQ). This control block is structured much like the SSVT; it has two

DSQ

matrices (which relate SSI function codes and JES3 destination codes to receivers) and entries which contain queue origin pointers and data used to post the receiver of a request.

Return a Response to Requester

In the example case we are using, that of an initiator requesting a job, the poster will queue the staging area for JES3's GMS function and post GMS that it has a request to service. GMS, when dispatched by the multifunction monitor, uses its algorithms to select a job to send to the initiator, placing scheduling information for the initiator into the staging area it received. Next, GMS begins the process of getting the scheduling information back to the initiator by issuing a JES3 macro - JSERV. This macro is the JES3 form of SSISERV, and causes entry to the router routine of SSI common services ... same process as used before, but now the code for the router is executed under control of the JES3 address space.

JSERV MACRO

Again, the router issues either SRB SCHEDULE or STARTIO to send the staging area back to its receiver (which is, of course, the original sender) by executing the code for the SSI poster routine. What address spaces do the routines run under control of this time? The router is controlled by the response sender (the original receiver), and the poster runs under the receiving address space (the original sender).

The response to the original request (for wait and reply types) is now matched to the original request to be able to return the response information. Responses are then moved from the staging area, in which they came from the sender, to the original SSOB, and the user is posted for "receipt of response". (We see that the return of a response is not the reverse of making a request, it is almost exactly the same process.) Remember that the original requestor, if WAITing, is waiting in the common services router routine. Common services, when posted, returns control to the SSI function routine which in turn returns control to the user's code following the IEFSSREQ macro. The user code (in our case, an initiator) now finds in the SSOB

the information it requested and may continue processing.

Review

A basic understanding of the subsystem interface is essential if you are to grasp the concepts of MVS to JES3 communications. In this chapter, we have discussed many codes, several macros, and multiple routines. Perhaps a short review would serve us well.

Codes

The MVS subsystem interface function codes are used in those cases in which some user address space needs to communicate with JES3. Should one JES3 address space have a requirement to communicate with another JES3 address space, JES3 destination codes are used instead of MVS SSI function codes.

Macros

An SSI request being made by a user address space is started on its way by issuance of an IEFSSREQ macro. Subsequently, an SSISERV macro is issued by a function routine to pass the request along to SSI common services (the router). The router routine uses a STARTIO or SCHEDULE macro to send the request through the poster routine of common services to global JES3. Once the request has been satisfied, a JSERV macro is issued by the JES3 component to route the response (if one is required) back to the requestor.

Routines

The first routine entered during normal course of processing of an SSI request is the request routine. After validation of the request, a function routine, which is responsible for supporting the request, is given control. In many cases, the request is satisfied directly by the function routine. When the function routine does not directly satisfy the request, it is passed to a JES3 function via the common services of the SSI. The common service

routines are also used in returning the response to the original requestor.

Chapter 4

Spool Data Management

Introduction

One of the significant features of JES3 is its spooling capability. Though a typical, large JES3 complex may have 7-10 spool volumes, JES3 supports as many as 30. These spool volumes may reside on DASDs which are IBM 3330 (Models 1 or 11), 3340s or 3350s; the environment may consist of a single device type or any combination of the supported device types.

Since all JES3 processors read and write data from and to the spool volumes, each processor must have access to the spool space. Several different types of data are recorded in the spool space. First, spool contains the information (originally taken from the initialization statements) necessary to bring up JES3 in both the global and local processors. The spool volumes also contain two types of information directly related to jobs which are to be run:

- the JES3 control blocks which define the scheduling and operational characteristics of the jobs.

- the SYSIN (DD * or DD DATA) data sets and SYSOUT data sets for jobs.

The management of spool space and recording/retrieval of spooled data are the responsibilities of our two special access methods. JES3 DSPs use the JES Spool Access Method (JSAM) to read and write data. User address spaces utilize the User Spool Access Method (USAM) for handling their spooled data, though USAM is transparent to user programs.

JSAM

USAM

Functional Overview

If we examine the component parts of JES3 Spool Data Management (SDM), we quickly see that some of the parts are distinct to JSAM, some are USAM related, and others are common to both access methods. To increase our perspective, let's itemize these components by the categories just

SDM

mentioned. We can break JSAM into about four separate logical parts.

JSAM Components

There are two recording techniques (from a programming standpoint) used for spooled data. We read/write the job related control blocks as Single Record Files (SRF). This simply means that a control block, irrespective of its size, is recorded as a single buffer image (a block of data on a spool volume). While most of our control blocks will fit into a single buffer image, some (due to the variable lengths consisting of multiple parts) may extend across multiple buffers. When this occurs, the buffers are chained together so that there is still a single "record" in the file.

SRF

It is more logical to record such files as a job's JCL, its SYSIN, and SYSOUT by a technique which allows us to pack multiple records (card images, print lines) into a single buffer image. We identify these files as MultiRecord Files (MRF).

MRF

The first of our four JSAM components consists of the set of macros used by DSPs to record and retrieve both SRFs and MRFs. Macros are provided to acquire spool space, create spool files, read and write records, and to terminate the existance of files when they are no longer needed. Many of these macros involve the acquisition of a place on spool (a DASD block) in which to record the JES3 record. Notice that our use of the term "record" is inconsistant with normal DASD terminology ... our spool access methods do not include the concept of "blocked" records.

I 4.1

Our second JSAM component is a set of routines responsible for the allocation of space. The unit of JES3 spool space is called a track group. Track groups consist of multiple physical tracks on DASD as defined by a JES3 initialization parameter. A track group is made up of either a half cylinder or a full cylinder - the quantity being fixed, once it is defined. So, then we have routines to allocate track

TRACK GROUP

groups and other routines to suballocate individual blocks within track groups.

Since each of our JES3 processors has access to spool, the single-system image concept demands that management of spool space be concentrated into a global-only responsibility. The third JSAM component is an interface routine between user address spaces requesting track groups and the allocation routines mentioned earlier. User address spaces are assigned one or more track groups (prior to MVS execution) as the job is being processed by Input Service. This initial allocation is used to hold the job control blocks, its SYSIN data, and the MVS SWA control blocks generated by the converter/interpreter. While the job is executing, SYSOUT data sets will be written into the remaining portion of the first allocation as long as they fit. Should the initial allocation be insufficient to hold all the job's SYSOUT, later requests are satisfied by global JES3 via the subsystem interface routines.

Our fourth JSAM component is a JES3 function (it happens to be one of the resident functions) named "JSAM". This function is responsible for routing all JSAM I/O completions to the appropriate DSP by posting flags which allow a waiting DSP to again become dispatchable.

USAM Components

We will group the responsibilities of USAM into a smaller number of categories - only two. First, there is a set of macros, access to which is through a standard (both JES2 and JES3) compatibility interface. User problem programs require access to spool for SYSIN and SYSOUT, which are being processed by normal BSAM or QSAM. Routines hidden away inside the compatibility interface, not the problem program, actually issue the macros which cause entry to the routines provided by JES3. When a buffer has been filled during creation of a SYSOUT data set, the second USAM component comes into play.

Though USAM cannot allocate track groups of spool space, USAM does have responsibility for

the allocation of individual blocks on spool into which buffer contents are to be written. These blocks are suballocated from the original set of track groups assigned to the user job until, of course, that space is exhausted. If additional track groups are required, requests are forwarded to JSAM's track group allocation routine through the subsystem interface and the allocation interface routine.

Common Component (JSAM and USAM)

You have probably noticed that to this point in our discussion we have not mentioned the "starting" of I/O operations to spool. JES3 manages I/O to spool at the STARTIO macro level, and our routine serves as an IOS driver. The routine which actually schedules the I/O operations is common to JSAM and USAM. So, we see both JSAM and USAM requests to read and write spool passing through a pageable link pack area module, housed in the processor for which I/O is being performed. This module is responsible for managing the termination of I/O requests as well as initiating them. When JSAM requests are completed, the common routine posts the JSAM function (we mentioned this earlier) to pass the completion to the proper DSP. USAM completions cause the common module to post the proper user address space that its I/O is complete. You should review the components by referring to the more detailed illustration. I 4.2

Buffers and Buffer Pools

Spool I/O is performed to and from spool volumes and buffers in memory. Both the size and number of buffers involved are predefined at JES3 initialization time with parameters in initialization statements.

JES3 Buffers

The buffers we use for spool I/O have a number of interesting characteristics. Each buffer contains a fixed area which is not transferred to spool volumes. This area contains the channel program necessary to read or write the data portion of the buffer, flags to indicate the status of the buffer, and any required

chaining fields. Following the fixed area, we find the data area which is formatted differently for SRFs and MRFs. How about user memory buffers? They are formatted as MRFs. Remember that a SRF contains a control block, a MRF will contain packed card images or print line images.

The buffer size can vary, but is usually either 1248, 1952, or 4000 bytes in length. These represent sizes which, when increased by the size of the fixed area (96 bytes), result in 3 or 2 or 1 buffer, respectively, per page of virtual memory. The 1952 byte size appears to be the one most commonly used by JES3 installations.

BUFFER POOLS

JES3 buffers are always grouped into buffer pools, of which there are three types:

- JESIO buffer pool - This pool is in the global JES3 address space and is used by DSPs doing SRF/MRF processing. I/O takes place directly to and from these buffers.

- User memory buffer pools - These buffer pools, inside the user address space, consist of 1 page of virtual memory for a SYSIN data set or multiple pages for SYSOUT data sets. The contents of these buffers are not directly transmitted to/from spool, but instead to/from I/O areas provided by the problem program. User memory buffers are transmitted to (when filled for output) and from (when emptied for input) buffers in the protected pool for transmission to and from the spool volumes.

- Protected buffer pool - Each JES3 processor contains, in CSA, a buffer pool used in the performance of I/O between user memories and spool. This pool is "protected" for two reasons:

- to maintain the integrity of data sets between address spaces.

- the Disabled Interrupt Exit included with SDM runs under control of JES3 segment and page tables ... thus requiring common addressability.

Data Flow for USAM Requests

Since it is assumed that most readers of this text are primarily concerned with the relationships between user address spaces and JES3, our SDM data flow discussion will be directed at USAM requests.

The accompanying illustration is a memory map which high lights the SDM components mentioned in the following description. You will, more than likely, need to refer to the diagram several times. | 4.3

SYSIN Processing

Our discussion begins at the point of MVS initiation of a job step for a user memory. The MVS allocation routines, when allocating a SYSIN (or SYSOUT) data set, issue the IEFSSREQ macro to ask the Job Entry Subsystem for assistance. In the case of JES3, the SSI function routine satisfies the request, without help from global JES3, by constructing a control block, the Data Set Block, in CSA. This control block contains many of the DCB-like characteristics of the data set being allocated, as well as several SDM-related fields to be used during processing of the data set. After building the DSB, the SSI function routine returns to the user code. Later, when the problem program OPENs the SYSIN data set, another IEFSSREQ macro is issued by the compatibility interface code, since the data set being OPENed is a spool data set. The SSI function routine which processes OPEN requests has a number of responsibilities. It is at this time that the user buffer pool (1 page of virtual memory) is constructed in subpool 230 of the user memory. Management of this buffer pool is facilitated by control fields in the DSB. The function routine finds itself facing a lack of information ... only global JES3 knows the location of the SYSIN data set on spool. A

ALLOCATION

DSB

OPEN

staging area containing descriptive information about the data set being OPENed is SSISERVed to global JES3.

Global JES3 has to respond (via JSERV) with the disk location of the first MRF buffer of the SYSIN data set. This information is kept in a control block which was built for the job during Input Service processing. Upon receipt of the response OPEN processing can continue by queueing requests to prime the user buffer. Remember that the I/O operation involved here first reads the data from spool to the protected buffer pool in CSA. At termination of the reading, the buffer content is moved to a user buffer.

The problem program now begins its normal processing by issuing GET or READ macros. Each logical record is passed to the program by the USAM routines given control by the compatibility interface. Then, as necessary, user buffers are refilled from protected buffers, I/O requests to refill the protected buffers are queued ... until end-of-file is reached.

GET

User buffers are freed at CLOSE time by the SSI function routine, thus "disconnecting" the data set and the problem program. At end of the job step, the MVS unallocation process again issues IEFSSREQ in order that the SSI function routine can free the DSB, and processing of the data set is now complete. Notice that only one communication between the SSI routine and JES3 was necessary.

CLOSE

UNALLOCATION

SYSOUT Processing

SYSOUT processing, though similar to that of SYSIN, is somewhat more complex. This additional complexity arises from the fact that JES3 has no need to maintain information about a SYSOUT data set until it is OPENed. As before, our involvement begins at allocation time. The SSI function routine constructs the DSB as the result of an IEFSSREQ request. For SYSOUT data sets, OPEN processing (in the SSI function routine) must construct an entry to be placed in the control block that JES3 uses to define a job's MRFs. This "new" entry is then sent to

global JES3, again via SSISERV, who places the entry into the control block. A user buffer pool is then allocated in subpool 230 ... this time, multiple pages of virtual memory (the default is 2 pages).

Following OPEN, the problem program issues PUT or WRITE macros. Each PUT causes entry to a USAM routine which counts the records, truncates any trailing blanks in the record, and moves the resulting record into a user buffer. When the user buffer is filled, the USAM "PUT" routine causes the full buffer to be moved to a protected buffer and queued for output to spool. PUT

After exhausting the input for the SYSOUT data set, the CLOSE macro is issued by the problem program. CLOSE issues IEFSSREQ giving control to the close routine in the SSI function code. CLOSE processing by the SSI function routine includes proper handling of the last user buffer (to cause it to be written to spool) and the freeing of the user buffer pool. Then, we see another difference between SYSIN and SYSOUT processing. Do you remember the control block entry sent to global JES3 at OPEN time? Well, it must be updated at unallocation time to reflect the final status of the new data set ... line count, disk location, etc. The updated entry is SSISERVed to global JES3 to complete the entry made at OPEN time. When the updating is complete, a notification is JSERVed back to the function routine which then frees the DSB.

Summary

Spooled data sets are clearly the responsibility of JES3. We have seen that, though it is not apparent to a user problem program, routines supplied as part of JES3 are deeply involved in the processing of both SYSIN and SYSOUT data sets. This code is found in two distinct sets of routines, one being branched into from the compatibility interface code, and the other entered as result of IEFSSREQ macros being issued. A quick review of I4.3 will help you locate these sets of routines, the buffer pools we discussed, and the major control blocks involved in JES3 SDM.

Chapter 5

Job Related Control Blocks

Places Control Blocks are Kept

JES3, like other systems, uses a control block scheme to define both the existence of jobs and how those jobs are to be processed. Some of a job's control blocks are kept in various areas of DASD until required for processing, at which time a copy of the required control block is brought into memory. Other control blocks reside only in memory, with no copy on DASD. Of those which exist in memory, there are two types - 1) those which exist for the life of the job, and 2) those which are created as job segments are scheduled. We will examine the general structure of our control blocks by first describing those areas on DASD in which control blocks are kept.

Spool Volumes

Control blocks are, you may remember from our discussion in Chapter 4, recorded on spool in the form of single record files. There are two areas of spool space which will contain control blocks. By generalizing, we can classify these two areas as:

- an area of spool space, used somewhat like CSA in memory, usable by many components of JES3 ... a commonly accessable area.

- the spool space "owned by a job".

Lets investigate the two areas individually.

Single Track Table

The part of spool space which is usable by many components of JES3 is the Single Track Table (STT) space. In order that this name be meaningful, we must relate back to a term we discussed back in Chapter 4. The term is "track", and a related term we used is "track group". We defined the unit of spool space as a track group, which consisted of a half or full cylinder of space. While we might think of a track group consisting of physical

SST

tracks, in JES3's jargon, a track is a disk record location or a DASD address at which a block (buffer image) is located. So, to be definitive (in JES3 terminology) a track group consists of a consecutive group of "tracks" or DASD addresses. Because it is often more efficient, in terms of spool space, to allocate a single "track" rather than a track group, the developers implemented the single track table. TRACK

The STT may consist of one or more areas inside (or, allocated from within) the DASD space defined to JES3 as its spool space. Physical placement of the STT is carefully considered by JES3 system programmers because of its relationship to system performance. Some, or in special cases, all of a job's control blocks will be written as SRFs in the STT. There is a table kept in the JES3 memory in the form of a bit map which defines the available tracks in the STT. This table is used to allocate or unallocate (make available for reuse) STT space, a single track at a time.

Job's Spool Space

With the exception of "called jobs", all jobs entering JES3 pass through Input Service processing and are assigned a minimum of a single track group of spool space. This original allocation of space is used to contain many of a job's control blocks. Control blocks for "called jobs" are written into the STT because the jobs own no spool space. The STT is utilized, as we shall see later, to contain some of the control blocks belonging to normal jobs, as well.

Job Control Table Data Set

I 5.1

One of a job's control blocks, the JCT entry (remember it from Chapter 2) is recorded into a unique data set instead of being placed in either STT or the job's spool space. This data set is allocated by MVS allocation as the JES3 address space is being initialized, and, consequently, the JCT data set does not lie within the DASD pace allocated for spool. From

a formatting standpoint, the JCT data set is of a fixed length, unblocked format, with each block capable of holding one JCT entry. You should remember that JCT entries vary in size - "called jobs" have two scheduler elements, normal jobs have four (or more), and non-standard jobs have a variable number up to an installation-defined maximum. Even though the JCT data set is allocated outside the spool space, the data set is read and written from the JESIO buffer pool with the JSAM SRF processing techniques. The JCT data set, depending upon how much space was allocated to it, contains a fixed number of blocks (or "slots"). These blocks are reused following a job's being PURGEd; allocation and unallocation of the blocks is managed via a memory-resident control block which, in structure, is identical to the control block which maps the STT.

JCT DATA SET

Control Blocks - Defining Jobs

Though there are typically many control blocks associated with a job, two are primary to the definition of the job. First, and most important, is the JCT entry, which we have discussed many times already. The JCT entry describes the processing to be performed on behalf of the job, the characteristics of the job, and serves as a continually-updated checkpoint of the status of the job as it progresses through its life in the system. A job "exists" when its JCT entry is created and ceases to exist when the JCT entry is removed.

There is, in addition to the JCT entry kept in the JCT data set, another closely related control block kept in memory for each job in the system. This new control block is named the Job Queue Element (JQE). The content of the JQE is essentially copied from the job's JCT entry. JQEs are present in JES3 to reduce the I/O that would otherwise result from frequent examination of JCT entries for scheduling work. The JQE is a highly abbreviated set of information about a job - only that necessary to make scheduling decisions and maintain an in-memory status for jobs. It is the JQE which is used to answer most inquiries about jobs ... further removing a need for JCT access.

JQE

The J2E is constructed at the same time as a job's JCT entry ... and in one sense, the J2E carries as much weight of importance as the JCT. It is the J2E which is first examined by JSS when determination of work to be done is being made. Once JSS has decided to schedule work for a job, the JCT is then read, additional information extracted, and the status checkpointed by rewriting the updated JCT entry.

From pointers in the J2E and JCT, JES3 DSPs working on behalf of a job can find all the other job-related information kept by JES3. These pointers are in the form of either a memory address or a disk address, depending upon the control block being accessed and the time at which the pointer is used.

Control Blocks - Job Processing

If we examine the job-related control block structure for a "typical" standard job, we find several control blocks involved. Lets discuss each of these control blocks individually. I 5.2

Job Description and Accounting Block (JDAB) JDAB

The JDAB is constructed by Input Service simultaneously with the JCT entry; many informational fields are common to both. JDABs give DSPs a source of data concerning a job such that JCT access is not necessary. Entries which represent each of the JCT scheduler elements are appended to the JDAB to provide DSP accounting for each segment of the job.

Job's Data Sets Block (JDS) JDS

All of a job's multirecord files will be identified and defined in the JDS. This block, built by Input Service, initially contains entries for the data sets whose contents are:

- inputted JCL for the job.
- two message data sets; one for messages generated by DSPs and one for system generated messages.

- the SWA control blocks generated by the Converter/Interpreter.

- any SYSIN (in individual entries) data sets for the job.

Entries for SYSOUT data sets are added to the JDS when the data sets are OPENed during MVS execution.

Job Summary Table (JST) JST

Though this control block is constructed at Input Service time, it is not filled in until Converter/Interpreter time. The entries consist of the requirements for preexecution setup for the job. Primary use of the entries is at MDS processing time when devices are being assigned to the job.

Job Track Allocation Table (JBTAT) JBTAT

Built at Input Service time, this block is a bit map identifying the track groups of spool space which have been allocated to the job. The table will be updated to reflect any subsequent allocations as they are required. At PURGE time, track groups are simply "given back" to the system or master track allocation table for TAT reassignment to other jobs.

Job Management Record (JMR) JMR

This control block is also built at Input Service time. It contains the job accounting information (fields) necessary to construct an SMF type 26 record for the job.

Format Parameter Buffer (FRP) FRP

If the user indicates, through the inclusion of the JES3 control statement FORMAT, that there is to be special handling for any of the SYSOUT data sets, a FRP is constructed at Input Service time for the job. The FRP is just a place to store the information, taken from FORMAT statements, relating to the individual SYSOUT data sets.

User Parameter Buffer (PARM) PARM

One of the JES3 control statements, PROCESS, allows the user to define a non-standard job; i.e., to let the user design the set of segments for the job. These PROCESS statements may be individually followed by parameter information to be passed to a DSP named in the PROCESS statement.

This control block is also generated for called jobs if the operator enters any optional parameter information in the CALL command.

Job Volume Table (JVT) JVT

This table, like the JST, is built to contain entries describing one of the resources assigned to jobs at MDS processing time. JVT entries are established to identify the volumes (by serial number) the job will require at MVS execution time. The JVT is built at Converter/Interpreter time.

Output Scheduling Element (OSE) OSE

OSE's contain a set of output (SYSOUT) data set characteristics relating to one or more data sets to be managed at Output Service time. The first OSE for a job is constructed at Input Service time and written into the STT. Later, during Output Service processing, the OSE will be completed and perhaps others generated for the job. These elements become the "output queue".

Control Blocks - Segment Processing

After the Job Segment Scheduler has been posted to make a scheduling pass, JQEs are examined to find an eligible job for which to schedule a job segment. Part of JSS's processing is to construct an FCT entry (remember from Chapter 2) if the function being scheduled is not a resident function. You should remember that FCTs represent the functions being performed by JES3. Also, in addition to constructing and enqueueing an FCT (if necessary), JSS will always build a Resident Queue Element. As the illustration shows, pointers from the JCT to the job's other SRFs are extracted and

| 5.3

RQ or RESQUEUE

placed into the RQ. The RQ, quite a large control block, is a storage area for status flags, informational fields, and queuing pointers during processing of a job segment (1 or more DSPs). RQs only last the life of a job segment.

Chapter 6

Additional Functions and Features

Introduction

There are many JES3 facilities we have not yet addressed in our discussions. Though it is beyond the intent of this text to describe all the features of JES3, a few which remain deserve our attention. Our purpose in this chapter is to examine several JES3 facilities that directly affect, or are affected by, users of the computer complex.

Deadline Scheduling

Most installations' production schedules demand that certain applications are completed at strategic times of day on particular days of the week. Deadline Scheduling, a JES3 DSP, is designed to facilitate scheduling of jobs in a controlled manner based upon time intervals, time of day, and associated job priority changes.

Implementation of Deadline Scheduling begins with one or more algorithm definitions (up to 36) in the initialization statements for JES3. Each algorithm is given a <u>type</u> name and contains the specifications regarding time intervals and priority changes. There are two required parts of a deadline algorithm: I 6.1

- an initial priority change - The change may be either in the form of a priority "set" value or a "change" value.

- a lead time value - Each job to use Deadline Scheduling will, via a JES3 control statement parameter, identify a time of day by which the job is to be completed - the deadline time. The <u>lead</u> time value, expressed in hours or minutes, represents a time prior to the deadline time at which the set or change value (priority) is to be applied.

Two other parts of a deadline algorithm are optional and will be described after a simple example involving only the required operands.

Let's suppose that our installation had defined an algorithm, type C, as

```
DEADLINE,C=(11,1H)
```

We can tell that at 1 hour before a job's deadline time, algorithm C will cause the job's priority to be set to 11. If a job with the JCL

```
//MYJOB   JOB  .....,PRTY=4
//STEP1   EXEC   ....
```

including the following JES3 control statement

```
//MYJOB   JOB   ....,PRTY=4
//*MAIN ...,DEADLINE=(1500,C)
//STEP1   EXEC   ....
```

enters our system at 9:00 am and has not completed by 1400 (1 hour prior to its deadline time of 1500), its priority is to be set to 11 at 1400. So, using this simple algorithm, if a job completes prior to 1400, Deadline Scheduling has had no effect on the job; however, if not completed by 1400, the change in priority will enhance the chances of the job's being scheduled. Should the job in our example have entered the system after 1500 (its deadline), but before 2400, the deadline would be assumed to be 1500 tomorrow.

Several other options are available for Deadline Scheduling:

- the set value in the algorithm could have been specified as a change value such as +3 ... meaning to increment the job's original priority by 3.

- additional increments and time intervals may also be specified. | 6.2

- the user may include a date or a relative cycle specification in the DEADLINE operand of //*MAIN.

More detailed information will either be supplied by your installation (with regard to its use of Deadline Scheduling) or may be found in the OS/VS2 JCL manual in the section describing JES3 control statements.

Dependent Job Control

Applications usually consist of multiple processing stages. These stages may relate to each other in very simple or very complex ways. Some stages may be optional, based upon successful or unsuccessful completion of prior stages. In prior operating systems environments, the stages of processing usually had to be related through the mechanism of job steps, since there was no job-to-job relationship. JES3 provides Dependent Job Control (DJC) as a means of relating the jobs which make up an application.

DJC

We use the term network to describe a set of related jobs which must run in a predetermined order. The "ordering" of jobs is usually necessitated by data set dependencies, though there may be other reasons as well. We include in each job's JCL a JES3 control statement, the //*NET statement, to define the structure of our network. In the illustrated network, pictured by what we refer to as a node diagram, we are showing that job B cannot run until after job A completes. Jobs C and D are to be run, in that order, after successful completion of job B, jobs E and F, in that order, only if job B abnormally terminates.

NETWORK

I 6.3

I 6.4

NODE

Without becoming too deeply involved in coding detail, let's examine our example network, whose name (ID) is "SAMPLE". The //*NET statements indicate:

- for JOBA - there are no predecessors indicated since HC (hold count) is defaulted to zero. Upon completion of JOBA, JOBB (successor to JOBA) is to be released (RL=JOBB) for normal scheduling by decrementing its hold count.

HOLD COUNT

- for JOBB - there is one predecessor (JOBA), indicated by HC=1. JOBB cannot proceed until its HC is reduced to zero. At completion of JOBB, hold counts for JOBC and JOBE are to be decremented ((RL=(JOBC,JOBE)).

- for JOBC - this job's hold count will only be decremented if JOBB normally completes (NC=D). If JOBB abnormally completes,

JOBC is to be flushed (AB=F) from the system; JOBD will also be flushed.

- for JOBD - run this job upon successful completion of JOBC (NC=D), retain it if JOBC abnormally completes (AB=R).

- for JOBE - run this job only if JOBB completes abnormally (AB=D). Should JOBB run normally, JOBE is to be flushed, along with JOBF.

Dependent Job Control has many items of noteworthy interest associated with it. Jobs as a network may be read into the system in essentially any order and at varying times. The structure and content of the control blocks that are built for DJC allow the DSP to schedule the proper job at the proper time, and we will "wait" for missing jobs of a network. This same control block structure disallows duplicate network names in the system; jobs in a second network with the same name will be considered part of the first network. Early completion of a long running job may be simulated via a console message (issued by a problem program) so that successor jobs may be scheduled earlier than otherwise. DJC is driven by job completion, not by job step completion. Within a job, however, the COND= entry coded on EXEC statements can still be used. Several large JES3 installations have production job stream generating programs which also generate the DJC control statements for their applications.

Recovery Management

JES3's recovery facilities have been greatly enhanced when compared to its predecessor, ASP. These recovery facilities can be categorized into three distinct groupings, allowing recovery for:

- software failures.
  - JES3
  - MVS
- hardware failures of global processor.
- DSP failures.

Our discussion will address the first two of these three categories.

System Recovery - Software Failures

In the event of software failure in the JES3 address space, recovery involves an initialization of JES3, but has no major effect upon user jobs executing at the time of the failure. User jobs throughout the complex continue to execute normally until they reach a point of requiring a global JES3 service. Until global JES3 is reinitialized, user jobs requiring global service are MVS WAITed. The normal, first attempt at recovery is a JES3 HOTSTART, which initializes JES3 from information spooled during the last incidence of reading the set of initialization statements (WARMSTART or COLDSTART). During HOTSTART, JES3 reestablishes itself based upon its checkpoints, validity checks control tables in CSA, and restructures each of its processing queues by reading the job queue (JCT entries). Jobs found to have been active at the time of failure are restarted at the scheduler element upon which the job was active; this restart is only an "internal" JES3 restart.

HOTSTART

JES3 HOTSTART may also be used if an MVS failure occurs on the global machine. In this case, however, since MVS must be IPLed, user jobs in execution on the global processor will be temporarily lost. They will be MVS restarted if they were journaled. Jobs not journaled will have JES3 failure option applied:

- restart - Restart the job at the beginning (of the job).
- cancel - Print any output generated to time of failure and cancel the job out of the system.
- hold - Place the job in operator hold for later restart.
- print - Print any output, then hold for restart.

In addition, because the global JES3 address space is lost in the case of MVS failure, local processors must resend any outstanding requests

for global service when the global has been HOTSTARTED.

If a WARMSTART is necessary, either because HOTSTART was unsuccessful, or because the system restart is a planned one, all JES3 processors must be re-IPLed. This means, of course, that all user jobs in execution will be temporarily (at least) lost. Those which were journaled will be MVS restarted; unjournaled jobs have the JES3 failure option applied.

WARMSTART

Failure of a local JES3 address space causes no loss of user jobs, even in the same processor as the failure. The operator LOCAL starts the JES3 address space, the local processor reconnects to the global, and the failure is transparent to user jobs.

A LOCAL start is also involved if MVS fails on a local processor. The effect is, however, a bit more drastic. Because MVS must be re-IPLed, user jobs in that machine must be restarted ... either MVS checkpoint-restart or JES3 failure option applied.

LOCAL START

System Recovery - Global Hardware Failure

The global JES3 address space is the "heart" of a JES3 complex. A hardware failure in the global processor is relatively more serious than the loss of a local processor, particularily from a total complex standpoint. The reaction to this global failure is to "move" the global JES3 facility to another processor in such a manner that we do not totally lose the JES3 complex. We call this procedure Dynamic System Interchange (DSI).

DSI

DSI, internally a complicated procedure, must have been thoroughly planned during the design of a JES3 installation. This is due to two reasons:

- we are going to move the global facility to a "new" global processor.
- we do not process an initialization deck ... during initialization of the original global processor, locals which are

potential globals must be identified. This identification is necessary in order to specify CTC paths to other locals from the machine that is becoming the global processor.

All, or a major subset, of the "old" global devices must be accessible by the "new" global. This accessibility usually involves hardware switching for unit record devices, some magnetic tapes, and teleprocessing lines used for Remote Job Processing.

The DSI process is begun and controlled by operations personnel, and it is a multistep transition of global capability. The local (to become global) JES3 address space is brought down in a special way, necessary devices switched to the "new" global, and a HOTSTART is simulated. The result is that we have, after DSI, a "new" global processor. Though we cannot downplay the loss of computing capacity, our complex can continue to supply JES3 services to its users.

When the "old" global is repaired, two options are available as choices as to how to use it. First, it may be LOCAL started and simply run as a local processor. The installation may, on the other hand, wish to reestablish that processor as global. If the latter option is chosen, the normal course of events is to bring up the repaired processor as a local, then DSI the global facility back to it. Now again, we have a "new" global and the "old" global must be LOCAL started to bring the complex back to its normal operating status.

GLOSSARY

This glossary provides definitions of OS/VS2 and JES3 terms used in this publication. For definitions not included, see IBM Data Processing Glossary, GC20-1699.

address space. The virtual storage assigned to a job, TSO user, or a task initiated by the START command. Each address space consists of the same range of addresses.

ASP. Asymmetric multiprocessing system. An extension of the IBM operating system (OS/MVT and OS/VS2 Release 1) that provides increased automation of computer operations for large scale data processing users.

ASP main processor. A processor supported by JES3 that is executing under either OS/MVT or OS/VS2 Release 1. The ASP MAINTASK modules must be present to communicate with JES3.

channel-to-channel adapter. A hardware device that is used to connect two channels on different systems.

cold start. The first start up after system generation. Cold start invokes the initialization process and may also be required after a catastrophic failure.

console service. Routines that provide two way communication between the operator and JES3.

converter/interpreter. The job segment that converts and interprets JCL for jobs that execute on MVS processors.

CTC. See channel-to-channel adapter.

DDR. See dynamic device reconfiguration.

dependent job control (DJC). The organizing of a collection of jobs that must execute in a specific

order. DJC manages jobs that are dependent upon one another.

DSI. See dynamic system interchange.

DSP. See dynamic support program.

dynamic allocation. Assignment of system resources to a job at the time the job is executed rather than at the time it is read into the system.

dynamic device reconfiguration (DDR). A facility that allows a demountable volume to be moved, and repositioned if necessary, without abnormally terminating the job or repeating the initial program load procedure.

dynamic support program (DSP). The executable code segments of JES3 that perform JES3 processing facilities on behalf of jobs in the system. Card reading and printing are examples.

dynamic system interchange (DSI). Allows the operator to switch the global JES3 functions to a local processor in the case of global processor failure.

dynamic writer. An output service function that controls printing or punching of data sets with characteristics that are not assigned to a specific device but are assigned by JES3 to appropriate devices as they become available.

early resource release. The releasing of resources (devices volumes, and data sets) after they are no longer needed.

explicit setup. The programmer specifies on a JES3 control card precisely which devices are to be set up.

external writer. In OS/VS2, a program that supports the ability to write SYSOUT to devices not supported by the job entry subsystem.

generalized main scheduling (GMS). A set of algorithms that allow the JES3 system programmer to tailor job scheduling and selection to the specific needs of the installation.

global main processor. The MVS processor in the JES3 installation in which the JES3 address space manages jobs for all main processors, including itself.

global processor. See global main processor.

GMS. See generalized main scheduling.

HASP. An extension to the System/360 operating system that provides supplementary job management, data management, and task management functions such as control of job flow.

high watermark setup. An attempt to allocate a minimum number of unique device types that fulfill the requirements for each job step. Devices used in one step can be released and used again in later steps.

hot start. A restart that reinitializes JES3 automatically without allowing changes to the initialization control cards. Recovery of all jobs in execution is attempted.

hot writer. An output service program that controls printing or punching of data sets wth characteristics that are assigned for processing on a specific device.

initialization. In JES3, the process that reads the JES3 initialization control statements and creates the tables and control blocks used throughout the JES3 program.

input service. The function that accepts and queues all jobs entering the JES3 system.

internal reader. A facility that transfers jobs to the JES3 input sevice function.

JES2. A functional extension of the HASPII program that receives jobs into the system and processes all output data produced by the job.

JES3 spool access method (JSAM). Data management routines that serve JES3 address space requests such as allocation and deallocation of JES3 buffers.

JSAM. See JES3 spool access method.

local main processor. The MVS processor(s) connected to and controlled by the global main processor and sharing the JES3 queue. JES3 is active in the local main processor for communication with the global main processor.

local processor. See local main processor.

loosely-coupled multiprocessing. Two or more computing systems interconnected by an I/O channel-to-channel adapter. The CPUs can be of different types and have their own unique configurations.

main device scheduler (MDS). Controls the setup of I/O devices associated with job execution.

main processor. Any processor in the JES3 installation on which jobs can execute. The three types of main processors are (1) global main, (2) local main, and (3) ASP main.

main service. A dynamic support program that schedules problem programs for execution and manages the flow of data (system input, print, and punch) across the channel-to-channel adapter to and from the global processor.

MDS. See main device scheduler.

migration. Term used to define the changing over from an installation's production operating system to an upgraded or entirely new operating system, such as from OS/MVT to OS/VS2.

multiple virtual storage (MVS). A virtual storage facility that allows each user a private address space. MVS is provided at OS/VS2 Release 2 and subsequent releases. JES3 is provided at OS/VS2 Release 3 and subsequent releases.

multiprocessing system. A computing system employing two or more interconnected processing units to execute programs simultaneously.

MVS. See multiple virtual storage.

network job processing (NJP). A facility that permits the input, processing, and output of jobs to and from compatible, remotely-located JES3 installations.

NJP. See network job processing.

output service. The function that processes SYSOUT data sets. Processing includes printing, punching, or directing output to an external writer.

pre-execution setup. That portion of setup performed by MDS prior to a job entering execution.

primary job-entry subsystem. The active job-entry subsystem. The primary job-entry subsystem is determined during the system generation process.

reader/interpreter. The job segment that reads and interprets JCL for jobs that execute on ASP main processors.

remote job processing (RJP). A facility that permits the input, processing, and output of jobs to and from terminals remote from the JES3 installation.

remote terminal processor (RTP). A programmable remote workstation.

RJP. See remote job processing.

RMT. See remote terminal processor program.

remote terminal processor program (RMT). A self-loading object deck created as a result of an RMT generation. RTP programs allow for JES3 to communicate with programmable remote workstations.

RTP. See remote terminal processor.

setup. Consists of volume fetching, allocation, mounting, and deallocation.

SSI. Subsystem interface.

subsystem interface (SSI). A set of program routines that allow two way communication between a JES3 address space and other address spaces.

USAM. See user spool access method.

user exit. A subroutine in an IBM provided program module that allows users to insert their own unique analysis and processing routine.

user spool access method (USAM). Data management routines that do not execute in the JES3 address space but provide the subsystem interface for allocation, deallocation, SYSIN/SYSOUT, OPEN, and CLOSE functions of user data sets.

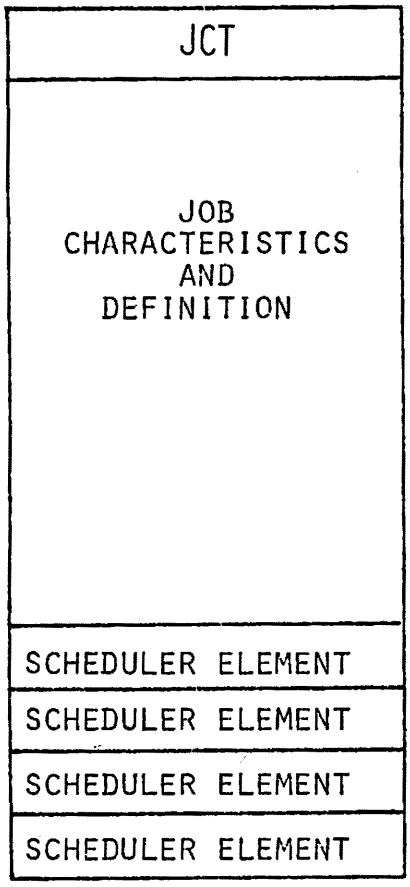
warm start. A restart that allows reinitialization. Jobs that were in execution remain in the job queue; however, an IPL must be performed for each processor. In ASP, this type of start is called "restart".

workstation. A terminal device that may or may not be a CPU. At a workstation, an operator can connect into a central system via SIGNON, enter jobs, and receive output.

APPENDIX

The illustrations which follow are ordered to conform with the references within the text material.

THE JOB CONTROL TABLE ENTRY

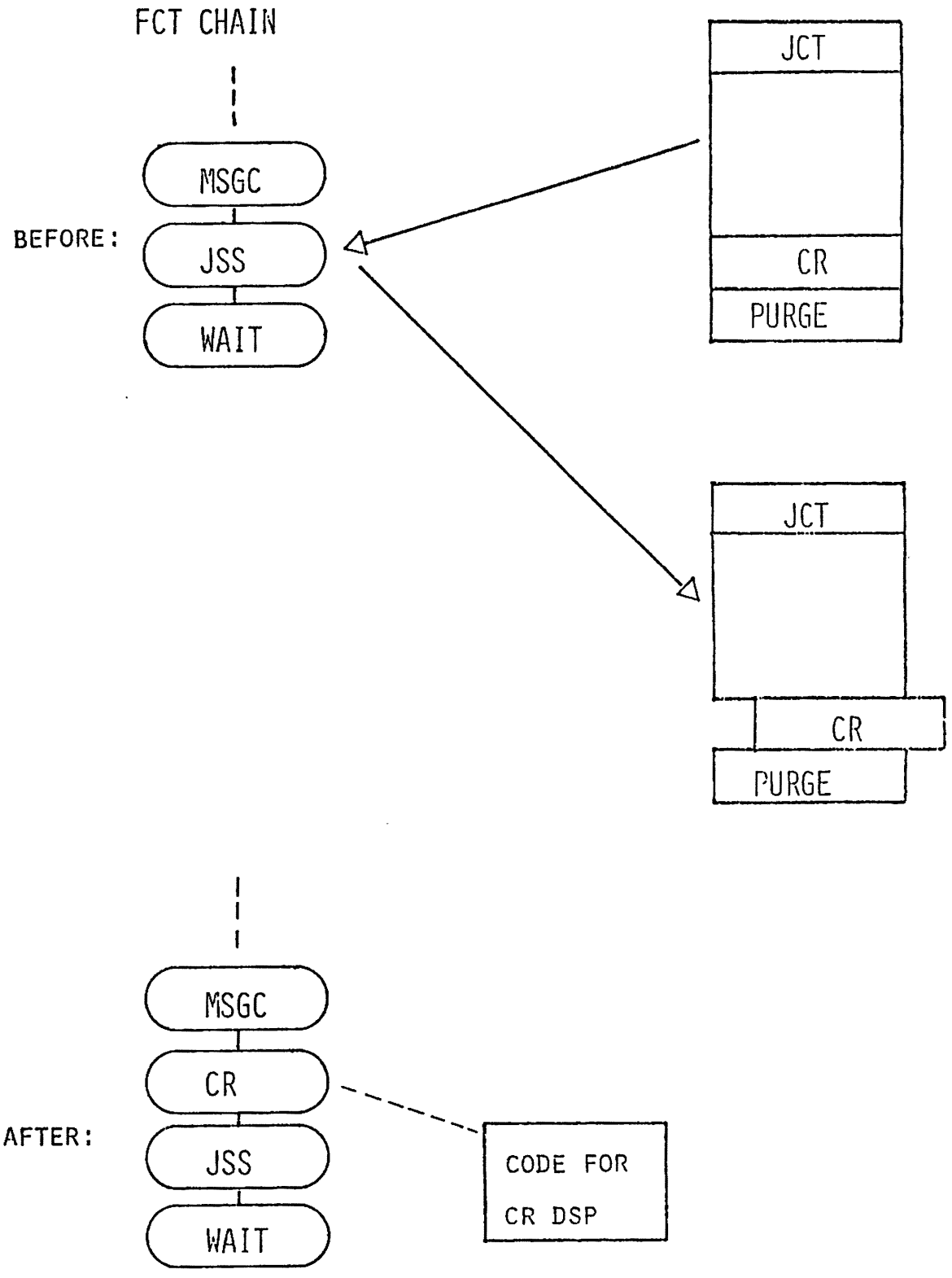


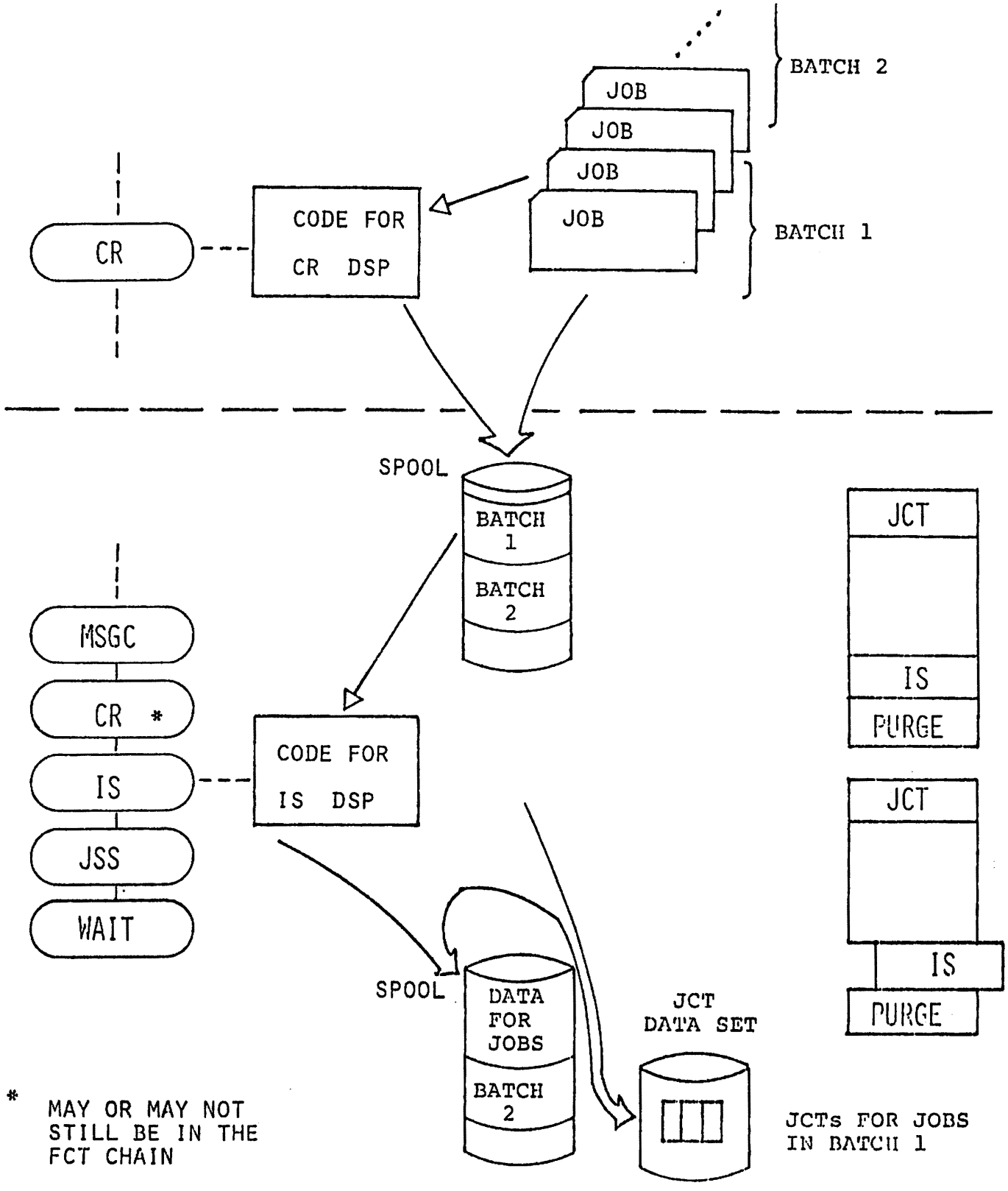
THE FIXED PORTION OF THE JCT ENTRY CONTAINS THE DEFINITION OF THE JOB... ITS CHARACTERISTICS AND VARIOUS FLAGS WHICH IDENTIFY THE STATUS OF THE JOB.

SCHEDULER ELEMENTS REPRESENT THE SEGMENTS OF WORK TO BE PERFORMED ON BEHALF OF THE JOB BY JES3.

RESIDENT JES3 FCTS

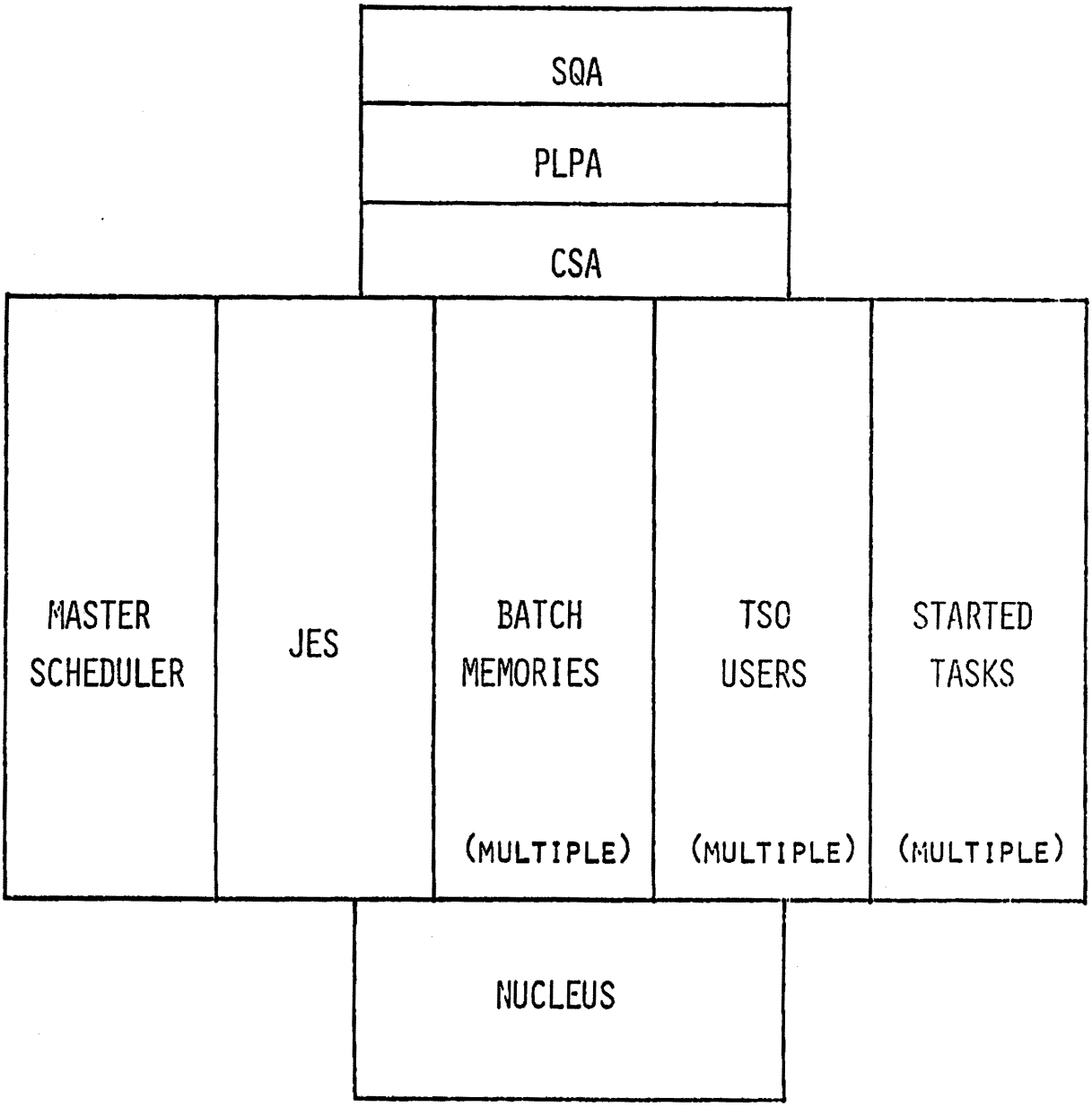
FCTENTRY	PRIORITY	DESCRIPTION	DRIVER
CONSOLES	255	CONSOLES	IATCNSL
CONSERV	254	CONSOLE SERVICE	IATCNSV
JSAM	250	DSP SPOOL I/O	IATDMBG
CONSDM	240	CONSOLE SPOOL I/O	IATCNDM
MAIN	53	MVS CTC I/O	IATMSMI
MAIN	52	ASP CTC I/O	IATMSIO
MAIN	51	MVS GENERALIZED MAIN SCHEDULING	IATMSMS
MAIN	50	ASP GENERALIZED MAIN SCHEDULING	IATMSMN
DYNAL	35	DYNAMIC ALLOCATION	IATDYDR
OUTSERV	30	OUTPUT SERVICE	IATOSDR
VERIFY	30	VOLUME VERIFICATION	IATLVVR
SETUP	30	MAIN DEVICE SCHEDULING	IATMDDR
SMFWGST	30	SMF RECORD WRITER	IATOSDR
MODDRVR	15	MODIFY PROCESSING	IATMODV
INQDRVR	14	INQUIRY PROCESSING	IATIQDV
WTDDRVR	12	WORK-TO-DO PROCESSING	IATGRWD
LOCATE	5	LOCATE	IATLVIN
MSGC	4	MAIN SERVICE GENERAL COLLECTION	IATMSGC
JSS	2	JOB SEGMENT SCHEDULER	IATGRJS
FAILSOFT	1	FAILING CODE PROCESSOR	IATFSLG
WAIT	0	ENTRY TO MULTIFUNCTION MONITOR	IATGRCT


FCT CHAIN
BEFORE:
MSGC
JSS
WAIT
JCT
CR
PURGE
JCT
CR
PURGE
AFTER:
MSGC
CR
JSS
WAIT
CODE FOR
CR DSP

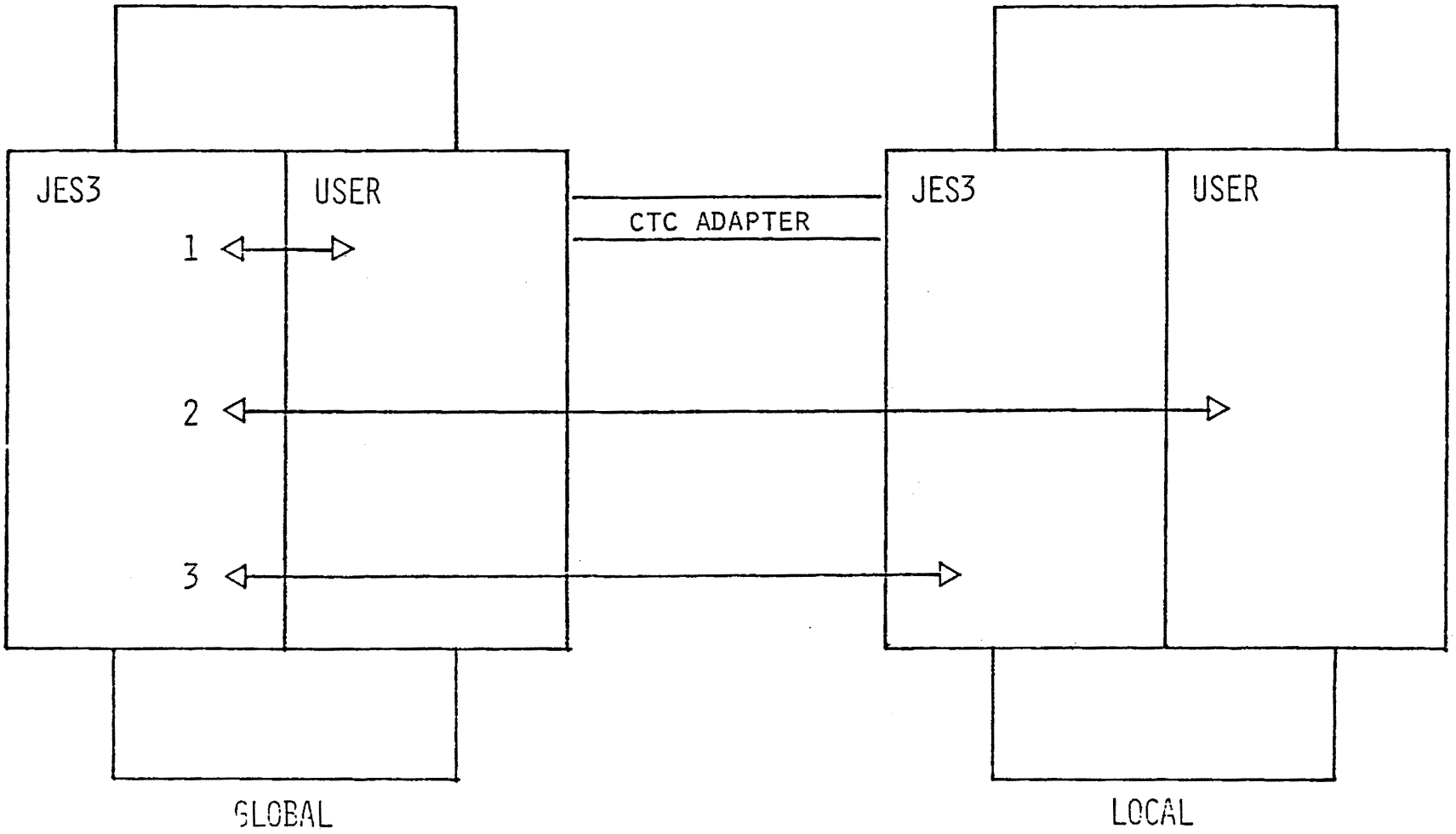


* MAY OR MAY NOT STILL BE IN THE FCT CHAIN

MVS MEMORY MAP



ADDRESS SPACE COMMUNICATIONS



1 GLOBAL JES3 TO GLOBAL USER VIA SRB SCHEDULE

2 GLOBAL JES3 TO LOCAL USER VIA SSISERV MACRO

3 GLOBAL JES3 TO LOCAL JES3 VIA JSERV MACRO

SSI FUNCTION CODES

SCHEDULING

	5	JOB SELECTION
	22	STEP INITIATION
*	4	STEP TERMINATION
	12	JOB TERMINATION
	13	JOB RE-ENQUEUE
*	8	END OF MEMORY
	32	"S INIT" COMMAND FAILURE

SYSIN/SYSOUT

*	6	ALLOCATION
	16	OPEN
*	17	CLOSE
	7	UNALLOCATION
*	18	CHECKPOINT
*	19	RESTART

DEVICE/DATA SET MGMT

	23	DYNAMIC ALLOCATION
*	24	COMMON ALLOCATION
	25	COMMON UNALLOCATION
	26	CHANGE DDNAME
	27	CHANGE ENQUEUE USE ATTRIBUTE
	35	MSS VOLUME INVENTORY
	36	MSS MOUNT EQUALIZATION
	37	MSS OPEN/EOV

TSO

	1	OUTPUT
	2	CANCEL
	3	STATUS
	11	VALIDATE

DDR

	28	CANDIDATE SELECTION
	29	CANDIDATE VERIFICATION
	30	SWAP NOTIFICATION
	31	SWAP COMPLETE

CONSOLES

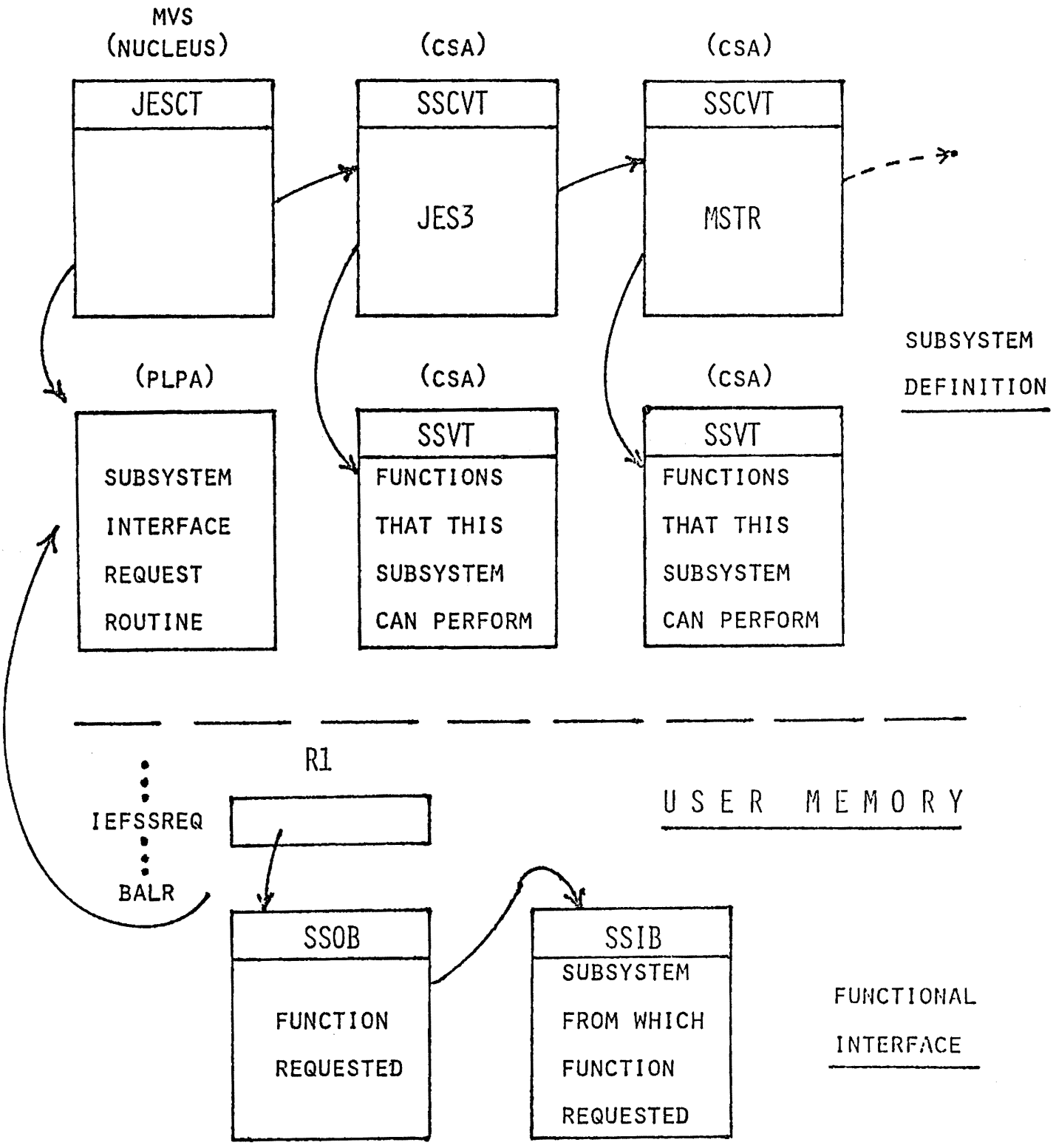
	9	WTO/WTOR
	10	SVC 34
	14	DOM
	33	MASTER CONSOLE SWITCH
	34	WTL

MISCELLANEOUS

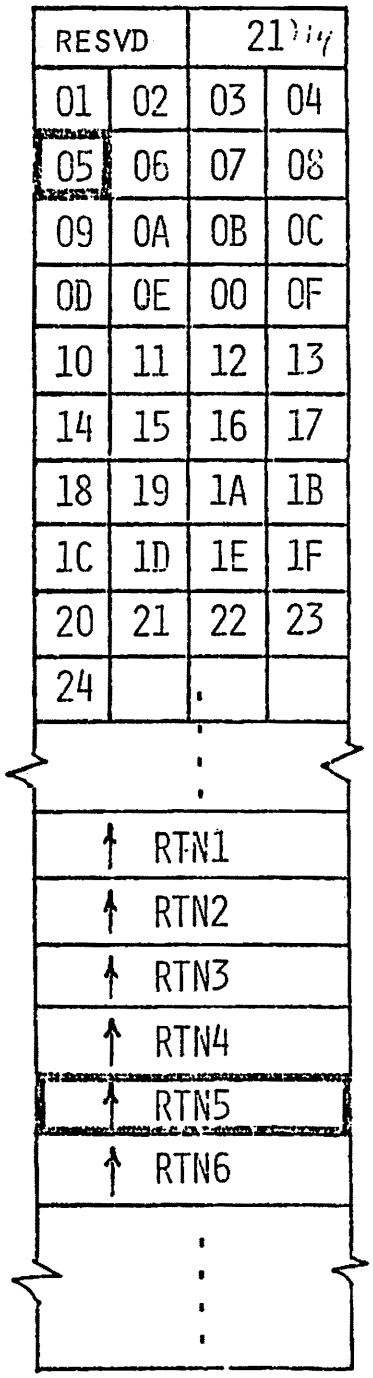
	15	VERIFY SUBSYSTEM ID (UNSUPPORTED)
	20	REQUEST JOB ID
	21	RETURN JOB ID

* PROCESSED BY FUNCTION ROUTINE

SSI-RELATED CONTROL BLOCKS



STRUCTURE OF
SUBSYSTEM VECTOR
TABLE



SPOOL DATA MANAGEMENT
FUNCTIONAL COMPONENTS

JSAM

1. Macro Routines
2. Spool Space Allocation
3. User Track Group Allocation Interface
4. JSAM Function

USAM

1. Macro Routines
2. DASD Block Allocation

COMMON

1. Spool I/O Initiation and Termination

SPOOL DATA MANAGEMENT
FUNCTIONAL COMPONENTS

JSAM

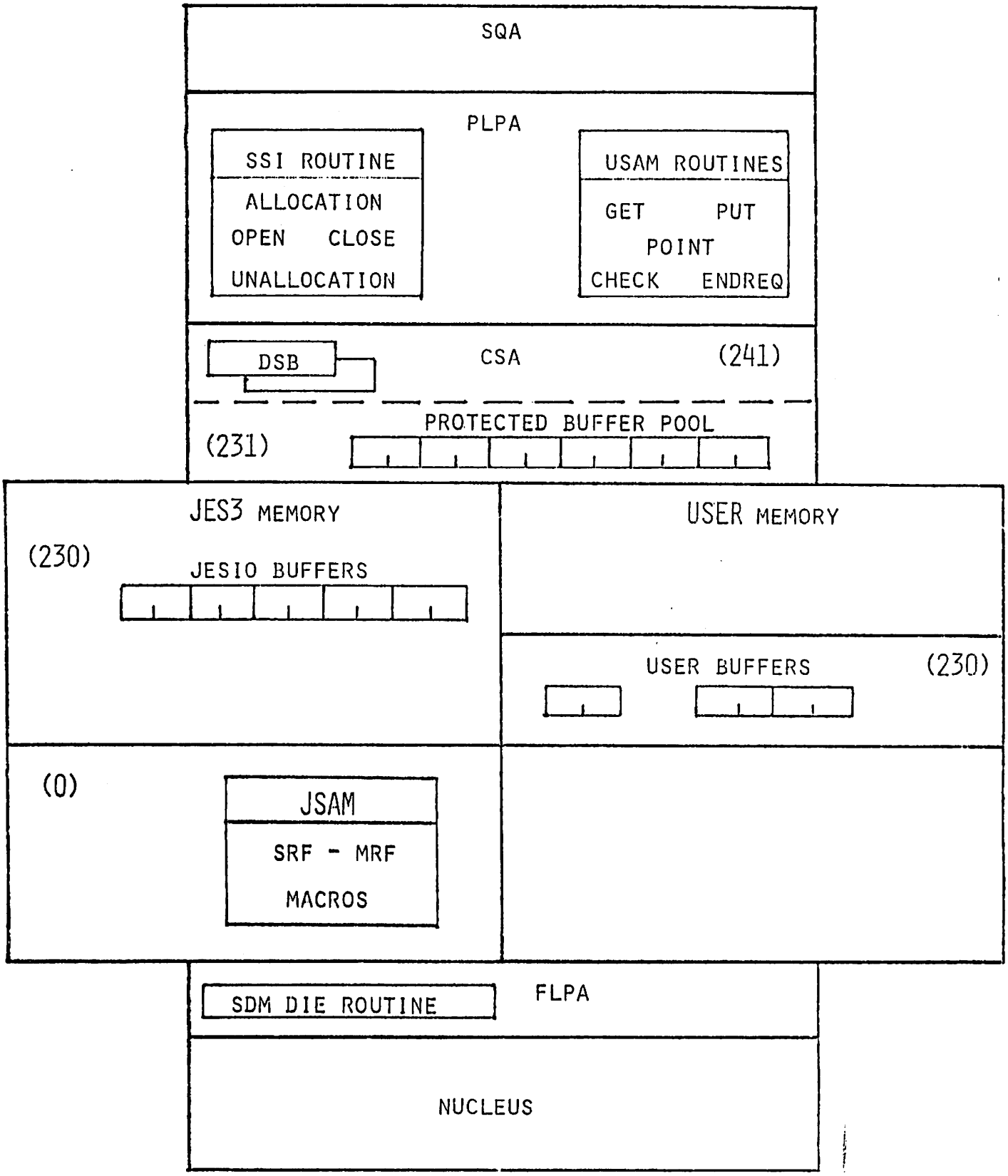
1. Macro Routines
   - SRF Processing
   - MRF Processing
2. Spool Space Allocation
   - ALLOCATES TRACK GROUPS FOR DSPs AND USER JOBS
   - ALLOCATES BLOCKS WITHIN TRACK GROUPS
3. User Track Group Allocation Interface
   - SERVES AS PATH TO TRACK GROUP ALLOCATION, ABOVE
4. JSAM Function
   - SENDS NOTIFICATION OF JSAM COMPLETIONS TO DSPs

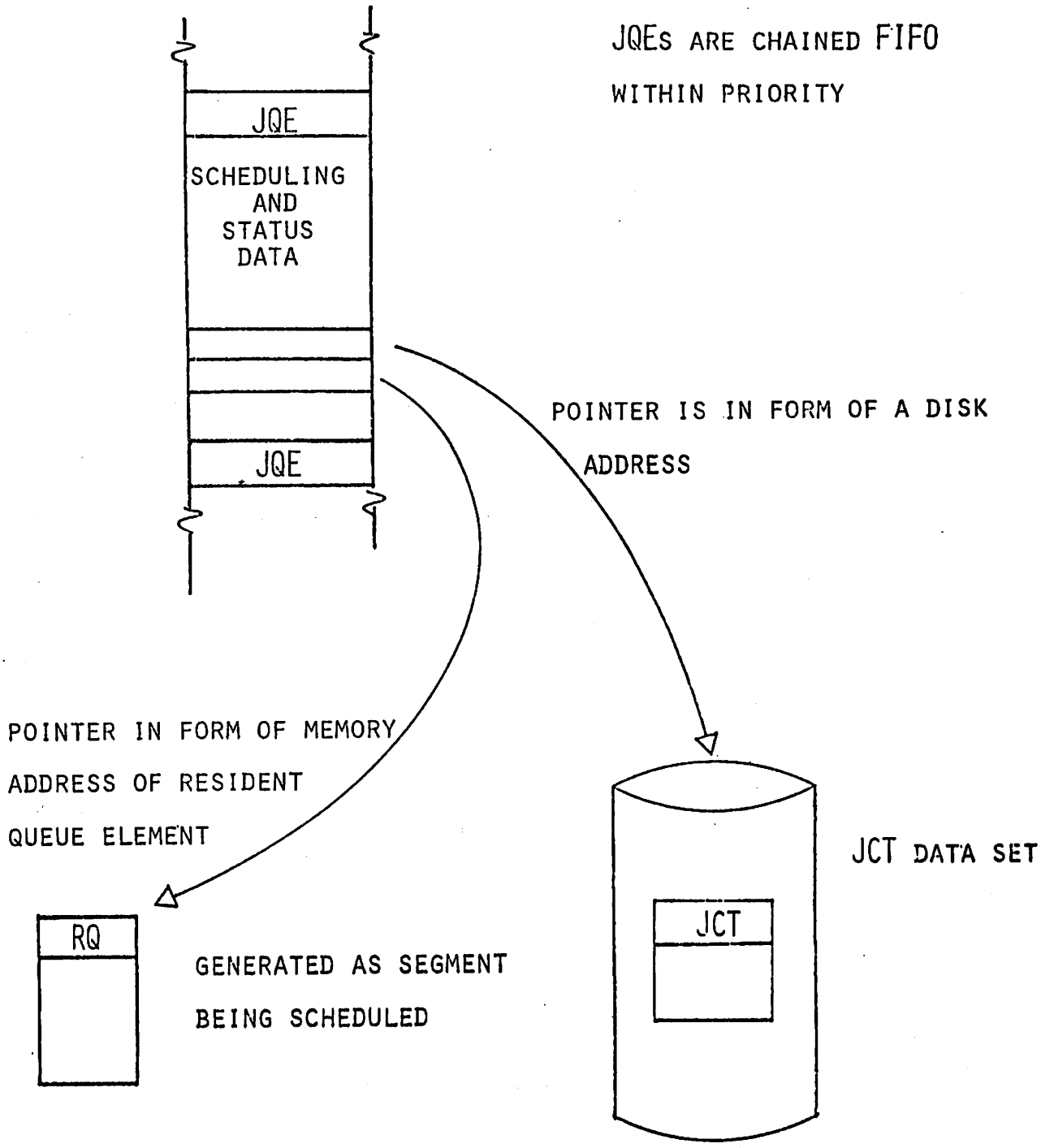
USAM

1. Macro Routines
   - ENTERED BY MACROS ISSUED IN COMPATIBILITY INTERFACE
2. Record Allocation
   - ALLOCATES BLOCKS WITHIN TRACK GROUPS

COMMON

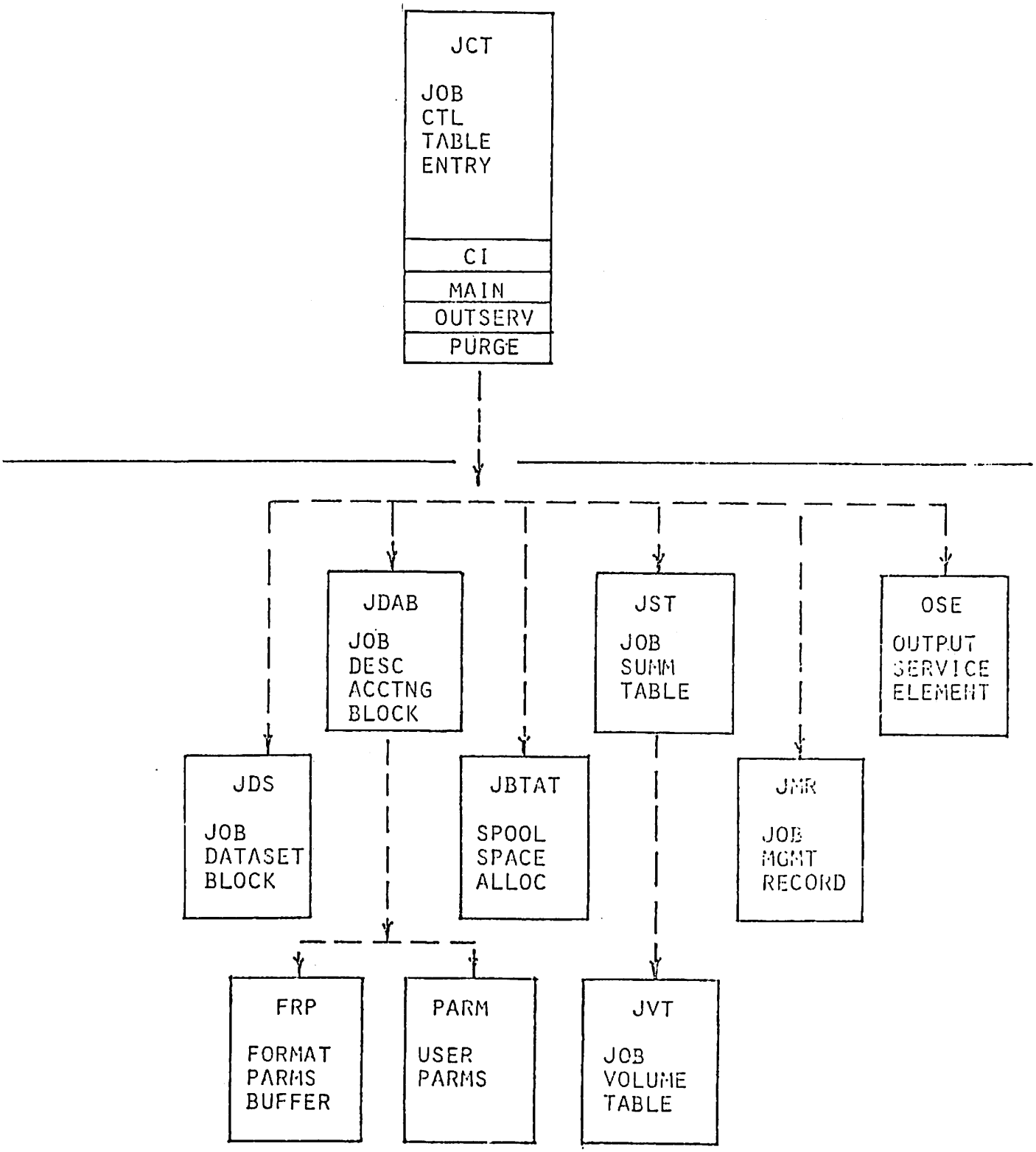
1. Spool I/O Initiation and Termination
   - STARTS ALL REQUESTS BY SRB SCHEDULING
   - SENDS JSAM COMPLETIONS TO JSAM FUNCTION
   - SENDS USAM COMPLETIONS TO USER ADDRESS SPACES


SQA
PLPA
SSI ROUTINE
ALLOCATION
OPEN CLOSE
UNALLOCATION
USAM ROUTINES
GET PUT
POINT
CHECK ENDREQ
DSB
CSA
(241)
(231)
PROTECTED BUFFER POOL
JES3 MEMORY
(230)
JESIO BUFFERS
USER MEMORY
USER BUFFERS
(230)
(0)
JSAM
SRF - MRF
MACROS
SDM DIE ROUTINE
FLPA
NUCLEUS


JQE
SCHEDULING AND STATUS DATA
JQE
JQEs ARE CHAINED FIFO WITHIN PRIORITY
POINTER IS IN FORM OF A DISK ADDRESS
POINTER IN FORM OF MEMORY ADDRESS OF RESIDENT QUEUE ELEMENT
RQ
GENERATED AS SEGMENT BEING SCHEDULED
JCT
JCT DATA SET

JOB CONTROL BLOCKS

"LIFE OF JOB"

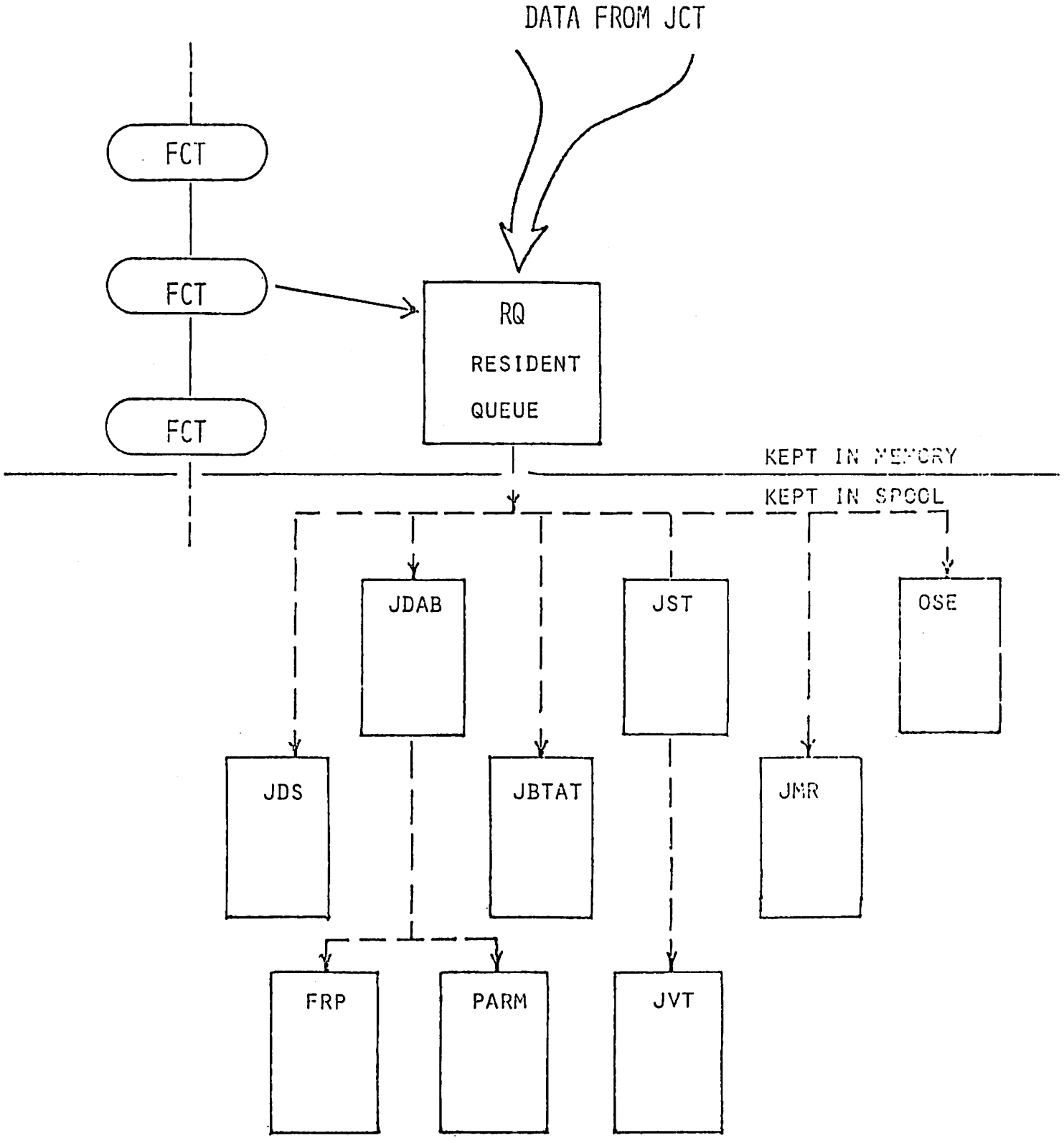


JCT KEPT IN UNIQUE DATA SET CONTAINING ALL JCTS

ALL OTHERS KEPT IN SPOOL

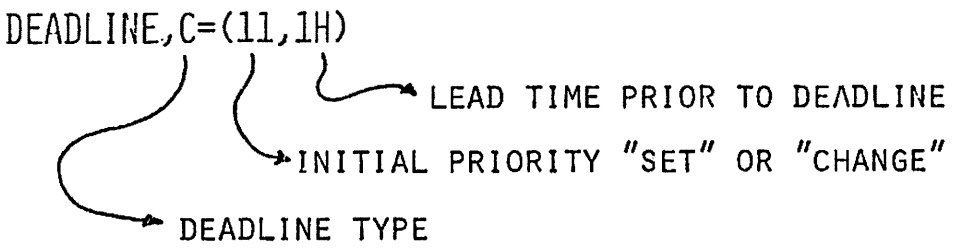
JOB CONTROL BLOCKS

"ACTIVE FUNCTION"



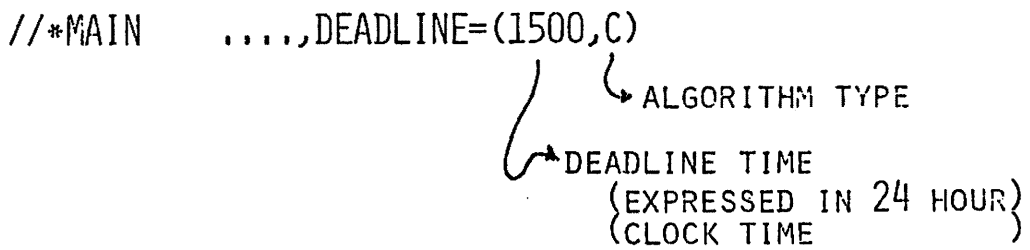
1. IN THE JES3 INITIALIZATION STATEMENTS, DEFINE NEEDED ALGORITHM TYPES.

   EXAMPLE -



2. IDENTIFY THE NEED FOR DEADLINE SCHEDULING WITH JES3 CONTROL STATEMENT OPERANDS FOR INDIVIDUAL JOBS.

   EXAMPLE -



1. ALGORITHM DEFINITION:

DEADLINE,B=(+3,2H,+1,20M)

THESE OPERANDS SPECIFY ADDITIONAL PRIORITY INCREMENTS OF +1 EACH TWENTY MINUTES FOLLOWING THE INITIAL PRIORITY CHANGE, UNTIL THE JOB COMPLETES OR A PRIORITY OF 14 IS REACHED.

NOTE THAT THE INITIAL PRIORITY IS SHOWN HERE AS AN INCREMENT RATHER THAN A "SET" VALUE

2. CONTROL STATEMENT:

//*MAIN ....,DEADLINE=(1600,B,OPTION)

THE OPTION FIELD MAY BE CODED AS A DATE - MMDDYY, OR A RELATIVE CYCLE SPECIFICATION:

3,WEEKLY	3RD DAY OF WEEK
12,MONTHLY	12TH DAY OF MONTH
117,YEARLY	117TH DAY OF YEAR

DEPENDENT JOB CONTROL

- DEFINE A JOB WITHIN A NETWORK OF JOBS
- SPECIFY THE RELATIONSHIPS OF THE DEPENDENT JOBS

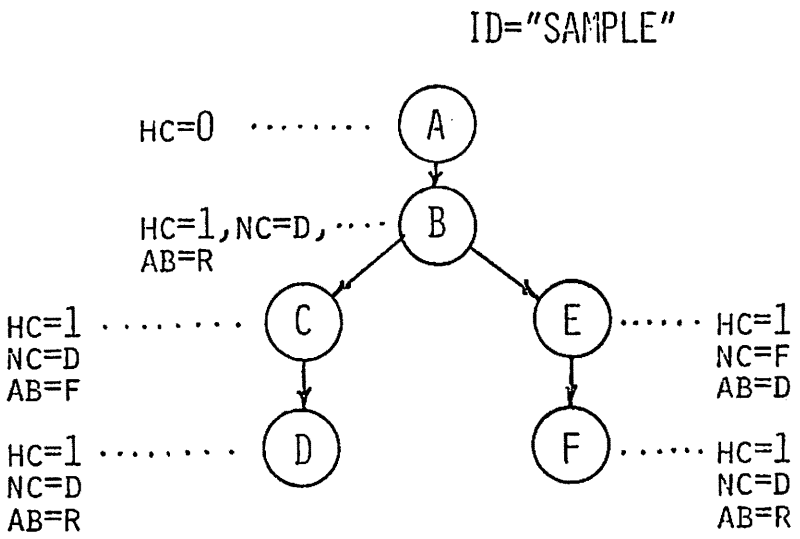
//*NET		FACTORS TO BE CONSIDERED
NETID	ID=NAME	- THE NETWORK I BELONG TO
NHOLD	HC=N	- HOW MANY IMMEDIATE PREDECESSORS DO I HAVE ?
RELEASE	RL=(JOBNAME,...)	- WHAT ARE MY IMMEDIATE SUCCESSORS
NORMAL	NC= D | F | R	ACTION TAKEN FOR NORMAL OR
ABNORMAL	AB= D | F | R	- ABNORMAL TERMINATION OF ANY IMMEDIATE PREDECESSOR

D - DECREMENT NHOLD COUNT

F - FLUSH THE JOB AND IT'S SUCCESSORS

R - RETAIN ; DO NOT DECREMENT NHOLD COUNT

NODE DIAGRAM FOR SAMPLE NEWWORK



JOB CONTROL LANGUAGE

```
//JOBA   JOB   ...
//*NET   ID=SAMPLE,HC=1,RL=(JOBB)
         :
//JOBB   JOB   ...
//*NET   ID=SAMPLE,HC=1,RL=(JOBC,JOBE)
         :
//JOBC   JOB   ...
//*NET   ID=SAMPLE,HC=1,NC=D,AB=F,RL=(JOBD)
         :
//JOBD   JOB   ...
//*NET   ID=SAMPLE,HC=1,NC=D,AB=R
         :
//JOBE   JOB   ...
//*NET   ID=SAMPLE,HC=1,NC=F,AB=D,RL=(JOBF)
         :
//JOBF   JOB   ...
//*NET   ID=SAMPLE,HC=1,NC=D,AB=R
```

READER'S COMMENTS

Title: JES3 - A PRIMER
Washington Systems Center
Technical Bulletin GG22-9200-00

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may of course, continue to use the information you supply.

Please state your occupation: _____

Comments:

Please mail to: J. M. Eubanks
IBM Corporation
18100 Frederick Pike
Gaithersburg, Md. 20760

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601