RTOS—Extending OS/360 for real time spaceflight control

by J. L. JOHNSTONE

International Business Machines Corporation
Houston, Texas

INTRODUCTION

The Real Time Operating System/360 (RTOS/360), a modified version of the standard IBM System/360 Operating System (OS/360)*, was developed by the Federal Systems Division (FSD) of IBM for support of the Real Time Computer Complex (RTCC) during NASA's Apollo spaceflights. RTOS/360 is a real time, multi-tasking, multi-jobbing operating system that extends the basic features of OS/360 and adds additional features to:

- Process real time data
- Provide simplicity of use for the applications programmer
- Ensure fast response system activity (requirements range from one-tenth of a second to one second)
- Improve efficiency
- Provide support for special devices not supported by OS/360
- Provide a fail-safe system
- Increase job shop throughput

The presence of these features in OS/360 does not deter from its basic capabilities; i.e., all the facilities of the current IBM released OS/360 that operate in a standard or non-real time mode of execution are available in RTOS/360.

Some of the major functional areas which were developed at IBM's FSD Houston Operations and added to OS/360 in the formation of RTOS/360 were:

- Independent Task Management
- System Task Capability
- Queue Management
- Data and Time Routing
- Time Management
- Real Time Input/Output Control System
- Data Tables
- Display Formatting Language (DFL)
- Real Time Linkages
- Large Core Storage Support
- Logging
- Simulated Input Control
- Fastime
- Fail-Safe Programs
- Background Utilities
- Houston Automatic Spooling Priority (HASP)
- Statistics Gathering System
- Job Accounting System
- Multi-jobbing

The RTOS environment

Although RTOS/360 can be used in a variety of applications and computer system configurations, it is pertinent prior to discussing its functional areas that we establish the environment in which it was designed to operate, i.e., the Real Time Computer Complex (RTCC), the RTCC hardware, and the RTCC applications programs.

The RTCC

The RTCC is a ground-based computing and data processing complex for NASA's manned spaceflight program. It includes the computer equipment, associated peripheral equipment and program packages to monitor and support—in real time—Apollo missions, simulations, and training exercises.[1]

RTCC is the core of NASA's Mission Control Center

* IBM System/360 Operating System (OS/360) consists of a comprehensive set of language translators and service programs operating under the supervisory control and coordination of an integrated set of control routines. The operating system is designed for use with Models 30, 40, 50, 65, and 75 of Computing System/360.

(MCC) at Houston, Texas. Flight controllers at MCC monitor every phase of a manned spaceflight, from launch through orbit, reentry, and splashdown. During a lunar mission, flight controllers also monitor and support the astronauts during their flight to the moon, the descent to the moon's surface, the liftoff and rendezvous with the mother ship, and the return to earth.

RTCC provides flight controllers with the information they need to monitor the flight and make decisions regarding the mission. This simply means flight controllers sitting at consoles in Houston have precise information in real time such as the status of every onboard system, the condition of the astronauts, their position in space at any desired time up to 40 hours in advance, or the effect that any planned maneuver would have on the spacecraft or the astronauts.

The RTCC is called on to do many things during a mission. Some of the more important or more common requirements include:

- Process radar data during launch and provide flight controllers with present position and velocity
- Provide flight controllers with information on whether or not the spacecraft will achieve orbit
- Process telemetry data and provide flight controllers with vital information such as amount of oxygen remaining in astronaut environmental control system
- Compute the orbital path of the spacecraft from radar data
- Predict the position of the spacecraft at some time in the future
- Compute how and when the spacecraft must accomplish a particular maneuver to change its orbital characteristics
- Compute navigation information to update the Apollo Guidance Computer on board the spacecraft
- Process radar range data and let flight controllers know the spacecraft is on correct lunar transfer flight path, and if not, what maneuvers are necessary to get it back on the correct path
- Monitor the Apollo Guidance Computer during reentry and predict the spacecraft landing point.

In addition to these tasks, and thousands more performed during a typical Apollo mission, the RTCC also has a key role in flight controller and crew training.

To perform the different requirements of the RTCC, each of five IBM System/360 Model 75 computers are assigned a different role and the RTCC is engineered so that these roles can be exchanged at any moment. This unified set of computers allows NASA to run either two actual missions at the same time, two simulated missions at the same time, or a simulated mission and an actual mission at the same time. Figure 1, The RTCC, demonstrates the five systems at work in the latter configuration. In the mission configuration, network data flows into the RTCC from one of the Communications Command and Telemetry Systems (CCATS) at MCC. The data are then sent to the Mission Operational Computer (MOC) in the RTCC, which processes all the real time processing tasks of the Mission, and the Dynamic Standby Computer (DSC), which performs redundancy processing and is ready to function as the MOC, if necessary. In the simulation and training exercise, an Apollo trainer, either at the MCC or Cape Kennedy, is in a closed loop with one of the identical Mission Operational Control Rooms (MOCR). (The other MOCR is being used for the mission in progress.) One simulation computer contains an application program which is generating simulated network data; the other computer is being used as a simulated operational computer. The fifth computer is a standby computer for both exercises; however, it is not idle, but performing job shop checkout for future application program development.

The hardware configurations

There are several System/360 hardware configurations used in the development and execution of the computing systems developed for the real time applications at NASA. Each configuration is supported by a single RTOS/360 system. The configuration used on each of the five computer systems in the RTCC itself consists of a System/360 Model 75 computer with a one-million byte main memory (IBM 2705). (See Figure 2, System/360 Model 75 for Mission Support.) An IBM 2361 Large Core Storage (LCS) acts as a four-million byte extension of main memory as well as a buffering

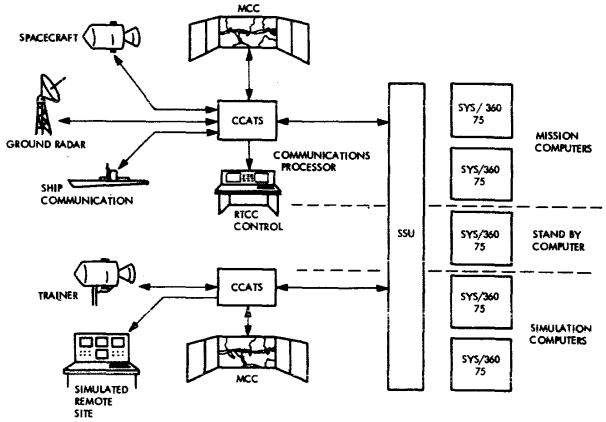

Figure 1—The RTCC

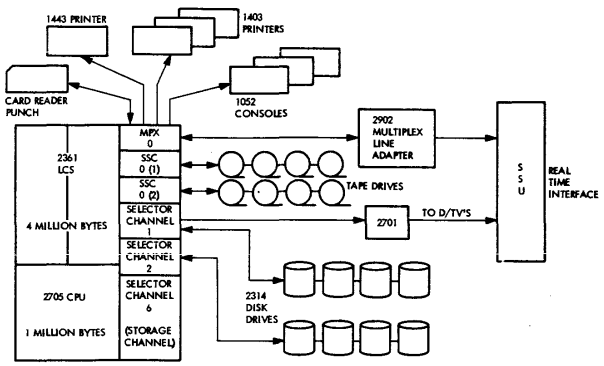

Figure 2—System/360 model 75 for mission support

device for retrieving data and programs from the IBM 2314 disk drives. The IBM 2701 provides a rapid demand response interface to the digital display (D/TV) system in the MOCR and RTCC. Real time acceptance and transmission of large amounts of data and control information are accomplished through the use of the IBM 2902 Multiplex Line Adapter (MLA). A card reader/punch, an IBM 1443 printer, three IBM 1403 printers, two IBM 1052 consoles, and eight tape drives complete the configuration.

Another System/360 Model 75 configuration is used primarily for Simulation exercises. In addition to those devices given in the previous configuration, this Model 75 configuration supports a special Apollo Simulation Processor Channel (ASPC), which receives data from a Multichannel Demultiplexor and Distributor (MDD), an IBM 2260 Display Device, and an IBM 2844 which acts as a control unit for the IBM 2314 disk drives. Several different System/360 Model 50 configurations are also supported by RTOS/360 at the RTCC.

The applications

The applications programming packages used to perform in these various configurations include:

- The Apollo Mission Systems
- The Ground Support Simulation Computer Systems
- The Dynamic Network Data Generation Systems
- The Simulation Checkout and Training Systems
- The Operational Readiness and Confidence Testing Systems.

We've now placed RTOS/360 in its environment; i.e., the RTCC, the RTCC hardware configurations, and the RTCC applications used for real time processing under RTOS/360 control. With this environment in mind, we can now turn to a description of the various functional areas and features designed to extend OS/360 to form RTOS/360. First, let's look at the functional areas.

Functional areas of RTOS/360

Independent task management

In OS/360, all processing is done in conjunction with *units of work* defined as tasks. Tasks are not programs in core storage nor are data for a program a task. A task is a unit of work (programs and data) requiring resources (CPU etc.) to complete its functions. A task exists only when a Task Control Block (TCB) is established and its location is known to the supervisor portion of the operating system. The TCB contains information on such things as pointers to data (I/O), the list of programs needed to operate under the task, the priority of the task, etc. In OS/360, the word "multiprogramming" is replaced by "multi-tasking"; however, the meaning is still the same, i.e., many tasks processing asynchronously—through various paths of logic with the usage of the CPU being switched according to the requirements of the system. (Figure 3, A Task, gives a graphic illustration of a task.)

Some of the characteristics of an OS/360 task are not functionally oriented toward the types of work required to be performed by a real time system. An OS/360 task requires the existence of its creator in order to exist; i.e., it is *dependent* on its creator. This OS/360 concept has been extended within RTOS/360 to include tasks which are *independent* of their creators. This causes a distinction between *dependent* and *independent* tasks. (A dependent task is identical to an OS/360 task.) Therefore, an independent task does not require the existence of its creator in order to exist.

How do the characteristics of an independent task render it more especially suited to real time systems? First, a real time system must be able to receive and process varying data loads rapidly and efficiently. In RTOS/360, an independent task may be defined for each type of data to be processed in real time and be

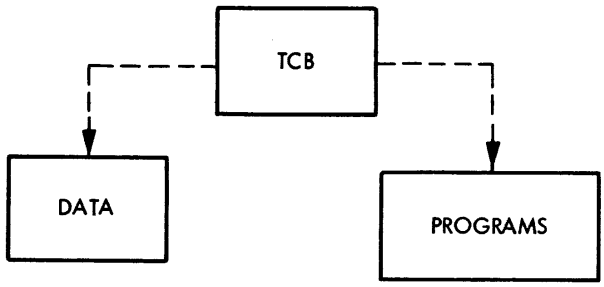

Figure 3—A task

available to receive work at all times even if the data rate is low or random. The major distinction here between dependent and independent tasks is that the independent task will continue to exist in the system when it has no data to process. During this time it is *dormant*. The OS/360 task requires at least one load module executing in order to exist. Each independent task is assigned an area in main core called a resource table. This is a private area that can be used by the programs running under the task. Usually, information is stored in this area which is derived from the processing of earlier data. In this way, the task can "remember" information through periods of dormancy. When data are received by the system for an independent task, they are sent to the task in the form of a request. Each request has its own priority which in turn becomes the task's dispatching priority while processing that request. The OS/360 task has only the priority of its creator. If an independent task is processing a request when another request is generated for it, the new request is enqueued according to its priority. Requests in this queue will be given to the task as it completes the processing of higher priority or older requests.

When an independent task becomes active, it is assigned a unique protect key. This protect key is given to all dependent tasks created by the independent task while processing a request. Therefore, a program running under an independent task or its descendents will be protected from all programs controlled by other active independent tasks or their descendents. Since dependent tasks are assigned the same protect key as their creator's, all tasks of a job step in OS/360 have the same protect key. This is not practical in large real time, multiprogramming systems where many tasks handle various types of data. Independent tasks ensure that unique protect keys will be assigned to unique functions.

Figure 4, OS/360 Task Structure, represents the logical structure of tasks operating as a job step in OS/360. This structure is obviously pyramidal in form. All tasks depend either directly or indirectly on the Job Step Task. The Job Step Task can create dependent tasks (subtasks) which in turn can create tasks dependent upon them, etc. All tasks compete for system resources (CPU, I/O, etc.), and OS/360 awards those resources according to the priority assigned to each task.

Figure 5, RTOS/360 Task Structure, represents the logical structure of tasks operating as a job step in RTOS/360. One can see that a new dimension has been added. The Job Step Task and its subtasks exist as in OS/360 while each independent task forms the basis of another set of tasks which operate independently of and

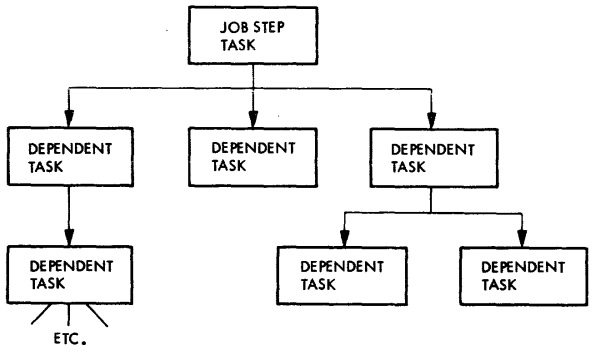

Figure 4—OS/360 task structure

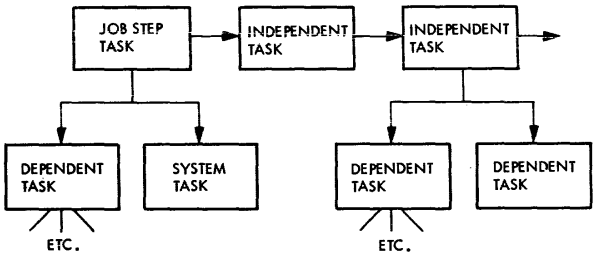

Figure 5—RTOS/360 task structure

parallel to the Job Step Task and each other. This structure is comparable to multi-jobbing in OS/360 with each independent task analogous to the Job Step Task of each active job. However, in RTOS/360, all independent tasks and their subtasks function within a single job step, and all tasks in that job step are awarded the system resources according to their dispatching priority.

System task capability

In the processing of real time data, it was found that many units of work (tasks) were unrelated to an existing task or could be performed asynchronously to existing tasks. These tasks were really tasks of the system. Therefore, a capability was developed in RTOS/360 for these systems tasks. System tasks perform services for the RTOS Supervisor or user-created tasks such as message writing and logging of real time input data. System tasks can be created and returned from within 1/25th the system overhead time required for either an OS/360 defined dependent task or RTOS defined independent task. This reduction in overhead to perform required system services in a real time environment can prove tremendously important during those CPU critical periods of high, real time, data processing.

The large difference in overhead is due to the following:

- All control blocks required for a System Task have been pre-allocated and pre-initialized for efficient utilization.
- The entry point for a System Task is an absolute location instead of a load module name, as is the case for dependent and independent tasks.

Queue management

If an independent task is processing a work request all other requests for that task must be held by the system until the task is ready to begin processing a new request. Therefore, RTOS/360 must build and maintain a queue of work requests which are waiting to be processed by an active independent task. Information concerning each request is held in a Real Time Queue Element (RTQEL). (Figure 6, Independent Task and RTQEL's, shows the logical structure of an independent task and its RTQEL's which are waiting to be processed.) Each active independent task will be processing one work request and that request is represented by the active RTQEL. All other work requests for the independent task are placed in a queue of waiting RTQEL's. This queue is ordered by dispatching priority and, in the case of equal priorities, it is first-in first-out (FIFO). When the task completes processing of the active RTQEL, the top RTQEL in the queue of waiting RTQEL's is made active and is given to the task. If there are no work requests (RTQEL's) waiting for the task, then the task is made dormant and waits in the system for the arrival of new work. All work requests for independent tasks can be optionally placed under queue management controls by directing each RTQEL into a Real Time Queue (RTQ). Each RTQ is created by a user macro instruction which defines the five attributes of the queue:

- Its unique name, which identifies the RTQ.
- Its length, which is the maximum number of RTQEL's to be held in the RTQ before an overflow condition occurs.
- The sequence in which RTQEL's are to be removed from the RTQ and given to independent tasks for processing (dispatching priority, FIFO, LIFO).
- The overflow disposition which identifies the RTQEL to be removed from the RTQ and discarded if the queue overflows (newest, oldest, lowest priority RTQEL).
- Whether the RTQ is currently able to give RTQEL's to independent tasks (enabled or disabled).

Figure 7, Real Time Queue Element Control, gives an example of the logical structure of RTQEL's controlled by an RTQ. The five attributes and other control information pertaining to the RTQ are held in the Real Time Queue Control Block. In the example, the RTQEL's would be given to the independent task in order one, two, three, four, if they were not controlled by the RTQ. That is the sequence of their relative dispatching priorities. However, the RTQ has a FIFO order attribute; therefore, the RTQEL's will be given to the task in the order three, one, two, four.

If queue management is not used, the RTQEL's for independent tasks in the waiting queues can accumulate indefinitely unless the tasks can process their work requests faster than they are generated. Queue man-

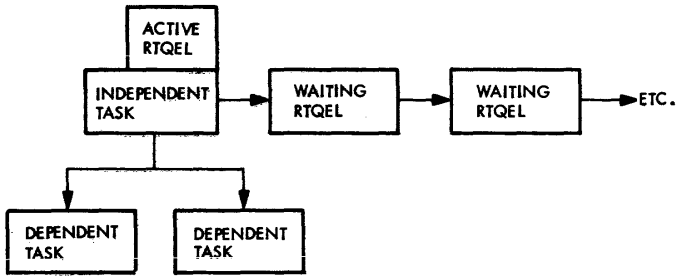

Figure 6—Independent task and RTQEL's

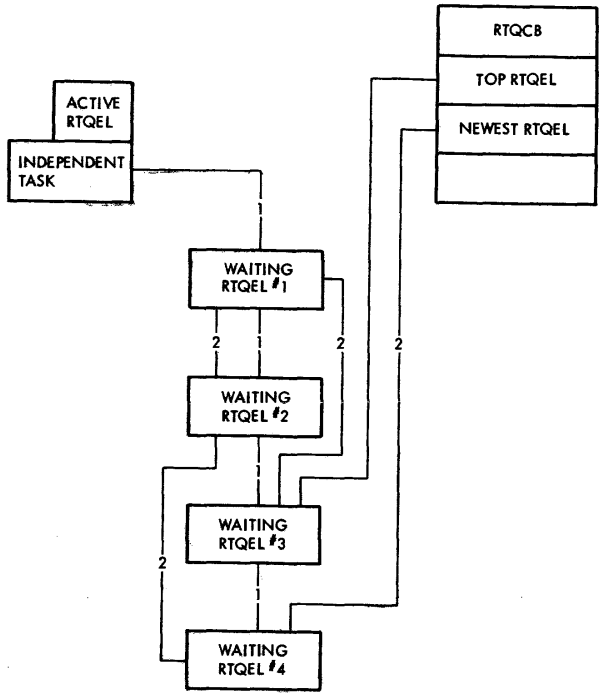

Figure 7—Real time queue element control

agement provides additional controls over the requests in the waiting queues by limiting the maximum number of RTQEL's held for independent tasks. It can also be used to indirectly control the system load by not giving work to an independent task until another independent task has completed processing.

The number and structure of RTQ's is determined entirely by the user. An RTQ can contain work requests for any number of tasks, and any number of RTQ's can contain work requests for the same independent task. The point to be made here is that queue management is very versatile in that it can be used in many ways to regulate the system's work flow.

Data and time routing concept

One of the characteristics of many real time, on-line systems is that they are driven by the arrival of data to be processed and by the passage of time; i.e., some processing is accomplished by the programming system because certain data have arrived while other processing is accomplished because certain reports and displays are required at specific times. Another characteristic found in the RTCC system is that much of the processing is very repetitive; i.e., the same kinds of data come again and again, representing different positions of the spacecraft or different data points for the various telemetered activities that are being monitored. In developing RTOS/360, and the independent task concept, it was recognized that a mechanism was required which would examine all types of input data and cause them to be sent to the appropriate independent tasks for processing. This mechanism is called *data routing*, and it acts as an interface between the hardware interrupt servicing function and the resident nucleus of RTOS/360. Data routing is a simple mechanism which requires only that the applications programs execute a macro instruction to identify the directives to RTOS which link a type of input data to an independent task. When input data is received in the system via the 2902 MLA, RTOS compares the data with the current data definitions established by the applications programs. If a match is found, the data are routed to the independent task that will process them. If no match is found, the message is discarded. Data routing can also be instructed to accumulate a number of data messages (for example, input messages) for the same independent task and generate a request for the task only after the number of messages specified by the user have been received. In this case, all the accumulated messages will be sent to the independent task as one request.

As stated above, work requests may be generated according to the passage of time also. For example, an independent task may be created to control a program which updates the position of a space vehicle every second. The only data necessary to perform this operation is the position of the vehicle at the last second and some orbit and velocity parameters. Since this operation is controlled by an independent task, the necessary data and parameters can be saved in the task's resource table while it is dormant (possibly out of main memory). Since data arrive in a random manner and not necessarily sequentially or on a time cycle, there is no method using input data which will cause requests to be generated for the task which must process a request each second. Therefore, *time routing* must be used to generate the required results. To use time routing, a problem program requests that a certain independent task is to be activated at a certain time or cyclical when a given delta time has elapsed. RTOS/360 routing and time management functions will then activate the independent task at the time requested. If the activation is to be continuous, it is left to the problem programmer to request the activation's end.

The data and time routing functions (which operate under a system task) have been constructed so that their functions can be combined. For example, it is possible to request the accumulation of data under data routing with requests generated by time routing on some specified interval. Each request generated will contain all the data accumulated during the last interval. Another way of using the combined functions is that messages can be accumulated over a timed interval and request generated either when the interval expires or when specified numbers of data messages have been received in the system, whichever event occurs first.

Time management

The System/360 Model 75 computers used to support NASA's real time applications are equipped with a special high-resolution (10μs accuracy) GMT (Greenwich Mean Time) clock and interval timer. In order to provide support for this special hardware, a time management supervisor was developed for RTOS/360 which functions in parallel with the standard OS/360 time management routines. The time management supervisor maintains the system time in a job step pseudo clock, and it controls the setting and interrupt processing from the GMT hardware to keep time and service interval timeout requests from the routing function and other areas of RTOS/360. Additional functions have been added to the time supervisor which provide optional controls over the job step pseudo clock.

Real time input/output control system (RTIOCS)

It was necessary to develop a Real Time Input/Output Control System in RTOS/360 which would service real time input/output requests rapidly and efficiently, perform special device-dependent data manipulation, and support the special real time input/output devices at the RTCC. RTIOCS is comprised of five logical parts discussed in the following paragraphs.

A *real time access method* performs device-dependent data manipulation and sends output messages to the special real time output devices at the RTCC. In addition, standard sequential System/360 output devices (2400 tapes, 1403/1443 printers) may be substituted for the special RTCC devices simply by altering the UNIT designation on cards in the user's input job stream. The real time access method is also used to control the reading of information from the IBM 2250 and 2260 graphic display units. This section of the real time access method functions closely with the graphic display attention control routine, and together, these two areas of the real time I/O control system provide RTOS/360 users the ability to read information from the IBM 2250 or 2260 devices. Writing on the display devices is controlled by the real time access method alone. In this case, the displays are processed as normal real time output requests.

The *real time interrupt servicer and start-stop input routine* provides software control over the real time input devices at the RTCC. The interrupt servicer passes input data to the data routing and logging functions in RTOS/360. The start-stop input routine accepts data whenever an active routing request is present for each particular device. An OS/360 OPEN/CLOSE is not required.

The *digital display control routine* provides centralized and simplified control of the special RTCC devices called digital television displays (D/TV). This control program is entered by user tasks signaling the change of status in one or more of the displays. The current status of the display is updated by the control routine, and it then gives control to the real time access method which updates the actual hardware display.

The digital/TV display control routine provides a software support for the Philco digital/TV display system at the RTCC. This program services all digital/TV display requests, maintains information indicating which displays are currently being viewed and the console which is viewing them, controls the dynamic allocation of the digital/TV channels, and generates work requests for the user tasks which create and update the actual numbers or figures within each display.

Data management—data tables

The large amounts of data required to be accessed by the Mission Systems at the RTCC during spaceflights prompted a careful evaluation of the OS/360 Data Management methods. First, it was found that although the methods were adequate for the environments for which they were designed, the RTCC real time environment produced a unique situation in which system overhead needed to be reduced for reading and writing data. Second, there was no efficient means to enable RTOS independent tasks to share data. Third, due to the critical importance of data in the system, a means to ensure data integrity and consistency had to be developed. Finally, an easy method had to be developed to allow users a simple method of reading and writing data, thereby eliminating the need for complicated coding techniques. The resolution to these RTOS data management problems was the development of control programs to support *data tables*.

Data tables are blocks or arrays of data maintained on direct access devices (2314 disk) in the partitioned format. (Data tables are treated as members of partitioned data sets.) Each data table is identified by its unique EBCDIC name and is defined by its block size and number of blocks. A data table generation program employs these parameters in allocating direct access space for each table, providing the controls required to access it, and storing its initial data in the direct access space provided.

The main utility of data tables is the additional facilities provided by the data table control programs. Here, the standard OS/360 Data Management OPEN/CLOSE logic has been eliminated, thereby increasing the speed at which data can be read or updated. Data can be used commonly by any number of different tasks. The data table programs provide methods of "locking" data tables which ensure data integrity and consistency by delaying any tasks which try to write into a data table until the table is "unlocked." In this way, various portions of a table can be read through different requests and the user is ensured that no update has taken place between requests.

Functional area—summary

Briefly, we placed RTOS/360 in its environment and outlined the major modifications made to OS/360 in its functional areas of Task Management, I/O Management, Time Management, and Data Management to extend it for real time spaceflight control. In addition, we have shown the addition of two new functional areas, Routing and Queue Management, which add additional controls necessary for RTOS/360 to effi-

ciently perform the strenuous requirements of real time processing. However, RTOS/360 development does not end here. Experience had taught us that many additional features and facilities would be necessary in an operating system to process and develop real time programming packages. These features are outlined in the following section.

Special features and facilities of RTOS/360

Large core storage support

The IBM 2361 four-megabyte Large Core Storage (LCS) is supported in three modes of operation by RTOS/360. The first mode is to use the LCS as a means for *improving job shop operations.* This is accomplished by: (1) using the LCS as assembler work space instead of tapes or disks, thereby improving assembler execution time; (2) using the LCS as work storage for compilers to allow larger compilations to be performed in main memory, thereby decreasing compile time and increasing job throughput; (3) placing job control information on the LCS, thereby job throughput is increased; (4) using the LCS as a system residence device for nonresident operating systems programs, thereby giving faster access to them and increasing throughput.

The second mode is to use the LCS as an *addressable extension of main memory.* This is especially applicable to large applications packages being developed on the one-half megabyte main memory System/360 Model 50's.

The third mode of operation was initiated by the fact that it was known from the initial development of the Apollo mission application package that the package would exceed the capacity of main memory and the LCS. (The Lunar Landing Mission exceeds six megabytes.) Therefore, an LCS algorithm was developed that dynamically allows the funneling of data and programs into main memory (see Figure 8, Allocation of Main Memory). Basically, this dynamic LCS allocation means that the LCS is used as a high-speed dynamically changing residence device for load modules and data tables which are heavily used but which cannot be contained in main storage for the duration of the need for them. A load module or data table will be put on the LCS when it is requested and is not presently on the LCS. As long as the load module or data table is frequently used, it will be retained on the LCS; when it appears that the load module or data table is no longer required on the LCS, it may be replaced with another load module or data table.

It is possible to identify load modules and data tables with such low response requirements that they need never be placed on the LCS, i.e., residence on a direct access device is sufficient. Conversely, some load

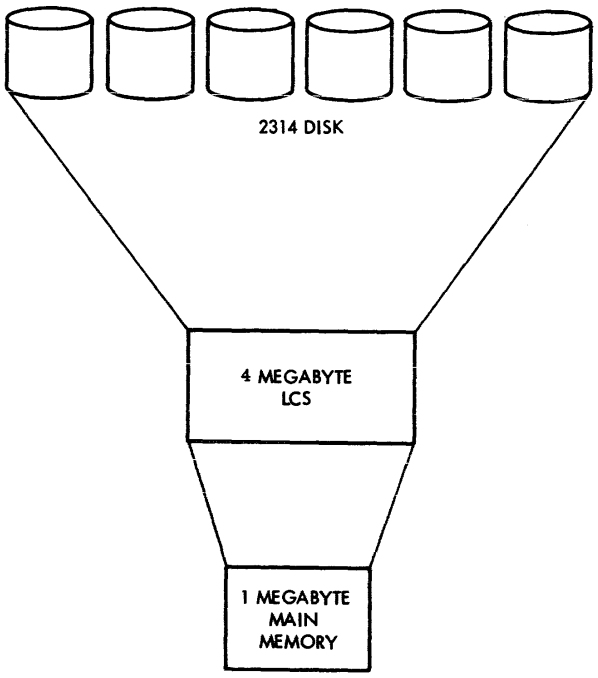

Figure 8—Allocation of main memory

modules and data tables are very critical; therefore, these may be permanently "locked" on the LCS.

To support the third mode of LCS operation, a Large Core Storage Access Method (LCSAM) was developed to provide the RTOS control program with a facility of moving blocks of storage from the LCS to main storage or from main storage to LCS. LCSAM will perform the data move either with the normal System/360 instruction set or by performing an I/O operation through the storage channel, depending on the size of the block of data.

Real time linkages

Two problems encountered in large real time systems required the development of a feature in RTOS/360 called Real Time Linkages. The first problem pertains to the fact that the system library subroutines referenced by standard OS/360 load modules (programs) must be included within each module when built by the OS/360 linkage editor. This requirement often results in a large duplication of system subroutines present in main core at one time. This duplication can be very wasteful since the amount of main core available is reduced, and the amount of time required to load a module is increased, and the amount of space required to hold the module on a direct access device is increased. Real time linkages solve this problem by allowing load

modules to reference common resident reentrant library subroutines.

A second problem pertains to the fact that certain constants (such as the diameter of the earth) used in the real time missions must be identical to all programs and be under close control by the coordinators of the total application mission system. The real time linkage mechanism solves this problem.

By holding task priorities in a common parameter table, the system can be "tuned" by simply changing those priorities found in that single table rather than performing a reassembly of a large number of programs.

Real time linkages resolve all the external references that a load module contains for system subroutines or common parameters when the module is loaded into core for execution. The system subroutines and common parameters are loaded into main core during real time initialization and held there for the duration of the run (job step). Therefore, the addresses of these routines and parameters can be inserted into the appropriate external address constant fields contained in a module as it is loaded so that the cost at execution is no greater than if they appeared in the load modules in the normal fashion.

Logging

In most real time applications, especially those which require post-run analysis, it becomes important to perform some type of recording activity which saves the data received, transmitted, and processed by the system. This feature is referred to as logging in RTOS/360. Logging automatically records all real time input and output messages on magnetic tape. Also, a macro instruction has been provided which will write problem program generated information on the log tape if an application programmer wants a record of selected data or processing results.

Simulated input control

One important factor, which is almost essential in the development of real time systems, is the ability to send simulated input data to the applications programs. In real time environments, it is impossible to employ or always obtain the necessary equipment to produce "live" data for all applications program checkout. To solve this situation, RTOS/360 contains a feature termed Simulated Input Control (SIC), which allows the user to run his development programs with simulated input data in an attempt to find most interface problems between modules and programming errors prior to final checkout with actual data.

The SIC programs which operate as part of RTOS/360 obtain the simulated input data from cards or tape, or both. All data have a time of receipt associated with each data message which allows SIC to send each one to the data routing function when the time of receipt on the message equals the current internal computer time (job step pseudo clock time). This in turn generates requests for independent tasks which will process the data as if they were a real time message. For convenience, the SIC package has been designed so that magnetic tapes produced by the logging function can be used as SIC input sources without special editing. The SIC programs will pass over all output messages on the log tape and send only the input messages to the data routing function.

Fastime

Another special RTOS/360 function that has become very valuable at RTCC is Fastime. Fastime is often used in conjunction with SIC when testing new areas of the user's system. Its only function is to step the job step pseudo clock when there is no system activity. Fastime operates as the lowest priority task in the system so that it is entered when there is no other activity. If the Fastime program running under this task determines that there is no further work to be performed before the next routing request, the time management function is signaled to step the pseudo clock to the time of the next routing request. One can see that many hours of computer time can be saved because the system will not wait for the actual passage of time to generate a time queue if the system becomes inactive, as time queues will be generated immediately when idle CPU time occurs. This function is further enhanced in SIC runs because the SIC programs use time queues in determining the exact moment a message is to be sent to data routing. Therefore, in a SIC run, time may be stepped to the time of the next data message. This message will be immediately sent to data routing and then to a task for processing. In this way, simulated data messages can be given to tasks as fast as the tasks can process them, thereby reducing the actual computer time to test new programs. By using Fastime with SIC, the checkout of an 80-minute orbit can be performed in about 10 minutes. Fastime and simulated input control have no place and are not used when the system is performing its real time production work. These functions are used only in testing new versions of the application systems.

Display formatting language

There is a large variety of display devices at the RTCC that have different internal format requirements.

There is a high probability of change in these devices, their internal formats, and the displays shown on them. This changeable character of the display devices increased the need for a series of display formatting programs. To meet this need, RTOS/360 programmers designed and developed a versatile display formatting language (DFL) which isolates the applications programmers from the unique characteristics of each display device and the internal format changes resulting from modifications to those devices.

The display formatting process consists of two steps. First, the user must define his display by assembling the DFL *format* macro instruction with his program. The format macro instruction expands into a "format statement" or character stream when assembled.

This character stream provides the display format controls used by the DFL conversion routines in the building of an actual output block for the desired display. During execution, the applications programmer can prepare output data for a display by executing the DFL conversion macro instruction. When the conversion routines complete processing and return control to the calling program, the data are in converted form and ready to be sent to the device specified by a particular control card (DD card) in the job control language for the job step. The user can subsequently output the data to the display device via the real time access method portion of the real time I/O control system. The conversion routines build output blocks for a particular device. The device is specified by identifying to the conversion routines the DD name of the DD card for the data set. From this information, the conversion routines can identify the particular device which is to receive the converted data and activate the appropriate conversion modules. This means that the DFL package provides complete device independence among those devices supported (see Figure 9, Device Independent Display Language).

Through the simple alteration of control cards in the user's job stream, the user can alter his display devices. This feature can be very valuable when the actual display devices are not readily available. The applications programmers can code and debug their display programs using common or standard devices, such as IBM 1403 printers. When the display devices become available, the programs will be ready for actual production work after changing the appropriate control cards. The devices currently supported by the DFL package are printers: IBM 1403, IBM 1443; display devices: IBM 2250, IBM 2260, Raytheon MCVG, Philco RTCC Digital/TV; plotters: RTCC X-Y Plotboards, RTCC Scribers; and Teletypes. The device independence feature and some of the devices supported by DFL are also shown in Figure 9.

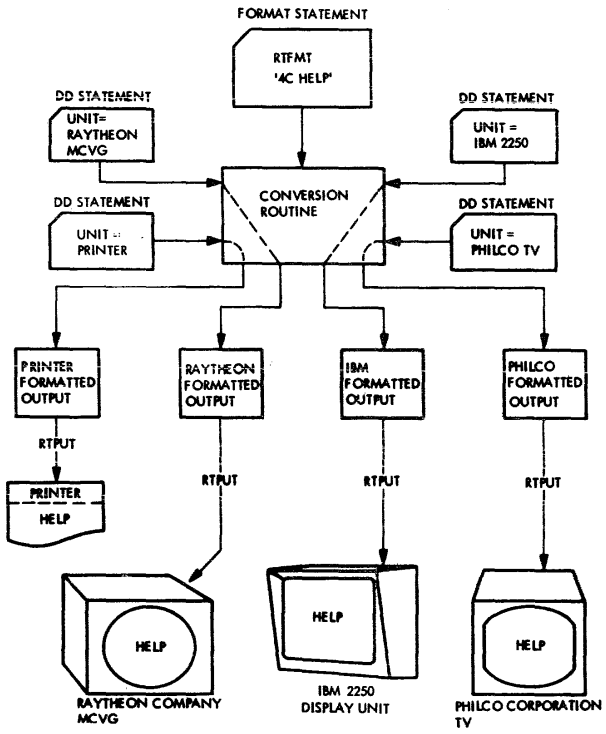

Figure 9—Device independent display language

Fail-safe programs

Because of the critical nature of real time manned spaceflights, it is extremely important that RTOS/360 be able to process abnormal conditions so that it is virtually impossible for a portion of a flight to go unmonitored because of a software, data, or hardware failure. Four areas of software support have been developed and included in RTOS/360 to meet this need: *selectover*, *high-speed restart*, *error recovery*, and *time-out*.

Selectover is performed by exchanging the operational roles of the Mission Operational Computer (MOC) and the Dynamic Standby Computer (DSC) without interruption to the input/output data on the real time interfaces. During Selectover, the integrity of the mission outputs is maintained.

The Apollo mission support system operating under RTOS/360 in a System/360 Model 75 with one megabyte main storage and four megabytes LCS, may be restarted in less than 10 seconds on an alternate Model 75 computer system which may be idle, processing job shop, or performing real time test operations. This is accomplished through IBM Channel-to-Channel Adapters (CCA) which link each combination of two out of five machines. An Initial Program Load (IPL) sequence is generated from a remote console to the proper CCA on the operational computer system, simultaneously

enabling the CCA path between the two systems. A special IPL hardware modification enables a restart even if the machine to be restarted is in manual state. All of allocated storage is then transferred over the CCA from the operational system to the selected standby system before resuming in the restarted system. A similar restart can be performed from magnetic tape by creating the tape on an operational computer and carrying it to the standby computer for IPL. (This operation takes about five minutes.)

The error recovery package of RTOS allows the system to recover from errors due to the program errors, hardware malfunctions, or abnormal conditions arising within the system itself. As recovery occurs, appropriate messages and recommendations are printed which indicate the current status of the system. A part of the error recovery activity includes device switching, i.e., if one I/O device fails, RTOS will automatically (or by external signal) select another device of that type. When the control program detects an error condition, an end-of-tape, or when a user requests a device switch, Alternate Device Support (ADS) is invoked to locate an alternate unit of the same device type and perform the necessary adjustments to allow the alternate to replace the primary device. RTOS/360 programs contain built-in logic which allows recovery from a situation where an alternate is unavailable. The computer operator is informed on the console typewriter of all device switching operations. Currently, device switching is provided for the 1052 typewriter, tapes, printers, and 2314 disks.

Certain I/O device failures are such that an interrupt to the CPU is never generated to signal the completion of the I/O operation or an error condition. A software timeout facility exists in RTOS/360 which will check once per second to determine if an I/O operation has not completed in a period of time which is normal for the particular device. When such an occurrence is detected, the I/O operation will be purged and appropriate messages will be printed. Normal use of the device will be attempted on subsequent requests.

Houston automatic spooling priority system

The tremendous development effort required to meet critical mission schedules requires that the computer systems be used to the maximum at all times during job shop operations. It was known from the onset of Apollo development that either a large number of "peripheral" off-line support computers would be required to perform such operations as loading jobs for execution and printing the vast amounts of output (usually large core and LCS dumps) or the Model 75 computers would have to be used to their maximum CPU availability. Since the former was far too expensive, a programming system was developed specifically for RTOS/360 that allowed all the peripheral functions normally associated with off-line support computers to be performed in the single Model 75 CPU. The system was called the Houston Automatic Spooling Priority (HASP) system. HASP acts as a dependent task under RTOS/360 (cohabitates in a single CPU with other RTOS/360 operations) and uses small amounts of primary CPU time to operate the peripheral functions. These functions include transferring the job stream to direct access to await execution, collecting job output on direct access, and printing and punching job output from direct access following job execution. Jobs awaiting any stage of processing (print, punch, or execution) are queued on a priority basis so that the effect of a true priority scheduler is gained not only for normal job execution but for associated peripheral functions as well.

A complete "warm start" capability also exists in HASP so that untimely interruptions of the system will cause no loss of job input or output queued for processing under HASP. Sophisticated operator communications exist that provide control over the number of input job streams, the number of output devices, and the order of job executions.

Background utilities

There are many utility functions in any data processing operation, especially a system which employs disks, that must be performed. These include: dumping direct access volumes to tape, restoring direct access volumes from tape, copying and comparing tapes, labeling tapes, changing volume serial numbers on direct access devices, etc. This is especially true when a large variety of applications systems are under development as in the RTCC. Utility operations usually require the complete dedication of the computer while they are being performed. This dedication was found to be unrealistic from both the cost and time required; therefore, *all* utility operations were designed so that they could operate in "background" under RTOS/360 control as dependent tasks. These background utilities execute asynchronously with the normal job processing and can be initiated and terminated by the computer operator at the console typewriter.

Job accounting system

With the large number of computers being used by a vast array of development groups, it was found that the RTCC required a means to report accounting and system measurement data. This was accomplished by the inclusion of a set of programs called the Job Ac-

counting System (JAS). Since all computer operations at the RTCC are under control of RTOS/360, JAS automatically generates, through punched cards, a data base for three types of reports which are valuable for both the accounting and system measurement purposes. The three report types are:

- A Job Shop Analysis Report which provides job mix and computer system performance statistics.
- A Computer Utilization Report which is used to charge computer time to user.
- A Management Report which provides information on program development costs through statistics on the use of the computer by individual programmers.

Statistics gathering and modeling activities

Each Apollo mission presents the RTCC with a unique set of processing requirements. For example, real time data sources may change in number, arrival rate, or message size. These and other such factors cause changes in the performance of real time computing systems. So that changes do not cause the systems to perform below acceptable limits, performance of current systems is measured and that of future systems is modeled.[2]

To measure the performance of a real time system and monitor its execution, a comprehensive Statistic Gathering System (SGS) was developed. SGS is a program and not a hardware device attached to the computer. It provides an accurate means of measuring performance on RTOS/360 by collecting:

- Timing information on control program services and application programs
- Percentage figures showing how definable system functions use the CPU resource
- Elapsed time figures showing task response time in a multiprogramming environment.

The SGS design for RTOS/360 is patterned after an earlier version used with the Gemini 7094 Executive Control Program.

The Real Time Computer Complex is not a project that is blessed with a firm definition of mission requirements. Results of each mission impose requirements for future missions and, thus, levy new demands for real time support. It is essential to the orderly development of RTCC real time systems to anticipate problems in computer system configuration or system program design that could impair the success of future missions. To analyze future system performance, RTCC uses models written in the language of the General Purpose Simulation System (GPSS/360).

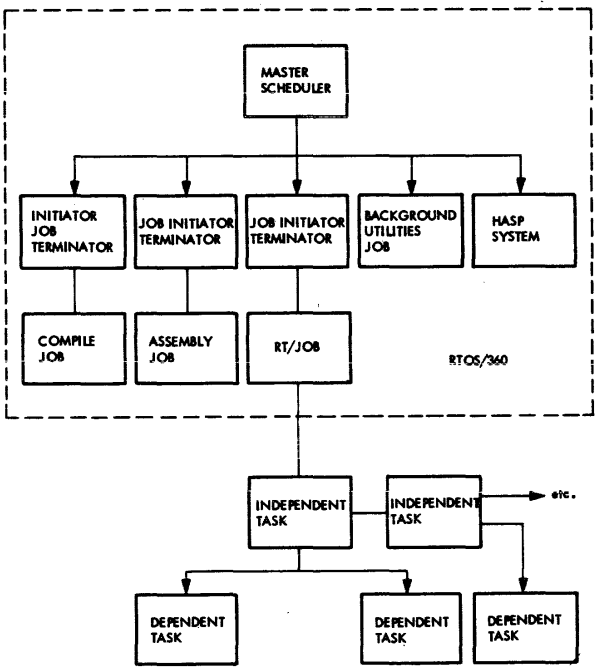

Figure 10—A multi-jobbing/multi-tasking RTOS/360

Information obtained from SGS is used in these modeling activities.

Multi-jobbing in RTOS

Within this paper, it has been shown that RTOS/360 in the real time environment is a multi-jobbing system in the broad sense; i.e., a real time job step can be processing several independent paths of logic (independent tasks which could be termed jobs since they share the system resources) at the same time the multi-tasks are performing, and the background utilities and HASP are also vying for the CPU. However, after careful investigation of statistics obtained from SGS and JAS, it was found that large amounts of time were still spent in the I/O wait state, and that the full computing power of the System/360 Model 75 was not being used. Therefore, the final feature to RTOS/360 was developed—real time multi-jobbing under RTOS/360 control. The RTOS/360 multi-jobbing differs from the OS/360 multi-jobbing in that RTOS/360 does not require partitioned memory nor, of course, a fixed number of tasks. An illustration of RTOS/360 multi-jobbing is shown in Figure 10, A Multi-jobbing/Multi-tasking RTOS/360.

CONCLUDING REMARKS

The development of real time control systems for NASA's spaceflight programs has been an evolutionary

process. For the Mercury Program, IBM developed the Mercury Monitor, which performed only real time control and occupied only a small portion of an IBM 7090 computer. Next came the development of the real time Executive Control System for the Gemini program. Executive occupied about 13,000 words of an IBM 7094-II computer. The third system in this evolutionary process was the RTOS/360, which is presented in this paper. RTOS/360, a 150,000 byte system, was the first system not only containing real time control facilities, as in the Mercury Monitor and Executive, but also containing the complete gambit of operating system functions (assemblers, compilers, job shop processing techniques, etc.).

Today, RTOS/360 has not only successfully supported several NASA Apollo Missions, but because of its real time facilities and special features, coupled with the current OS/360 System, is being used by other installations outside the RTCC to meet their special requirements for a real time operating system.

ACKNOWLEDGMENTS

I wish to acknowledge the contributions of all those members of the RTOS departments whose documents and comments aided in the preparation of this paper. Special thanks go to Ray Strecker and Ken Adams, whose technical documents on RTOS were a major source of information. For the encouragement to write the paper, I wish to thank W. D. Pollan.

REFERENCES

1 J JOHNSTONE
A real time executive system for manned spaceflight
Proc F J C C 1967

2 W STANLEY H HERTEL
Statistics gathering and simulation for the apollo real time operating system
IBM Systems Journal Vol 7 No 2 1968