MULTIPLE-TASK REAL-TIME PDP-15 OPERATING SYSTEM
FOR DATA ACQUISITION AND ANALYSIS*

W. R. Myers
Aerojet Nuclear Company
Idaho National Engineering Laboratory
Idaho Falls, Idaho 83401

ABSTRACT

The RAMOS operating system is capable of handling up to 72 simultaneous tasks in an interrupt-driven environment. The minimum viable hardware configuration includes a Digital Equipment Corporation PDP-15 computer with 16384 words of memory, an extended arithmetic element, automatic priority interrupt, a 256K-word RS09 DECdisk, two DECtape transports, and an alphanumeric keyboard/typer. The monitor executes major tasks by loading disk-resident modules to memory for execution; modules are written in a format that allows page-relocation by the monitor, and can be loaded into any available page.

All requests for monitor service by tasks, including input/output, floating point arithmetic, request for additional memory, task initiation, etc., are implemented by privileged monitor calls (CAL). All 10 device handlers are capable of queuing requests for service, allowing several tasks "simultaneous" use of all resources. All alphanumeric IO (including the PC05) is completely buffered and handled by a single multiplexing routine. The floating point arithmetic software is re-entrant to all operating modules and includes matrix arithmetic functions.

One of the system tasks can be a "batch" job, controlled by simulating an alphanumeric command terminal through cooperative functions of the disk handler and alphanumeric device software. An alphanumeric control sequence may be executed, automatically accessing disk-resident tasks in any prescribed order; a library of control sequences is maintained on bulk storage for access by the monitor.

A multitask real-time operating system (RAMOS) has been developed for a relatively small PDP-15 computer configuration. The machine was originally acquired for use in a dedicated data acquisition application and was limited in capacity relative even to the minimal RSX-15 requirements. The computer is a PDP-15/30 with 16 Kwords of core memory, 256 Kwords of disk storage (1 RS09 platter), three DECtape transports, an NP02 PHA interface, EAE and API (and memory protect and relocate which are not currently used), two teletyps and a refresh display. The system has been augmented by adding a 200 lpm line printer, a (special design) 24-channel digital data multiplexer and a control panel and function keyboard for display and analysis control.

The system is used in a Radiation Measurements Laboratory where it is interfaced to several stand-alone multichannel analyzer configurations, automatic sample changer gross counting systems, a NOVA-based PHA system, auto-dial and auto-answer dataphones for control of remote data acquisition systems, and to a PDP-8 and a PDP-9 in nearby laboratories. It is required to accept data, on demand, from the several laboratory instruments and te dataphones, and to execute analysis procedures, requested from the local and telephone terminals and the interfaced processors, in a timely manner.

Because of the original hardware limitations, the limited prospects for augmenting the system, and the need for several special-purpose unique interfaces, it was necessary to develop the software system in-house. Prior extensive experience in developing real-time data acquisition and on-line analysis systems for small computers defined the starting point of the effort; the real-time capabilities are of primary importance with multitasking data analysis a close second.

Basic Structure

A multitasking system constrained to operate in 16 Kwords of memory must have a small executive program and 10 handlers. Because these data requirements are well defined it is practical to implement dump-mode only in the bulk storage 10 to minimize handler size and complexity, and because of the many character-oriented terminals and devices, use of a common character-10 handler is indicated. Floating-point arithmetic functions are required by both the real-time and background tasks, requiring a re-entrant floating-point arithmetic software package to obviate loading multiple copies of identical routines. The

+ Work performed under the auspices of the Energy Research and Development Administration.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED.
RAMOS operating system, as currently configured, utilizes core memory in the lower two pages:

- Executive, CAL interrupt: 1400 locations
- Disk and DECtape: 240
- Display and control: 740
- Character-10 handler: 1700
- Command decode, Batch: 1200
- Real-time Routines: 1100
- Floating Arithmetic, matrix: 4400
- Alphanumeric 10 buffers: 1400
- Interrupt stack and CAL save: 400

Total: 15400

The remainder of core memory, up to 32 Kwords, is reserved for execution of disk-resident data analysis modules. The overlay modules are written in a page-locatable format, and are restricted to be less than 4096 words in length, including all required data and 10 buffers, so that they may be loaded to any available page for execution. Complex analyses may require sequential execution of several program modules, each one of which performs a specific operation on the data and leaves results in memory or disk common areas for use by the next module in the sequence.

**Control Language**

The data reduction overlay modules are driven by commands from alphanumeric terminals, including telephones and interprocessor data links, or from the batch control text. Each module implements mnemonic commands that are both appropriate to the functions performed and intelligible to the (non-programmer) user. Batch control programs may be written in the control language and catalogued in a bulk-storage library for automatic or demand execution. Laboratory personnel may write data analysis procedures in a (high-level) language that they understand, make modifications and variations as required without programmer assistance, and perform the data analyses simply by telling the system to "execute name".

To illustrate the mnemonic command programming technique, the following procedure might be used to perform a non-linear least-squares gaussian* analysis of a Ge(Li) 4096-channel gamma-ray spectrum:

```
SEARCH(LP)
SPC 3 SEN 2 SRC PRT Z
GAUSS(LP)
SPC 3 BKG 6 FIT PRT Z
GASS(LP)
SPC 3 LIN 58 EFF 75 DO PRT Z
GEFOR
LIBRARY XCT REPORT
END
```

Spectrum #3 is searched for photopeaks at sensitivity level #2, gaussian functions are fitted to the peaks using background type #6, then linearity and detector efficiency tables are merged with the results to produce energy and intensity values and a modified focal interpreter is entered to generate a formatted report.

**Real-Time Operation**

In the capacity of a laboratory data collection center, the PDP-15 must accept and record data on demand from several independent systems which are normally interfaced through serial data links. Data are transmitted to the computer in ascii or bcd character format and are buffered as line images by an interrupt-level routine. On completion of a line, the interrupt handler (typically at API level 3) requests execution of a task at one of the API software levels (usually level 5 or 6) which converts the character string to binary data and augments the scratch data file, on bulk storage, that belongs to the transmitting device. When the entire spectrum is transmitted, the recording task may request execution of a non-resident procedure to move the data to a library and/or analyze the data and print a summary report.

The laboratory data lines are always enabled, allowing the data transmission to be initiated at the experiment station. The system must be capable of responding to several simultaneous data transfers at speeds of 10 to 200 characters per second. High-speed 18-bit parallel interfaces to local 4096-channel analyzers are similar but permit processor intervention to control record lengths and inter-record delays. Interfaces to other computer systems operate at 5000 characters per second, but, by convention, all communication is in line format and appropriate delays are generated by the transmitting system.

**Task Priorities**

Tasks may be initiated at several priority levels, but, once started, they are all executed at the same level (API mainstream level 8). Because the system is primarily designed for multi-task operation, it is assumed that there is always a suspended task waiting to regain control. Allocation of CPU time is performed, in the simplest manner, by returning control to the task that has the most recent interrupt. A task is always suspended during an IO operation, permitting another task to use the system; tasks requiring the most IO do most of the waiting but are also the ones that are most often restarted.

**The 'CAL' Handler**

All requests for monitor service by tasks and programs, including input/output, floating point arithmetic, requests for memory, scheduling of tasks, etc., are implemented using the 'CAL' instruction. The floating point arithmetic interpreter is accessed by the indirect 'CAL' to by-pass the normal set-up delays and provide faster execution. For all other functions, the 'CAL' interpreter acquires a parameter block to save the index and limit registers, the parameter address, the return address, and the function code. After examining the device code in the middle six bits of the instruction, the interpreter switches the API level to that required to protect multiple entry due to an interrupt, then dispatches to device-specific (handler) coding, or to a monitor function.

**Input/Output Queuing**

If the 'CAL' function specifies IO, the device handler proceeds to start the operation or to queue the request depending on the status of its busy flag. If the flag word is zero, the device is not busy and the operation is started. If the flag is
mut zero, it is the address of the parameter save block belonging to the currently active request. If queuing is required, the address of the new parameter block is inserted in the linkage word of the active block (or the last block in a queue) and the task is left in a waiting state.

When an active I0 task is completed, the interrupt portion of the handler examines the linkage pointer to determine if another request has been made; if there is a request pending, it is made current and the device is re-activated before returning control to the (ready) task.

**Bulk Storage I0**

Because all of the operations in the system are fairly well defined, it is appropriate to restrict bulk-storage I0 to dump-mode transfers. This restriction permits the handlers to be small, fast and very simple. Where directory-oriented I0 is necessary, as in the 'FOCAL' program library, the batch control text library, and certain data libraries, the appropriate application program must perform the required directory operations.

**Common "Byte-I0" Handler**

Because our application utilizes several devices such as teletypewriters, terminals, serial data lines, and paper tape reader and punch, which are functionally very similar, a common character-handling I0 routine is used. The various devices are interfaced to the PDP-15 in several ways: the console teletype and the paper punch through the program interrupt, the LT-15 and the paper tape reader through API, and special data interfaces, telephones, and the line printer through a (special design) digital data multiplexer. Some devices may therefore require a few words of unique code to be compatible with the major routine, and a hardware modification is needed to change the paper tape reader API level to the same as that used by the other devices.

All devices have separate input and output line buffers and individual control tables. The tables contain the buffer origins and limiting addresses, the input and output instructions which are normally IOTs, busy flags, and indicators for echo, code-conversion, carriage-return delay, multiplexer channel indicators where appropriate, and pointers to the 'CAL' control block acquired for the function.

There are two specific advantages to this handler structure: several devices are supported by a routine that is only a little larger, except for table and buffer areas, than that required for a single device, and implementation of additional devices requires very little programming effort. To interface a new device into the routine, one need only allocate I0 buffer space, set up a control table, and add a two-word prologue to the 'CAL' routine and to the interrupt routine.

Functions that are supported include octal, decimal, character and ascii (sixbit) input and output, code control (for reading BCD paper tapes), and waiting for a line or a character on input. A cooperative function between the common character handler and the disk handler allows use of a portion of the RS09 disk as an input terminal to implement a batch operating mode in the system.

**Memory Management and Program Relocation**

The monitor includes rudimentary memory management operations accessed by a 'CAL' instruction. The memory handler functions allow programs to request and release 4096-word memory pages and to modify (relocate) 15-bit address constants to correspond to the appropriate program page.

The memory request/release functions are implemented by using a one-word variable that has a bit set for each used page; the request function clears the lowest available page, clears the appropriate bit in the flag word, and returns the page address to the requester. When a page is to be released, the program must indicate the page number as an operation parameter.

The request/release operations are used primarily by the command interpreter when a disk-resident program must be loaded into core memory for execution. They may also be used by resident data acquisition tasks to reserve data space and by utility modules to establish special purpose I0 buffers.

The relocation function is used when a disk-resident program is read to a memory page for execution. All executable overlay modules are required to be less than 4096 words, to be assembled to run in page (index) mode, and to have their origins on page boundaries; each must include as part of the coding, a contiguous list of the addresses of all address constants in the module. The relocation function operates through the address list to modify the indicated address constants to have values within the page assigned to the module.

This page-relocation technique provides an excellent compromise between the extremes of using a relocating loader for each execution of a non-resident task and the necessity to assign specific memory partitions to dump-mode program modules.

**Reentrant Floating Arithmetic Software**

In order to conserve space in the program overlay areas and still provide a complete floating-mode arithmetic capability, a reentrant arithmetic software package is included in the resident monitor. The package implements 35 instructions including five immediate operand instructions, seven common functions, three-way compare and branch, inverse divide, and three I0 instructions. Four operand formats are available: 18-bit (positive) data, 18-bit (signed) integers, and two- and three-word floating point formats. The package is interpretive and floating mode coding follows the PDP-15 page-mode addressing conventions. Operand mode may be changed in the course of a computation by one of the "operate-class" instructions allowing mixed mode arithmetic.

Input and output are handled through a "floating accumulator" utilizing user-written character handling routines. The input interpreter is free-format, allowing input of integer, floating, or fixed-point numbers. The output controller transmits formatted ASCII character strings to the user output routine. The output format may be integer, fixed-point or floating-point. The six common functions and four operations on symmetric matrices are also included in the reentrant operations.
Reenterability is accomplished by maintaining a 32-word 'scratch' area in each user program. The scratch area contains all necessary permanent data, such as the floating accumulator, operand data mode indicator, loop and format control words, and temporary scratch locations needed by the interpreter. The floating accumulator components, operand mode indicator and format words are available for normal-mode initialization or manipulation.

**SUMMARY**

The RAMOS software system has been in use for almost two years, in support of a Radiation Measurements Laboratory, on a 16K PDP-15/30 with an RS09 DEC Disk. It has met all expectations in providing a versatile real-time and multi-task capability in this well-defined environment.

Incorporation of frequently used software in the resident monitor has permitted economies in memory utilization and in programming efforts. There have been very few situations in which more than 2048 words (of coding) are needed for the most complex analysis modules.

RAMOS could easily be adapted to operate in a more restricted hardware configuration, particularly in situations which do not require extensive computation or large data buffers. It could be advantageously utilized as a real-time operating system in systems as small as 8 or 12K with one or two DECTapes; EAE and API are, however, required.

With a recent augmentation of the core memory to 32K and acquisition of a 5 Kword disk, capabilities are being added for line printer spooling, simultaneous execution of batch-mode jobs, and task scheduling on a time-of-day basis. The size of the monitor will remain under 12K and the restriction of overlay modules to 4K will be maintained. The system control language will include more sophisticated expression evaluation, conditional execution and iteration control.

**REFERENCES**