MULTITASKING IN A USER PARTITION
WITH A CONTOUR MODEL OF PROCESSES

by

LEE ALLEN

B. S., Georgia Institute of Technology, 1972

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1973

Approved by:

V. E. Wallentine
V.E. Wallentine

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Purpose.

With the introduction in recent years of new concepts such as task communication, virtual access to all resources, and the necessity for system support of more advanced features in higher-level languages such as retention of valid storage upon block exit and indirect reference, a need for more sophisticated machine architecture and operating system design has become increasingly more apparent. The compiler writer as well as the operating system designer must bear the burden of implementing desired features using the limited facilities of the computer systems which are available.

One particular difficulty has been the implementation of retention in block-structured environments. Recently, the contour model of block structured processes has been given some attention as a solution to this problem. The purpose of this research has been to demonstrate the feasibility of a contour model in support of an operating system by implementing a *multi-tasking monitor* which resides in the user partition. The secondary purpose has been to indicate the possible design of a contour model machine by implementing some hardware functions as supervisor calls to the monitor.

Configuration.

The configuration chosen for implementation was the IBM 360 OS/MFT because of its availability and wide-spread acceptance. The monitor interfaces directly with the OS system; however, since this interface only includes using system-supplied routines such as

GETMAIN (to allocate storage), and STIMER (to set time intervals), the interface is completely transparent to the user.

<u>Report Structure.</u>

The rest of this paper is divided into four major sections. The first section describes the concept of a process and how this concept is related to tasks running under the monitor. Some of the features which apply to processes as a whole are described such as the system queues, processor blocks, and deadlock detection.

The second section is devoted to describing contours and how they were used in the implementation of multi-tasking. The philosophy of reentrant and recursive code is explained with reference to how contour modeling facilitates implementation of this philosophy. Retention in block-structured processes is explained with an example illustrating the advantage of contour modeling over stack modeling for retention.

The third section describes the language used to take advantage of the features of the monitor. Each of the macros which were added to regular IBM assembler language is described along with the particular function or purpose which the macro performs. An example is given written in ALGOL and the extended assembler language to illustrate some of the features of the monitor.

The last section presents some of the conclusions which the author derived from the research. Some practical as well as theoretical applications of the monitor are given such as its use as a real operating system and the possible construction of a machine.

II. PROCESS

Process Definition.

Johnston defines a process as a "time-invariant algorithm and a time-varying record of execution of that algorithm."[2] This philosophy was followed in the implementation of the monitor. Each user in the system is considered as a process. However, each user can also create other processes or tasks with which he can interact. Each process is independent in that it competes with all other processes for computer time; however, each process is dependent on its parent for global variables. Each process communicates directly with the monitor and may communicate with other processes in the system through the use of system variables.

When a process is created, the current environment in which it is created is marked so that even though the parent process may terminate, the environment is not destroyed and the descendant process can continue to execute.

Processor.

Each process is assigned a processor block consisting of 108 bytes of storage, see figure 1. It is this block with which the monitor keeps track of the process as to its current status and environment.

The first part of the processor block is the display or current environment of the process. For each level that the process has entered a block, there is an entry in the display to indicate the bounds of storage that the process can reference. In addition, the process can reference any of the environment in

Displacement	Field	
0	Display Length	
4	Display lower & upper bounds for contours	
84	Link	
88	Time	
92	Current Contour Address	
96	Next Instruction Address	
100	Highest Priority	Current Priority
104	Times Passed Over	

Figure 1--Processor Block

which it was created by using a link which resides in its addressable area. Therefore, complete memory protection is available which would normally be a hardware function on a hypothetical contour model machine.

Since the processor is frequently placed on queues such as the ready queue or wait queues while it is waiting for resources, space is left in the processor for a link to the next processor on a queue. In this way the only space which need be allocated for a queue is the head of the queue which points to the first processor block on the queue.

Process Suspension.

There are two conditions under which execution of a process may be suspended. First of all, a proceess is allowed a certain amount of time to perform any execution using the CPU. If during this time, the process does not request any resources, execution will be suspended, and the processor block will be put on the ready queue according to its priority. If there are no other processes currently ready for execution the process will be restored to execution.

The ready queue can be considered as a queue for a resource too. In this case the resource is the CPU. In a multiprocessing machine, whenever a processor is free it would be assigned a processor block if any are ready to execute.

Further movement of a processor block among queues is upon processor request for a resource. This resource may be an I/O channel in which case the prosess is suspended (moved from ready queue to

I/O queue) while the I/O operation takes place. In order to maintain concurrency of processing, a process could create a subtask (process) the purpose of which is to do the I/O. This would allow the parent process to continue while the descendant process is suspended waiting on completion of the I/O request. At some point in execution, the parent task will need the information and will need to check and see if the I/O operation is complete. This can be done by setting a shared variable to a certain value or by communicating through the system variables.

Communication using the system variables represents the second type of resource request. When a process wishes to check for a certain condition, it may check the system variables. If the variable has been set by another process (or possibly the same process) execution continues. If the variable has not been set, the process is placed on a wait queue. At any point in execution when a process sets that particular variable the waiting process is removed from the wait queue and placed on the ready queue according to its priority. In the case where more than one process requests a resource represented by a system variable, the processes are placed on the wait queue in the order of their respective priorities as explained in the "Priority" section of this paper.

There are four fields in the processor block which facilitate placing the process on a queue. The link field has already been mentioned and is used to indicate the next processor block on the queue.

When a process is suspended because it is waiting on an I/O request, all time left in its particular time slice is deleted. When is resumes execution, it is given a full time slice. When a process is suspended for any other reason, the amount of time left in its time slice is stored in the processor block, and when the process resumes execution it is started with however much time is remaining. This scheme was used for a variety of reasons. When a process is suspended because it requested a resource which another process already controls, the reasoning was that if the process was able to gain exclusive control of the resource, it should not be given another full time slice to execute in. For technical reasons, if the process is unable to gain control of the resource it is likewise only allowed the time which it had left in the last execution to use the CPU. When a process creates another process it is suspended in order to give the created process a chance to transfer the parameters. In this way the original process may create additional processes and use the same area for parameter passing. Because the original process is only suspended so that the parameters can be passed, it is only allowed the remaining time to use the CPU when it resumes execution.

For most of the calls to the monitor for such features as block entry and exit, pointer change, and resource release, it was felt that the CPU time should not be subtracted from the requesting process for time used by the monitor to respond to the requests. Also, some portions of the monitor's routines need to be noninterruptable. In these cases also, the process's remaining time is stored in the processor block and the remaining portion of the time slice is restored at the end of the monitor routine.

Whenever a process is suspended for any reason and whenever a process makes a supervisor call, the address of the next instruction to be executed along with a pointer to the current environment are stored in the processor block.

Deadlock Detection.

Since the system variables are controlled by the monitor rather than the processes, deadlock is user preventable by having the processes use only one of the variables to indicate that it desires exclusive control of some resource. However, it is still possible that one process may control a resource that another process needs to continue while that process already has control of a resource that the first process needs to continue by using two different system variables. Since execution takes place under control of the monitor, the monitor can never be blocked. When there are processes waiting on resources other than I/O requests and there are no processes on the ready queue, the monitor cancels the job after printing out an appropriate message and giving a core dump of all storage currently in use. This could easily be changed so that an arbitrary choice is made and one process is allowed to execute with all of the resources of the system in the hope that it would free up the resources when it was done.

Priority.

Each process is assigned a certain priority level which determines where it is placed on certain queues such as the ready queue and resource queues. Each process can change its own priority level but can never alter its priority level to be higher than a maximum fixed at creation of the process. The first process in the system is assigned a priority level equal to the second highest priority level in the system and has a maximum of the highest level since it is the father of all other processes.

Process creation is a tree structure. The first process can create other processes and each of these processes can create other processes. When a process creates another process it assigns a priority level equal to or lower than the parent process's priority level. This priority level then becomes the maximum priority level that the descendant can attain. Space is allocated in the processor block to store the current priority level and maximum priority level.

When a process is suspended for any reason it is placed on a queue. The priority level is used to determine where the process is placed on the queue. The processor block is located under any processor blocks of higher priorities and before any blocks of lower priority. However, if the block is placed before any lower priority blocks a counter is incremented in the lower blocks. A processor block can not be placed before any block whose counter has reached a value of 5, an arbitrary number. This counter is set to 0 whenever a block is removed from a queue.

III. CONTOUR MODELING

Contour.

Inherent to the contour model is the contour itself. It has already been shown that a system can run efficiently under the restriction that all instructions of an algorithm be reenterable thus requiring that all data be stored separately from the actual algorithm.[3] This is in complete agreement with the definition of process as presented in the section on processes in this paper. The contour provides the record of execution for the process.

The contour as implemented consists of 80 bytes of control information (see figure 2) and a variable amount of local storage. A contour is local to the process in whose processor block display the address of the contour appears. A contour may be referenced by the current process and any descendant processes which are created while the contour exists.

The contour model is especially applicable to block structured languages; however, other languages are easily implemented as one large block such as FORTRAN. At block entry, a contour is allocated for all of the space which the block will require in the way of variable storage. If the block is reentered before the contour is deallocated, a fresh contour is created. This scheme also provides an easy mechanism for recursion, since a new allocation of variable storage is made each time the procedure (block) is called (entered).

Retention.

Contours are allocated and deallocated as a whole. To keep

Displacement	
0	Reference Counter
4	Previous Contour
8	Next Contour
12	Register 14
16	Register 15
20	Register 0
24	Register 1
28	Register 2
32	Register 3
36	Register 4
40	Register 5
44	Register 6
48	Register 7
52	Register 8
56	Register 9
60	Register 10
64	Register 11
68	Register 12
72	Contour Name
76	Contour Length
80	Variable Storage

Figure 2--Contour

track of any variables in other contours which might reference storage in the current contour, a reference counter is placed in the contour. Each time a variable in another contour is changed to point to the current contour, the reference counter is incremented in the current contour and the reference counter in the contour which the variable previously pointed at is decremented. Pointer variables are discussed more fully in the "Pointers" section in this paper.

At block entry, when the contour is created, the reference count is set to 1. Whenever a block is exited, the reference count is decremented by one. There is one other case where the reference counter is incremented. When a process is created, so that its environment will not disappear while it is executing, the reference counters of all contours which are in the current environment are incremented. When the created process terminates all these reference counts are decremented.

If a contour's reference count is decremented and reaches a zero value, the contour is immediately deallocated. The monitor does all the necessary keeping track of reference counters and the reference count should be of no concern to the user.

In order to facilitate reference between contours two links are provided in each contour to point to the previous contour and to the next contour. The link to the previous contour can be considered as the dynamic link of the contour to its calling environment.

Contour vs. Stack.

An alternate model which can be used for block-structured processes is the stack model where all variable storage is kept on a stack rather than in separate contours. Generally, array storage is kept separate from the stack with some kind of descriptor in the stack. This type of model can provide almost all of the features found in the monitor with the exception of retention as illustrated in the following example.

```
BEGIN
RECORD STUDENT SF(NAME,ADDRESS,NEXT);
STRING FIELD NAME (0) [0:26],
     ADDRESS (3) [2:37];
STUDENT FIELD NEXT (8) [0:18];
STUDENT SP;
    .
    .
    .
      BEGIN
         .
         .
         .
           BEGIN
              .
              .
              .
           NEXT := STUDENT
           END
      END;
    .
    .
    .
END;
```

STUDENT is a record class identifier which is implemented in some versions of ALGOL. The record referenced by STUDENT consists of a field for the student's name, a field for his address, and a field which points to the next allocation of STUDENT. SP points to the first allocation of STUDENT. The statement in the innermost block has the effect of allocating a new student record and placing its address in the field of the first student record named NEXT.

In the stack model, the new allocation of STUDENT would be placed on the top of the stack. This allocation needs to be kept as long as NEXT points to it. However, the innermost block and second block are exited right after the allocation. Since the allocation is on the top of the stack the variable storage for the second and third blocks must also be kept. This makes the stack full of worthless information which could normally be deleted.

In the contour model as implemented in the monitor, only the storage for STUDENT is kept since a new block must be entered to allocate storage. This means that the storage reserved for blocks two and three could be deallocated.

Another advantage of the contour model over the stack model is the variable length of storage for contours. With the stack model, a fixed amount of contiguous storage must be set aside for use of the stack. With the contour model, only the exact amount of storage needed is allocated.

The stack model is slightly more efficient than the contour model however, because to increase the size of variable storage only a stack pointer needs to be incremented, while in the contour model a portion of main memory must be allocated. This could be improved with fast hardware memory allocation.

Register Saving and Restoration.

Whenever a process is suspended for any reason, the registers must be saved so that they can be restored when execution resumes. Space is provided in the contour to save these registers. The decision was made to put the register storage in the contour rather than the processor block since the monitor was designed to handle recursion and reentrant blocks as part of its normal load. Whenever a process enters a block a new contour is created and the current values of the registers are saved in the previous contour. When a block is exited, the registers are not automatically restored, but the user has the option of loading the previous values of the registers. Therefore recursive procedures (blocks) not only have new data areas for the variables, but also have new registers effectively. Of course, when a block is entered the registers contain the values they had when the entry was initiated.

Register Restrictions.

Since the monitor was implemented on an IBM computer, IBM conventions as to registers were used. The user is free to use registers 0, 1, 14, and 15; however, since all monitor calls destroy some or all of these registers, the user is warned that they may not contain the values that he expects. This is in keeping with IBM's philosophy of SVC's in which the same thing happens.

Another IBM convention is the "Save Area" which is used to save and restore registers. Normally register 13 contains the address of the "Save Area." Since the monitor performs the functions of saving the registers it reserves register 13 for this purpose. The programmer is not allowed to change register 13 although he may reference it at any time. Register 13 contains the address of the current contour so that any variable storage in the current contour may be directly referenced using register 13 as a base register. Also the user can restore registers when he exits a block by issuing "LM 14,12,12(13)."

There is one other restriction on the use of the IBM general registers. Because the monitor was designed to operate within the user partition, it must be relocatable. Since it is relocatable, the resident address of the monitor can only be determined at execution time. For this reason, the user must not change register 12 as it contains the base address of the monitor.

Contour Name and Length.

Each contour is given a name corresponding to the particular block for which it was created. Contour names are not unique in that a block may be reentered in which case two different processes contain contours with the same name. Also, if a process recurses it will contain more than one contour with the same name. Block names are unique, however. Space is reserved in the contour for its name.

The length of the contour is also stored in the contour so that the space allocated to a contour can be freed when the reference counter reaches zero. The base is already known by chaining through the links, but the length must also be known to deallocate the storage.

IV. IMPLEMENTATION

The Language.

In order to take advantage of the features implemented in the monitor a language had to be developed (but is not presently implemented fully) which would use all of the capabilities present in the monitor. Because it was felt that the most efficient level of programming, as to code generated, is at the assembly language level, the monitor was written in IBM assembly language, and the language used to interface with the monitor is an extension of this assembly language. Macros are used to generate instructions which require the monitor intervention.

Overhead.

There is a greater advantage in using IBM assembly language extensions to interface with the monitor in that the user only adds the amount of overhead that he needs. The monitor is set up so that it will accept and run a regular assembly language program in which case there would be no overhead added besides the regular OS overhead except for the necessary time-slicing involved in the multi-tasking. Of course, this would be the extreme case. If the user desires block structure only, the only overhead added would be that to handle block entry and exit. If the user wishes to include pointers in his program then he adds the overhead necessary for garbage collection.

Block Structure.

There are two macros implemented to allow block structure in the assembly language extension. These are the ENTER and EXIT macros. The necessary housekeeping that takes place when a block is entered is that a contour is created for the block, the name and length are placed in the current contour, the registers are stored in the previous contour, and the bounds of the contour are placed in the processor block for the process.

When a block is exited, the contour is deallocated if the reference counter is zero and a check is made to see if this block is the outermost block for the process. If this block is the last block, the processor block is deallocated and the process is terminated.

Regular block structure conventions are followed which pertain to local and global variables. If a variable is declared within a block it is local to that block and global to all other blocks nested within the current block. The particular variable is not referenceable in blocks which are outer blocks to the current block. The same principles apply to procedure names and labels.

As has already been mentioned, register 13 contains the base address of the current contour. For this reason all variables local to the current block are directly referenceable with register 13 as the base register. All global variables must be searched for since the base is not readily available. For this reason, a great deal of time may be saved if a certain global variable is referenced many times in a block by allocating local storage for the variable and moving the global value into the local storage before working with it, referencing the local variable when needed and then storing the final value back in the global storage before block exit. This will result in a savings any time a global variable is referenced more than two times within a block.

Process Creation.

Process creation is accomplished through the use of procedures. A special designation is used in the language extension for procedure declaration, PROC. This has the same effect as an ENTER macro except that a list of parameters may follow immediately after the PROC. Parameters are designated by the special pseudo-op DP. As many parameters as necessary may be declared immediately following the PROC declaration. Parameter definition is terminated by any other symbol appearing in the source statements. Procedure definition is terminated by the EXIT macro.

Process creation is done by calling a procedure with the special operand "TASK" in the call statement. The CALL statement as implemented has a variable number of operands. The first operand is the name of the procedure. The second and third operands are optional and indicate the priority of the procedure if the third operand is coded as "TASK." The fourth operand indicates the area which is reserved for holding the addresses of any parameters which are passed. The rest of the operands are any parameters which the calling process wishes to pass to the called process. If the TASK operand is left out, the assumption is that the procedure is to be called as part of the current process. In this case, the procedure must have a RETURN macro to indicate that return is to be made to the calling block.

The procedure name operand may also be a block name. In this case no parameters may be passed. This allows a process to start a subtask (process) at any block within the program.

Parameter Passage.

Parameters are passed by giving the called procedure the addresses of the actual parameters. These addresses are stored in the locations reserved for the formal parameters. The actual mechanism for referencing parameters can be envisioned by replacing every occurrence of the formal parameter name in the procedure by the actual parameter name. Because of the linkage involved, all references to a parameter require two instructions. Therefore, if a parameter is used more than twice it is more efficient for the user to store the value that the parameter points to in a local variable and restore the value before the procedure is exited.

Variable Declaration.

Because variable storage is allocated in a contour separate from the actual instructions a few pseudo-ops needed to be added to the assembly language for declaration of variables. These additions include DCL and DCLEND pseudo-ops. Every block may have one and only one DCL...DCLEND pair. All variables declared between the DCL and DCLEND become local to the block and global to any contained blocks. All of the pseudo-ops available in IBM assembly language for declaring storage assignments are also available for use between the DCL and DCLEND with the addition of another pseudo-op for declaring pointers.

Pointers.

A special pseudo-op is included for declaring pointers. DP means to reserve a full word of storage for this variable as it is to be used as a pointer. Pointer arrays are also allowed by using a number in the operand portion of the declaration. "P DP 10" for example, means to reserve 10 full words of storage to be used as pointers. If no number appears in the operand portion, one word of storage is reserved.

Pointers may be used in any instruction where a full word is allowed. Whenever a pointer changes in value the appropriate contour's reference count is increased if the pointer now points to it or decreased if the pointer previously pointed to it.

Pointers are especially useful for indirect reference. Any level of indirect reference may be indicated by appending an appropriate number of "#"'s to a pointer variable name when it appears in the operand portion of a statement. There is no check made as to type of variable referenced or as to storage boundaries. These are left to the programmer and an appropriate message will be printed out and the offending process aborted if something goes wrong.

The user is warned that no check is made in the present implementation for circular reference in pointers. If a pointer points at a contour which has a pointer which points to the original contour, neither of the two contours will ever be deallocated. In order to avoid such references, the user should clear all pointer variables to zero before he exits the block in which they are local.

Input and Output.

The user is allowed to do any type of input and output normally allowed in IBM assembly language. The DCB for the data set referenced must be present in the user program. The user has the option of doing his own I/O or letting the monitor do it for him.

The user may code any READ or WRITE macros in the middle of his program, in which case there is no transfer to the monitor. The user may also use the IO macro which is one of the extensions to the assembly language. The IO macro is made up of the exact number of operands required for the normal operation plus one more operand to indicate whether the normal operation is a PUT (P), GET (G), READ (R), or WRITE (W). The particular code letter is the first operand of the macro and the rest of the operands are the same as would normally be required for that operation.

The monitor handles input and output operations by creating a subtask under control of the OS operating system for the exclusive purpose of doing the I/O operation. When the I/O operation completes the monitor is interrupted and the process which issued the I/O request is removed from the wait queue and put on the ready queue. As mentioned before, if the user wishes to maintain processing while his I/O request is waiting on completion, he should create a separate process which he calls when an I/O operation is needed.

System Variables.

Dijkstra has described elementary operations which facilitate process cooperation and synchronization.[4] These are the primitive P and V operations which affect integer-valued variables called semaphores. The system variables of the monitor are implemented as semaphores. When the first process is initiated, the system variables are set to 1. The user can increment or decrement the system variables to any value he chooses using the two macros implemented to use the system variables, as long as they stay non-negative.

A GRAB macro may be used to indicate that a process has entered a portion of his program which uses some resource which he wishes to have exclusive control over. The effect on the system variable is to decrement the variable. If the variable reaches a negative value, the implication is that the resource is already in use and the process which makes the request is put on a wait queue. There are ten system variables, an arbitrary maximum number, which may be used. The user could also define his own system variables in which case all reference to these global variables would be surrounded by GRAB and LETGO macros. The operand portion of the GRAB macro indicates which variable, by number, the process wishes to reference.

When a process wishes to release control of a resource or leaves that portion of the program which needs exclusive control of some resource, it may issue a LETGO macro. This has the effect of incrementing the system variable and removing one process from the wait queue, if there are any on the wait queue. The process which issues the LETGO macro is allowed to continue processing and the process which was on the wait queue is placed on the ready queue in its appropriate priority position.

Priority Setting.

There is one other macro implemented to allow a process to set its own priority. As mentioned before, a priority level can not be set higher than a set maximum; however, a process may set its priority level anywhere underneath that maximum. The SETPR macro is used to branch to the monitor to set the priority. If the priority level desired, coded in the operand portion as a number, is higher than the maximum, the priority level is set to the maximum.

An Example.

To illustrate some of the ideas discussed, an example is presented at this point. In figure 3 a sample ALGOL program is given. This program is a relatively simple program for finding the factorial of two numbers, 5 and 7. Two features have been added to the language. In lines 13, 15, 18 and 19, Dijkstra's P and V operations are presented as system procedures which act on semaphore 1. Also, the special keyword "TASK" is included in lines 16 and 17 to indicate that the procedures are called as tasks to run concurrently as opposed to line 9 where the factorial procedure is called as a regular procedure.

In figure 4 a possible compilation of the ALGOL program is

```
BEGIN
INTEGER F1, F2;
PROCEDURE FACT(N,R,FLAG);
INTEGER N, R, FLAG;
BEGIN
LABEL AROUND, AROUND2;
INTEGER X;
IF N = 0 THEN GO TO AROUND;
FACT(N-1,X,0);
R := N x X;
GO TO AROUND2;
AROUND: R := 1;
AROUND2: IF FLAG = 1 THEN V(1)
END;
P(1);
FACT(5, F1, 1),TASK;
FACT(7,F1,1),TASK;
P(1);
P(1)
END;
```

Figure 3--A Sample ALGOL Program

```
         ENTER
FACT     PROC
N        DP     1
R        DP     1
FLAG     DP     1
         L      3,N
         L      4,=F'0'
         SR     4,3
         BE     AROUND
         S      3,=F'1'
         ST     3,TEMP
         CALL   FACT,,,PARLIST,TEMP,X,=F'0'
         L      5,N
         LA     4,0
         M      4,X
         ST     5,R
         B      AROUND2
AROUND   L      3,=F'1'
         ST     3,R
AROUND2  L      3,FLAG
         L      4,=F'1'
         CR     3,4
         BNE    AROUND3
         LETGO  1
         DCL
TEMP     DS     F
X        DS     F
PARLIST  DS     3F
         DCLEND
AROUND3  RETURN
         GRAB   1
         L      3,=F'5'
         ST     3,TEMP
         CALL   FACT,1,TASK,PARLIST,TEMP,F1,=F'1'
         L      3,=F'7'
         ST     3,TEMP+4
         CALL   FACT,1,TASK,PARLIST,TEMP+4,F2,=F'1'
         GRAB   1
         GRAB   1
         DCL
F1       DS     F
F2       DS     F
TEMP     DS     2F
PARLIST  DS     3F
         DCLEND
```

Figure 4--A Typical Compilation of the ALGOL Program

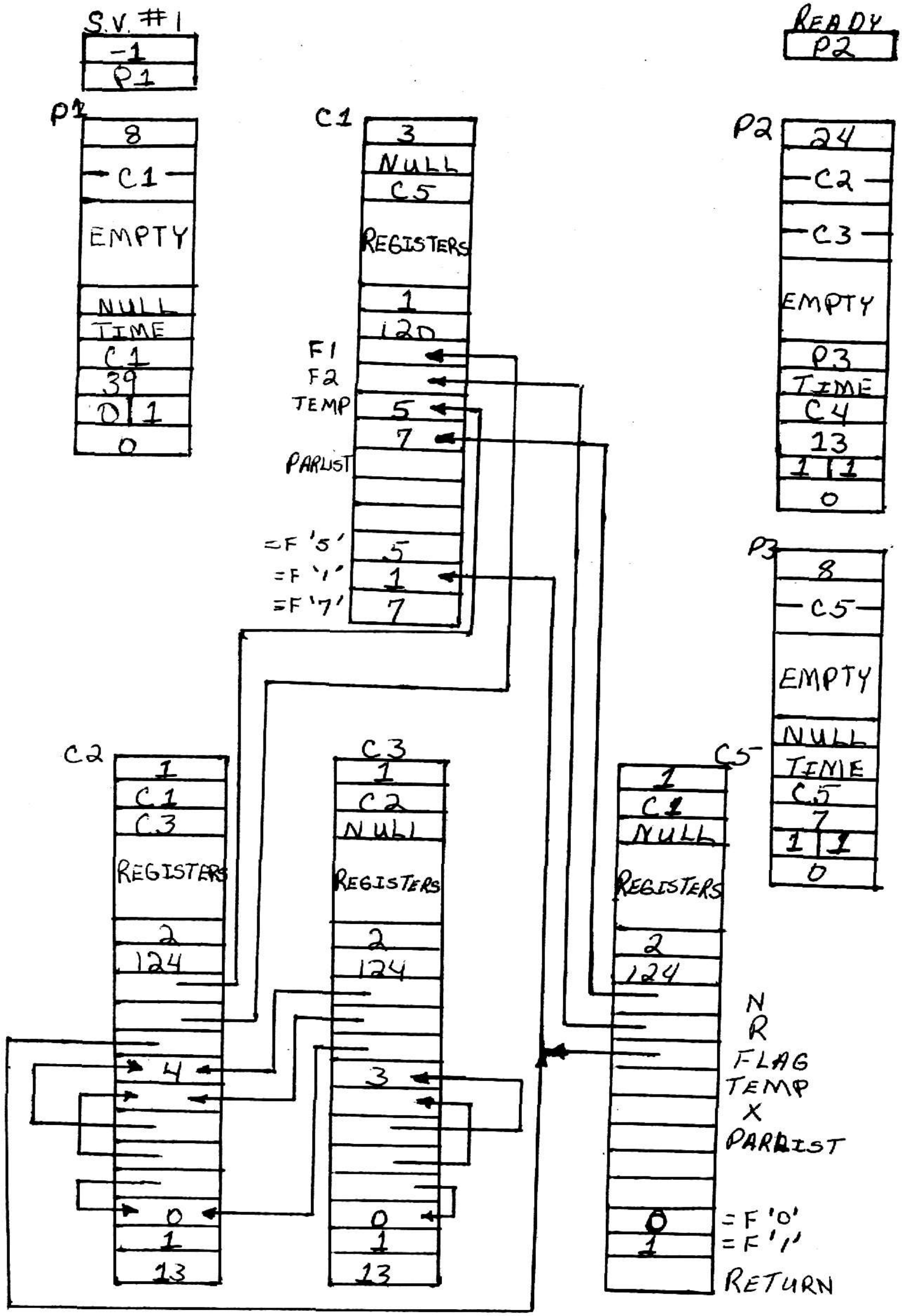

Figure 5--A Snapshot during Execution of the Program

given as it would be in the extended assembly language. Lines 15, 18, and 19 in the ALGOL program are used to wait for completion of the factorial procedure. These two lines are compiled into GRAB macros in figure 4. Also, line 13 in the ALGOL program is used to indicate completion of the procedure. This is compiled into a LETGO operation in line 24 of figure 4.

In figure 5, a snapshot is given of the system at a particular instant in time during execution of the algorithm. The main block has reached line 18 in the ALGOL program and thus has created two new processes. The original process (P1) is on a wait queue concerned with system variable #1. (Pointers which refer to contours or processor blocks are designated with C_ or P_ such as C2 or P3 to indicate which contour or processor block they point to.) P1 has only one contour active, but this contour's reference count is 3 since two other processes have been created. The next instruction location of the processor block for P1 has instruction 39 since it has just executed instruction 38 causing it to be placed on the wait queue.

The two created processes are currently on the ready queue. P2 is the process created to find the factorial of 5. P3 is the process created to find the factorial of 7. P3 has not had time to do much processing. When execution resumes, P3 will continue at instruction 7. P3 has not yet recursed.

Process P2 has already recursed 1 level and is about to recurse for the second time. At the point that the diagram illustrates, P2 has just executed instruction 12 which is the call to itself, but has not recuresed so that a new contour has not been created. P2 has two contours currently valid. C2 represents the first time that the factorial procedure was called, and C3 represents the first recursion caused by P2 calling FACT again.

It should be pointed out that the call of the factorial procedure in statement 9 of the ALGOL program and in instruction 12 of the compilation do not contain the TASK option. Because of this the procedure called must contain a RETURN statement. The return address is stored in the current contour. When a process reaches its outermost block, the process is terminated whether a RETURN statement is executed or not. Therefore, FACT may terminate with RETURN even though it is called as a TASK also. The actual listing of the example as it was run under the monitor with the answers printed out can be found in appendix B.

V. CONCLUSIONS

Evaluation.

As was mentioned in the introduction, the main purpose of this research was to demonstrate the feasibility of a contour model operating system. Because the monitor actually runs and executes processes in a multi-tasking environment using the contour model, the author feels that this feasibility has been demonstrated. A second issue which might be brought up however, is whether this contour model is practical and efficient. There was neither the time nor the facilities available to make the kind of studies to give that question a valid answer. The author will be the first to admit that the monitor as implemented is not in its most efficient form because of the time which would be required to optimize the code.

The system, in its present form, constitutes merely an exposure to the problems of multitasking as a typical user. It is skeletal in nature and not intended for production use. It does, however, provide a basis upon which a time sharing supervisor can be built, and its development effort was significantly less than systems which provide the same service. The time spent on the system was approximately 3 man-months as compared with over two years of many men working full-time for most multi-tasking operating systems.

A valid evaluation can be made on the basis of features included in the monitor which do not appear in other operating systems. Almost all of the features of the monitor appear also in the MCP operating system on the Burroughs machines with two exceptions. The stack mechanism which Burroughs uses does not immediately lend itself to retention unless, as was pointed out, it is desired to save worthless blocks of information on the stack. The Burroughs philosophy also is to destroy all subtasks when the parent task terminates since a process is needed in the system for each stack which is resident. One feature which the Burroughs MCP offers on the B-7700 which the monitor does not provide is virtual machine capabilities. This is because of the restriction of running under HASP on the 360.

Since the monitor operates under OS/360, all of the facilities of the OS supervisor are also available when running with the monitor. The monitor's biggest feature is retention. A second feature which is not already available on the 360 is block structure. With block structure already present in the operating system, a great burden is lifted from the compiler writer for such languages as PL/I and ALGOL.

It was felt that virtual memory should be a hardware function. If such hardware were available only a small routine would need be provided in the operating system to handle any interrupts caused by a page fault. This procedure could very easily be added if the monitor was to run on a machine with hardware virtual memory. The overhead involved to incorporate virtual memory without any hardware would be extremely prohibitive.

One feature which is available in the RC-4000 operating system that is not available directly in the monitor is the capability of sending an entire message to another process in the system. With the monitor it is only possible to communicate with another process through the system variables. If the process is a descendant or parent however, a message can be sent through global variables. The lines of communication must be agreed upon previous to the sending of the first message.

The Univac Exec VIII operating system which runs on the 1100 series has one feature, besides virtual memory, which is not incorporated in the operating system implemented by the monitor. The Exec VIII system has a complex protection and security arrangement with a variety of passwords and numbers needed to enter the system and get at any resources. The monitor uses the IBM protection scheme for file reference.

Use as a Teaching Aid.

The monitor and associated extended assembly language would be particularly useful in a first compiler design course. The features which are already implemented in the monitor such as block structure and reentrant code would release the instructor from some of the mechanics of implementing a compiler and allow him to concentrate on the more fundamental concepts of syntax parsing, lexical analysis, and symbol table construction. Of course, in a higher course the mechanics of implementation must be covered because of the limited design of present operating systems.

Use as a Real Operating System.

If the monitor were supplied with all of the routines which are referenced by the SVC's in IBM assembly language it could probably limp along as the actual operating system. This was not the intention in writing the monitor however. The monitor was written primarily to illustrate how a contour model operating system might be implemented and especially to demonstrate the feasibility and desirability of such an operating system. Secondly, the monitor was designed to run in the user partition providing him with all of the capabilities not available normally running under the OS operating system. On this matter, the monitor is a complete success.

Time-Sharing.

The monitor would be especially useful in a time-sharing environment where a certain partition is dedicated exclusively to the time-sharing users. The monitor never loses control while it is in execution. Under the priority schedule currently used at the Kansas State University Computing Center, the monitor would receive 300 milliseconds of CPU time at least every three seconds. Therefore the response time which a user at a terminal would experience would be usually about 3 seconds.

Systems Implementation Language.

A particularly good use of the monitor would be to provide the nucleus for an operating system designed for the 360 using the extended assembly language provided to take advantage of the capabilities of the monitor. The writing of an actual operating system takes many people years working full-time. This manpower and time were not available for the project; however, a good start was made toward a contour model operating system. The principles have been demonstrated and could very easily be put to use in the design of an actual operating system.

Contour Model Machine.

The secondary purpose of the research was to demonstrate the possible design of a contour model machine. The author feels that he has successfully demonstrated the facilities which would be needed on such a machine. It is hoped that sometime in the near future such a machine will be built with the idea of satisfying language requirements rather than tailoring the compiler to fit the machine.

One of the most important hardware features for a contour model machine would be automatic memory allocation and deallocation. This would require the hardware to poll the reference counters at some intervals to decide if they have reached a value of zero or not. Another important feature would be hardware virtual memory but that would be no new concept. One other important hardware concept which is necessary would be one machine instruction for the GRAB and LETGO operations. Some other hardware features would include automatic pointer reference checking every time a pointer variable is changed. This could be done with control bits on each word to indicate its type as Burroughs currently does in its machines.

FOOTNOTES

1. B-7700 System Characteristics Manual, Burroughs Corporation, Detroit, Michigan, January 1973.

2. Johnston, John B.,"The Contour Model of Block Structured Processes," Proc. Symposium on Data Structures in Programming Languages, J. T. Tou and P. Wegner (Eds.), SIGPLAN Notices, 6, 2 (Feb., 1971), pp. 55-82.

3. Organick, Elliot I., Computer System Organization: B5700, B6700 Series, Academic Press, New York, 1973.

4. Dijkstra, E. W., Cooperating Sequential Processes, Report EWD 123, Mathematical Department, Technological University, Eindhoven, The Netherlands, September 1965. (Reprinted in Programming Languages (F. Genuys Ed.) Academic Press, London, 1968.)

BIBLIOGRAPHY

Struble, George, Assembler Language Programming: The IBM System/360, Addison-Wesley Publishing Co., Reading, Mass., 1971.

Horning, J. J. and Randell, B., "Process Structuring," ACM Computing Surveys, 5, 1 (March, 1973), pp. 5-30.

Bell, C. G. and Newell, A., Computer Structures: Readings and Examples, McGraw-Hill, New York, 1971.

Dijkstra, E. W., Cooperating Sequential Processes, Report EWD 123, Mathematical Department, Technological University, Eindhoven, The Netherlands, September 1965. (Reprinted in Programming Languages (F. Genuys Ed.) Academic Press, London, 1968.)

Organick, Elliot I., Computer System Organization: B5700, B6700 Series, Academic Press, New York, 1973.

B-7700 System Characteristics Manual, Burroughs Corporation, Detroit, Michigan, January 1973.

Burroughs B-5500 Information Processing Systems Extended ALGOL Reference Manual, Burroughs Corporation, Detroit, Michigan, April, 1969.

GTL Programmers Reference Manual for the Burroughs B-5500, Rich Electronic Computer Center, Georgia Institute of Technology, Atlanta, December, 1971.

Programmers Reference Manual for the Univac 1108 Exec 8 Executive System, Rich Electronic Computer Center, Georgia Institute of Technology, Atlanta, (revised) 1972.

Johnston, John B., "The Contour Model of Block Structured Processes," Proc. Symposium on Data Structures in Programming Languages, J. T. Tou and P. Wegner (Eds.), SIGPLAN Notices, 6, 2 (Feb., 1971), pp. 55-82.

IBM System/360 Operating System System Control Blocks, Form GC28-6628-7, IBM Corporation, Data Processing Division, White Plains, N. Y., 1971.

IBM System/360 Operating System Programmer's Guide to Debugging, Form GC28-6670-4, IBM Corporation, Data Processing Division, White Plains, N. Y., 1971.

IBM System/360 Operating System Supervisor Services, Form GC28-6646-5, IBM Corporation, Data Processing Division, White Plains, N. Y., 1971.

IBM System/360 Operating System Data Management Services, Form GC26-3746-0, IBM Corporation, Data Processing Division, White Plains, N. Y., 1971.

IBM System/360 Operating System Supervisor and Data Management Macro Instructions, Form GC28-6647-5, IBM Corporation, Data Processing Division, White Plains, N. Y., 1971.

APPENDIX A

Monitor Listing

ILLEGIBLE

THE FOLLOWING DOCUMENT (S) IS ILLEGIBLE DUE TO THE PRINTING ON THE ORIGINAL BEING CUT OFF

ILLEGIBLE

```
  2            MACRO
  3            EXIT
  4            L      0,0(13)           LOAD REFERENCE COUNT
  5            S      0,ONE             SUBTRACT ONE
  6            ST     0,0(13)           STORE BACK REFERENCE COUNT
  7            BAL    14,SVC20          BRANCH TO MONITOR
  8            MEND

 10            MACRO
 11  &ADDR     ENTER  &LENGTH,&NAME
 12  &ADDR     STM    14,12,12(13)
 13            LA     0,&LENGTH         LOAD LENGTH OF VARIABLE STORAGE
 14            BALR   1,0
 15            BAL    1,4(1)            INDICATE GETMAIN
 16            SVC    10                ISSUE GETMAIN SVC
 17            ST     13,4(1)           STORE PREVIOUS SAVE AREA ADDRESS
 18            ST     1,8(13)           STORE CURRENT SAVE AREA ADDRESS IN PREV.
 19            LR     13,1              REG. 13 GETS NEW SAVE AREA ADDRESS
 20            LA     1,1
 21            ST     1,0(13)           STORE 1 IN REFERENCE COUNT
 22            LA     1,&NAME           LOAD BLOCK NUMBER
 23            ST     1,72(13)          STORE IN SAVE AREA
 24            LA     1,&LENGTH         LOAD LENGTH OF VARIABLE STORAGE
 25            ST     1,76(13)          STORE LENGTH IN CONTOUR
 26            BAL    14,SVC4           BRANCH TO MONITOR
 27            MEND

 29            MACRO
 30            GRAB   &NUMBER
 31            LA     1,&NUMBER         LOAD LOCK NUMBER
 32            BAL    14,SVC28          BRANCH TO MONITOR
 33            MEND

 35            MACRO
 36            LETGO  &NUMBER
 37            LA     1,&NUMBER         LOAD LOCK NUMBER
 38            BAL    14,SVC24          BRANCH TO MONITOR
 39            MEND

 41            MACRO
 42            SETPR  &NUMBER
 43            LA     0,&NUMBER         RO GETS PRIORITY DESIRED
 44            BAL    14,SVC32          BRANCH TO MONITOR
 45            MEND
```

```
  47            MACRO
  48            IO     &RORW,&DCBA,&AREA
  49            LA     0,32            LOAD LENGTH OF DECB
  50            BALR   1,0
  51            BAL    1,4(1)          INDICATE GETMAIN
  52            SVC    10              ISSUE GETMAIN SVC
  53            AIF    ('&RORW' EQ 'R').READL IF READ GO TO .READL
  54            MVI    5(1),X'20'      INDICATE TYPE OF 'WRITE'
  55            AGO    .DONE           GO TO .DONE
  56 .READL     MVI    5(1),X'80'      INDICATE TYPE OF 'READ'
  57 .DONE      BALR   15,0            GET CURRENT ADDRESS
  58            LA     14,&DCBA.(15)   LOAD DCB ADDRESS
  59            ST     14,8(1)         STORE IN DECB
  60            LA     0,&AREA         LOAD I/O BUFFER ADDRESS
  61            ST     0,12(1)         STORE IN DECB
  62            MVI    4(1),X'00'      INDICATE TYPE
  63            LH     0,82(14)        LOAD LENGTH OF BUFFER
  64            STH    0,6(1)          STORE LENGTH IN DECB
  65            L      15,48(14)       LOAD READ/WRITE ROUTINE ADDRESS
  66            BAL    14,SVC36        BRANCH TO MONITOR TO HANDLE I/O
  67            MEND
```

```
         MACRO
         CALLP &PORM,&ADDR,&PRIOR,&TASK,&DEST,&P1,&P2,&P3
         BALR  1,0             GET CURRENT ADDRESS
         LA    0,&ADDR         LOAD DISPLACEMENT
         AIF   ('&PORM' EQ '-').SUBTR
         AR    0,1             ADD DISPLACEMENT TO BASE
         AGO   .SKIP
.SUBTR   SR    1,0             SUBTRACT DISPLACEMENT
         LR    0,1             PUT ADDRESS IN RO
.SKIP    AIF   ('&PRIOR' EQ '').SKIP2
         LA    14,&PRIOR       LOAD PRIORITY
         SLL   14,24           SHIFT PRIORITY INTO FIRST BYTE
         OR    0,14            PUT PRIORITY INTO ADDRESS REGISTER
.SKIP2   AIF   ('&P1' EQ '').DONE
         LA    14,0            GET A ZERO IN R14
         LA    1,&P1           LOAD PARAMETER ADDRESS
         ST    1,&DEST.(14,13) STORE PARAMETER ADDRESS IN PARAMETER L
         AIF   ('&P2' EQ '').DONE
         A     14,FOUR         INCREMENT TO NEXT LIST ELEMNNT
         LA    1,&P2           LOAD ADDRESS OF SECOND PARAMETER
         ST    1,&DEST.(14,13) STORE IN PARAMETER ADDRESS LIST
         AIF   ('&P3' EQ '').DONE
         A     14,FOUR
         LA    1,&P3           LOAD ADDRESS OFTHIRD PARAMETER
         ST    1,&DEST.(14,13) STORE IN PARAMETER ADDRESS LIST
.DONE    AIF   ('&TASK' EQ '').SPROC
         LA    1,&DEST.(13)
         BAL   14,SVC8         BRANCH TO MONITOR
         AGO   .DONE2
.SPROC   BALR  1,0             GET CURRENT ADDRESS
         LA    14,24(1)        LOAD RETURN ADDRESS
         L     15,76(13)       LOAD LENGTH OF CONTOUR
         S     15,FOUR         SUBTRACT TO GET LAST WORD
         ST    14,0(15,13)     STORE RETURN ADDRESS IN CONTOUR
         LA    14,&DEST.(13) LOAD PARAMETER ADDRESS LIST ADDRESS
         LR    15,0            LOAD PROCEDURE ADDRESS
         BR    15              BRANCH TO PROCEDURE
.DONE2   ANOP
         MEND


         MACRO
         RETURNP
         EXIT                  REGULAR EXIT FROM PROCEDURE
         L     1,76(13)        LOAD LENGTH OF CONTOUR
         S     1,FOUR          DECREMENT TO LAST WORD
         L     15,0(1,13)      LOAD RETURN ADDRESS
         BR    15              RETURN TO CALLING PLACE
         MEND
```

```
ADDR2  STMT   SOURCE STATEMENT                                         F01MAY7

        118            MACRO
        119            MOPEN &DCBA,&IORO
        120            LA    0,4          LOAD LENGTH OF DATA BLOCK
        121            BALR  1,0          GET CURRENT ADDRESS
        122            BAL   1,4(1)       INDICATE GETMAIN
        123            SVC   10           ISSUE GETMAIN SVC
        124            AIF   ('&IORO' EQ 'INPUT').IN
        125            LA    15,143       LOAD CODE BYTE
        126            AGO   .REST
        127 .IN        LA    15,128       LOAD CODE WORD
        128 .REST      SLL   15,24        SHIFT INTO FIRST BYTE
        129            BALR  14,0         GET CURRENT ADDRESS
        130            LA    14,&DCBA.(14) LOAD ADDRESS OF DCB
        131            OR    15,14        STORE CODE WORD AND DCB ADDRESS
        132            ST    15,0(1)      STORE COMPLETE DATA IN DATA WORD
        133            BAL   14,SVC40     BRANCH TO MONITOR
        134            MEND
```

```
ADDR2  STMT    SOURCE STATEMENT

        136              START 0
        137              PRINT NOGEN
        138 GOMONI       BALR  12,0
        139              USING *,12          12 IS BASE REGISTER
0000C   140              STM   14,12,12(13)
00144   141              B     AROUND        BRANCH AROUND CONSTANTS

        143 * THE FOLLOWING IS A CONSTANT AREA FOR THE ATTACH MACRO

        145 ATCHLIST DC    A(DUMMY)          ADDRESS OF TASK ENTRY POINT NAME
        146          DC    F'0'
        147 ATCHECB  DS    F                 ADDRESS OF EVENT CONTROL BLOCK
        148          DC    2F'0'
        149          DC    AL1(2)
        150          DC    AL3(IOCOMPL)      ADDRESS OF TASK COMPLETION ROUTINE
        151          DC    F'0'
        152 ATCHNAME DS    CL8
        153          DC    4F'0'
```

```
         155 READY    DS     F               READY LIST
         156 DUMMY    DC     CL8'LINK'       DUMMY ROUTINE TO INITIATE SUBTASKS
         157 IOCMASK  DC     X'80000000'     MASK TO CHECK FOR LAST EVENT POINTER
         158 IOMASK   DC     X'40000000'     MASK TO CHECK FOR EVENT COMPLETION
         159 EVCOUNT  DC     F'0'            NUMBER OF EVENTS AWAITING COMPLETION
         160 EVENTS   DC     10F'0'          POINTERS TO THE ECB OF EACH EVENT
         161 TIMESET  DC     F'5770'
         162 FOUR     DC     F'4'            ****************
         163 ONE      DC     F'1'            *CONSTANTS USED*
         164 EIGHT    DC     F'8'            *IN ARITHMETIC *
         165 N16      DC     F'16'           ****************
         166 LOCKS    DC     10F'1'          LIST OF COMMON VARIABLES
         167 QUEUES   DC     10F'0'          WAITING LISTS
         168 TINSAVE  DS     CL4             STORAGE TO SAVE INSTRUCTION AFTER TIMER
00186    169 BINST    BAL    14,TIMEXR       BRANCH TO TIMER EXIT RESTORE (AFTER INT.
         170 ERREX    DS     F               DUMMY ERROR ROUTINE FOR LINE PRINTER
00104    171 INTEREX  STM    14,13,HOLD      STORE REGISTERS TO LOOK AT IN DUMP
         172                 ABEND 18,DUMP   THIS WILL BE INTERRUPT EXIT ROUTINE
         180 HOLD     DS     16F             STORAGE FOR REGISTERS
```

```
ADDR2  STMT   SOURCE STATEMENT                                                    F01MAY72

        182 AROUND   GETMAIN R,LV=108      GET AREA FOR FIRST PROCESSOR
00040   186          ST    1,READY         STORE ADDRESS OF AREA IN READY LIST
00000   187          LA    0,0
00000   188          ST    0,0(1)
00000   189          MVC   1(107,1),0(1)  CLEAR PROCESSOR AREA TO ZERO
00084   190          MVC   0(4,1),FOUR     STORE COUNT IN AREA
00088   191          MVC   100(4,1),ONE    SET PRIORITY TO ONE
        192          SPIE  INTEREX,((1,15)) SET INTERRUPT EXIT ADDRESS
0085E   201          LA    14,PBEGIN       LOAD FIRST PROGRAM ADDRESS
001C4   202          B     PSTART          BRANCH TO ADDRESS IN TIMER EXIT TO BEGIN
```

```
0000C    204 TIMEX      STM    14,12,12(13)  INTERVAL TIMER INTERRUPT ROUTINE
00010    205            L      2,16          ADDRESS OF COMMUNICATIONS VECTOR(CVT)
00000    206            L      2,0(2)        ADDRESS OF TCB WORDS
00000    207            L      2,0(2)        ADDRESS OF TCB
00000    208            L      2,0(2)        ADDRESS OF IRB
0001C    209            L      2,28(2)       ADDRESS OF PREVIOUS RB
00010    210            LM     3,4,16(2)     LOAD OLD PSW
00000    211            LA     4,0(4)        CLEAR FLAG BYTE OF PSW
00000    212            MVC    TINSAVE,0(4)  SAVE NEXT INSTRUCTION TO BE EXECUTED
000E8    213            MVC    0(4,4),BINST  STORE BRANCH TO MONITOR IN NEXT INSTRUCT
0000C    214            LM     14,12,12(13)
         215            BR     14            RETURN TO SUPERVISOR


0000C    217 TIMEXR     STM    14,12,12(13)  TIMER EXIT RESTORE ROUTINE
00084    218            S      14,FOUR       GO BACK TO ORIGINAL INTERRUPT POINT
000F4    219            MVC    0(4,14),TINSAVE RESTORE ORIGINAL INSTRUCTION
00040    220 PSTART     LA     5,READY       LOAD ADDRESS OF CURRENT PROC. AREA
00000    221            L      7,0(5)        LOAD CURRENT PROCESSOR ADDRESS
00080    222            MVC    88(4,7),TIMESET STORE NEW TIME INTERVAL IN AREA
00054    223            MVC    0(4,5),84(7)  STORE NEXT PROCESSOR AT HEAD OF LIST
0005C    224            STM    13,14,92(7)   SAVE NEXT INSTRUCTION AND CURRENT CONTOU
00790    225            BAL    14,SEARCH     BRANCH TO PLACE ON QUEUE
00040    226            L      5,READY       GET ADDRESS OF NEXT
0005C    227            LM     13,14,92(5)   RESTORE NEXT INSTRUCTION AND CONTOUR
         228            STIMER TASK,TIMEX,TUINTVL=88(5) SET NEW INTERVAL FOR NEXT TA
00010    232            LM     15,12,16(13)  RESTORE REGISTERS
         233            BR     14            BRANCH TO NEW TASK
```

```
00040    235 SVC4     L      15,READY        STORE AREA BOUNDS ROUTINE
00000    236          A      15,0(15)        GET ADDRESS OF NEXT DISPLAY POSITION
00000    237          ST     13,0(15)        STORE LOWER AREA BOUND IN DISPLAY
         238          AR     1,13            ADD LENGTH
00004    239          ST     1,4(15)         STORE UPPER BOUND
00008    240          LA     1,8
00040    241          L      15,READY
00000    242          A      1,0(15)         ADD PREVIOUS LENGTH OF DISPLAY
00000    243          ST     1,0(15)         STORE NEW LENGTH
         244          BR     14              RETURN TO PROGRAM
```

```
0000C    246 SVC8     STM    14,12,12(13)  COROUTINE CALLING ROUTINE
         247          TTIMER CANCEL        NO TIMER INTERRUPT ALLOWED
00040    250          L      2,READY       LOAD CURRENT PROCESSOR AREA ADDRESS
00000    251          LA     5,0           GET A ZERO IN R5
         252          LR     7,13          LOAD CURRENT CONTOUR ADDRESS
00001    253 CCLOOP1  LA     6,1           GET A 1 IN R6
00000    254          A      6,0(7)        INCREMENT REFERENCECOUNTER
00000    255          ST     6,0(7)        STORE REFERENCE COUNTER BACK
00004    256          L      7,4(7)        LOAD PREVIOUS CONTOUR ADDRESS
00004    257          C      5,4(7)        CHECK TO SEE IF ANY MORE IN CURRENT ENVI
00230    258          BNE    CCLOOP1       MORE, SO GO TO LOOP
00008    259          L      7,8(7)        LOAD LAST CONTOUR
00058    260          ST     0,88(2)       STORE REMAINING TIME
0005C    261          STM    13,14,92(2)   SAVE NEXT INSTRUCTION AND CURRENT CONTOU
         262          GETMAIN R,LV=108     CREATE NEW PROCESSOR AREA
00004    266          LA     0,4
00000    267          ST     0,0(1)        STORE BEGINNING LENGTH IN AREA
         268          SR     0,0
00004    269          ST     0,4(1)
00004    270          MVC    5(103,1),4(1)  CLEAR AREA
00054    271          ST     2,84(1)       LINK PREVIOUS TO NEXT
00040    272          ST     1,READY       PUT NEXT ON READY LIST AT TOP
00014    273          L      14,20(13)     LOAD R1 WHICH CONTIANS PRIORITY
00018    274          SRL    14,24         SHIFT TO GET PRIORITY IN RIGHT BYTE
00064    275          LH     7,100(2)      LOAD MAX PRIORITY FOR CALLING PROCESS
         276          CR     14,7          CHECK TO SEE IF NEW IS GREATER THAN MAS
0028E    277          BNL    CCOK          OK, SO GO TO SET PRIORITY
         278          LR     14,7          CHANGE NEW TO BE MAX
00064    279 CCOK     STH    14,100(1)     STORE MAX PRIORITY
00066    280          STH    14,102(1)     STORE CURRENT PRIORITY
         281          STIMER TASK,TIMEX,TUINTVL=TIMESET
0000C    285          LM     14,12,12(13)  RESTORE REGISTERS
00000    286          LA     14,0(1)
         287          LR     15,0          LOAD ADDRESS OF NEW TASK
         288          BR     15            BRANCH TO NEW TASK
```

```
         290 SVC12    LR    15,13       LOAD CURRENT SAVE AREA ADDRESS
         291          SR    0,0         GET A ZERO IN R0
00048    292 BALOOP   C     1,72(15)    COMPARE BLOCK # WANTED TO SAVE AREA #
         293          BCR   8,14        IF EQUAL THEN RETURN TO PROGRAM
00004    294          C     0,4(15)     COMPARE TO SEE IF OUT OF CONTOURS
002C6    295          BE    BANOSUCH    YES, SO GO TO ERROR
00004    296          L     15,4(15)    LOAD SAVE AREA ADDRESS OF OUTER BLOCK
002B0    297          B     BALOOP      BRANCH TO COMPARE AGAIN
         298 BANOSUCH ABEND 19,DUMP
```

```
ADDR2  STMT    SOURCE STATEMENT                                                      F01MAY7

0000C   307 SVC16     STM    14,12,12(13)   POINTER CHANGE ROUTINE
00040   308           L      5,READY        LOAD CURRENT PROCESSOR AREA
00000   309           L      3,0(5)         LOAD LENGTH OF DISPLAY
        310           SR     8,8
        311           CR     0,8            CHECK PREVIOUS POINTER REFERENCE WITH ZE
00322   312           BE     PTRNORIG       IF ZERO, NO ORIGINAL VALUE
00004   313           LA     4,4
00000   314 PTRLOOP   C      0,0(4,5)       CHECK TO SEE IF GREATER THAN LOWER BOUND
0034C   315           BL     PTRLSS
00004   316           C      0,4(4,5)       CHECK TO SEE IF LESS THAN UPPER BOUND
0034C   317           BH     PTRLSS
00000   318           L      11,0(4,5)      FOUND AT THIS POINT
00000   319           L      10,0(11)       LOAD REFERENCE COUNT
00088   320           S      10,ONE         SUBTRACT 1 FROM REFERENCE COUNT
00000   321           ST     10,0(11)       STORE REFERENCE COUNT
        322           CR     1,11
0031C   323           BL     PTRAROUN
00004   324           C      1,4(4,5)
0033E   325           BL     PTRSAME        IF LOWER THAN UPPER THEN SAME CONYOUR
        326 PTRAROUN  CR     10,8
00368   327           BE     PTRDEALO       BRANCH TO DEALLOCATE AREA
00004   328 PTRNORIG  LA     4,4
00000   329 PTRLOOP2  C      1,0(4,5)
0035A   330           BL     PTRNXT         IF LOWER THAN LOWER THEN NOT FOUND
00004   331           C      1,4(4,5)
0035A   332           BH     PTRNXT         IF HIGHER THAN UPPER THEN NOT FOUND
00000   333           L      11,0(4,5)      LOAD BASE ADDRESS
00000   334           L      10,0(11)       LOAD REFERENCE COUNT
00088   335 PTRSAME   A      10,ONE         ADD 1 TO REFERENCE COUNT
00000   336           ST     10,0(11)       STORE BACK REF. COUNT
0000C   337 PTREND    LM     14,12,12(13)   RESTORE REGISTERS
        338           BR     14             RETURN TO PROGRAM
0008C   339 PTRLSS    A      4,EIGHT        INCREMENT INDEX TO NEXT AREA BOUNDS
        340           CR     4,3            CHECK TO SEE IF PAST DISPLAY BOUNDS
00322   341           BE     PTRNORIG       IF EQUAL THEN NOT FOUND
002EE   342           B      PTRLOOP
0008C   343 PTRNXT    A      4,EIGHT        INCREMENT INDEX TO NEXT BOUND
        344           CR     4,3            CHECK TO SEE IF PAST DISPLAY BOUNDS
00346   345           BE     PTREND         RETURN TO PROGRAM IF EQUAL
00326   346           B      PTRLOOP2
00004   347 PTRDEALO  L      0,4(4,5)       LOAD UPPER BOUND
00000   348           L      1,0(4,5)       LOAD BASE ADDRESS
        349           SR     0,1            SUBTRACT TO FIND LENGTH
        350           FREEMAIN R,LV=(0),A=(1) FREE THE NON-REFERENCED STORAGE
00000   353           ST     8,0(4,5)       CLEAR DISPLAY WHERE BOUNDS WERE
00004   354           ST     8,4(4,5)
0008C   355           A      4,EIGHT
        356           CR     4,3            CHECK TO SEE IF THAT WAS LAST DISPLAY
003A6   357           BNE    PTRNEW         CONTINUE WITH CHECKING IF NOT LAST
0008C   358           S      4,EIGHT        RESTORE 4 TO ORIGINAL POSITION
0008C   359 PTRLOOP3  S      3,EIGHT        DECREMENT DISPLAY LENGTH
00000   360           ST     3,0(5)         STORE DISPLAY LENGTH
0008C   361           S      4,EIGHT        DECREMENT TO CHECK PREVIOUS DISPLAY
00000   362           C      8,0(4,5)       CHECK TO SEE IF PREVIOUS WAS ZERO
003A6   363           BNE    PTRNEW         RETURN TO PROGRAM
```

```
0038F   364              B       PTRLOOP3      GO BACK AND DECREMENT AGAIN
00018   365 PTRNEW       L       1,24(13)      RESTORE ADDRES CURRENTLY IN POINTER
00322   366              B       PTRNORIG      CHECK CURRENT POINTER ADDRESS
```

```
0000C   368 SVC20    STM   14,12,12(13)   BLOCK EXIT ROUTINE
00040   369          L     5,READY        LOAD CURRENT CONTOUR
00000   370          LA    8,0            LOAD R8 WITH ZERO
00004   371          C     13,4(5)        CHECK TO SEE IF LAST BLOCK
003F4   372          BNE   BKEXARON       NO, SO CONTINUE
        373          LR    9,13           LOAD CURRENT CONTOUR ADDRESS
00004   374 LOOP8BK  L     9,4(9)         LOAD PREVIOUS CONTOUR ADDRESS
00004   375          C     8,4(9)         CHECK TO SEE IF ANY PREVIOUS CONTOURS
00468   376          BE    PROCEND        GO TO END OF PROCESS ROUTINE
00000   377          L     7,0(9)         LOAD REFERENCE COUNT
00088   378          S     7,ONE          SUBTRACT ONE
00000   379          ST    7,0(9)         STORE REFERENCE COUNT BACK
        380          CR    8,7            CHECK TO SEE IF ZERO OR LESS
003C4   381          BL    LOOP8BK        BRANCH TO CHECK NEXT PREVIOUS
00000   382          LA    1,0(9)         LOAD ADDRES OF CONTOUR
0004C   383          L     0,76(9)        LOAD LENGTH OF CONTOUR
        384          FREEMAIN R,LV=(0),A=(1) DEALLOCATE THE CONTOUR
003C4   387          B     LOOP8BK        RETURN TO CHECK NEXT PREVIOUS
        388 BKEXARON CR    0,8            CHECK REFERENCE COUNT FOR ZERO
00452   389          BH    BKEXRTRN       NO DEALLOCATION
00004   390          LA    4,4
00000   391 BKEXLOOP C     13,0(4,5)      CHECK BASE AGAINST DISPLAY
0040E   392          BE    BKEXFOUN       BASE FOUND IN DISPLAY
0008C   393          A     4,EIGHT        INCREMENT TO CHECK NEXT DISPLAY
003FE   394          B     BKEXLOOP
00004   395 BKEXFOUN L     0,4(4,5)       LOAD UPPER BOUND
00000   396          L     1,0(4,5)       LOAD LOWER BOUND
        397          SR    0,1            SUBTRACT TO FIND LENGTH
        398          FREEMAIN R,LV=(0),A=(1) FREE THE STORAGE
00000   401          ST    8,0(4,5)       CLEAR DISPLAY TO ZERO WHERE BOUNDS WERE
00004   402          ST    8,4(4,5)
0008C   403          A     4,EIGHT
00000   404          C     4,0(5)         CHECK TO SEE IF DISPLAY IS LARGER
0045C   405          BNE   BKEXNOD3       NO DECREMENT
0008C   406 BKEXLP2  S     4,EIGHT
00000   407          C     8,0(4,5)       CHECK TO SEE IF PREVIOUS DISPLAY IS ZERO
0044A   408          BNE   BKEXNOD2       NO DECREMENT
00084   409          C     4,FOUR         CHECK TO SEE IF LAST BLOCK
00432   410          BNE   BKEXLP2
00468   411          B     PROCEND        BRANCH TO PROCESSOR END
0008C   412 BKEXNOD2 A     4,EIGHT        SET LENGTH AT ONE POSITION HIGHER
00000   413          ST    4,0(5)         STORE DISPLAY LENGTH
0000C   414 BKEXRTRN LM    14,12,12(13)   RESTORE REGISTERS
00004   415          L     13,4(13)       GET CALLING BLOCK SAVE AREA
        416          BR    14
0008C   417 BKEXNOD3 S     4,EIGHT        GET CURRENT DISPLAY INDEX
00084   418          C     4,FOUR         CHECK TO SEE IF LAST DISPLAY
00452   419          BNE   BKEXRTRN       IF NOT THEN RETURN TO PROGRAM
```

```
00000   421 PROCEND  L      4,0(5)          LOAD DISPLAY LENGTH
00084   422          C      4,FOUR          CHECK TO SEE IF NONE
004A8   423          BE     PEENDDE         NO MORE DEALLOCATION
0008C   424 PELOOP   S      4,EIGHT         GET PREVIOUS DISPLAY
00084   425          C      4,FOUR          CHECK TO SEE IF LAST
004A8   426          BE     PEENDDE
00000   427          C      8,0(4,5)        CHECK TO SEE IF ALREADY DEALLOCATED
00474   428          BE     PELOOP
00000   429          L      10,0(4,5)       LOAD REFERENCE COUNT
00000   430          C      8,0(10)         CHECK TO SEE IF REFERENCE IS ABOVE ZERO
00474   431          BL     PELOOP          YES, SO CONTINUE
00004   432          L      0,4(4,5)        LOAD UPPER BOUND
00000   433          L      1,0(4,5)        LOAD LOWER BOUND
        434          SR     0,1             SUBTRACT TO GET LENGTH
        435          FREEMAIN R,LV=(0),A=(1) FREE STORAGE
00474   438          B      PELOOP
00008   439 PEENDDE  C      8,8(5)          CHECK LAST AREA
004C0   440          BE     PEALL           NO MORE DEALLOCATION
00008   441          L      0,8(5)          GET UPPER BOUND
00004   442          L      1,4(5)          GET LOWER BOUND
        443          SR     0,1             SUBTRACT TO GET LENGTH
        444          FREEMAIN R,LV=(0),A=(1) FREE STORAGE
        447 PEALL    TTIMER CANCEL          CANCEL REMAINING TIME
00054   450          MVC    READY,84(5)     MOVE NEXT PROCESSOR ADDRESS TO READY LIS
00000   451          LA     1,0(5)          LOAD REG. 1 WITH BASE ADDRESS FROM AREA
        452          FREEMAIN R,LV=100,A=(1) FREE PROCESSOR AREA
00040   456          C      8,READY         CHECK TO SEE IF ANY ON READY LIST
00818   457          BE     WAIT            NO, SO GO TO CHECK FOR I/O WAIT
00040   458 WOVER    L      5,READY         LOAD NEXT PROCESSOR
0005C   459          LM     13,14,92(5)     LOAD NEXT INSTRUCTION AND SAVE AREA ADDR.
        460          STIMER TASK,TIMEX,TUINTVL=88(5) SET INTERVAL TIMER
00010   464          LM     15,12,16(13)    RESTORE REGISTERS
        465          BR     14              RETURN TO NEXT PROGRAM
```

```
0000C    467 SVC24     STM    14,12,12(13)   V OPERATIONS ROUTINE
         468           LR     9,1            SAVE COMMON VARIABLE NUMBER
         469           TTIMER CANCEL         NO TIMER INTERRUPTS ALLOWED
00040    472           L      5,READY        GET PROCESSOR AREA ADDRESS
00058    473           ST     0,88(5)        STORE REMAINING TIME
         474           SR     8,8
00084    475           M      8,FOUR         MULTIPLY LOCK NUMBER BY FOUR TO GET INDE
00094    476           L      15,LOCKS(9)    LOAD CORRECT COMMON VARIABLE
00088    477           A      15,ONE         ADD ONE
00094    478           ST     15,LOCKS(9)    STORE NEW VALUE BACK
000BC    479           L      7,QUEUES(9)    GET CORRESPONDING QUEUE ADDRESS
         480           CR     7,8            CHECK TO SEE IF NONE ON WAITING LIST
0053E    481           BE     VNONE
00054    482           L      6,84(7)        LOAD NEXT PROCESSOR ON QUEUE
000BC    483           ST     6,QUEUES(9)    PUT NEXT TO BE FIRST ON QUEUE
00054    484           LA     5,84(5)        GET ADDRESS OF DESIRED TOP OF QUEUE
00790    485           BAL    14,SEARCH      BRANCH TO ROUTINE TO PLACE ON QUEUE
00040    486           L      5,READY        RESTORE ORIGINAL PROCESSOR
         487 VNONE     STIMER TASK,TIMEX,TUINTVL=88(5) RESTORE TIME
0000C    491           LM     14,12,12(13)   RESTORE REGISTERS
         492           BR     14             RETURN TO PROGRAM
```

```
ADDR2  STMT   SOURCE STATEMENT                                          F01MAY7

0000C   494 SVC28    STM   14,12,12(13)  P OPERATIONS ROUTINE
        495          LR    9,1           SAVE LOCK NUMBER
        496          TTIMER CANCEL       NO TIMER INTERRUPTS ALLOWED
00040   499          L     5,READY       GET CURRENT PROCESSOR AREA
00058   500          ST    0,88(5)       STORE AWAY REMAINING TIME
        501          SR    8,8
00084   502          M     8,FOUR        MULTIPLY LOCK NUMBER TO GET INDEX
00094   503          L     15,LOCKS(9)   LOAD PARTICULAR COMMON VARIABLE
00088   504          S     15,ONE        SUBTRACT ONE
00094   505          ST    15,LOCKS(9)   STORE COMMON VARIABLE BACK
        506          CR    15,8          CHECK TO SEE IF ALREADY IN USE
0058A   507          BL    PALREADY
        508          STIMER TASK,TIMEX,TUINTVL=88(5) RESTORE TIME
0000C   512          LM    14,12,12(13)  RESTORE REGISTERS
        513          BR    14            RETURN TO PROGRAM
0005C   514 PALREADY STM   13,14,92(5)   SAVE RETURN AND SAVE AREA
00054   515          MVC   READY,84(5)   SET NEXT ON TOP OF READY QUEUE
        516          LR    7,5           LOAD NEXT PROCESSOR ADDRESS
000BC   517          LA    5,QUEUES(9)   LOAD TOP OF QUEUE ADDRESS
00790   518          BAL   14,SEARCH     BRANCH TO PUT ON QUEUE
00040   519 PAROUND  C     8,READY
00818   520          BE    WAIT
00040   521          L     5,READY
0005C   522          LM    13,14,92(5)   LOAD NEXT INSTRUCTION AND SAVE AREA ADDR
        523          STIMER TASK,TIMEX,TUINTVL=88(5) SET TIME INTERVAL
00010   527          LM    15,12,16(13)  RESTORE REGISTERS
        528          BR    14            RETURN TO PROGRAM
```

```
       530 * PRIORITY SET ROUTINE
00040  531 SVC32     L     15,READY          GET CURRENT PROCESSOR ADDRESS
00064  532           LH    1,100(15)         LOAD MAXIMUM PRIORITY
       533           CR    0,1               CHECK TO SEE IF DESIRED GREATER THAN MA
005CE  534           BNL   PSOK              NO, SO GO TO SET
       535           LR    0,1               CHANGE DESIRED TO MAX.
00066  536 PSOK      STH   0,102(15)         STORE NEW PRIORITY IN PROCESSOR BLOCK
       537           BR    14                RETURN TO USER PROGRAM
```

```
0000C    539 SVC36     STM    14,12,12(13)  I/O ROUTINE
00620    540           BAL    11,EVENTSET   GO TO SET UP ECB FOR SUBTASK
         541           LR     11,1          SAVE ECB ADDRESS
         542           GETMAIN R,LV=52      GET STORAGE FOR CONSTANT LIST
         546           LR     10,1          SAVE CONSTANT LIST ADDRESS
0000C    547           MVC    0(52,10),ATCHLIST MOVE ALREADY BUILT CONSTANTS TO LIST
00008    548           ST     11,8(10)      STORE ECB ADDRESS IN CONSTANT LIST
         549           GETMAIN R,LV=8       GET STORAGE FOR PARAMETERS
00000    553           ST     13,0(1)       PASS SAVE AREA ADDRESS AS A PARAMETER
00674    554           LA     5,IOTASK      LOAD ADDRESS OF DESIRED TASK
00004    555           ST     5,4(1)        STORE ADDRESS IN PARAMETER LIST
         556           LR     15,10         PASS CONSTANT LIST TO SVC IN R15
         557           SVC    42            ISSUE ATTACH SVC
00040    558           L      5,READY       LOAD CURRENT PROCESSOR BLOCK ADDRESS
00054    559           MVC    READY,84(5)   SET NEXT TASK ON TOP OF READY LIST
00054    560           ST     8,84(5)       SET LINK TO ZERO
0059E    561           B      PAROUND       GO TO LOAD NEXT PROCESSOR



         563 EVENTSET TTIMER CANCEL         NO INTERRUPTS ALLOWED
00040    566           L      5,READY       LOAD CURRENT PROCESSOR ADDRESS
0005C    567           STM    13,14,92(5)   SAVE ADDRESSES
00080    568           MVC    88(4,5),TIMESET SET TIME INTERVAL FOR NEXT EXECUTION
         569           GETMAIN R,LV=8       GET ECB FOR NEW TASK
         573           MVI    0(1),X'00'    SET TASK ECB TO NOT COMPLETED
00004    574           ST     5,4(1)        STORE PROCESSOR ADDRESS IN ECB
00054    575           L      6,EVCOUNT     LOAD NUMBER OF TASKS
00058    576           LA     7,EVENTS      LOAD ADDRESS OF EVENTS POINTERS
         577           AR     7,6           ADD TO FIND NEXT EVENT POINTER
00000    578           ST     1,0(7)        STORE ECB ADDRESS IN POINTER
         579           OI     0(7),X'80'    INDICATE LAST EVENT
00000    580           LA     8,0           GET A ZERO IN R8
         581           CR     8,6           SEE IF ONLY EVENT
0066A    582           BE     IOFIRST       YES, SO SKIP CLEARING FLAG
00084    583           S      7,FOUR        GET TO PREVIOUS EVENT
         584           NI     0(7),X'7F'    CLEAR FLAG INDICATING LAST EVENT
00084    585 IOFIRST   A      6,FOUR        INCREMENT EVENT COUNTER
00054    586           ST     6,EVCOUNT     STORE BACK COUNT
         587           BR     11            RETURN TO CALLING ROUTINE
```

```
0000C    589 IOTASK    ST      14,12(13)      SAVE RETURN ADDRESS TO SUPERVISOR
00000    590           L       7,0(1)         RESTORE OLD SAVE AREA ADDRESS
         591           GETMAIN R,LV=72        GET A SAVE AREA
00004    595           ST      13,4(1)        SAVE PREVIOUS SAVE AREA ADDRESS
00008    596           ST      1,8(13)        STORE CURRENT IN PREVIOUS
         597           LR      13,1           LOAD NEW SAVE AREA ADDRESS
00010    598           LM      15,1,16(7)     RESTORE REGISTER VALUES
         599           LR      7,1            SAVE DECB ADDRESS
         600           BALR    14,15          BRANCH TO READ/WRITE ROUTINE
         601           CHECK   0(7),DSORG=ALL WAIT FOR COMPLETION OF I/O
         609           LR      1,13           GET SAVE AREA ADDRESS
00004    610           L       13,4(13)       RESTORE ORIGINAL SAVE AREA
         611           FREEMAIN R,LV=72,A=(1) FREE TEMPORARY SAVE AREA
0000C    615           L       14,12(13)      RESTORE RETURN ADDRESS
         616           BR      14             RETURN TO SUPERVISOR INDICATING TERM.
```

```
0000C    618 IOCOMPL  STM   14,1,12(13)      COMPLETION EXIT ROUTINE FOR I/O SUBTASK
         619          TTIMER CANCEL          NO TIMER INTERRUPTS ALLOWED
         622          LR    9,0              SAVE TIME LEFT IF ANY
00058    623          LA    7,EVENTS         LOAD ADDRESS OF EVENT POINTERS
00000    624          LA    4,0              GET A ZERO IN R4
00050    625          L     11,IOMASK        LOAD MASK TO CHECK FOR COMPLETION
00000    626 IOCLOOP  L     3,0(4,7)         LOAD ECB ADDRESS
00000    627          L     10,0(3)          LOAD ECB VALUE
         628          NR    10,11            CHECK FOR COMPLETION
006F6    629          BNZ   IOCFOUND         NOT ZERO SO FOUND
00084    630          A     4,FOUR           ZERO, SO 2NCREMENT TO NEXT ONE
006E0    631          B     IOCLOOP          GO TO CHECK NEXT ONE
00000    632 IOCFOUND LA    8,0              GET A ZERO IN R8
00040    633          LA    5,READY          GET ADDRESS OF READY QUEUE IN R5
00040    634          C     8,READY          CHECK TO SEE IF NONE READY
0070E    635          BE    IOCNREDY         YES SO GO TO HANDLE THIS
00040    636          L     5,READY          LOAD PROCESSOR WHICH IS READY
00054    637          LA    5,84(5)          PRETEND THAT LINK IS HEAD OF QUEUE
00004    638 IOCNREDY L     7,4(3)           LOAD PROCESSOR ADDRESS
00790    639          BAL   14,SEARCH
         640          LR    1,3              LOAD ADDRESS OF ECB
         641          FREEMAIN R,LV=8,A=(1) FREE ECB STORAGE
00018    645          L     1,24(13)         LOAD TCB ADDRESS
0078C    646          ST    1,TCBHOLD        STORE TCB ADDRESS
         647          DETACH TCBHOLD         DETACH SUBTASK FROM SYSTEM
00058    650          LA    7,EVENTS         LOAD ADDRESS OF EVENT POINTERS
         651          AR    4,7              GET POINTER TO CURRENT EVENT
0004C    652 IOCLOOP2 L     10,IOCMASK       LOAD MASK TO CHECK FOR LAST POINTER
00000    653          N     10,0(4)          CHECK TO SEE IF LAST EVENT
00750    654          BNZ   IOCDONE          YES, SO GO TO COMPLETION
00004    655          MVC   0(4,4),4(4)      NO, SO MOVE NEXT INTO CURRENT
00084    656          A     4,FOUR           INCREMENT TO NEXT BLOCK
00736    657          B     IOCLOOP2         GO TO CHECK NEXT EVENT POINTER
00000    658 IOCDONE  LA    8,0              GET A ZERO IN R8
00000    659          ST    8,0(4)           CLEAR LAST EVENT POINTER
         660          SR    4,7              GET INDEX ONLY IN R4
00054    661          ST    4,EVCOUNT        RESET EVENT COUNT
         662          CR    8,4              CHECK FOR A ZERO INDEX
00772    663          BE    IOCSKIP          SKIP IF NO EVENT POINTERS
         664          AR    4,7              RESTORE POINTER TO LAST EVENT POINTER
00084    665          S     4,FOUR           DECREMENT TO PREVIOUS
         666          OI    0(4),X'80'
         667          OI    0(4),X'80'       PLACE A ONE IN HIGH ORDER BIT
         668 IOCSKIP  LPR   8,9              CHECK FOR A ZERO TIME
00786    669          BZ    IOCSKST          YES, SO SKIP SETTING TIMER
00020    670          ST    8,32(13)         PLACE TIME REMAINING IN ACCESIBLE PLACE
         671          STIMER TASK,TIMEX,TUINTVL=32(13) SET TIME INTERVAL
0000C    675 IOCSKST  LM    14,1,12(13)      RESTORE REGISTERS
         676          BR    14               RETURN TO OPERATING SYSTEM
         677 TCBHOLD  DS    F                STORAGE TO HOLD TCB ADDRESS FOR DETACH
```

```
00054    679 SEARCH   LA    11,84         GET AN 84 IN R11
00000    680          L     6,0(5)        LOAD PROCESSOR ADDRESS
         681          SR    5,11          SUBTRACT 84 FROM QUEUE ADDRESS
00000    682          LA    11,0          GET A ZERO IN R11
00054    683          ST    11,84(7)      CLEAR LINK OF NEW PROCESSOR
         684          CR    11,6          CHECK TO SEE IF NO PROCESSOR ON QUEUE
007C6    685          BE    SRCHNONE      NO, SO GO TO PLACE ON TOP OF QUEUE
00066    686          LH    10,102(7)     LOAD PRIORITY OF NEW PROCESSOR
00066    687 SRCHLOOP CH    10,102(6)     CHECK TO SEE IF GREATER THAN CURRENT
007D2    688          BL    SRCHPASS      NO, SO GO TO PLACE ON QUEUE
00054    689          C     11,84(6)      CHECK TO SEE IF NO MORE PROCESSORS
007CC    690          BE    SRCHDONE      YES, SO GO TO PLACE ON QUEUE
         691          LR    5,6           SAVE PREVIOUS POINTER
00054    692          L     6,84(6)       LOAD POINTER TO NEXT PROCESSOR
007AC    693          B     SRCHLOOP      GO TO CHECK NEXT PROCESSOR
00054    694 SRCHNONE ST    7,84(5)       STORE NEW PROCESSOR AT HEAD OF QUEUE
         695          BR    14            RETURN TO CALLING ROUTINE
00054    696 SRCHDONE ST    7,84(6)       STORE NEW PROCESSOR AT END OF QUEUE
         697          BR    14            RETURN TO CALLING ROUTINE
00005    698 SRCHPASS LA    10,5          LOAD HIGHEST NUMBER OF TIMES PASSED OVER
00068    699 SC2LOOP  C     10,104(6)     CHECK TO SEE IF PASSED UP TOO MANY TIMES
00802    700          BH    SC2MANY       GO TO INCREMENT TO NEXT PROCESS
00054    701          ST    7,84(5)       STORE NEW PROCESSOR IN PROPER POSITION
00054    702          ST    6,84(7)       RESET LINKS
00001    703          LA    7,1           GET A 1 IN R7
00068    704 SRCHLOP2 L     10,104(6)     GET TIMES PASSED OVER COUNT
         705          AR    10,7          INCREMENT TIMES PASSED OVER
00068    706          ST    10,104(6)     STORE TIMES PASSED OVER BACK
00054    707          C     11,84(6)      CHECK TO SEE IF ANY MORE PROCESSORS
         708          BCR   8,14          NO, SO RETURN TO CALLING ROUTINE
00054    709          L     6,84(6)       LOAD POINTER TO NEXT PROCESSOR
007EA    710          B     SRCHLOP2      GO TO INCREMENT NEXT PROCESSOR COUNT
00054    711 SC2MANY  C     11,84(6)      CHECK TO SEE IF NO MORE PROCESSORS
007CC    712          BE    SRCHDONE      NO MORE, SO GO TO STORE AT END OF QUEUE
         713          LR    5,6           SAVE POINTER
00054    714          L     6,84(6)       GET NEXT PROCESSOR ADDRESS
00066    715          LH    10,102(7)     RELOAD PRIORITY
007AC    716          B     SRCHLOOP      GO TO CHECK PLACEMENT AGAIN
```

```
00054    718 WAIT      C     8,EVCOUNT      CHECK TO SEE IF ANY EVENTS WAITING
0082E    719           BL    WAROUND
         720           ABEND 16,DUMP
         728 WAROUND   WAIT  1,ECBLIST=EVENTS WAIT FOR COMPLETION OF I/O
004E2    733           B     WOVER          GO TO LOAD PROCESSOR
```

```
00018    735 SVC40    ST     1,24(13)        SAVE DATA BLOCK FOR OPEN ROUTINE
         736          TTIMER CANCEL          NO INTERRUPTS ALLOWED
00014    739          ST     0,20(13)        SAVE TIME REMAINING
00018    740          L      1,24(13)        RESTORE DATA BLOCK ADDRESS
         741          SVC    19              ISSUE OPEN SVC
         742          STIMER TASK,TIMEX,TUINTVL=20(13) RESTORE TIME INTERVAL
         746          BR     14              RETURN TO PROGRAM
```

```
ADDR2  STMT   SOURCE STATEMENT                                                   F01MAY7

          1             START
          2 * THIS IS THE TASK WHICH IS ATTACHED TO INITIATE I/O.  IT DOES NOTHIN
          3 * BUT BRANCH TO THE DESIRED POINT IN THE MONITOR--IOTASK.
00004     4 LINK        L     15,4(1)
          5             BR    15
          6             END   LINK
```

APPENDIX B

A Sample Program Run

```
ADDR2  STMT   SOURCE STATEMENT                                               F01MAY72

       748 PBEGIN   ENTER 252,1        ENTER
00005  764          LA    1,5          GET A 5 IN R1
000F0  765          ST    1,240(13)    STORE IN CONSTANT LOCATION
00001  766          LA    1,1          GET A 1 IN R1
000F4  767          ST    1,244(13)    STORE IN CONSTANT LOCATION
00007  768          LA    1,7          GET A 7 IN R1
000F8  769          ST    1,248(13)    STORE IN CONSTANT LOCATION
       770          BALR  1,0          GET CURRENT ADDRESS
00136  771          B     310(1)       BRANCH AROUND PROCEDURE
       772          ENTER 124,2        ENTER 'FACT' PROCEDURE
00004  788          L     1,4(13)      LOAD PREVIOUS SAVE AREA
0000C  789          L     14,12(1)     REGISTER REGISTER 14
00000  790          L     1,0(14)      LOAD ADDRESS OF ACTUAL PARAMETER #1
00050  791          ST    1,80(13)     STORE IN FORMAL PARAMETER LOCATION
00004  792          L     1,4(14)      LOAD ADDRESS OF SECOND ACTUAL PARAMETER
00054  793          ST    1,84(13)     STORE IN FORMAL PARAMETER LOAATION
00008  794          L     1,8(14)      LOAD THIRD ACTUAL PARAMETER ADDRESS
00058  795          ST    1,88(13)     STORE IN FORMAL PARAMETER LOCATION
00000  796          LA    1,0          GET A ZERO IN R1
00070  797          ST    1,112(13)    STORE IN CONSTANT
00001  798          LA    1,1          GET A 1 IN R1
00074  799          ST    1,116(13)    STORE IN CONSTANT
00050  800          L     1,80(13)     LOAD ACTUAL PARAMETER ADDRESS
00000  801          L     3,0(1)       L     3,N
00070  802          L     4,112(13)    L     4,=F'0'
       803          SR    4,3          SR    4,3
       804          BALR  1,0          GET CURRENT ADDRESS
00076  805          BE    118(1)       BE    AROUND
00074  806          S     3,116(13)    S     3,=F'1'
0005C  807          ST    3,92(13)     ST    3,TEMP
00054  808          L     1,84(13)     LOAD ACTUAL PARAMETER ADDRESS
       809          CALLP -,136,,,100,92(13),96(13),112(13)
00050  831          L     1,80(13)     LOAD ACTUAL PARAMETER ADDRESS
00000  832          L     5,0(1)       L     5,N
00000  833          LA    4,0          LA    4,0
00060  834          M     4,96(13)     M     4,X
00054  835          L     1,84(13)     LOAD ACTUAL PARAMETER ADDRESS
00000  836          ST    5,0(1)       ST    5,R
       837          BALR  1,0          GET CURRENT ADDRESS
00010  838          B     16(1)        B     AROUND2
00074  839          L     3,116(13)    L     3,=F'1'
00054  840          L     1,84(13)     LOAD CURRENT ADDRESS
00000  841          ST    3,0(1)       ST    3,R
00058  842          L     1,88(13)     LOAD ACTUAL PARAMETER ADDRESS
00000  843          L     3,0(1)       L     3,FLAG
00074  844          L     4,116(13)    L     4,=F'1'
       845          CR    3,4          CR    3,4
       846          BALR  1,0          GET CURRENT ADDRESS
0000C  847          BNE   12(1)        BNE   AROUND3
       848          LETGO 1            LETGO 1
       851          RETURNP            RETURN
       860          GRAB  1            GRAB  1
000F0  863          L     3,240(13)    L     3,=F'5'
00058  864          ST    3,88(13)     ST    3,TEMP
       865          CALLP -,324,1,TASK,96,88(13),80(13),244(13)
```

```
000F8    884            L      3,248(13)        L      3,=F'7'
0005C    885            ST     3,92(13)         ST     3,TEMP+4
         886            CALLP -,396,1,TASK,96,92(13),84(13),244(13)
         905            GRAB   1                GRAB   1
         908            GRAB   1                GRAB   1


         912 * THIS ROUTINE HAS BEEN ADDED TO PRINT OUT THE ANSWERS


         914 ***********************************************************************
         915            MOPEN 206,OUTPUT
00050    927            L      3,80(13)
00070    928            CVD    3,112(13)
00070    929            UNPK   80(4,13),112(8,13)
         930            MVI    108(13),X'40'
0006C    931            MVC    109(131,13),108(13)
         932            OI     83(13),X'F0'
00050    933            MVC    158(4,13),80(13)
         934            IO     W,140,108(13)
00054    950            L      3,84(13)
00070    951            CVD    3,112(13)
00070    952            UNPK   84(4,13),112(8,13)
         953            MVI    108(13),X'40'
0006C    954            MVC    109(131,13),108(13)
         955            OI     87(13),X'F0'
00054    956            MVC    158(4,13),84(13)
         957            IO     W,52,108(13)
         973 ***********************************************************************

         975            EXIT                    EXIT
         980            DCB    DDNAME=PRT,DSORG=PS,MACRF=W,BFTEK=R,RECFM=F,
                               LRECL=132,BLKSIZE=132,SYNAD=ERREX
        1031            END    GOMONI
```

0120
5040

APPENDIX C

English Flowchart of the Monitor

GOMONI--Initial Housekeeping

1. Store the registers in the operating system save area and get enough storage for the first processor block.

2. Store the address of the processor block on the ready queue and initialize the contents of the processor block.

3. Issue a SPIE macro so that all user interrupts will be given to the monitor.

4. Load address of processor block and set initial time interval in processor block.

5. Place processor block address back on ready queue, set interval timer for first process, and branch to the start of the program.

INTEREX--Internal Interrupt Handler

1. Store registers so that they may be viewed in the dump of storage.

2. Set user completion code to 18, dump out main memory, and quit.

TIMEX--Interval Timer Interrupt Handler

1. Chain through the system control blocks until the address of the program's request block is found.

2. Load the old PSW from the request block to get the address where the program will resume.

3. Save the next instruction which would have been executed and replace it by a branch to the monitor.

4. Return to the OS supervisor.

5. When the program resumes it will branch to TIMEXR.

6. Save the registers and restore the saved instruction.

7. Get processor block address and store new time interval in the block.

8. Branch to the routine which places the suspended processor block on the queue according to priority.

9. Get block which is on the top of the ready queue, restore its environment and set a new time interval.

10. Start the process at the point of suspension.

SVC4--Keep Track of Display(After Block Entry)

1. Get current processor block address.

2. Store the upper and lower bounds for the new contour in the appropriate positions in the display. Set display count to point to next position in display.

3. Return to program.

SVC8--Process Creation Routine

1. Save the registers and cancel the timer interval saving the remaining time by placing it in the current processor.

2. Chain back through all of the contours in the current environment incrementing the reference counters of each.

3. Save the next instruction to be executed and the current contour address in the processor block.

4. Allocate new storage for the new processor block.

5. Store control values in the processor block.

6. Check the requested priority against the current process's priority. If lower than current priority then store requested priority in new processor block, else store current priority in new block.

7. Set new time interval after placing new process at top of the ready queue and branch to the new process.

SVC12--Find Base for Global Variable

1. Chain back through the current environment looking for the desired block name.

2. If block name is found, return to program with base address in register 15.

3. If block name is not found, set user completion code to 19 and quit.

SVC16--Change Reference Count after Pointer Value is Changed.

1. Save registers and load processor block address.

2. Check to see if previous value of pointer was zero. If yes, go to step 5.

3. Check previous pointer value against the bounds in the display. If value is not between any of the bounds in the current environment, go to step 5.

4. Decrement the reference counter and if it reaches zero, deallocate the contour.

5. Compare the new value of the pointer against all of the bounds in the current display. If the value is found to be between any of the bounds, increment the reference counter.

6. Return to the program, after restoring the registers.

SVC20--Block Exit Routine

1. Save registers and get processor block address.

2. Check to see if this is the last block of the process. If this is not the last block go to step 5.

3. Chain back through all of the contours in the current environment decrementing the reference counter of each. If any of the reference counters reach zero, deallocate the contour.

4. Go to step 1 of PROCEND.

5. Check the reference counter of the current contour. If it is not zero go to step 8.

6. Free the storage for the current contour, and clear the display where the bounds were.

7. Check the previous position of the display to see if it is zero. If so decrement the display count and go to step 7.

8. Return to program after restoring the registers.

PROCEND--Routine for Process Termination

1. Check to see if any contours remain in the display which have not been deallocated. If there are none, go to step 3.

2. If any contours remain in the display whose reference counters are zero or less, deallocate them.

3. Cancel the remaining time in the interval.

4. Place next processor block on top of ready queue and deallocate the terminating process's processor block.

5. Check to see if any process is ready to execute. If not go to the WAIT routine, step 1.

6. Load the new process's current environment and next instruction, set the new time interval and branch to the new process after restoring its registers.

SVC24--LETGO Operations Routine

1. Save the registers and cancel the time interval saving the remaining time in the processor block.

2. Increment the particular system variable requested.

3. Check to see if any processes are waiting on this system variable. If none, go to step 6.

4. Take the top process from the queue.

5. Branch to routine which places new process in its appropriate place on the ready queue according to priority, but not on the top of the ready queue.

6. Restore the registers and time interval for the process which issued the LETGO macro.

7. Return to execution of the suspended process.

SVC28--GRAB Operations Routine

1. Save the registers and cancel the time interval saving the remaining time in the processor block.

2. Decrement the requested system variable. If its value goes below zero go to step 4.

3. Restore the remaining time in the time interval and return to the suspended process after restoring the registers.

4. Save the current environment and next instruction in the processor block.

5. Branch to the routine which places the current process on the particular system variable queue according to priority.

6. Check to see if there is another process ready to execute. If there is none, go to the WAIT routine, step 1.

7. Restore the environment and next instruction for the new process, set the time interval, and branch to the new process.

SVC32--Priority Set Routine

1. Load address of current processor block and load maximum priority.

2. If desired priority is greater than the maximum, set the desired priority equeal to the maximum.

3. Store the desired priority in the processor block and return to the requesting program.

SVC36--I/O Routine

1. Save the registers.

2. Branch to EVENTSET, step 1, which allocates an event control block for the I/O subtask.

3. Save the address of the ECB and get storage for the constant area needed to attach a subtask to the system.

4. Place the address of the allocated constant area in register 15 and move a preset constant area into the allocated area.

5. Store the ECB address in the constant area and allocate storage to hold the parameters which are passed to the task.

6. Store the save area address and the address of IOTASK, the routine which actually handles the I/O, in the parameter area.

7. Issue the supervisor call which attaches LINK, a subtask, to the system.

8. Set the next processor block on the ready list to be at the top and branch to step 6 of SVC28.

EVENTSET--Set up ECB and place Process in Wait State

1. Cancel the time interval, load the processor block address, and save the current environment and next instruction in the processor block.

2. Get an area to be used as an event control block.

3. Set the first byte of the ECB to indicate not completed yet and store the processor block address in the second word of the ECB area.

4. Store address of the ECB on the EVENTS list.

5. Indicate that this is the last address in the list by placing a 1 in the high-order bit of the address and clearing all other high-order bits in the list of address.

6. Set the event count to point to the next available address space and return to the calling routine.

IOTASK--Do I/O Concurrently with Processing

1. Save the return address and restore the save area address of the requesting process.

2. Allocate an area for a new save area since the I/O routine supplied by IBM destroys the current save area.

3. Save the data event control block address and branch to the IBM supplied READ/WRITE routine.

4. Wait for completion of the I/O.

5. Free the temporary save area and return to the OS supervisor thus indicating completion of the routine and posting the ECB so that the monitor will get interrupted.

IOCOMPL--I/O Completion Interrupt Handler

1. Save the registers and cancel the time interval saving the remaining time in register 9.

2. Check all of the ECB's pointed to by the EVENTS list until one event is found which has completed.

3. Get the processor block address from the completed ECB.

4. If a process is currently executing set register 5 to point at the next processor block on the ready queue so that the executing process is not blocked, otherwise register 5 points to the top of the ready queue.

5. Branch to the SEARCH routine which places the new processor block on the ready queue according to its priority.

6. Deallocate the storage which was used for the ECB.

7. Detach the subtask from the system.

8. Restructure the EVENTS list to reflect the fact that one of the ECB addresses has been deleted.

9. If no interval was in effect go to step 11.

10. Restore the time interval.

11. Restore the registers and return to the OS operating system.

SEARCH--Place a Processor Block on a Queue According to its Priority

1. Load the processor block address and register 5 already contains the address of the top of the queue.

2. Subtract 84 from register 5 to indicate that the new process address will go in the link of the process immediately above it.

3. Look at each processor block on the queue until one is found which has a lower priority, in which case go to step 5.

4. If the end of the queue is reached, store the new processor block at the end of the queue and return to the calling routine.

5. Check to see if this processor block has already been passed over 5 times. If so, continue where the search was left off in step 3.

6. If the processor block with the lower priority has not been passed over 5 times yet, store the new processor block at the front of the process with the lower priority.

7. Increment the times passed over field of each of the processor blocks which are lower on the queue.

8. Return to the calling routine.

WAIT--No Processes are Ready to Execute

1. Check to see if any processes are waiting on I/O. If any are waiting, go to step 3.

2. Set the user completion code to 16 indicating either deadlock or termination of all processes, dump out the main storage, and quit.

3. Wait for one of the processes to complete the I/O operation.

4. Go to step 6 of PROCEND.

SVC40--Open Files

1. Save the data block address needed for the open and cancel the time interval.

2. Save the time remaining and place the data block address in register 1.

3. Issue the open supervisor call.

4. Restore the time interval and return to the process.

LINK--Branch to Desired Subtask

1. Load the desired entry point address.

2. Branch to the desired entry point.

MULTITASKING IN A USER PARTITION WITH A CONTOUR MODEL OF PROCESSES

by

LEE ALLEN

B. S., Georgia Institute of Technology, 1972

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1973

This paper attempts to demonstrate the feasibility of using the contour model of block structured processes to design an operating system. The lack of technology to handle innovations in Computer Science and higher level languages is pointed out as a partial justification for implementing such an operating system. The model chosen to demonstrate the feasibility was a multi-tasking monitor which resides in the user partition running under OS/MFT on the IBM system 360.

A process is defined as a "time-invariant algorithm and a time-varying record of execution of that algorithm." The monitor handles each task in the system as a process. Each process is assigned a processor block consisting of certain control information needed to coordinate the system.

Process execution may be suspended for various reasons. In this case the processor block is put on a queue until the conditions that caused suspension are satisfied. When a process is ready for execution its processor block is placed on the ready queue waiting on CPU time. Deadlock is detected when all of the processes are on queues waiting on resources which can never be satisfied. Each process is assigned a certain priority level which designates where the processor block is placed on queues waiting for resources. The monitor guarantees that a process can not be continually passed up by placing higher priority processes ahead of it.

Variable storage is kept separate from the instructions in a portion of storage called a contour. Contours are retained in the system as long as any other variable in another contour points at them. The contour consists of some control information along with the variable storage.

The language used to take advantage of all of the capabilities of the monitor is described. Generally, the language is an extension of IBM 360 assembly language. There are certain register restrictions when using the language and these are described. There is only the amount of overhead added to regular assembly language processing as is needed by the user. A block structure is used in the language in order to make the transition from higher-level languages more natural. Processes are created by calling a procedure as a task. Since variables are reserved storage apart from the instruction, special pseudo-ops are used in the language to declare variables. Pointers are implemented to facilitate indirect reference. Input and output are handled by the monitor so that processing can continue while the I/O operations are taking place. There are system variables implemented to facilitate process communication and cooperation.

An ALGOL language example is given to illustrate some of the features of the monitor. The ALGOL version and compilation are given along with a diagram of a moment in time during execution of the program.

Finally, a comparison is made between the monitor and other current operating systems. Some practical applications of the monitor are discussed such as time-sharing in a single partition and the use of the monitor as a nucleus upon which could be built an operating system. A contour model machine is discussed as a future possibility.