ROUGH DRAFT
Art Rosenberg
2/7/67

Introduction

The Sigma 7 computer has been designed to provide for maximum exploitation of computer technology. However, hardware facilities merely provide the environment for software systems which must reflect practical implementation of computer utilization techniques. To this end, the Sigma 7 software provides the Universal Time-Sharing (UTS) Monitor System, which encompasses those facilities and services appropriate for general-purpose, multi-programmed on-line, time-shared operations. As such, the UTS Monitor extends the operating capability of the Sigma 7 from the Batch Processing mode to a more flexible and dynamic form of computer processing.

I. CONCEPTS OF THE UNIVERSAL TIME-SHARING MONITOR

The UTS Monitor has been designed to provide multi-programming services for online (interactive) user programs, in addition to batch-mode production jobs, symbiont I/O, and critical real-time processes.

To clarify the operational environments which the UTS Monitor is designed to service, it is appropriate to define a number of terms which will be used in this manual.

Multiprogramming: A technique for maximizing the efficient use of a computer system by overlapping computation with I/O operations. In particular, where one processing job cannot provide maximum overlap, other jobs are operated concurrently to achieve such system efficiency at a reasonable cost in overhead.

Time-Sharing: The sharing of computer system resources to provide processing service for several user functions (related or independent), where an individual job does not require all of the computer's time. The allocation of such resources is based upon satisfying the response-time needs of the participating tasks. Thus, critical on-line processes can be interleaved with on-line, interactive

Time-Sharing (Cont'd) operations, and non-priority production jobs can absorb any remaining computer time.

Interactive (Conversational) Processing: One of the major benefits of the time-sharing approach to computer utilization. Direct access to computational processes affords convenient, personalized services for a variety of new users and applications. The needs of interactive computation include a satisfactory response time for non-machine dialogue (e.g., 1-5 seconds), comprehensive, easy-to-use service facilities, and system reliability.

Remote Access: The physical extension of computer system access via communication lines. Remote access may be provided for interactive, production, and critical real-time operations.

Reentrant Processes: Those processes which permit independent users (or user programs) to execute them concurrently or in an interruptable sequence. Reentrant processes are in "pure procedure" form, where instructions are never modified and user-dependent context is separable from the process through isolable data and machine-register storage areas. The scheduling and switching of user environments is performed by an external process, i.e., an operating system Executive.

Recursive Processes: Processes which must control successive use by preserving user contexts themselves. Recursive routines may be those which call upon themselves in a nested fashion (last-in, first-out, LIFO) or may be first-level, machine-dependent routines (i.e., interrupt routines) which cannot expect context to be preserved by other processes.

(more definitions coming)

The UTS Monitor is designed to provide a spectrum of services ranging from Batch operations through real-time tasks. For each functional level, there exists certain fundamental differences in scheduling logic as well as certain services which must be rendered.

Batch Level

User jobs are organized in a stack fashion, where the stack is a first-in, first-out queue ordered by explicit priority. Batch jobs are considered as Background operations, with the lowest implicit priority in the system. However, operational efficiency for production jobs will not be jeopardized unnecessarily because of priority levels, specifically in the area of system facility allocations. At System Generation time, or by operator action, resources may be permanently or temporarily allocated for batch operations so that implicit priority operations will not impede production throughput. Once a production job in the background has been initiated, its allocated facilities (I/O devices, secondary storage) will not be preempted except by a foreground task or operator action. Actual Monitor service for compute time and I/O operations will, however, be given to a batch job at the lowest level of implicit priority unless compute time is explicitly dedicated for such operations.

At the Background level, there does exist a second level of priority, which is explicit. Normally, jobs can be entered into the stack with whatever administrative priority is appropriate. However, this will not cause a current Background job to be suspended. Operator action or a service call from a foreground process can cause a checkpoint of the current background job (with "draining" of current I/O operations) so that an important Background job can be initiated as soon as possible. In the worse case, the current Background job can be terminated (abort) immediately and reinitiated completely at a later point in time.

By definition, Batch jobs are unattended operations. As such, run parameters must be provided to the UTS system which define maximum operating limits which should not be exceeded; if such limits are violated, the job will be terminated. Where such limit parameters are not provided by the user, system limits are imposed by default. Similarly, in the event of system-detected program errors, the default case will be a job abort. However, provision is made for user recovery or "end-action", whereby, if expressly indicated, a user-supplied recovery routine will be given information concerning the type of error detected and the user-program's environment (registers, status double-word) at the time of error detection. This capability is particularly useful for run-time debugging.

It is possible to have user programs which are designed to operate in either a production mode or conversational mode. Similarly, production programs can be initiated from a job stack or on-line from a user terminal. In the first case, the user program must be designed to determine its mode of operation from the type of control device used. This is set by the UTS Monitor when it loads the program, i.e., from an on-line terminal or a control card file in the production stack. In the former case, however, the user can also initiate a batch job from a remote terminal using the BATCH command, which will set the mode to production, not conversational. The program, designed for both types of use, will organize its elements to respond interactively or process batch inputs and outputs; this may be done by having two types of "front ends".

User programs can also be designed through the above medium to switch their mode of operation from conversational to production, but not the reverse. One should not confuse the choice of output media with the scheduling queue for a user's job. Once a job has been assigned to the Batch queue, it cannot be communicated with by the user although its status can be controlled as any other job in the stack. If, however, voluminous output is not desired at a low-speed terminal, the output can be shunted to a secondary storage file, magnetic tape, high-speed printer, etc. The job can still remain in one of the on-line queues for input control purposes.

Interactive Processing

The most significant extension of computer utilization which is accommodated by Sigma 7 and the Universal Time-Sharing Monitor is in the area of Interactive Processing. For this environment, the UTS Monitor recognizes the more dynamic needs of interactive users as well as certain implicit priorities of their operations.

Interactive users impose an implicit priority upon a computer system by the simple fact that they are "on-line" to the system. In providing this direct access to the computer, it is necessary to satisfy the user with a response time geared to a man-machine dialogue. Actually, the interactive user can expect two levels of responsiveness:

"Conversational" Response: When the interactive user makes simple data entries, issues Control Commands, or requests a simple computational process, he expects a rapid acknowledgment of his input or (in the last case) a computational result.

"On-Line" Computation: It is possible for the interactive user to initiate a long and involved computer process of indeterminate length. Obviously, a response time for completion of this process cannot be guar~~r~~anteed, but the operating system must do the best it can on an equitable basis for all such users.

By defining system performance goals, the conversational response-time cycle can be made secure. That is, by establishing the maximum number of users, the cut-off point in CPU time allocation for a "conversational" process (the "conversational quantum"), etc., it is possible to insure the proper turnaround time to a conversational input. For a general-purpose time-sharing system, it is not possible to make assurances for turnaround-time of any user computational process; however, for special applications where the process is a known quantity, system design can account for some guarantee of response time at this level.

The UTS Monitor provides a Scheduling package for general interactive users with installation-dependent parameters to govern the user service at the conversational and on-line computing level. Special application needs can be accommodated by modification or extensions to this scheduling logic.

To accommodate effective interactive processing, the UTS Monitor exploits the Sigma 7 Map feature for dynamic relocation of user programs and/or data. This facility permits users to share memory easily, not only in terms of preserving virtual memory addressing, but in allowing the use of non-contiguous physical core memory. This is essential for maximized efficiency in a time-sharing environment, where secondary storage (disc) is used for inactive program residency and swapping.

Selective memory protection is also provided through the Sigma 7 Map, to insure the integrity of the operating system and individual user processes and data in a time-shared environment. User programs cannot accidentally (or otherwise) gain illegal access to virtual and physical memory areas not specifically allocated for their use.

The Map feature is of great value for reentrant programs, which are of major utility in a time-shared environment. Common processes can be shared by several users concurrently, since their unique data areas appear identical in virtual memory, but are actually different in physical memory. By simply changing the user's map and program status doubleword (registers, program counter, etc.), a common process can immediately start (or continue) operating for a different user.

Real Time

Critical Real-Time processes are given privileged services as in the Batch Processing Monitor. These include:

- Master mode operation
- Dedication of core memory space, I/O devices, I/O channels, CALL trap locations, External Interrupts, clocks, fast register blocks, etc.
- I/O priority servicing
- Interrupt service
- Data and Command Chaining
- End-Action I/O service

In addition, for the UTS environment, the following services are possible:

- Clock-watching service (synchronous scheduling)
- Virtual memory space dedicated in all user maps
- Automatic swapping for non-resident foreground tasks
- Non-mapped operation (for critical, resident foreground tasks)

Real-time processes fall into two general categories, as far as responsiveness is concerned:

- Critical Real-Time
- Tolerant Real-Time

In the first case, responsiveness must be insured by constant core residency, direct connection to interrupts, dedication of system facilities, etc. In short, system overhead must be reduced to a minimum. Tolerant real-time processes are those which can afford some reasonable delay between the activating signal and the time for process execution. In this case, the real-time process can be dynamically serviced by the UTS Monitor without unnecessary dedication of all operating facility needs, particularly core memory. However, the implicit priority of any real-time process is higher than a background job (production) or interactive users and are classed as "foreground" operations.

Foreground processes can be either permanently installed for continual operation at System Generation time or dynamically loaded as a new job by the Operator. In both cases, once the real-time task has been properly loaded into the system and activated, it can utilize the privileged services provided by the UTS Monitor. A foreground task can operate either in the Master Mode or in the Slave Mode, depending upon its operating requirements. This facility will require that <u>any user-supplied foreground task which operates in the Master Mode must be thoroughly checked out before it can be safely operated in the full UTS environment.</u>

Although the UTS Monitor will provide a maximum of <u>generalized</u> services and facilities for real-time operations, it cannot, by definition, protect the user from himself. Most specifically, in a time-shared environment it is quite tempting and easy to overload the system. In attempting to operate several foreground tasks concurrently, it is essential to provide sufficient system resources if proper responsiveness is to be maintained.

Peripheral Processing

One of the important facilities available and important to the Universal Time-Sharing Monitor is the capability of performing concurrent bulk-input-output transfers between various peripheral devices (particularly to and from disc storage) with minimal interference to other computations. To accomplish this, two types of service functions are provided by the UTS Monitor:

- SYMBIONTS
- COOPERATIVES

In essence, peripheral processing consists primarily of I/O transfers with a small amount of computational effort to drive the peripheral devices. Symbionts are the driving routines for peripheral processing and there can be input and output Symbionts for every Sigma 7 I/O device. They operate on the basis of bulk transfers, independent of any user program needs.

To accommodate format needs of a user program, the Cooperative routine is employed to feed I/O data to and from a Symbiont. On input, a Cooperative will break down (unpack) the data for the user program; for output, a Cooperative would pack the data for the Symbiont.

Symbionts have an implicit priority higher than background production jobs, but lower than interactive or foreground tasks. Symbiont operations are initiated by Operator action and can be terminated by the Operator as well. Symbionts are also automatically activated for normal Batch job operations, e.g., printer output via the disc. User programs use Symbionts in conjunction with Cooperatives for interfacing, as described above.

Multi-Processing

Several features of the Sigma 7 computer, coupled with the UTS Monitor, could be used for a multi-processing environment. The hardware map, tasking service, reentrant routines of the Monitor, etc., lend themselves to task-oriented multi-processing.

The UTS Monitor does not provide for multi-processing, however, since additional extensions are required for this purpose, primarily in the scheduling logic to permit execution of more than one process at a time. In addition, control would have to be exercised over the bookkeeping tables of the Monitor such that only one CPU could modify common data at any given time. Other complications would be caused by I/O and external interrupt configurations which could require CPU-to-CPU communications not provided for in the UTS Monitor.

Resource Allocation

The fundamental requirement for the UTS system is to perform effective resource allocation to concurrent system users on a dynamic basis. The resources are varied in type, number and mode of allocation; some are more static allocations, others, very dynamic. Certain resources are very critical to system loading capability and must be governable by a user installation.

Resources Available to a User Program Under UTS

1. Core memory space for execution
2. Compute time (CPU)
3. I/O channel time and/or assignment (depending on type)
4. I/O peripheral devices
5. Active swapping storage on fast-access disc
6. Permanent file storage on high-capacity disc
7. Trap location
8. Interrupt lines
9. Communication channels
10. Last memory blocks
11. Map space (virtual memory space)

The above resources affect users in the following areas:

1. Convenient access for loading (time)
2. Convenient access for execution (time)
3. Restrictions on entry into the system (executable or not)
4. Restrictions on program execution (size, time, facilities) and priority

Those resources which are sharable (i.e., core storage, auxiliary random-access storage, CPU time) will be controlled by System Generation and Operator Key-in parameters so that any one user or user program cannot freely degrad system performance or preempt service from other users. Through the use of such parameters, a system may allocate maximum amount of these resources, special-case (privileged) allocations, or first-come, first-serve unrestricted use of such resources.

Non-sharable resources will be allocated on a first-come, first-served basis and/or by dedication to privileged foreground programs. This will be done either by specific assignment at System Generation time or upon user program demand.

II. UTS STRUCTURE

The Universal Time-Sharing Monitor encompasses all of the features of the Sigma 7 Batch Processing Monitor, since a comprehensive time-sharing environment includes production job processing as a background operation. As such, the UTS Monitor is actually structured upon the functional elements of the Batch Processing Monitor with several notable extensions.

- Core residency and overlay organization of the Monitor is different because of the more dynamic environment for interactive users and additional Executive services.
- The Sigma 7 map facility is used to advantage wherever possible for dynamic relocatibility, context switching, core space management, etc.
- Provision is made for automatic swapping between core memory and fast disc storage to provide maximum computation and I/O overlap.
- Scheduling is provided for the multi-user and multi-task environment.
- File management control is extended to cover the on-line multi-user operating environment.
- The accounting mechanism is extended for the on-line multi-user operating environment.
- Management of public subroutine virtual memory space is provided for.
- Dynamic core space allocation is provided through the Sigma 7 map.
- Provision is made for character-oriented communications processing and interactive user console services.
- A conversational Executive Control Language processor is added to supplement the normal Batch-Mode Control Card Interpreter.

The UTS Monitor occupies a portion of the total virtual memory space (128K) available to each user. This sharing of the map permits the UTS Monitor itself to take advantage of dynamic relocation for greater organizational flexibility. It also enables more efficient Monitor interaction with a user's program by providing a common virtual memory reference between them at all times.

The UTS Monitor is composed of a number of functional modules all of which are pure procedures and therefore reentrant. Those processing modules which are not required to be permanently in core can therefore always be restored with a fresh copy from disc instead of being saved each time. The monitor processing modules are linked together by central communication and bookkeeping data tables. Any Monitor process may be interrupted before completion and reentered, providing the common bookkeeping tasks have been completed, or if the interrupting task is independent of the central bookkeeping functions (i.e., a real-time foreground process with dedicated facilities). In only a few strategic areas will the interrupt problem be regularly forestalled by interrupt disabling while critical bookkeeping is performed; that is, pushing machine register contents.

Privileged Functions

In the UTS Monitor system, all machine controls are considered to be privileged to the Monitor and those real-time tasks which require such privileges. Thus, a user program can normally only exercise a machine operation (other than computation) indirectly by asking the Monitor to perform such a function. This applies also to making changes to data located in Monitor-controlled storage areas in a program's virtual memory. This is necessary to minimize or eliminate any accidental catastrophic system hang-up due to a software error. By centralizing privileged operations as much as possible, even within the Monitor area itself, system maintenance, extensibility, and modularity is greatly enhanced.

Executive Functions

Executive functions of the UTS Monitor are distinguished from the more fundamental Monitor services by the fact that they involve the administrative tasks, initialization or termination functions, secondary operations for user program operations, or non-critical yet useful functions for system users. The Executive elements of the UTS Monitor do not themselves perform any privileged tasks directly, but cause the main Monitor elements to do so when necessary. Data associated with Executive functions can often be made non-resident in core memory because of less-frequent use or lower priority for response when core space is at a premium.

Executive services of the UTS Monitor include the following:

- Accounting
- Job Scheduler (Batch)
- Error processing and abort service
- I/O resource allocation
- Storage allocation (secondary storage)
- File security
- User information services (system status, etc.)

- Checkpoint service (save for restart)
- Console Communications services
- Overlay loader
- Public subroutine control

Main Monitor

The hard core of the UTS Monitor consists of those Monitor functions which are constantly required to be in core memory for program execution.

- I/O Dispatcher
- I/O Handlers (standard peripherals)
- Task Scheduler
- Interrupt processing
- Clock processing
- Trap processing
- Secondary storage access control
- Operator communication
- Communications handler (for interactive users)
- Symbionts and cooperatives (when active)
- Real-time service routines
- Swap module
- Map Maintenance (core allocation)
- File Management Basic Tables

Service Subsystems

The UTS Monitor provides the operating environment for user programs which can be system service programs or private programs. System services are public programs, designed for multi-use (reentrant) in the interactive case, or any installation program for the batch mode of operation. Such programs are available from secondary storage and are recognized in the UTS Executive command list. (Private programs require specific file management service for loading and operation.) Service subsystems are often in high demand for convenient response to interactive usage and can, at System Generation time, be made permanent core residents. However, in general, the Service Subsystems are basic programs for system users which are treated as user programs when operating.

The following subsystems are currently available in the UTS system:

- Symbol
- Meta-Symbol
- FORTRAN IV (Debug and High-Efficiency)
- SYMBUG (DDT)
- EDITOR
- HELP
- COBOL (Batch mode only)
- 1401 Simulator (Batch mode only)

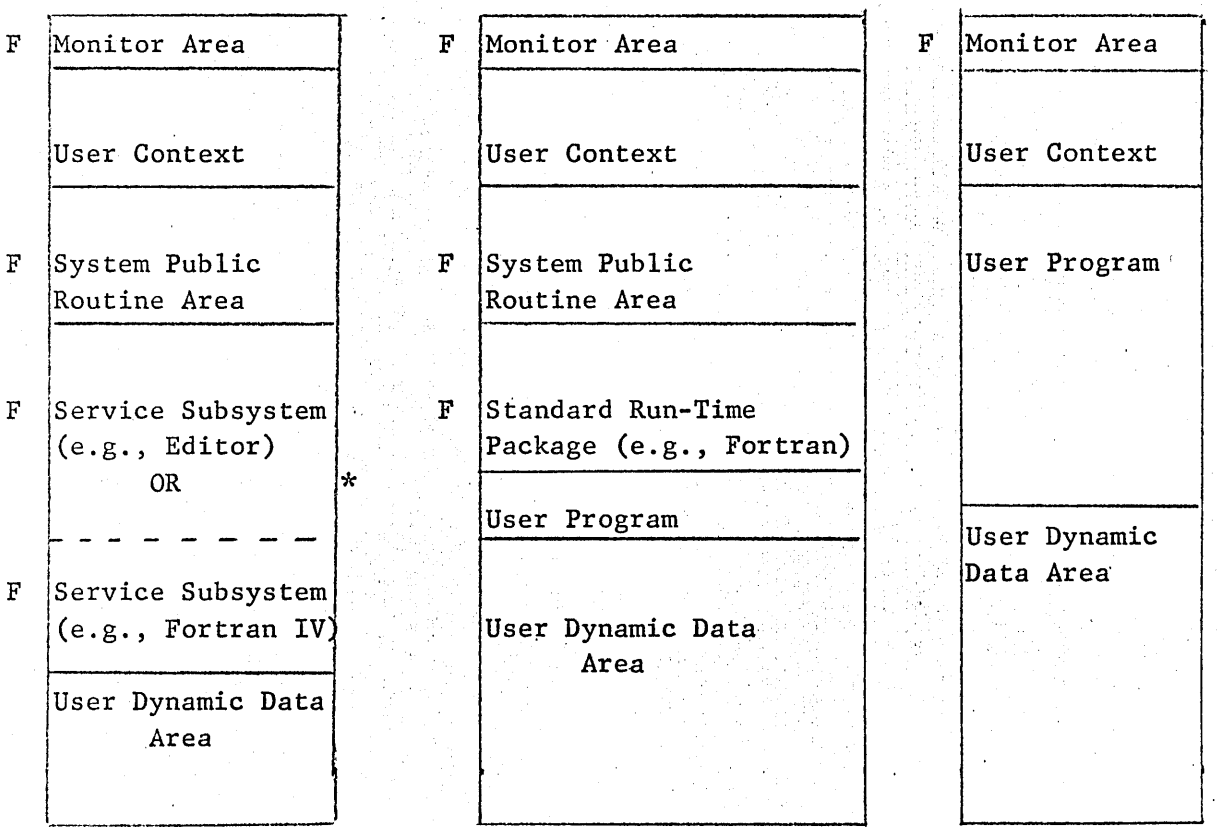
Other service subsystems can be added to the System at System Generation time to serve particular application needs of a Sigma 7 installation. Such subsystems must be carefully administered as far as becoming public system services, since they should be properly checked out, documented, and be appropriate for installation needs. Other useful routines and programs can be made available to all users by being entered into a public library and loaded as a private program through the file management facility.

Map-Sharing

All programs and routines, including the UTS Monitor, which will be operating together in core memory, must share a common virtual memory, i.e., the same hardware map. This affords maximum operating efficiency under the hardware architecture of Sigma 7. Public functions must be consistently allocated the same virtual memory areas to allow for concurrent use by multiple users. This means that a user's memory map will include all public, reentrant routines of the UTS system that are required for the user's operation and these routines will always be in the same virtual address locations. For public library routines, which are available to the user but not on a reentrant basis (or for popular multi-use), any available virtual space is used at load time.

Map-sharing should not be confused with the sharing of physical core memory. Routines and/or data used by a program may be in physical core memory but not necessarily in the program's map. Thus a program which uses overlays may have the overlay segments in core memory and the overlay action consists of simply changing the physical addresses in the memory map. Programs designed for Sigma machines without the map hardware and smaller core memories will be operable in a more efficient manner by using the map in UTS for virtual memory overlays.

Map assignments of virtual memory will typically appear as follows:


F Monitor Area
User Context
F System Public Routine Area
F Service Subsystem (e.g., Editor)
OR
*
F Service Subsystem (e.g., Fortran IV)
User Dynamic Data Area
F Monitor Area
User Context
F System Public Routine Area
F Standard Run-Time Package (e.g., Fortran)
User Program
User Dynamic Data Area
F Monitor Area
User Context
User Program
User Dynamic Data Area

F = fixed virtual memory locations in all users' maps.

* = overlayable processes in virtual memory, but resident in actual core memory.

Certain functions are assigned to permanent physical locations. Such functions are tied to the hardware (interrupts, traps) and, because they are not relocated, can operate without the map. A dedicated foreground process can be so treated and avoid map-sharing of virtual memory. (Physical core memory, of course, is still being shared.)

Public Subroutines

A significant facility of the UTS System is the ability to provide useful subroutines (reentrant) (e.g., mathematical routines, byte string routines, etc.) for concurrent utilization by several user programs. This feature minimizes core storage requirements in a shared memory environment as well as swapping overhead.

For Sigma 7, the public subroutine area is a dedicated portion of the virtual memory space. This space is dedicated in each users' map, as is the Monitor space, since program addressing within the public subroutines must remain constant. Inasmuch as it is uncertain how much of this "public" space should be allocated for a given installation, it becomes a System Generation parameter. Furthermore, each installation may require different subroutines to be resident and therefore only those administratively identified for public residency will be loaded into the public space.

Public Routine virtual space is selective, depending upon installation needs. At System Generation time, a specific set of subroutines is permanently established in the public space. These will never be changed during system operation and any other subroutines from the system library will be loaded as a private copy in a user's program. There is no demand swapping of public subroutines because of the impracticality of dynamically moving routines in virtual memory.

For these users who wish to utilize the maximum amount of virtual memory space in their maps, they may ask for all subroutines to be private copies and the public sub-routine space in their map will be made available for private use. This permits a user program which uses few public subroutines to gain more virtual memory than normally available.

User Program Context

The UTS Monitor is structured along the lines of distributed centralization. That is, although various data must be maintained by the Monitor for an individual user program, such data is not centralized in Monitor Core Memory space. Rather, each user program will carry within its addressing space areas devoted to Monitor-Controlled data storage. Thus, when a user is not active in memory, it is possible to liberate a maximum amount of his physical core memory (swapping) for other users. The size of this context space is variable, depending on the user program's operating needs and it can expand dynamically during execution by linking the user's virtual memory pages.

The contents of the user's context area includes the following: (See Appendix ____)

- Map Image
- Task Control Blocks
- Data Control Blocks
- Program Status Doubleword
- Simulated Sense Switches
- Program status indicators (Monitor-set)
- Temporary storage pointer
- Communication Echo table (private version)

The context block information is selectively protected so that only the UTS Monitor can modify its contents. However, the user program can read the information therein (those set by the user or Monitor at known locations) or use the data for indirect addressing. Furthermore, changes can be made to context block contents through Monitor service calls.

CAL Traps

Since user programs are prohibited from going directly to the Monitor core memory area by the memory protection hardware of the Sigma 7 computer, the CAL instruction provides the mechanism for user programs - Monitor intercommunication. By executing a CAL instruction, one of sixty-four memory locations will be reached where control reverts to the Master mode (XPSD) and the user's service request is examined further.

Not all the CAL locations are used by the UTS Monitor and the available ones can be employed by user foreground programs for various privileged functions (Real-Time Services). They can also be used by an installation to extend the UTS Monitor services to user programs (System Generation).

Selective Memory Protection

The UTS Monitor allocates core memory according to operating needs and by memory protection needs. That is, the number of memory pages required is governed by the total memory size required as well as selective partitioning into different memory protection areas.

Monitor:	No Access
Public Routines:	Read and Execute Only
Context Area:	Read Only
User Program (Instructions):	Read and Execute Only
User Data:	All Access, Read Only

User programs must provide indications at load time of their process and data partitioning, otherwise an "All Access" protection will be assigned. "No Access" protection is assigned to any dedicated virtual memory space (i.e., to a critical real-time process) while it is not active, thereby minimizing environment switching (no map changes).

Lock and key protection is used to guarantee the integrity of the Monitor and Executive areas from foreground tasks operating in the Master mode. It will restrict such tasks from violating areas other than its own.

Reentrancy

The map feature of Sigma 7 has been designed to provide reentrancy capabilities for UTS programs. By changing the user's map, a common program process can be applied to a different user context and data area, even though virtual addresses in the common process are the same. The requirement for reentrant process is that it be "pure procedure", never modifying any instructions, only data. Since it is difficult to verify whether a program has indeed been implemented for reentrancy when written in assembly language, and since "public" service programs should be properly controlled, the assumption will be made that a program is not reentrant unless administratively entered into the system.

The UTS system requires reentrancy for all programs and subroutines which will be public and used concurrently by multiple users. By definition, reentrant programs must be pure procedure; i.e., no modification of instructions and separation of process (instructions) and context data from data, temporary storage, and context on a page-protected basis. All reentrant programs will utilize the map to switch from user to user for a reentrant program.

Subroutines, both for the standard System Public Subroutine area and for the standard Run-time package for a language processor, must be organized by the loader to have any local storage separated from the process into independent pages. To accomplish this, the standard subroutines must be organized into two blocks; one block of m pages will have the processes (instructions and constant data only), the other block of n pages will contain all local storage for the subroutines. The loader will adjust all addresses in the subroutines to access the relocated temporary storage. Every user map will have the same virtual space dedicated to these public subroutines and/or run-time packages. Thus, by switching the map, only the subroutine storage areas will have to be changed to reflect a new user's context location in physical core memory. This scheme will not permit dynamic storage to be used; any subroutine requiring dynamic storage will have to be a private copy in the user's program.

A user program must indicate at load time whether or not the standard subroutine package(s) should be used or only private copies should be loaded.

III. UTS FUNCTIONS

Core Memory Management

The UTS system allocates core memory on a dynamic basis in order to support the multi-programming environment. It is required that sufficient core memory be available during system operation to accommodate enough user processes so that program execution can take place while I/O and swapping are being performed.

A System Generation parameter, which can be overridden by the Operator during system operation is the maximum size of physical core memory which will normally be made available to a user program. This is necessary in order to avoid having a particular user program degrade system performance for other users. If a larger program (and/or data) is required by a user, a logical overlay at that program's expense will be performed.

Physical core memory areas will be allocated on a dynamic basis to all processes requiring such storage, based on implicit and explicit priority. When physical pages are necessary, low-priority processes (non-reentrant) or data areas will be purged to secondary storage (swapped). Reentrant processes will not be saved, since a fresh copy will be brought in when necessary. For I/O operations in progress for a user's program, the I/O buffer pages will be retained in core along with its context blocks continuing the DCB's, etc. If no operation is being performed for a user program, all of that program's core space will be purged, leaving only queue information for future activation.

To control core space allocation, a bit map of core memory pages is maintained by the UTS Monitor. In addition, the scheduling queues with their implicit and explicit priorities are used to find what must be brought in and what can be swapped out. Each program's context area will contain a map image used to both establish the hardware map when executing and to reset the bit map for memory when core space is released by swapping.

Core memory management of the UTS system will recognize special needs for user programs. These include:

. Core space dedication for foreground residency

. Specific core memory addressing space required for special hardware processors. This would include restricting the use of this space for processes which could not easily be preempted, (i.e., "sensitive" areas). This information will be defined at System Generation time.

When necessary, depending upon priority, the required available space will be derived by swapping out the current residents. If time permits, special core needs will be accommodated by moving current occupants in core memory.

~~From a system efficiency point-of-view, physical core memory allocation may be profitably organized so that protection types are located together. For example, all reentrant processes are physically in certain areas of core, user context areas in others, foreground tasks are physically together, and the more dynamic, swappable elements of user programs are also co-located.~~

Swapping

As part of the dynamic scheduling of user programs in the UTS system, a swapping function is used. Swapping involves the removal from physical core memory of a user's program and/or data to make room for another user. It is closely tied to the scheduling logic of the UTS system in that practical decisions must be made for the following:

. Who executes next

. Who is transferred in next

. Who is removed from core next (purge)

The logic of scheduled swapping is to always have a user process available for execution while other user processes are being swapped. This requires sufficient core memory space to be available so that enough users can be resident to guarantee maximum overlap of swapping (I/O) time by user execution time. Resident user execution times must be equal to or greater than the time to get other users ready (swapped) for execution.

Part of the efficiency in swapping chores will be derived from being judicious about what has to be swapped. By being selective, physical core space can be made available with minimal overhead or with more practical pay-off in scheduling operations.

User core space will not be released unless required for other users or if the user program action explicitly calls for a release function, e.g., program termination. Given that a scheduling decision requires physical core space to be made available to an inbound user process or an existing one, the following logical selections for swapping will take place:

1. If a resident user process is not operating, but is awaiting execution in the current queue (priority level) it will not be swapped out before less "appropriate" user space has been exhausted.

2. "Appropriate" user space for swapping is determined by selecting a user process which is not ready for execution and is awaiting a Monitor service or input (i.e., suspended).

3. Among all user processes in item 2, the choice of space to be made available is selected in the following order:
   - Protected space (non-modifiable) unique to a single user (not a common process or data area).
   - Protected space (non-modifiable) common to several users, but not currently required for execution.
   - Unprotected space (modifiable) belonging to a suspended user process. Suspended will refer to most recently suspended, e.g., having just been executed within the current queue cycle.

* NOTE: *
REENTRANT COMMON PROCESSES ARE RELEASED FIRST WHEN THE SCHEDULER EXECUTES ALL USERS OF SUCH A PROCESS AS A GROUP. THIS INSURES THAT NO USER FOR THE PROCESS REMAINS TO BE EXECUTED DURING THIS RESPONSE CYCLE.

A use counter for process or data elements will be required for users sharing common elements, based on load-time information, to satisfy retention of multi-use elements.

Reentrant elements are appropriate for available space, since saving is not required (no full swap), but only some bookkeeping is necessary. However, such bookkeeping will include map maintenance for users and if a reentrant element is moved, it will cause many maps to be modified.

Since hardware facilities are not available to handle actual core modifications or references, software checking of such history will be a source of overhead for questionable pay-off.

Swapping storage will be assigned only on fast, random-access storage. It will consist of space for all active, unique user elements and common processes (reentrant) or common data (active). In order to maintain efficient swapping transfers, it is necessary to compromise the amount of selectivity in saving user process elements. That is, it may be more expedient to save large blocks of user elements than to pick out little pieces for transfer to and from the high-speed disc. For example, all data pages should be swapped, if modifiable and protected elements are intermixed.

Logical Page-Turning

The UTS Monitor will offer a facility for a user program to operate within a smaller amount of physical core storage when the virtual addressing memory is large. This facility can be used where there is a maximum limit set upon a user's physical core space allocation or where there just is not enough physical memory available.

~~The mechanism to be used will not be demand paging on a single page basis. This has been shown to be disastrous to system operating efficiency. Instead,~~ An overlay structure will be required from the user's program at load-time which will describe program segments for physical rather than virtual memory. That is, although the program will be written as if it had sufficient core space available to it, a tree structure will show what logical pieces do indeed have to be resident at any given time.

The Batch system overlay mechanism can be used to provide this form of logical page-turning, since the organization requirements are similar. The tree structure will show which pieces must always be resident, which pieces must be in core concurrently, and which pieces can be removed at certain times. By making a reference to an unavailable VIRTUAL location, the overlay tree will be consulted to determine what overlay action is to be followed.

The UTS Monitor will employ the map to make changes in the user's physical memory relative to the virtual space. Thus, a limited number of physical core memory pages can be used to service a larger number of virtual memory pages. This approach will afford a reasonable and practical method for efficient core space utilization without dangerous guessing and excessive system overhead.

In addition to the above method of "reference loading", explicit overlays can be called for by an executi~~ve~~NG program. This is an overlay in the virtual memory space and consequently in the physical space as well.

Auxiliary Storage Organization for UTS

Auxiliary storage requirements for the UTS system are greater than for those of the Batch Processing Monitor for a number of reasons:

1. UTS System is larger in size.
2. Additional forms of storage required.
3. Volume of storage requirements significant.

Auxiliary storage in the UTS environment is divided into three fundamental categories:

1. Active random-access storage
2. Permanent random-access storage
3. Permanent external storage

Active Random-Access Storage

This storage requires the SDS high-performance rapid-access disc for maximizing multiprogramming efficiency in the UTS system. Contained in this storage are the following types of files:

1. UTS Main Monitor and Executive Routines
2. System Bookkeeping data (e.g., file management directions)
3. Service programs
4. Swap storage for active programs
5. Scratch storage for user programs
6. Active non-core resident foreground programs
7. Overlay elements for active user programs (active load modules)

This storage is a critical resource for the UTS system and is a limiting factor on system performance and loading. It must be kept relatively unclogged by excessive transfer requests and priority for residency goes to the storage items listed above. Access time IS MINIMIZED AS OPPOSED TO TRANSFER TIME, SINCE MOST ACTIVITY WILL BE IN MANY, SMALL TRANSFERS. RAD ORGANIZATION AND [illegible] QUEUEING WILL BE EMPLOYED TO OPTIMIZE FOR SWAPPING [illegible]

Permanent Random-Access Storage

This type of storage must be random-access in nature but speed of accessibility need not be as great as the Active Random Access storage. ~~Either high performance~~ EITHER THE STANDARD RAD's or high-capacity ~~RAD~~ DISC can be used, although the latter is preferable recommended. Residency in this storage includes the following:

1. Public libraries (programs, subroutines)
2. Private libraries (user-owned)
3. Batch jobs awaiting execution (Job Stack)
4. *User permanent input and output files
5. Symbiont buffering for I/O peripherals
6. Accounting data
7. Save-for-Restart (check point) programs

*User data files for input or output to an activated program will not be moved to Active Storage unless explicitly called for by the program. This is done (for input files) by a request for file movement (copy) from Permanent Storage to Active scratch storage. Output files generated by a user program in Active scratch storage must be explicitly "saved" before termination in order for them to be preserved in the system. Otherwise, input and output files will be accessed by a user program directly from Permanent Storage.

System Generation parameters are used to define the allocation of all random-access storage available in the system, including maximum limits for individual users, both Permanent and Active storage. The allocation of Permanent and Active storage will have a more significant effect on UTS system configurations that have high-speed discs and high-capacity discs; where only high-speed discs are used, movement from Permanent to Active storage will not be necessary.

Permanent External Storage

Magnetic tape is required to back-up random-access storage and will be used for purge storage when Permanent Random-Access storage becomes full or when permanent copies of programs or data are required. Purging will be under Computer Operator Control in terms of selecting files to be purged. Age of files (when last used) as well as priority will be taken into consideration according to installation administrative rules. The UTS File Management system will provide inventory display of current files on Permanent random storage and will purge to tape either specific files or files selected by age of use or owner criteria.

INSERT A

Scheduling

Two forms of scheduling are operative in the UTS system; Job Scheduling and Task Scheduling. The former applies to a user job operating in the Batch Mode, while the latter refers to elements of jobs (of any kind) for the multiprogramming environment, i.e., interactive processing.

Job Scheduling

Job scheduling is performed as under the Batch Processing Monitor, i.e., a job stack is maintained on the disc which is organized on a rank-ordered priority basis. This priority is an administrative number assigned externally to incoming jobs.

In addition, a foreground or interactive job can activate a background job by placing it in the stack (via a monitor service call). An interactive job can place itself (or be placed by the user) on the production stack, so that it is no longer interactive. In this last instance, the job is in save-for-restart form for subsequent execution.

Job Scheduling in the UTS (and BPM) system operates one job at a time from the production stack. From a practical point of view, it does not assume special situations where more than one background job is active or provide sophisticated scheduling rules (e.g., automatic priority upgrading based on elapsed time and deadliness). However, users can, if they so wish, tailor job scheduling to their own needs by installing their own job scheduling modules at System Generation time.

Task Scheduling

Task scheduling applies to the dynamic scheduling of user jobs in the multiprogrammed environment of a time-sharing system. Based on user service needs and the load on the system, the task scheduling function determines the operational behavior of any given system. Because such needs will vary from installation to installation, the UTS system recognizes that the task scheduling logic must be highly flexible for proper adjustment and tailoring. Consequently, the major scheduling elements have been made installation parameters which can be specified at System Generation time and/or while the system is in operation.

The scheduling logic which can be so adjusted lies primarily in the area of interactive use. Overall scheduling consists of three user areas:

1. Foreground (real-time)
2. Interactive users
3. Background (Batch) jobs

Foreground jobs are assigned priority in absolute rank-order, based on priority interrupts. "Synchronous" foreground jobs have higher priority than interactive users and are activated when clock "timer" runs out. (The UTS Monitor does the clock watching.) The time allocation for foreground activities should be preplanned by the user installation to insure against overloading the system.

The interactive user level is the area where great variations can occur and scheduling can be tuned accordingly. Within this level are two types of operations:

1. Conversational Processing
2. On-Line Computation

Dynamic parameters for conversational processing are:

1. Maximum number of interactive users allowed in the system.
2. Conversational time quantum (maximum amount of time allowed for conversational mode).
3. Conversational duty cycle.

Parameters for on-line computation are:

1. Number of queue levels (multi-level queue)
2. Time quantum for each level

Multi-level queues are functionally useful for the on-line compute mode of interactive processing for several reasons. First, it is difficult to ascertain the length of compute time required to service this mode for all users in order to produce the response output. Second, since time must be equitably shared (normally), it is appropriate in the system to "feel" its way in a dynamic and uncertain environment. Finally, it allows a finer categorization of interactive use in a generalized system such as UTS; e.g., large-size processes may be immediately relegated to a lower priority queue when in the on-line compute mode.

The multi-level scheme can be employed either as a fixed assignment method for known types of processes (which is difficult without installation experience) or, as has been done with several prototype time-sharing systems, programs can be "pushed down" to lower priority levels (with accompanying longer time "quanta") as required dynamically. The goal of such scheduling is to favor conversational response and fast, short on-line computation as opposed to longer on-line computing.

Needless to say, the algorithm for scheduling can and should be adjusted to an installation's needs after such needs have become empirically determined. For this reason, maximum modularity and flexibility to make such changes are provided in the UTS design.

The rules for a user to move from one queue level to any other (including the conversational queue) are also variable for installations and can be changed by assembly and system generation loading. They are not dynamically changeable during system operation.

Effectively, queue management logic for the interactive level involves the decision rules for the following known events:

1. Time quantum exceeded
2. Program return for various monitor services (I/O calls, interactive input or output, overlay, linking, etc.)

Another parameter, specifiable at System Generation time and during system operation, is the amount of time to be allocated for batch jobs. This time is essentially a guaranteed portion of the interactive duty cycle which is always given over to background jobs. It, therefore, will implicitly limit the maximum number of interactive users which can be served simultaneously. The amount of time allocated to a Batch operation is cumulative; that is, if I/O is requested, CPU time is still available to it during the interactive duty cycle. Thus, a Batch job can operate intermittantly within the duty cycle to drive I/O rapidly. However, the requirement for accommodating Batch operations in this manner is that core space for a Batch job must be pre-allocated so that no swapping interference takes place. The Batch job must be resident while it is active (getting CPU time) all during the duty cycle. It may be removed only when its CPU time is exhausted; however, ~~even then~~ this ~~may~~ IS not ~~be practical~~ VERY EFFICACIOUS. in terms of maximum efficiency.

Shifting Between Conversation and On-Line Compute Modes

One of the difficulties encountered with interactive time-sharing systems has been to differentiate between the conversational and on-line compute~~d~~s phases of an interactive process. Early systems simply used compute~~d~~s time utilized as a measure of this differential. However, other clues can be employed effectively.

The following ground rules are follwed by the UTS system to determine the status of an interactive process:

1. Any I/O call for interactive input, i.e., keyboard with a WAIT option, will be considered as a sign that the program is indeed in a conversational phase.
2. Any I/O request, or Monitor service requiring I/O, except input or output to the interactive terminal, will be considered as a sign that the program has shifted to an on-line compute mode.
3. When the time quantum for the conversational queue has been exceeded, the program is dropped to the on-line compute mode.

Needless to say, such ground rules for scheduling can and should be modifiable to handle special cases or for an installation's environmental requirements.

The UTS Monitor provides a Queue Status module which will be activated every time an interactive user program makes a service call to the Monitor or is trapped by the system. This module will determine, by a parameterized decision table, whether or not the user's queue status should be changed. This decision is taken before any monitor service is rendered.

System Scheduling

Above and beyond the scheduling control described above for user execution, the UTS system services programs on an implicit scheduling priority. This applies to all Monitor functions required by user programs. That is, services are rendered according to the following implicit priorities:

First Level

1. Real-Time foreground tasks (by interrupt level)
   a. Interrupt connected
   b. Resident or non-resident in core

2. Synchronous Foreground tasks
   a. Serviced by Monitor clock-watching function

Second Level

1. Interactive Programs - Conversational Mode
2. Interactive Programs - On-Line Compute Mode

Third Level

1. Batch Jobs (by explicit priority)

The second level, interactive processing, is broken down further into the following priorities for UTS servicing:

1. Executive Command Processing
2. User Program Ready for interactive output
3. User Program Ready to execute and in core (conversational mode)

4. User Program Ready to execute and in core (on-line compute mode)
5. User Program Ready to execute and not in core (conversational mode)
6. User Program Ready to execute and not in core (on-line compute mode)

Executive Command Processor

In the UTS environment, Executive Commands are processed in an interactive manner. (For Batch Jobs, the normal Control Card Interpreter is also used.) The UTS Executive will use the same communication input buffer that the user's program employs. When the user has explicitly or implicitly set the communication mode to Executive, the Executive Command Processor will then process the input message using its own echo table (System Generation).

Every Executive Command will be treated as a priority interactive input which must be processed immediately. It is not of higher priority than normal system I/O functions. For every Executive Command, full conversational features must be used. This means that a response to the user will be made immediately (< 3 seconds). The response may be a "WAIT" to indicate that it will take some time for the Executive Request to be serviced, such as loading a user program. If an Executive Command format is in error, a diagnostic specifically indicating the error type will be returned. Those Executive Commands which do not produce any particular output will respond with a simple acknowledgement showing that the request has been received and that the user MAY issue another Executive Command (or other input).

The Executive Command processor uses free formats and will minimize any unnecessary user input. To this end, an action routine to scan incoming characters and complete the printing of implicit portions of a Command input is used. This feature will be disabled if the user has informed the system that he is an "expert". In the "expert" mode, only the minimal characters necessary for explicit command definition will be recognized by the Executive Command processor. The user can leave the "expert" mode at any time by informing the system that he wants a "fill" for his inputs. The Executive Command processor can process input messages in both abbreviated or completed input form.

The Executive mode for inputs is explicitly entered anytime the ! sign detected upon input. Once in this mode, it remains there until an Executive Command which will start executing the user's program is given; the user mode will then automatically be entered. To explicitly enter the user mode, ~~the~~ another special (β) character will be used. This mode is illegal unless there is an executing user program and any messages sent in this mode are rejected accordingly.

Identification of Input Requests

Any conversational input will be logically terminated on the basis of a New Line character code. Acknowledgement of message receipt would be simply the echo back of a New Line code and the physical Carriage Return with Line Feed movement of the console printer. However, it is essential to identify the source of the process which is acknowledging the input and requesting further input. Thus, in addition to any output messages which may be generated, the UTS Executive Command processor will always start a new line followed by an ! to indicate to the user that Executive Command input is currently being expected and that the terminal is in the Executive input mode. This approach will also be used by any public service program, such as the basic Text Editor, Debug, Help, etc., where, not only is the fact that input requested, but what process is expecting it. This is particularly useful when dealing with process-to-process linking and it may not be clear to the user which process is in control.

Calling Service Systems

The Executive Command processor will recognize all requests for Executive action as defined in Appendix ( ). In addition, it will recognize, by name, the user's request for a system service program to be loaded and initiated.

Executable Command Files and Program Calls to the Executive Command Processor

The UTS system provides a flexible mechanism for allowing a user program to generate an executable file for system processing. The user program can set up, in symbolic text form, the same commands that can be typed into the Executive Command processor. An Executive Command file "reader" then scans the input text, executing each request sequentially until the file end, as if these were a set of control cards. The executable file may also be created by the ON-LINE user and given as a canned package or "cliche" to the Executive Command processor for execution. This facility permits complex operations to be "wrapped up" for repeated executions and it also allows program to generate a linked set of system processes for batch-mode operations.

Program-to-Program Intercommunication

The UTS system will permit programs to call for the execution of other programs. This mechanism will utilize the Load and Link facility whereby the program being called is treated as a subroutine. This means that a return to the calling program is anticipated at some time after the ~~upon~~ completion of the called program's operation. Communication between the two programs is accomplished by the registers and a communication file on secondary storage or in common dynamic storage (top). The calling program is assumed to be stopped while the called program is executing and may or may not be removed from physical core memory (swapped). There is no automatic copying of the calling program's context data for the called program; the called programs will initialize its own operating needs, based upon information transferred via the communication file or data area. There is no sharing of virtual memory other than the common dynamic storage area for intercommunication.

A program which is to be used in a linked manner must be organized with a special front end (convention) as opposed to the normal program structure, which will determine if it has been linked or loaded by the user. In the first case it must preserve the registers used for arguments and will control the exit of the program for returning to the calling program. Return will be accomplished by the UTS Monitor providing the identity and the next program location of the calling program at the time of linking. This information is given in the registers, along with the parameters for communication data (file on disc or in common core). If the check reveals that the program has been directly loaded by the user, input will be handled in the normal way and the normal exit (i.e., return to the Monitor) will take place.

This mode of program intercommunication will be used initially for UTS instead of the tasking service required for PL/1. Tasking may be implemented at some later time when the PL/1 language is supported under UTS.

Intervention and Error Recovery

User programs, which under the Batch system would normally be aborted because of an error, will merely be suspended (stopped) if the process is an interactive one. Control is given either to the user (console control, Executive Mode) or to a recovery routine specified by the controlling process or the object program itself.

If, during the execution of the user's program, an error occurs which is trapped by the system or the user intervenes with an Executive Command to stop execution and perform some Debug operation, an entry point for error recovery and for manual intervention must be made known to the UTS system prior to execution. The routines at these entry points must be given the current user program status and register environment preserved for subsequent processing. An error-type code will also be passed to the error-recovery routine.

Given the case of a control program or run-time and a user's program, such as the Debug service program and the user's object program, provision must be made for proper return to a control point upon user intervention or an error condition. If the user's program is being modified by the run-time control program (Debug), then it must be loaded as part of the control program and treated as data. They must share virtual memory.

Accounting

The Accounting function under UTS will take into consideration the fact that multiple users are using system resources concurrently. This means that for each user and job, the time and space utilized by a user will be specifically accounted for. A common process will be charged to a user only when actually being used. By definition, all elements in the user's map which are actually in core memory will be chargeable (except for the Monitor and, where necessary, a critical foreground process in all maps). All secondary storage being held for a user will be charged, including swap storage.

Wherever possible, Monitor service for a user program will be clocked on the user's time. That is, if a user program executes a service call to the Monitor, the time will continue to be charged to the user until he is no longer executing or being serviced.

In addition to those items accounted for under the Batch Processing Monitor, elapsed-time must be recorded for terminal usage. Core residency will not be recorded when the user's program or parts of his program are not required to be in core memory (i.e., they are eligible to be swapped out). Actual swapping costs are not chargeable to the user, except for overlay service.

The UTS accounting package will provide recording of all chargeable resources on a summary count basis. This data will optionally be dumped onto secondary storage, according to System Generation parameters. The reduction of such data is delegated to an optional, user-supplied process to produce an accounting report for administrative purposes. The on-line user will be able to retrieve current accounting data through a UTS information service command.

Start-Up and Shut-Down

The UTS system operation will utilize a start-up and shut-down procedure, for iterative use which maintains maximum operational continuity for on-line system users. The start-up and shut-down procedure will apply to a system which is going on or off the air completely or when scheduled interactive use begins and ends while other functions (e.g., batch and foreground) may still operate.

At start-up time, all local terminals will be turned on to receive a "system on" message, after which the terminals will be turned off until a user begins normal operations.

No other Executive actions will take place other than normal set-up for interactive (and other) requests; that is, communication processing and buffering for Executive requests will be provided. (User program communication buffers are not required until the user's process is loaded and active.)

At shutdown time, a system message to all active users will precede actually shut-down operations. No new users will be accepted into the system for the interval between the initial shut-down announcement and system shut-down. After the interval (System Generation, an operator-controlled parameter), all currently active users still executing will have their processes automatically saved-for-rest~~ou~~art or terminated (System Generation parameters or user option).

Communications Service (Low-Speed)

The UTS system includes a comprehensive and flexible communications package to service interactive operations. It is dependent around character-oriented, full-duplex hardware facilities which, via the software, provides message-oriented input and output functions for user programs and user terminals (ASR, KSR, Teletypes, SDS Keyboard Display device).

The UTS communications service consists of the following elements:

- Terminal Status Management
- Low-speed, character-oriented communications handlers (input, output)
- Inter-terminal service (Dial, Link)
- Dial-up service (from the computer)
- JOIN service for multi-user programs

Character-Oriented Communications Handler

The fundamental software module for UTS communications service are the Character-Oriented Communications Handlers. They perform the following functions:

Input

- The normal status of all UTS terminals will be Active (waiting for characters) unless the power is turned off or the station is disconnected (remote). EXECUTIVE MODE MESSAGES WILL ALWAYS BE ACCEPTED; USER MODE MESSAGES (TO THE USER'S PROGRAM) WILL ONLY BE ACCEPTED WHEN THE PROGRAM REQUESTS INPUT.

- On external interrupt, the handler examines a common input table for new characters which have arrived since the last inspection. The common input table is designed as a data-chained double buffer so that even if characters are not emptied from one half, the other half can be used for further inputs. It also causes character processing to be done at least at the critical point when one of the buffers is full, if not earlier. A pointer is maintained to indicate the word location where new, unprocessed characters begin.

- Each input word contains a single character and an identification number of the user's ~~channel~~ LINE. The handler checks an "echo" table associated with the given ~~channel~~ LINE and generates an output character or no output according to the echo table indication for the input character.

- A check, using the echo table, to determine if the input character is a control character is also made. Control characters either indicate a change to characters already received (backspace, cancel), a mode switch for the UTS Monitor (Executive Command, user program input), or an end-of-message signal. If a control character is found, the appropriate action routine is activated, i.e., move user buffer pointer, select user or system input flag, set message complete flag, etc.

- The handler makes a conversion from ASCII code to EBCDIC (unless the user's ~~channel~~ PROGRAM does not wish conversion) and places the character in the user's fixed length input buffer. Message assembly in the user's buffer continues until an end-of-message character is received or the buffer is full. If the user's buffer is full, a NAK character is generated to the console to lock out further inputs and the user program "input complete" flag is set. For end-of-message control characters, the NAK output is optionally generated for locking the keyboard as per the echo table indications.

- When the incoming character has been properly processed, the handler clears its interrupt level and exits.

- Paper-tape input is treated as keyboard input except that a rub-out code (DELETE) is recognized as a character to be ignored. Furthermore, the Teletype requirements for remote paper tape input are that start and stop paper tape reader functions (XON, XOFF) be provided. This permits proper computer control of paper-tape reading. The handler (in paper-tape mode) will generate a Stop Reader (XOFF) code in response to an end-of-message control character or buffer full condition and a Start Reader code when an Input Request is made (always).

Output

The communications handler for output accepts characters stored in a user program output buffer of fixed length for the given output channel. Characters are converted to ASCII code from EBCDIC before transmission by the handler. Illegal transmission characters such as EOT are detected and cause ~~the~~ a program ~~to be stopped~~ error. (Requests affecting communication lines must be explicitly made to the Executive for proper bookkeeping.)

The user program calls for output with a buffer full of characters in the program's virtual memory. As the handler takes control, the buffer is emptied into a Monitor buffer with conversion to ASCII. Characters are transmitted for output until an internal end-of-message is detected (DEL) or the record count has been reached, whichever comes first. The buffer will be released to the user program when empty, making it available for reloading. The user program will test the DCB to determine when the buffer is empty. In this manner, the user program may keep a steady output flow going at a maximum rate. It will be held up as long as the buffers are full.

Carriage Return Standard Convention for UTS

The standard physical message length for the UTS terminals will be a line of 72 characters, not including the NEW LINE code. The NEW LINE code will indicate a logical End-of-Message. A Carriage Return code will be echoed back ~~with~~ AS a New Line code and will be considered as a text character at all times. It will also be ~~connected~~ CONVERTED to a New Line code when detected in an output message text.

New Line codes will not be automatically generated by the Communications Output Handler at the end of a message. However, an output character count will be cumulatively made and if 72 spacing characters (graphics) have been transmitted without any New Line characters (or carriage return), a New Line code will be sent out if the 73rd character is not one.

Terminal Status Management

An important part of the communications package deals with maintaining proper bookkeeping of each user terminal. Each user software "channel" is activated when a connection is made by dial-up or when power is turned on at a terminal. A mode flag indicates whether the user is currently talking to the Executive or a user program. This flag is changed by an input control character (!) and ( ) or automatically by a GO Executive Command. User identification by name is kept for each channel for information purposes in using the interconsole communication services. Input or output activity status is maintained to avoid conflict between inter-console messages or Monitor messages and user program input or output. Inter-console link status and a conversational communication mode flag is maintained for each user channel. Whenever a User

THE USER'S ACTUAL LINE NUMBER IS INDEPENDENT OF HIS INTERNAL "CHANNEL, SUCH THAT IF A REMOTE USER IS ACCIDENTALLY IS DISCONNECTED, HE CAN BE RECONNECTED WHEN DIALS IN ON ANOTHER LINE (ROTARY). EACH USER TERMINAL IS IDENTIFIED BY TRANSMISSION LINE TYPE AND FORMAT GROUP WHEN A DIAL-UP CAPABILITY EXISTS IN THE UTS HARDWARE CONFIGURATION.

Termination Command is received (i.e., LOGOUT), the terminal will be disconnected or power turned off (EOT). Provision will be made for installation assigment of special consoles for message addressing, privilege functions, and/or priority service.

Inter-Console Service

It is ~~often~~ useful and ~~sometimes~~ OFTEN essential to provide direct communications between user terminals in an interactive time-sharing system. This capability is provided by the UTS Inter-terminal service and consists of two user functions and a program function. The first user function is a simple store-and-forward message-switching operation, DIAL, whereby a user may address any other CONNECTED terminal, expecially the operator's console for delivery of a text message. The second service provides linkage of two or more terminals to a single user program, where the program normally operates for only a single user terminal. This feature is extremely useful for training, demonstrations, group debugging, etc. The program function involves ~~JOINING~~ OF two or more terminals to it, under controlled circumstances.

For both of the first two services above, it is necessary to have the addressed terminal indicate acceptance of a message or linkage. Thus each such terminal must alert the system via an Executive Command (ENABLE) that it is ready for receipt of messages or linkage. If a terminal is not active, it will be turned on by the system (if a direct, internal communication line) for the duration of the output. Addressing of terminals is by ~~channel~~ LINE number; the system's user identification can be used to locate the proper ~~channel~~ LINE number by querying the UTS Executive as to the whereabouts or existance of an active user. No remote dial-up (if such hardware exists) will be performed for DIAL or LINK; remote users must be active in the system. If, during the course of a linked operation, a remote user disconnects, the linkage will be automaticall unlinked by the UTS system.

DIAL Service

A user may send a message of a fixed maximum length to any single console other than his own that is active and enabled (indicated acceptance of messages). Carriage Returns within the message may be used, since the New Line code is placed in the text instead and ~~a Line Feed is~~ echoed back. The end of the message text is indicated by a New Line character.

The receiver's terminal will not receive the message if it is in the process of input or output. At the first opportunity where either has just been completed, the message will be delivered, with the sender's channel number appended to it. If a terminal has not indicated message acceptance, a DIAL command will be rejected.

A special case of the DIAL function is available to the Computer Operator's console whereby he may dial all active terminals for public notices of importance.

The operator's console is addressed by DIAL 0 (OH) and his terminal may be assigned to any terminal during system operation.

Terminal Linking

Linking of several terminals is initiated by a logged-in user, whether or not a program is loaded or operating or not (on that terminal); linking can be initiated at any time. Any Active linked terminal can cause a program to be loaded, executed, stopped, or terminated. Linked consoles can be made Passive or Active by the link originator; that is they may either be received only for any program output or they may input and receive output as well. All terminals may always issue Executive Commands. While any Active linked terminal is inputting in the User Mode (to the program), all other Active terminals cannot input to the program. This is necessary to avoid confused input to the program input buffer. All inputs from every linked terminal are seen by all the others, as well as all outputs to any terminal.

As any terminal links a terminal to his (or vice versa) a message will be issued by the UTS system indicating who is linked to whom and the type of linkage on both terminals. A terminal or group of terminals may be linked to only one active user program at a time.

~~A linked terminal may Unlink any of the other unlinked terminals from itself.~~ Any linked terminal may unlink by giving the UNLINK command to the UTS system at which point a message to the remaining linked terminals and the unlinked one will confirm the Executive request. It should be noted that each linked terminal is not independently in the Executive or User Mode according to the mode switch last used by a terminal; the current mode is applicable to all linked terminals no matter which one set it.

All linked terminals can conveniently converse directly without using the DIAL command. All that is necessary is that each line of input be terminated with a Cancel character (CAN) which will be echoed with a New Line.

JOIN Function

The JOIN function is exercised by an operating user program for a group operation, via input from the Originator console. The Originator terminal, which activated the program, provides the program with the terminal ~~channel~~ LINE numbers which are to be used in the Monitor service calls.

Active or Inactive terminals (except Inactive Remote stations which would have to be dialed-up) can be joined. Active stations must be enabled for the JOIN function and cannot have any program currently in operation. Inactive stations may always be joined and the JOIN function starts up those terminals.

Only the originator terminal may Join or Unjoin other terminals, or issue any Executive Commands which may affect the common program. The only Executive Commands which a Joined terminal may exercise are the DIAL and QUIT commands. Once a terminal has Quit or been Unjoined, that terminal reverts to normal usage. When the Originator terminal terminates the ~~C~~ommon program, all Joined terminals are released and turned off.

Joined terminals cannot link or be linked to. Inputs and outputs to and from each Joined terminal are not seen by the other terminals; each has its own communication buffers in the common program. Each terminal thus must be specifically addressed by the program. The originating terminal is the only one implicitely addressed, i.e., any input or output calls not naming the terminal (channel number) will refer to the Originator terminal.

Checkpoint and Save for Restart

User programs which are in operation may be stopped and saved on secondary storage for resumed operation in the future. This facility is operable in two modes, Checkpoint and Save-for-Restart.

Checkpoint

Checkpoint is a mechanism for a long computational program (Batch Mode) to periodically snapshot all elements of its current operation so that the program need not be completely reinitiated because of a system hardware or software failure. Every snapshot will include core memory contents and secondary storage files as well as the program's environmental context data (registers, DCB's, etc.). The conditions for checkpointing can be based on time or at logical phases of the program's operation. Successive snapshots of a program's operation will always replace the one last saved.

The checkpointed program will be contained in a permanent file, identified by the user's account number and a unique user-supplied file name (not the name of the original program, which has its own file name). If the checkpointed program is to be activated, it is run as a new job. Furthermore, unless explicitly preserved, the checkpoint file is automatically deleted from permanent storage when it is executed.

An operating program can call for checkpointing itself directly by an Executive service call or it can have a run-time routine which will respond to periodic time interrupts and call upon the checkpoint service.

At the time of checkpointing, any on-going I/O operations will be completed and all I/O status information preserved in the program's context area. In restarting a checkpointed program, a restart routine location must be provided which will reinitiate all environmental conditions (such as open files, magnetic tape positioning, etc.). It is assumed that the self-checkpointing program maintains the proper restart information for itself at all times. The UTS Monitor will provide the proper communication to the Restart Module at load-time to activate the re-initiation information.

Save-for-Restart

A program, which is stopped externally for future continued operation, will be saved for restart. In effect, it is a dump of the program and its operating environment. A save-for-restart may be generated by an operator [illegible] or by a high-priority real-time program. It is assumed that the program will be resumed exactly as saved; ~~therefore, no external I/O peripherals can be involved which require prepositioning, i.e., magnetic tape, unless these devices are maintained as in initial position. This will be [illegible] from periodic preemption.~~

A save-for-restart pre-emption by a high-priority program in the UTS system is accomplished by simple swapping to secondary swap storage. Resumption of the program's operation is automatically caused by the exit of the priority program. Files are not closed for the saved program, since continuation of execution will take place relatively quickly. (I/O transactions will be allowed to complete, if they are in process.)

On the other hand, a program which is manually saved-for-restart, will not necessarily be expected to be continued for a while. Shut-down procedures will be necessary to preserve the environment indefinitely. This means that open files must be closed, current I/O transactions completed, and scratch storage saved, not released. A unique identification must be provided for future restart, since it will not be a fresh copy of the program. The restart procedure will require the UTS Executive to recognize that it is a restart version in the user's account file and reopens all data files and DCB's that were active at the time of shutdown. Program execution will resume exactly from the point of shut-down or at the location supplied with the execution command from the user (GO). There will be a maximum limit established at System Generation time for, user save-for-restart file storage, since such operations can be over-utilized without proper purging.

System Generation Processor and System Modification

Since time-sharing systems lend themselves to being highly variable in terms of configuration sizes, number of users, types of user processes, response needs, etc., it becomes critical to a user installation to be able to specify particular operating needs for its specific system configuration. The mechanisms for this capability are provided primarily through System Generation parameters. For those parameters which do not involve structural changes in the system (i.e., space allocation), additional changes in parameter values are made dynamically from the Operator Console.

The System Generation processor will operate in the minimal system configuration, taking advantage of auxiliary storage. Where more core memory is available, time benefits are derived from the additional SPACE, ~~system resources~~.

All changes to a UTS system, which can normally be expected to occur on a day-to-day basis, will not only be changeable via the System Generation processor, but through UTS service routines. Such routines operate under the UTS Monitor and enable the system to continue normal operations as well as avoiding time consumption for a complete System Generation process. The following system modifications are provided under both the System Generation processor and the UTS Monitor:

- Subroutine Library maintenance
- New (additions, deletions, replacements) Public Service Programs
- Scheduling parameter changes
    - Quantum values for user queues
    - Number of interactive users in system (MAXIMUM)
    - Response-time duty cycles
    - Batch job time allocation
    - Interactive status change conditions (for various Monitor service requests)
- Foreground identification and resource allocation
- Peripheral device availability
- Auxiliary storage allocation changes
- Limits on user resource allocations
- User priorities
- Terminal directory changes (e.g., system operator's station number)

Changes which are made only at System Generation time include:

- Total system hardware configuration
- UTS system core organization and overlay structure
- Public Subroutine space assignments
- Scheduling queue structures (number of queues, decision rules)
- Executive commands
- Accounting items
- UTS service extensions

IV.

UTS System Configuration Considerations

The UTS system will require sufficient core residency for Monitor elements to execute in order to minimize time overhead for interactive and foreground operations. This means that any UTS service required by an executing interactive (conversational mode) or foreground process must be core resident and should not be overlayed. This applies also to any bookkeeping tables. Administrative functions for initiating or terminating a process can be non-resident. Likewise, service functions which may be required in the on-line compute mode or Batch mode are candidates for overlay.

User programs which are swapped will also be handled to minimize response time. That is, user program areas (e.g., context data), which are necessary for on-going Monitor's processing of users, will be the last elements to be swapped or never swapped (e.g., communication buffers, parts of context area).

The minimal amount of core space required for an efficient UTS system (Interactive plus Batch) will be that space for resident monitor or processes and data, monitor overlay area(s), user resident areas (dynamically proportional to the number of active users currently on the system), and sufficient available space for submerging interactive swapping overhead. The latter factor is dependent upon the typical conversational process and/or context size which must be swapped, plus the typical Batch job size (in core at one time) and any foreground residency. The size and number of such spaces will be determined by actual interactive process sizes and execution times. This will be the conversational FORTRAN plus the Text Editor facility. The minimal core memory for a reasonable UTS system will be 48-65K; however, until the specific sizes and times for interactive processes are known, capacity and performance figures cannot be established. The average figures should be under 8K per user conversational context (not including a common reentrant process) and 50-100 μseconds for the conversational process time.

Auxiliary storage for a minimal UTS system will be based upon UTS system storage and the number of active users accommodated concurrently (which is a function of core space and system efficiency). This latter number dictates the amount of:

- Swap storage } fast RAD
- Scratch storage } fast RAD
- Permanent files } High-capacity disc
- Library files } High-capacity disc

Magnetic tapes are required for system generat~~ing~~ion, system entry of user inputs (programs, data) on to the discs, disc purging and spill, direct user access (where necessary for user processes). The minimum requirement at present appears to be at least two magnetic tapes (9-channel).

Other hardware requirements are:

2 Fast Memory Blocks

2 Clocks

Communications Interface (Character-oriented) including interrupts

1 Multiplexor IOP

2 Selector IOP

Memory Protect (Lock and Key)

MAP

Decimal and Floating Point options

Card Reader

Typewriter

Printer

At the present time the memory module organization and number of ports cannot be specified. However, consideration must be given to swapping activity and CPU interference.

Performance

The performance characteristics for the UTS system are:

1. Monitor overhead for disc and core storage management, scheduling, accounting, context switching, I/O control, and Executive command processing will not exceed 5% of the total operating time (CPU) of the system. This includes overlay of Executive functions where necessary, but does not include user-requested I/O transfer time.

2. Interactive communications processing (character-oriented) will not exceed 5% of the total operating time of the system with one CPU.

3. Swapping time will be masked by having enough "user spaces" in core memory such that one (or more) users, executing their full conversational quantun, will equal the time to make a swap of two other conversational processes. The "user space" size is the size of a typical conversational process and/or user context data.

4. The core residency for the UTS Monitor will not exceed 20% of the total minimum core space configuration. This does not include user context storage.

5. A conversational process is an interactive process which will have a maximum response time of 3 seconds from the time of receipt of an end-of-message character to the start of the response output. This response time is not guaranteed for a user interactive process which requires any I/O service other than console I/O and normal system swapping (one swap). The 3-second time interval is a system parameter and may be changed by the customer installation. It is only valid for a maximum number of users in a given configuration, depending upon whether a worst case or a probablistic loading environment is anticipated.

6. Performance will be based on SDS standard configurations and typical user software sizes. Any variations in configurations or user software behavior (e.g., sizes, scheduling, etc.) will affect performance figures and are the responsibility of the customer's installation.

Debug Service

The UTS system provides an interactive Debug service program for use with Assembly language user programs. When loading a program for debugging, the user's map will share virtual memory between the object program and the Debug program. Both will be loaded into core memory for execution.

The Debug program will provide the following services:

- Dump memory locations in hexadecimal, octal, decimal, instruction mnemonics, and symbolically (global symbols).
- Locations will be referenced relatively to global symbols.
- New symbols may be defined for the object program.
- Location contents can be altered.
- Program insertions can be made.
- Conditional selective snapshot dumps can be inserted and removed.
- Conditional, selective trace of program areas.
- Start execution at any program location.
- Dump or trace program environment data, i.e., registers, temp stack.
- Modify program environment data.
- Search program or data areas ,for masked values
- Save a modified binary program

The interactive Debug program will utilize both the Teletype Keyboard/Printer or the Keyboard/Display devices. It will operate under the UTS Monitor service for any privileged functions. These include:

- Error recovery control
- User interrupt control
- I/O service
- Change of protection for user program areas
- File changes (replacing or saving a binary program file)

Modifications to a user program or patching the program for Debug calls requires that the program protection (map) be set to ALL ACCESS.

Provision must be made for Master mode access to be made, for checkout out Executive or foreground functions. Such access must be guarded so that only system programmers can use this facility at proper times (e.g., Operator control over when such access is permitted, "privileged" consoles, passwords, etc.).

Editor Service

The UTS system will provide a basic, general text editor which will operate both in a batch mode and conversationally. It is usable to create text files and will link to language processors for conversational programming. It will be reentrant for multiple users.

The text editor will provide the following services:

- Initialization of a new symbolic file (via File Management service).
- Modifying an existing text file (insertion, deletion, replacement, addition).
- Line number referencing.
- Line number insertion (using decimal system). (Line numbers will be provided by user)
- Automatic resequencing (removing decimal point values) upon user request.
- List with or without line numbers (line numbers will not be considered necessarily as part of the text.
- Rename a current symbolic file.
- Merge symbolic files into one file.
- Tab input lines by filling with spaces according to software tab settings.
- Provide in-line editing (character editing).

The Text Editor will be called by an Executive Command. Further commands are processed by the Editor itself.

When operating in the Batch mode, inputs are card images. No interactive response is given for any input request. If any error occurs with the input, the error message is logged and the process will abort.

Editing and the SDS Keyboard/Display

The SDS Keyboard/Display device has implicit requirements for editing service above and beyond normal terminal input and output. The first level handler functions for K/D device are based upon the standard Sigma character-oriented communications equipment with several differences.

- A different "echo" table is required, since new control characters are used.
- Message-mode capability of the K/D requires identification of a "message" (leading ETX character), and no echo of the message characters. (Dynamic switching of echo table pointer.)
- Output message generation requires control character buffering with NULL characters for timing purposes.

The above functions require that the K/D devices be serviced by a superset routine which has as a base the normal communication software. Identification of a K/D "channel" must be made dynamically, since a remote station could be either a Teletype or K/D.

A second-level K/D processor will be provided as a system routine for servicing mechanical editing functions. This routine will specifically provide for page-turning and line-rolling of a user program-supplied text buffer. Other text editing functions which change text content, such as insertion, deletions, addition, etc., must be handled directly by the user process such as the UTS Text Editor. The user process will also be required to maintain the buffer for text page-turning or line rolling when the terminal's request requires a new buffer full (end of buffer reached, top or bottom).

Insert A

User File Types

In the UTS environment, user files will be of several types, which can contain different forms of the same entity. Specifically, the following types of files can exist for a user program:

1. User-generated source symbolic code
2. Processor (FORTRAN) - generated symbolic code
3. Binary object code (load module)
(4. Save-for-restart executable code)

Clearly, these file types can co-exist and must be identifiable. The user cannot be expected to create unique special names for each of these forms of a single program. Provision must be made in the File Management facility to accommodate a file type identifier. This identifier can be explicitly employed directly by a user or implicitly through a processor. For example, if the user wishes his program to be loaded for execution, it must be the binary object file. If it is to be compiled, it must be in the processor-acceptable form. If editing is required, the user-generated form is called for.

When files are modified or deleted, again there must be some control for assuring across-the-board coordination. For example, when a user-generated source file is updated, the processor-formatted source file must likewise be changed.

a —A—

Old. ART Rosenberg
UTS Spec — maybe
early 1967

Foreground Servicing

The UTS Monitor will provide the same services to foreground tasks as specified for the Batch Processing Monitor. However, because the map hardware is available, certain factors must be considered.

Foreground tasks, both resident and non-resident, can be efficiently operated in the non-mapped mode. Non-resident foreground tasks are assumed to be "tolerant" in terms of response time and can afford to wait for physical core space to be made available and to be relocated to that space for execution. If a non-resident task is not tolerant, it should be made resident. Furthermore, once a non-resident task starts executing, it should not be suspended (swapped out) and then resumed.

Unless a foreground task requires the common use of public reentrant subroutines, the map mode should not be used for foreground tasks. Private copies of any required subroutines will be provided by the loader. If common reentrant subroutines are to be used, then the map mode must be employed. In this case, it will be an appropriate real-time service to dedicate virtual memory space in all maps, so that map-switching is eliminated (except for FORTRAN subroutine local storage).

Foreground tasks can preempt core storage of all Background (Batch) and Interactive users, except for I/O buffers in use. However, other foreground tasks (of lower priority) will not be moved out of core memory unless they are mapped. A foreground task, operating in the Master mode, can abort all I/O service to other users and take over the machine, but it is normally an indication of system overload when foreground tasks cannot complete their executions.

Every foreground task connected to an interrupt will be responsible for preserving and restoring the current operational environment (registers, program status doubleword). This will be done directly, if the foreground task includes its own interrupt routine, and by a centralized (MONITOR) interrupt handler when the foreground process is serviced by the system. The latter approach is required for non-resident foreground tasks.

A resident foreground task must be activated by an interrupt within 100 microseconds after the occurrence of the interrupt. This means that a foreground task will not be delayed by any UTS Monitor overhead or current service function for more than that amount of time. After the foreground task starts execution, it may be interrupted by Monitor Service functions (i.e., I/O interrupts) which will only execute minimal bookkeeping tasks before returning control to the foreground task. If the foreground task cannot tolerate any interruption from the Monitor, it must be at a higher interrupt level than the I/O and communications. In such cases, the amount of time required FOR the foreground task (frequency of interrupt within a response duty cycle, process time for the task) must be considered as additional system overhead to be subtracted from the conversational duty cycle, and will impose additional limits ~~to~~ ON the number of interactive users accommodated by the system.

If a foreground task uses any monitor service, it must do so by a normal request (CAL). The priority of its interrupt level will be recognized by the monitor for queuing purposes. Unless the foreground task interrupt has been ~~permitted~~ ALLOWED to interfere with monitor bookkeeping operations by not permitting a safety disabling (minimal) of all interrupts of the monitor, monitor service can always be requested at any time via the normal CAL.

All assignable system resources will be dedicated to a foreground task when it is active, resident or not, except for core memory when it is not a resident foreground. This will be done at System Generation time or when the foreground task is initiated. It will be an Operator function to abort OR checkpoint any lower priority job in order to satisfy the resource allocation needs of a newly-activated foreground task; this does not apply to core memory.

Foreground tasks will not be normally timed out for execution, but will run until complete or waiting for I/O. As a special service for debugging parts of a foreground operation, a quantum time setting may be requested such that if the task has not returned to the monitor within a given period of time, it will be suspended as an error condition. The foreground tasks associated interrupt will be disarmed. It will be required that such a foreground task be in the Slave mode. (at least that part which is being checked out,) and that the monitor be called prior to execution IN ORDER to set the timer. In effect, it will be treated as a user program with additional privileges of dedicated resources.

Every foreground task is considered to be an independent job with its own context requirements, temp stack, etc. For economic reasons, it is appropriate to organize resident foreground tasks to appear to the monitor as one job sharing a common temp stack. This is possible because of the first-in, last-out order of absolute priority. Unless so organized, independent foreground tasks will require independent job-associated storage.

In the UTS (and BPM) environment, there exists an accounting problem where foreground tasks cannot be properly charged for CPU time. This occurs when such tasks service their own interrupts and do not allow for proper "stopping" of the clock for the interrupted job or user. If this time is significant and known (by being static), it may be accounted for administratively on a percentage basis, i.e., every user job will have a percentage of its compute time deducted because of foreground interference. Alternatively, if foreground tasks are aperiodically initiated, they should be required to clock themselves (or allow the monitor to do so) upon entry and exit. This differential can then be used to adjust the interrupted job's accounting information.