# What's different about tactical executive systems

*by* WILLIAM G. PHILLIPS

*Radio Corporation of America*
Moorestown, New Jersey

The program for a computerized command-and-control system is generally a combination of critically time-constrained real-time tasks, which directly control the tactical mission environment, and non-real time tasks, which support the system. This computer program structure is the basis for determining the allocation of the total available processing time for a complete mission cycle.

## TIMING ALLOCATION

Since tactical command-and-control systems (Figure 1) are triggered by a series of predictable and non-predictable events, the computer-program task allocations must be designed for complete flexibility within the total available processing time period. In the case of predictable event triggers, the design may be simple to the extent of repetitive processing of a single chain of tasks, called a "thread." In the case of unpredictable event triggers, such as special-threat target detections, the design must be more complex to the point of interleaving threads. The process of interleaving threads presents an interesting timing problem within this type of system because of the requirement that a real-time thread must complete its processing in a predefined critical time period. This time period is frequently a function of the design requirements of the interfacing tactical equipment.
equipment.

Figure 2 illustrates a processing sequence where the triggering events are predictably separated and therefore the thread allocations ($P_i$) and their respective critical time periods ($Q_i$) are predictably separated. The slots which occur between threads ($R_i$) are available for processing of non-real time tasks. The real-time tasks ($q_{ij}$), as individual items, must all satisfy their individual time allocations within their respective critical time thread period $Q_i$. This timing constraint can be represented by the following inequality:

$$Q_i > \sum t(q_{ij}) \tag{1}$$

where $t(q_{ij})$ is the time allocation associated with task $q_{ij}$.

If the non-real-time tasks are also time constrained to a fixed completion period ($T_p$), then the general equation for timing allocation within a complete processing sequence ($T_p$) is:

$$T_p > \sum_i (P_i + R_i) \tag{2}$$

which describes the inequality to be satisfied by the combination of real-time and non-real time tasks over the total available processing period, $T_p$. This period represents a complete cycle of tactical events, such as radar-track processing, weapons assignment and firing, and special-threat target processing. The critical time-thread period ($Q_i$) of Ineq. 1 represents intervals of tactical events, such as radar-target detection, weapons designation, and special-threat target-assignment processing.

This time allocation can also be easily applied to non-tactical systems, which frequently allocate a range of time ($Q_i$), in equal quantums, to a group of application tasks. Any unused time ($R_i$) between the actual completion of a quantum period cycle, i.e. all process state tasks have completed their respective quantum period of execution ($P_i$), and the beginning of period $Q_i+1$ is allocated to system background processing such as on-line fault analysis or some accounting procedures.

The major difference between the two types of systems is the criticality of satisfying $Q_i$ in inequality (1). Non-tactical systems most frequently are responsible for the scheduling and processing of a group of non-related homogeneous tasks, which are not critically dependent upon when they initiate or complete processing. That is, the tasks will not have failed their intended purpose if they complete processing two or three seconds later than if they had been run in a "batch" environment. This philosophy can also be applied to some real-time systems, such as a communications network which, while a two or three second delay would postpone the completion or initiation of a call, it would not cause the system to fail its intended "mission" of initiating and completing phone calls in sufficient time to be compatible with human reaction speed. Tactical systems, on the other hand, are constrained in time by high speed device interface requirements which frequently must be satisfied within tolerances no greater than a few milliseconds. Any perturbation to tactical task scheduling could cause these tolerances to be violated, thereby possibly causing the

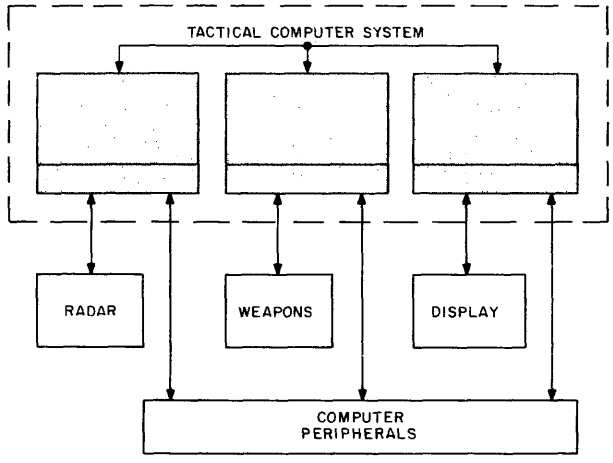



Figure 1—Command and control computer program complex

intended mission to degrade or fail. The degree of impact due to mission failure (criticality) is far greater in a tactical system primarily because of the direct relationship between mission success and human lives.

An added perturbation to the system-timing allocation is the introduction of the executive program tasks that support the scheduling and dispatching of the system tasks. This additional allocation can be represented by expanding Ineq. 2:

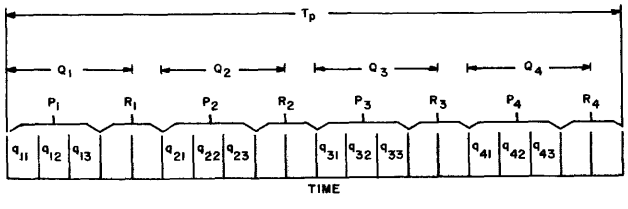
$$T_p > \sum_{ij}[t(q_{ij})+t(e_{ij})]+\sum_{ik}[t(r_{ik})+t(e_{ik})] \quad (3)$$

where $t(e_{ij,ik})$ is the time allocation associated with task $e_{if}$ or $e_{ik}$ and $t(r_{ik})$ is the time allocation associated with task $r_{ik}$.

This general inequality must be satisfied for timing allocation over the period $T_p$; it includes the real-time task times, $t(q_{ij})$; the non-real time task times, $t(r_{ik})$; and the executive task times, $t(e_{ij})$ and $t(e_{ik})$. We can also expand Ineq. 1 for the critical-thread periods ($Q_i$) to include the executive tasks:

$$Q_i \sum_{ij} t(q_{ij})+t(e_{ij}) \quad (4)$$

It is apparent from Inequalities 3 and 4 that there are many unknowns: in addition to ascertaining the process-




$T_p$ = Total available processing time for a mission cycle

$Q_i$ = Allocation of critical time period within which real-time threat $P_i$ must be completed

$P_i$ = Processing thread i, consisting of a processing sequence of real-time tasks, $q_{ij}$.

$R_i$ = Processing thread i, consisting of a processing sequence of non-real time background tasks, $r_{ij}$.

Figure 2—Processing time allocation non-interleaved

ing-time allocations for the real-time and non-real-time tasks, we must also determine the time expenditures of the executive tasks. This is further complicated by the fact that executive-task durations may vary because of the varying types of services to be performed (such as I/O scheduling), the type of real-time task scheduling (i.e., immediate with or without messages, time delayed, etc.), and the scheduling queue backlogs. Until all of these unknowns are determined, or at least closely predicted. Inequalities 3 and 4 cannot be credibly satisfied.

A practical solution to this problem is to arbitrarily allocate a budget of a fixed percentage of the total available processing time ($T_p$) to the executive tasks $\Sigma t(e_{ij})$ and $\Sigma t(e_{ik})$. A more precise procedure is to further allocate a fixed percentage of the available critical-thread period $Q_i$ to the executive tasks $\Sigma t(e_{ij})$. A typical allocation, at the beginning of system development, is 10 to 15 percent for both of these periods. As the development progresses and the task timings become more defined, the terms of the inequalities must be adjusted. This, in fact, is an excellent method of ensuring that the task design is meeting its timing allocations; for if the inequalities fail to be satisfied, the system integrity is compromised, and the system must be redesigned.

Real-time command and control systems differ in complexity. A simple system, with predictably sequenced trigger events, has its real-time-thread critical periods ($Q_i$) allocated somewhat as in Figure 2, with the required condition that the following inequality be satisfied:

$$T_p > \sum_i Q_i \quad (5)$$

However, some systems are more complex because of unpredictable sequences of trigger events. These types of systems frequently require that real-time threads overlap each other, but that each thread must still complete processing in its allocated critical period, as illustrated in Figure 3. This figure shows the critical real-time periods, $Q_i$, overlapping the non-real time tasks, $R_i$, naturally being delayed until the completion of all real-time threads. This allocation permits us to then concentrate on the critical real-time periods, $Q_i$, and to process the low level, $R_i$, tasks, in a background mode, if and when time is available during $T_p$. This overlapping (interleaving) process is described by the following inequalities:

$$\sum_i Q_i > T_p \quad (6)$$

$$Q_i > \sum_j t(q_{ij})+e_i+\sum t(q) \quad (7)$$

where $e_t$ represents the total fixed executive overhead time allocated for the period, $Q_i$, and $t(q)$ represents the tasks interleaved in $Q_i$.

This type of complex system requires that the executive program, through a scheduling mechanism, manage the interleaving process to ensure that inequalities 3 and 7 remain satisfied during task processing. To accomplish this, the executive-program scheduling mechanism must be designed to manage a dynamic queue based on task

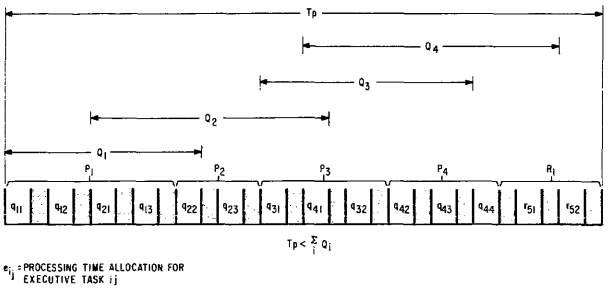



Figure 3—Processing time allocation—interleaved

priorities and to resolve any timing conflicts between competing tasks across threads, as described by inequality 7.

It is not unusual to encounter system requirements that dictate that the first task within a thread, $q_{it_1}$ (Figure 3)—and therefore the thread itself—shall be repetitively triggered at some frequency relative to the occurrence of an event; that is, the thread initiates a processing sequence, $P_i$, (Figure 3) repeatedly at some frequency after some initial event trigger. This may occur for three general reasons in a command and control system:

(1) Periodic interface requirements with time-pulsed radars or other similar equipment.
(2) Periodic interface requirements with display consoles, which require refreshed data.
(3) Periodic polling of interfacing equipments for input messages.

In the case where a thread is scheduled in constant intervals, relative to a single event (i.e., the triggering event always occurs at the same time within each $T_p$ interval), the thread timing allocations in Figure 3 are identical for each succeeding $T_p$ interval. However, in complex systems, the possibility exists that some periodic threads will be scheduled some constant frequency after, or possibly before, the occurrence of an unpredictable event. This case will then cause the thread critical allocation times, $Q_i$, to drift from one $T_p$ period to another, as illustrated in Figure 4. This drifting would also occur for those cases where a thread was scheduled with a variable frequency.

It is important to understand, at this point, that tactical tasks are not amenable to a multiprogramming scheduling technique because of their homogeneous functional properties. Unlike a commercial data center environment, where each task in the processing queue is completely heterogeneous and consequently is not dependent on the processing state of any other task; the tactical system tasks, within a thread, are dependent upon their predecessor/s to supply both data and initiation triggers. This dependency is required primarily because tactical tasks frequently interface with equipments which require time tagged data from other equipments. For example, it is unrealistic to execute a task which supplies data to a display console, prior to the completion of a predecessor task whose function was to pre-process the data from a radar buffer.

Let us now summarize the four common types of critical real-time tasks:

(1) The dynamic task that must be scheduled strictly according to a priority sequence.
(2) A periodic task that must be scheduled repetitively at a fixed frequency relative to a predictable event occurrence.
(3) A periodic task that must be scheduled repetitively at a fixed frequency relative to an unpredictable event occurrence.
(4) A periodic task that is scheduled repetitively at a variable frequency relative to a predictable event occurrence.

Since a mixture of these type tasks may be required to complete processing within the same critical-thread period and since each task will perform a unique tactical function, a priority scheduling philosophy must be developed, which will ensure the hierarchy of tasks in relation to one another. This is especially true in the case where a periodic task, and possibly its associated thread, is unpredictably triggered while a lower relative priority task is processing. Inequality 7 represents the total time allocated to a complete mix of real-time tasks over a critical processing period, $Q_i$, assuming, of course, that random-interrupt processing is included in the appropriate allocation; and therefore is the principal timing requirement to be satisfied by the design of the executive scheduling mechanism.

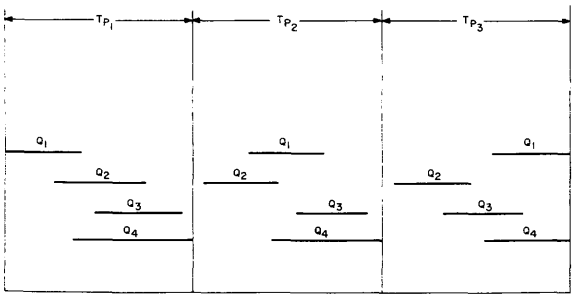



Figure 4—Drifting critical time periods

## EXECUTIVE DESIGN APPROACH

The executive program, to satisfy the above timing allocations, must provide efficient mechanisms for performing the following functions:

*Scheduling* critical real-time tasks according to a dynamically changing priority-sensitive environment.
*Interleaving* processing threads.
*Monitoring* the processing of all tasks and threads to ensure critical time periods and total available processing time periods are not violated.

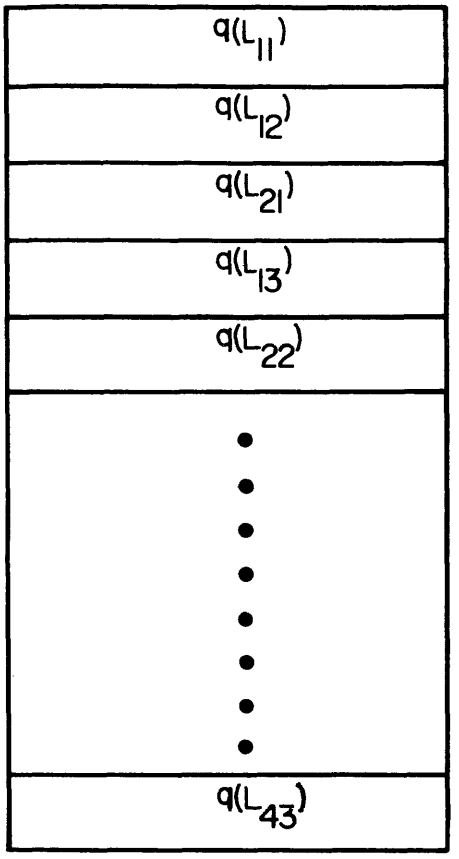



Figure 5—Single-level queuing model

If a unique task program is equivalenced to each timing allocation, $q_{ij}$, in Figure 3, we can state the following general priority characteristics of threaded tasks in this type of command and control system:

$$q(L_{i1}) \geq q(L_{i2}) \geq \cdots \geq q(L_{in}) \tag{8}$$

where $q(L_{ij})$ is the priority of task $q_{ij}$, and

$$P(L_i) = q(L_{i1}) \tag{9}$$

where $P(L_i)$ is the priority of thread $P_i$.

Inequality 8 shows that tasks are always structured within their respective threads in descending priority order, independently dynamic. This priority structure differs considerably from most non-tactical systems, which contain tasks of different priorities within a single thread and the execution of any specific task is a function of both priority and associated I/O states.

The reason for the difference is, again, because of the heterogeneous characteristics of tactical tasks versus the homogeneous characteristics of most non-tactical systems.

As established in Eq. 9, the priority of thread $P_i$ is dictated by the priority of its first task, $q_{i1}$. These characteristics show that the only dynamically changing priorities in the system are those associated with the "lead" task of each thread; thus, a simple queuing model can be structured to satisfy this scheduling requirement. Figure 5 illustrates a standard queue structure in which the tasks, $q_{ij}$ (Figure 3), are randomly dispersed and are serviced by the executive according to their respective priorities. This queuing model will satisfy the task-scheduling requirements of our system, but will not provide the executive with an adequate mechanism to monitor the thread critical time periods for possible overrun conditions.

This dynamic process of time budget management is probably the greatest single difference between tactical and non-tactical computer systems. The tactical executive design must contain the capability to compensate automatically for as many perturbations to the processing norm as possible, while maintaining each critical thread period, whereas the typical non-tactical executive design logic usually will rely on an external operator to restore system integrity. The process of automatic time budget management contributes greatly to the sophistication of tactical executives, especially in the area of dynamic task queue maintenance.

An approach to provide an executive time-monitoring capability is to structure the basic queue with time bounds which correspond to the *critical thread periods*, $Q_i$, illustrated in Figure 3. The priorities of these time bounds could then be established according to the priorities of the threads they represent, $P_i$ (Figure 3). All of the tasks associated with a thread, and consequently a thread critical period, would then be contained within the corresponding thread priority level in the queue, as shown in Figure 6. This structuring is possible because of the characteristics described by Eqs. 8 and 9. If we now associate an overrun time parameter with each thread level and with each task, it is possible to predict the probability of

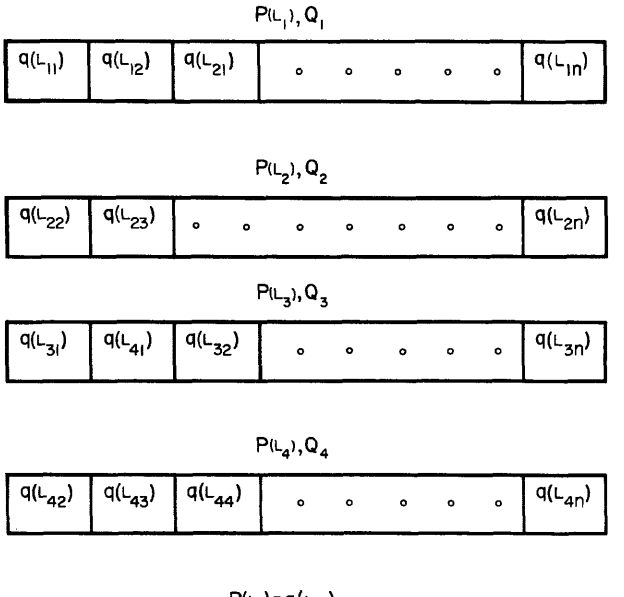



Figure 6—Multiple-level queue structure

achieving the required critical period constraint of $Q_i$ (Eq. 7) and $T_p$ (Eq. 3). If an overrun occurs at the thread level —i.e., $Q_i$ is not satisfied—the only recourse is to transfer to some error-processing state. However, it is simple to predict the varying probability of achieving $Q_i$ by carefully monitoring each intra-thread task for an overrun condition. If a timing problem should arise, the executive program has the capability to temporarily suspend the interleaved $Q_i+_j$ tasks (Figure 6), which are processing within the $P(L_i)$ thread priority level, in favor of keeping the $P(L_i)$ tasks within their time constraint, $Q_i$. This is a simple process whereby the tasks, which are interleaved, are simply moved to their normal thread priority level, thereby allocating the entire critical period, $Q_i$, to $q_i$ tasks only.

For example, task $q_{41}$ (Figure 6) would be moved out of the thread critical period, $Q_3$, and into the thread critical period, $Q_4$, if task $q_3$, was in a time overrun condition, which jeopardized the completion of $P(L_3)$ tasks within the thread critical period, $Q_3$. This methodology is possible because of the common priority structure of these type systems, as described in the following inequality:

$$q(L_i) \geq q(L_i+j) \tag{10}$$

for a thread critical period, $Q_i$.

Inequality 10 states that interleaved tasks ($q_{i+j}$) have a priority less than or equal to non-interleaved tasks, ($q_i$), within the same thread critical time period ($Q_i$). This is likely to be the case, except in the rare instances where a high priority task may be dynamically interleaved into a thread period, in which case the high priority task would be processed in priority order within the thread and the lower priority non-interleaved tasks could overrun their critical thread period. This requires a tradeoff on the part of the system analyst/designer of the tactical priority structure to determine whether it is more important to satisfy a critical thread period or immediately process a high priority task.

Under certain circumstances, a complete thread of tasks may require immediate processing because of the arrival of some unpredictable high priority event. If the priority of this event is higher than the thread priority level of the currently processing task, the executive program will initiate a special suspension process called "preemption". This preemption process is not unlike a multiprogrammed non-tactical system's interrupt logic, except that tactical system preemption takes place at the thread level (i.e., an entire group of tasks is interrupted), while most non-tactical systems interrupt at the single task level. This thread level preemption evolves from the functional properties associated with a tactical thread. For example, if a currently processing thread's primary function was to load a launcher and fire a missile, and at the instant of load, the computer system, by virtue of some event, decided to suspend the thread, the preempted thread may actually be recalled to support the preempting event by reloading the launcher and firing at another target. Thread level preemption then requires that the tactical executive scheduling logic be capable of suspending and awakening multiple tasks simultaneously. It is intuitively obvious from the previous scheduling queue structure discussions that a preemption could occur as the direct result of (1) the current processing task requesting the scheduling of a higher-thread-level successor, or (2) an external interrupt from a decrementing clock or an input/output operation. The most frequent cause for preemption is the arrival of an external interrupt from another computer subsystem announcing "special-threat" target detections. This event arrival will cause any processing task, and its associated thread, to be suspended during normal executive interrupt processing, and the appropriate higher priority event processing thread will be placed into its appropriate priority position in the scheduling queue. The executive will then examine the scheduling queue in search of the highest priority pending task (which in most cases would be the suspended task). In this hypothetical case, however, the highest priority pending task is the new arrival. This special case causes the executive to preempt the previously interrupted thread/task and save all registers and volatile data-base contents. Processing control is then transferred to the new candidate. The executive then increases the priority of the preempted task to the highest within its predefined thread level. This procedure ensures that the preempted task is "awakened" prior to any other pending candidate selection in its (the preempted tasks) thread priority level.

This scheduling logic and queuing model enables the executive program to manage the critical processing periods, $t_p$ (Eq. 3) and $Q_i$(Eq. 7): to provide instantaneous response to special high priority events while maintaining the system integrity; and to permit task interleaving between threads.

Let us now examine periodic task scheduling requirements which are represented by either of the following process initiation triggers:

$$t_i = t_1 + \sum_{i=1}^{j} \Delta t_i \tag{11}$$

or

$$t_i = t_l + \Delta t \tag{12}$$

Eq. 11 is the scheduling-time calculation for determining when to begin processing predictable periodic tasks. This is obvious because the term $t_1$ represents the first time the task was processed and $\Delta t_i$ represents the fixed frequency; therefore, the next processing time will always be initiated some $\Delta t$ factor after the first processing time. These types of periodic tasks are scheduled for processing in an identical manner to non-periodic tasks, as earlier described.

Eq. 12 represents the scheduling time required for unpredictable periodic tasks. This is evident because the term $t_i$ is the *last* time the task was processed and $\Delta t$

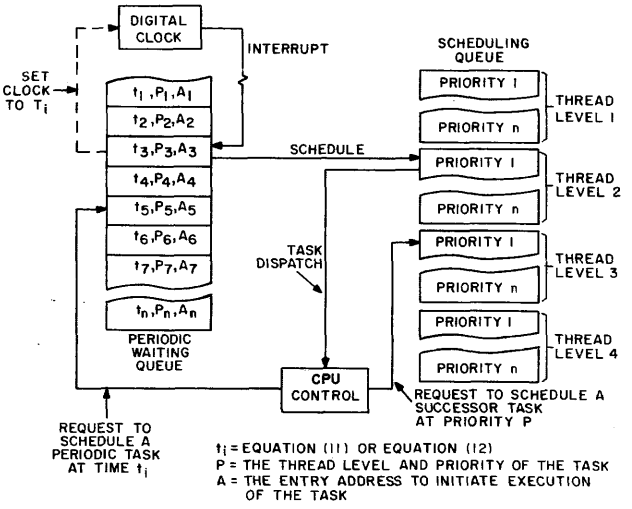



Figure 7—Executive scheduling philosophy

is a constant time increment, therefore in the event the last processing time $t_i$ was triggered by a random event, this type of task would not always be scheduled in exact $\Delta t$ intervals relative to the first scheduled time. As mentioned earlier, periodic tasks may be more critical than nonperiodic tasks, especially those represented by Eq. 12. This then requires that periodic tasks must begin processing in relation to their relative priority, when compared to all other system tasks. The executive program could satisfy this requirement by inserting all periodic tasks, which are ready for processing, per Eq. 11 or 12, into the scheduling queue, according to their thread level and priority. This technique will avoid having high priority pending tasks delayed because of lower priority periodic tasks instantly being processed upon achieving their respective scheduling times.

We can represent this design concept in Figure 7, where the "periodic waiting queue" is a "holding table" of unordered frequency dependent tasks awaiting their scheduling times, $(t_i)$, as represented by Eqs. 11 or 12. Periodic tasks are selected from the "periodic waiting queue" and inserted into the scheduling queue according to their respective thread level and priority; i.e., they will compete for processing time with the entire set of system tasks.

## CONCLUSIONS

The performance criteria for today's sophisticated tactical Command and Control systems imposes unprecedented requirements on the design of both the hardware and the software which control the systems. The most critical aspects include the time tolerances associated with the scheduling/dispatching and processing of tactical tasks, and the dynamic attribute of an ever changing, highly unpredictable tactical environment. The executive program, which is the nucleous of any tactical system, must be designed to operate not only in the classical non-tactical environment, but must additionally continually monitor the critical time periods associated with task group (threads) processing and automatically compensate, if possible, for any overruns. The executive scheduling and dispatching mechanism must also be sufficiently flexible to suspend and subsequently awaken groups of tasks in the event of unpredictable high priority event arrivals. These dynamic attributes of a tactical executive set it apart from the typical non-tactical executive, which operates in a well structured and fairly predictable environment, however it is obvious that many similarities do exist and in fact frequently outweigh the differences. Although most of the techniques discussed here are not new to the computer sciences, the method of implementation to solve the tactical problem is noteworthy. Most of the system characteristics described in this paper are representative of the U.S. Navy's AEGIS program presently being developed by RCA Corporation.

## REFERENCES

1. Phillips, W. G., "Executive Program Scheduling for Large Command and Control Systems," RCA Reprint No. RE 18-1-6, *RCA Engineer Magazine*, Vol. 18, No. 1, pp. 55-59, July 1972.