AT&T Bell Laboratories Technical Journal
Vol. 63, No. 8, October 1984
Printed in U.S.A.

The UNIX *System:*

The Fair Share Scheduler

By G. J. HENRY*

(Manuscript received July 19, 1983)

The Fair Share Scheduler (FSS) is a process scheduling scheme within the *UNIX*™ operating system that controls the distribution of resources to sets of related processes. This control offers features that are useful to many applications, including user control of service level, execution predictability, fair resource allocation, predictable and fair billing, and load insulation between user communities. This paper discusses the concepts of a fair share scheduler, the motivation for and history behind FSS, some practical FSS applications, the user and administrator interfaces to FSS, and the design philosophy of FSS.

I. INTRODUCTION

The primary motivation for the original versions of the *UNIX* operating system[1] was to create a powerful tool for the interactive user that was inexpensive in both equipment and human effort. Its most important implementation goals were to provide a system characterized by simplicity, elegance, and ease of use. The popularity of the system verifies the achievement of these goals. As the *UNIX* system enters production environments outside of the research world, enhancements are continually being made to allow it to adapt to different applications. This paper describes an enhancement to the process

* AT&T Bell Laboratories.

scheduler within the *UNIX* operating system, called the Fair Share Scheduler (FSS), that controls the distribution of resources to sets of related processes. This control is useful to a wide variety of environments, such as computation centers, project-managed facilities, and universities.

II. HISTORY

The original version of the process scheduler for the *UNIX* operating system was tailored to optimize process throughput for interactive users. It resolves Central Processing Unit (CPU) contention between processes by considering the recent CPU activity of each process. If the recent CPU activity of a process has low ratios of compute time versus real time, it will be associated with a good priority.[2] This implies that processes whose CPU activity is bursty in nature or small in total demand will be favored. These characteristics are typical of interactive processes, where there is some "think" time between each burst of processing.

This priority scheme works well because short tasks are predominate in the *UNIX* system. For example, editing is a common task on a *UNIX* system. Editing is interactive in nature and bursty with respect to system resource consumption requirements. The command interpreter for the *UNIX* system promotes joining small tasks to solve some larger task.[3] These small tasks typically request a single short burst of system resources and then exit.

The disadvantage of this scheduling scheme is the indeterminate nature of system response. Resource distribution to a process by the process scheduler within the standard *UNIX* operating system largely depends on the activity of the system as a whole. Since the total system activity is normally unknown over a given time period, the system response to a process (or user) is also unknown.

The original implementation of FSS was motivated by the computation center's need for giving a prespecified rate of system resources at a fixed cost to a related set of users. FSS provides a mechanism for contracting an average system response rate to a set of users that could predict their average system usage rate. This version has been used on production computation center systems since January of 1983. Since that time, FSS has proved to be beneficial to other applications and has been proposed as a desirable enhancement to the *UNIX* operating system.

III. CONCEPTS

Introducing FSS to the *UNIX* operating system changes conceptions inherent to the structure of the *UNIX* system. This section describes these changes, along with the new features provided by FSS.

3.1 System resources

System resources are the services provided to a process by the operating system, such as use of the processor or disk. Access to system resources by a process may be obtained only by going through the operating system. FSS maintains control over the distribution of all system resources by scheduling the processor based on the system resource consumption rate of a set of related processes.

3.2 Distribution of system resources

The standard *UNIX* operating system process scheduler considers the processor consumption rate of the full set of active processes on the system (see Fig. 1). It distributes all of the available system resources to n users (U), each having a domain of processes (p) owned by them. Each user may possess a different number of processes, each of which shows a variance in processing characteristics. The amount of resources available to a user is dependent on the number of active processes in a user's domain, the number of active processes in the system domain, and the type of activity exhibited by each process.

FSS considers the resource consumption rate of a related group of processes, along with the individual processor consumption rates for each active process on the system. A group of processes associated with the same resource consumption rate is called a *fair share group.* FSS controls the *UNIX* system by dividing the system resources into fair share groups and associating each fair share group with a set of users. The process scheduler for the standard *UNIX* operating system handles contention between processes within each fair share group. Thus, resource distribution by FSS to a user is also determined by the

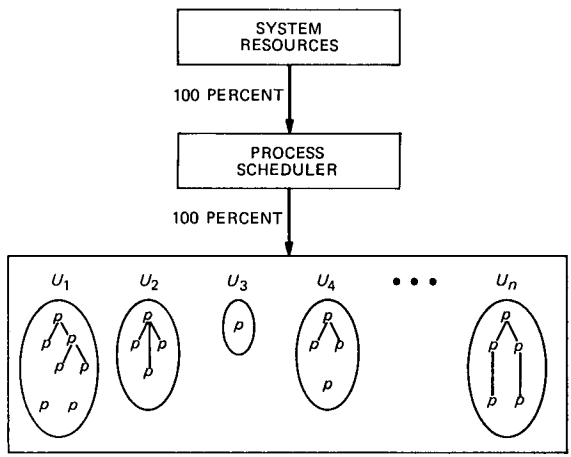


Fig. 1—Standard process scheduler.

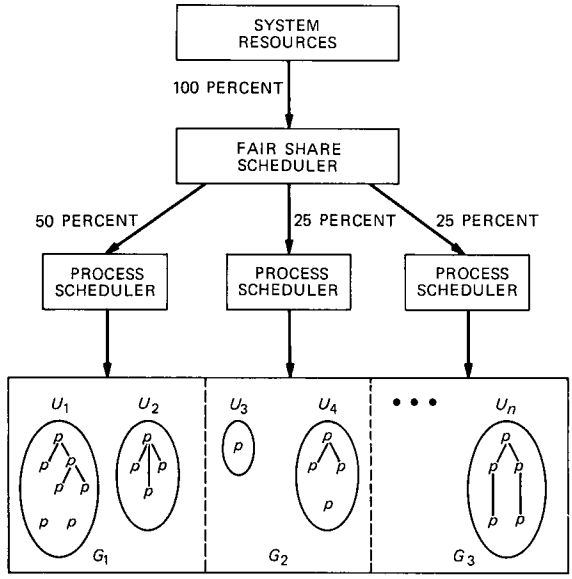


Fig. 2—Fair share scheduler.

user's association with a fair share group and the allocation of system resources to that fair share group.

Figure 2 shows the same set of users (U) and processes (p) as Fig. 1. Each user process domain is now bounded by a fair share group. In this example, FSS distributes the total resources to a set of three fair share groups (G_1, G_2, and G_3); G_1 is allocated 50 percent of the available resources; G_2 and G_3 are each allocated 25 percent of the available resources. In effect, each fair share group is provided with a virtual *UNIX* system.

3.3 Access to system resources

Access to system resources by a process with FSS is determined by the user that owns the process, the fair share group the user is associated with, and the resource consumption rate of the fair share group. A fair share group process association or resource consumption rate may change dynamically.

Controlling fair share group access and resource consumption rates allows a new set of administrative alternatives on *UNIX* systems that may be represented with a pie chart (see Fig. 3). The area of the pie chart represents the total amount of available system resources. Each pie chart slice is the amount of system resources allocated to a given fair share group. The filling of each pie chart slice is the number of users associated with a fair share group.

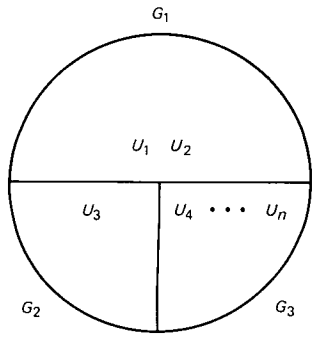


Fig. 3—Resource allocation.

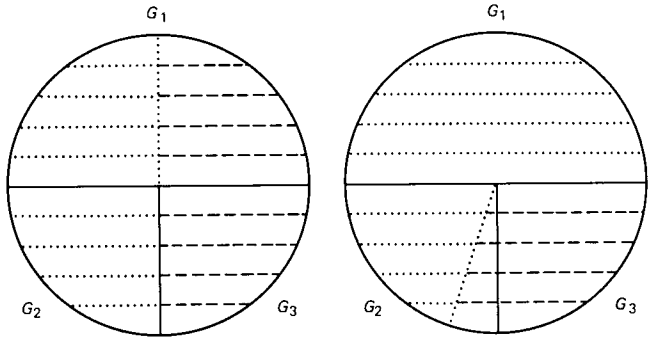


Fig. 4—Unused resource distribution.

3.4 Unused system resources

When a fair share group is not using its full resource portion, FSS distributes the extra resources in relative proportions to other fair share groups that show the demand. This has the desirable effect of using all the system resources when a demand exists, while maintaining boundaries for distributing resources that are relative to fair share group allocations.

Consider the fair share group allocations described in Fig. 3. If G_1 is not using any of its allocated resources, FSS gives those resources in equal portions to G_2 and G_3 because they are both allocated an equivalent amount of system resources (see Fig. 4a). If G_2 is not using any of its allocated resource, FSS gives G_1 twice as much of these resources as it gives G_3 because G_1 is allocated twice the amount of resources as G_3 (Fig. 4b). Therefore, G_1 receives two-thirds of the total system resources, while G_3 receives the rest.

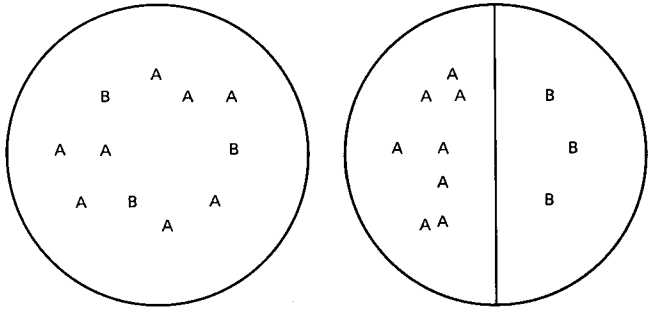
IV. APPLICATIONS

The general benefit of FSS is the dynamic control it has over the distribution of resources in the *UNIX* system. That is, knowing the number and type of processes that are using a virtual system of a given size allows greater predictability for the execution of a given task or the responsiveness for a given user. This section will point out some practical applications of how this control may be used.

A computation center may use FSS to allocate resources to a predefined set of users at a fixed cost. The cost is calculated by the relationship between the total cost of the real system versus the percentage allocated to a virtual system. The set of users have the advantage of being able to define their responsiveness and predict the charges that they will incur. The computation center has the advantage of providing a billing procedure that is both predictable and fair.

Providing a fixed processing rate to a set of users has the added advantage of insulating that set from other sets of users on the same system. A system with a heterogeneous user population has the potential for one set of users to monopolize system resources. This typically happens when a system is saturated with a set of users from a related project. For example, employees from the same department are reaching a project deadline or students from the same class have a project due. Users that are not related to that project must compete on equal terms with these users. Figure 5a shows the process scheduler for a standard *UNIX* operating system with two user groups, A and B. Group A has the potential to obtain more system resources than group B simply because group A owns more active processes. FSS can resolve this by associating each group with a fixed portion of the system (see Fig. 5b). This means that group B will be insulated from the activities of users in group A.

A system administrator can use FSS to achieve a resource-limiting



(a) (b)

Fig. 5—Load insulation.

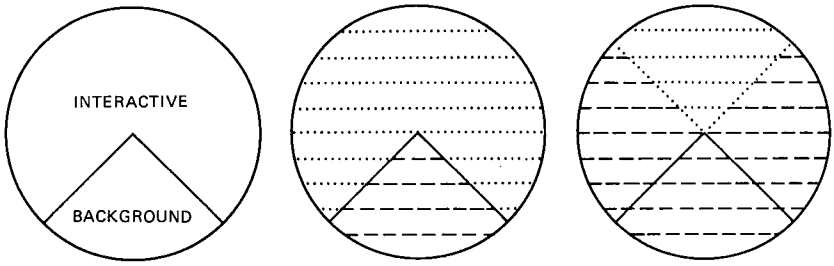


Fig. 6—Upper bound enforcement.

scheme by associating process sets with a small virtual system. Most *UNIX* systems have many active background processes competing with interactive processes for system resources, such as networking tasks or system monitors. FSS may be used to give the interactive processes a higher priority over the background processes by associating the background processes with a small resource consumption rate relative to the other fair share groups (see Fig. 6a). When there is a high interactive and background demand, the background processes will be confined to a fixed limit (see Fig. 6b). When there is a low interactive demand and high background demand, the unused resources go to the background processes (see Fig. 6c).

A project manager can use FSS to dynamically select the amount of resources available to users in (or not in) the critical path of a project. Raising the upper bound of a fair share group consumption rate to a large limit relative to other fair share groups and associating users in the critical path with this large fair share group may give good response to a set of users. Users in the small fair share group will have worse response but may use the resources left over when the demand decreases from users in the large group. This allows a project manager to allocate resources to critical activities without requiring a dedicated system.

A final example of FSS use is to divide the system resources evenly among the interactive users on the system, that is, to divide a system with n interactive users into n fair share groups, each provided with $1/n$ of the system resources. This has the advantage of insulating users from each other, while ensuring that each user has an equal share of the system resources.

V. INTERFACE

The user interface to FSS requires a small set of commands for administration and fair share group access. This section provides some

examples of their use and assumes knowledge of basic *UNIX* system concepts.

5.1 Establishing fair share groups

The system resource division in Fig. 3 may be established through the following sequence of commands:

```
fsadm -a -s 50 -i 1 G1
fsadm -a -s 25 -i 2 G2
fsadm -a -s 25 -i 3 G3
```

These commands inform FSS of three fair share groups, `G1`, `G2`, and `G3`, which are allocated `50`, `25`, and `25` shares, respectively, of the available system resources and are identified to FSS by the integers `1`, `2`, and `3`. (The number of shares associated with a fair share group determines its allocation of resources. In this example, there is a total of one hundred shares of system resources. Thus, one share is equivalent to 1 percent of the total system resources.)

Dynamic share modification may be done through the same command. The command sequence

```
fsadm -m -s 25 G1
fsadm -m -s 50 G2
```

reverses the resource allocation rates between fair share groups `G1` and `G2`, described previously.

5.2 Associating users with fair share groups

The fair share group administrator may provide user access to a fair share group by explicitly associating a user with a fair share group. Figure 3 shows two users (`U1`, `U2`) associated with fair share group `G1`. The command

```
fsgadm -a -g G1 U1
```

allows the user with the login name `U1` to access the fair share group named `G1`. If no other fair share groups are associated with this user, the fair share group `G1` will be the only one accessible to this user. Generally, one fair share group is used as a system default for those users not associated with any fair share group.

5.3 User access to fair share groups

The association between a fair share group and a user process is normally established when the user logs in. Each new process created by a user inherits the same fair share group association as its parent process. This association may be dynamically changed to an alternate fair share group by

```
chfsg -g G1 U1
```

which associates the *UNIX* system processes owned by the user with the login name `U1` to the fair share group named `G1`.

VI. DESIGN

FSS was designed to minimize the number of changes and amount of overhead in the process scheduler, while preserving the basic structure of the *UNIX* operating system. The resulting FSS implementation incurs less than one-percent operating system overhead and requires no change in any user-level programs. The following section describes an overview of the operation of FSS and is not intended as a complete proof of the algorithm.

6.1 Standard process scheduler

The process scheduler for the standard *UNIX* operating system distributes resources by using a prioritized round-robin queueing scheme (see Fig. 7). The priority is actually a number that is associated with each process. A logical queue exists for each priority value. When the process scheduler selects another process to run, it simply chooses the first runnable process on the highest-priority queue. The CPU is allocated in a round-robin fashion until a higher-priority event occurs,

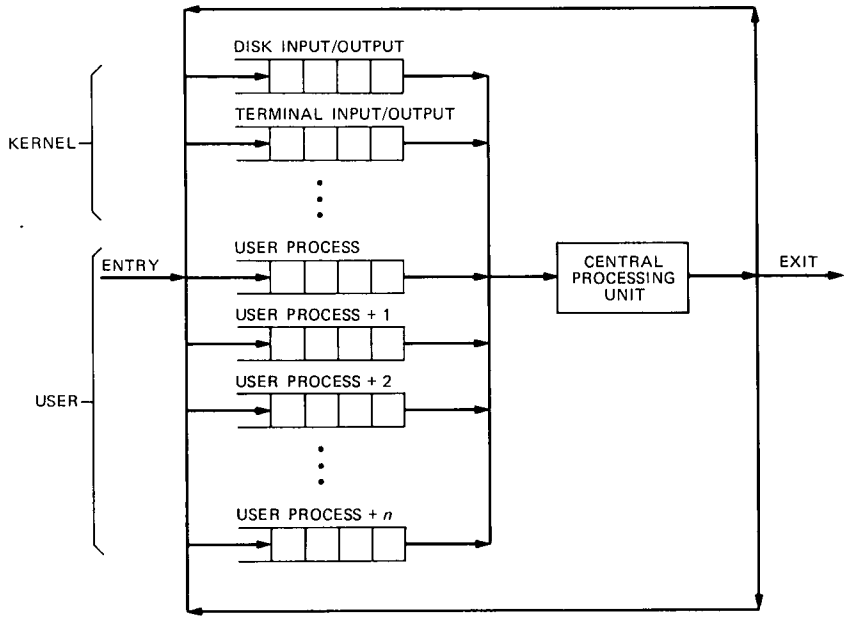


Fig. 7—Standard queueing model.

causing another process to become active; the process is done with the CPU; or a time limit expires. If the process still requires more service after relinquishing the CPU, the process is reinserted into a lower-priority queue. The priority structure is divided into two types: kernel and user.

Kernel priority is used when a process is executing within the operating system for a user process (for example, when a process requests a block from disk). Kernel priorities are the highest and are used for reserved operating system functions. Priorities at this level are roughly layered with respect to the response one would expect for a particular event. Disk events have high priority and terminal events have low. This structure is established to optimize throughput of critical resources.

A process generally has a *user* priority when it is contending for the use of the CPU. User priorities are lower than any of the kernel levels and are calculated at least every second for each process. The user-level process priority is considered to be high when it contains a low numerical value and may be represented by the following ratio:

$$UNIX \text{ system priority} = \frac{\text{recent CPU usage}}{\text{real time}}.$$

A process generally enters the system at the highest user priority because it has no recent CPU activity. The user priority drops as the process uses the CPU and rises when the process is kept from using the CPU.

6.2 Fair share scheduler

FSS maintains fair share group resource consumption rates by expanding the definition of user priority to include resource usage by a fair share group. Resource usage is a function of the entities provided to a process by the operating system, such as use of the CPU and access to disks. The expanded definition logically separates processes into another set of user priority queues, while maintaining the same kernel-level priorities (see Fig. 8). That is, the user priority queues are divided into a set of user priority queue structures, one set for each fair share group. The fair share group that is farthest from achieving its resource consumption rate will have its set of queues on top of the user queues, the set for the next farthest fair share group follows, and so on. Fair share group sets are reordered every second along with processes within each set. The FSS user-level process priority function is then expanded to

$$\text{FSS priority} = UNIX \text{ system priority} + \frac{\text{recent fair share group resource usage}}{\text{real time}}.$$

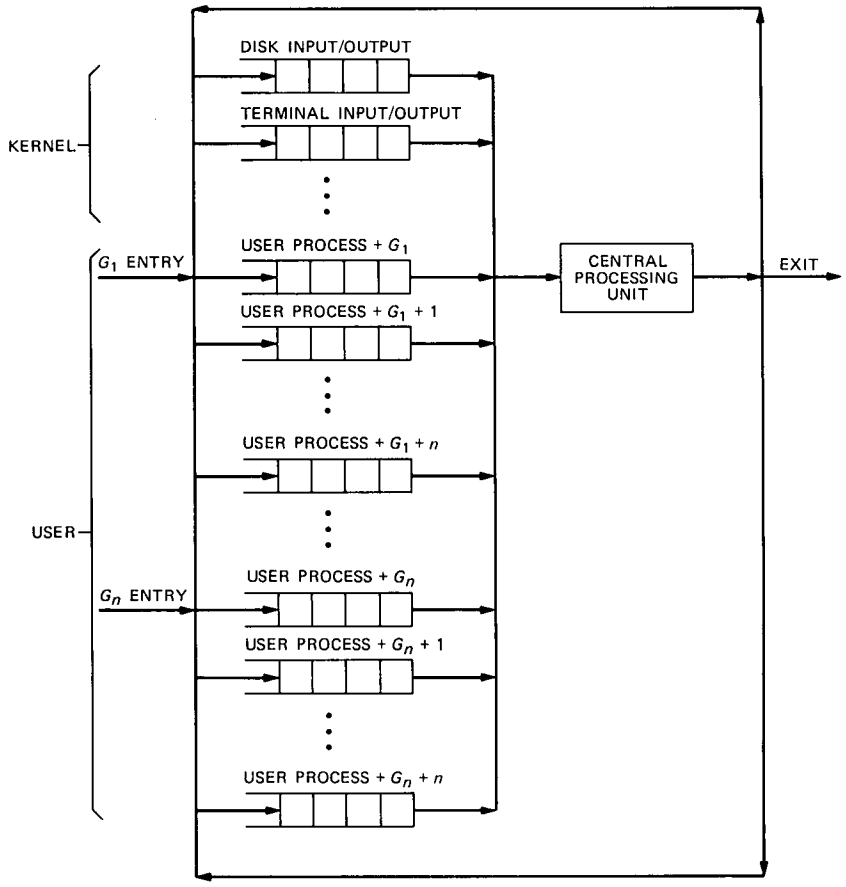


Fig. 8—FSS queueing model.

The fair share group resource usage is calculated by taking the exponentially weighted sum of the system resources recently used by all the processes within the fair share group. This sum is normalized to the allocated resource consumption rate associated with the fair share group and compared to a similar measure for all the other fair share groups. This additional priority function ratio has the same characteristics as the *UNIX* system priority. The priority will drop as the fair share group uses more resources than are allocated to it and rise when it is kept from using system resources. Thus, the new priority function distributes system resources according to the resource consumption rate associated with each fair share group, while maintaining the same scheduling philosophy as the standard *UNIX* operating system within each fair share group.

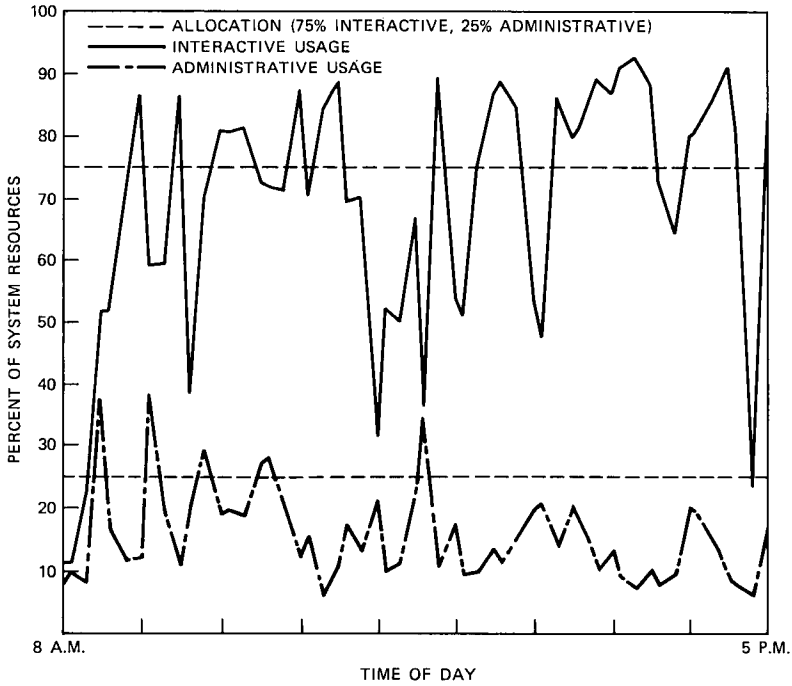


Fig. 9—Actual fair share group usage.

VII. LIMITATIONS

The queue model of the *UNIX* operating system suggests that precedence is given to the optimization of critical resources at the kernel level. This implies that it is not always possible to guarantee an exact resource consumption rate to a fair share group over a given period of time. However, the average resource consumption rate should approach the allocated fair share group resource consumption rate, providing that there is a sufficient demand. Figure 9 shows the actual usage of two fair share groups on a typical *UNIX* system. One fair share group is used for interactive users and is allocated 75 percent of the system resources, while the other is used for administrative tasks and is allocated the remaining system resources. The usage of each fair share group, in general, fluctuates around its corresponding allocated rate. Also notice that the peaks of the administrative fair share group, above its allocation, correspond to the valleys below the allocation of the fair share group for interactive users.

VIII. SUMMARY

FSS was designed to extend the process scheduling criteria in the *UNIX* operating system for the purpose of giving a prespecified rate

of service at a fixed cost to a related set of users in a computation center environment. The resulting implementation gives the *UNIX* system an additional control mechanism that is beneficial to many different applications. This control allows the division of system resources into parts and the constriction of user access to each part. The user interface requires a small set of commands for administration and user access. The implementation incurs a small amount of operating system overhead and relies on the existing process priority structure within the *UNIX* operating system.

IX. ACKNOWLEDGMENTS

I would like to thank Roger Polsley for the insight that he gave me through our discussions. Much of the FSS design and implementation is based on his pioneering efforts.

REFERENCES

1. D. M. Ritchie and K. Thompson, "*UNIX* Time-Sharing System: The *UNIX* Time-Sharing System," B.S.T.J, *57*, No. 6 (July-August 1978), pp. 1905–29.
2. K. Thompson, "*UNIX* Time-Sharing System: *UNIX* Implementation," B.S.T.J, *57*, No. 6 (July-August 1978), 1931–46.
3. S. R. Bourne, "*UNIX* Time-Sharing System: The *UNIX* Shell," B.S.T.J, *57*, No. 6 (July-August 1978), pp. 1971–90.

AUTHOR

Gary J. Henry, B.S. (Computer Science), 1977, University of Wisconsin, Madison, WI; Raytheon, Portsmouth, RI, 1978–1979; M.S. (Information Engineering), 1982, University of Illinois, Chicago, IL; AT&T Bell Laboratories, 1979—. From 1978 to 1979, Mr. Henry introduced the *UNIX* operating system to Raytheon as a programmers workbench for several ongoing projects. Since December of 1979, Mr. Henry has been providing technical *UNIX* system support for the Computing Technology department at AT&T Bell Laboratories at Naperville, Illinois, and is responsible for support development, consulting, problem solving, coordination of *UNIX* system release conversions, and configuration planning. His project involvement has included the Fair Share Scheduler, AT&T–Philips Joint Venture consultation and installation of *UNIX* system software for the first European AT&T 3B20S, *UNIX* system conversion coordinator for releases 4.0 and 5.0 for the Naperville computation center systems, *UNIX* system internals instructor, *UNIX* system 5.0 performance improvements, and shared memory implementation project member. Mr. Henry is currently responsible for global *UNIX* system support functions for all of the AT&T Bell Laboratories computation centers. Member, USENIX Association and/usr/group.