# SCHEDULING IN CONCURRENT PASCAL*

F. B. Schneider
A. J. Bernstein
Department of Computer Science
S.U.N.Y. at Stony Brook
Stony Brook, New York, 11794

## Introduction

The monitor construct [H74] [B75] and the programming language Concurrent Pascal [B75] have provided a basis for research into operating systems. Based on the experiences gained by the use of Concurrent Pascal in the construction of operating systems [B76] [G77] there has been some discussion about the monitor implementations proposed by Brinch Hansen [B75] and Hoare [H74]. For example, the (non) problem of nested monitor calls has received considerable attention [L77] [H77] [B78] [W78]. In addition, the lack of facilities for dynamic resource management in Concurrent Pascal has inspired research proposals which extend the language to solve those problems [S77] [K77] [A78]. It is unfortunate that much of this research culminates in new monitor-like objects which can only be used to solve particular problems. As pointed out by Parnas [P78] this seems to indicate that more primitive constructs from which various monitor-like objects may be built, are required. In this paper such a construct is introduced and its applicability in monitors used to construct systems which impose a scheduling discipline on a shared resource (e.g. disk) is studied.

## The Scheduling Scenario

Consider a system in which a number of concurrently executing processes access some shared resource such as a disk. Disk head seek times are usually very long compared with the actual data transfer time associated with a disk access. Consequently, more efficient disk utilization, as well as improved average waiting time for processes attempting to access the disk, can be realized by using a disk head scheduling algorithm such as the one described in [H74]. In this algorithm requests are ordered so that the disk head sweeps across the disk in one direction, then the other, analogous to the operation of an elevator in an office building.

An access graph [B72] for the implementation of a scheduling algorithm for a disk is illustrated in Figure 1. In order to access the disk it is required that a user first call the scheduler, which may cause the caller to be suspended until a time when the disk transfer can be performed efficiently. Upon returning

---

*This research was supported in part by NSF grant MCS 76-04828.

from the scheduler, the disk is called to perform the actual I/O operation. Lastly, the scheduler is called to report the completion of the transfer. This structure, however, is undesirable since it reveals to higher levels of the system (the user modules in this case) the functions of scheduling and disk I/O as separate entities. In addition, a user could easily defeat the scheduling algorithm by not calling the scheduler prior to accessing the disk.

A more desirable arrangement would be to provide a single call which invoked both the scheduling and transfer functions, and returned to the higher level on completion. It would then be impossible for a user to directly access the resource, save through the "scheduler". Although Figure 2 exhibits this type of structure it is unacceptable because entry to the scheduler is prevented if an I/O operation is in progress (due to the mutual exclusion associated with that monitor). Thus no real scheduling can take place because processes would be suspended at the entrance to the scheduler, instead of inside it where an ordering on the suspended processes would be imposed by some priority <u>wait</u> mechanism [H74].

## The New Construct

To solve this problem, we propose a new mechanism which can be used in designing a monitor. Ordinarily, processes attempting to enter monitor procedureswhile another process is active within the monitor are delayed by mutual exclusion at monitor entry. The order in which these processes ultimately enter is undefined. We propose the addition of a facility through which this order can be explicitly controlled. This is accomplished by associating with each entry procedure an integer valued function called a scheduling discipline, which yields a priority. This priority is used to order the processes waiting to enter the monitor.

A scheduling discipline is associated with a procedure entry by the <u>use</u> clause in the procedure heading. The named scheduling discipline is then used to compute the priority of a process attempting to enter that procedure. Scheduling disciplines are defined as PASCAL functions, declared global to all entry procedures. The function may reference permanent monitor variables, though it may not alter their values. The function's parameter list provides a mechanism for using monitor procedure entry parameters in the priority computation as well. Call by value is imposed to ensure that the function executes without side-effects.

Whenever a process attempts to enter a monitor while there is another process actively executing in the module the caller is blocked at mutual exclusion. When a process relinquishes control of the monitor either by exiting a monitor procedure, or by being suspended at a <u>wait</u> statement, a new process must be granted control of the monitor. To select this process, the scheduling discipline (priority function) is evaluated for each process which is blocked at mutual exclusion. These processes are then ordered on an entry queue, which is a queue associated with the monitor containing entries for processes arranged in ascending priority order. The process at the head of the queue is then granted entrance.

The evaluation of that scheduling discipline may only be performed while there is no process actively executing in the monitor, because permanent monitor variables will be referenced. It may be that a number of processes must be added to the entry queue each time control is relinquished. This is because the time spent at the scheduled resource (e.g. disk) by a process can be relatively long, and for that time the mutual exclusion at the monitor will be in effect due to the nested call (as in Figure 2).

Figure 3 illustrates the use of this feature by implementing the disk head scheduling algorithm discussed earlier.

It should be noted that it is not possible to write pathological scheduling disciplines in which each time a process relinquishes control of the monitor the relative ordering of the processes on the entry queue changes due to the altered values of the permanent variables. Such a capability does not appear to have any practical application and would incur a high execution overhead for the context switches associated with the repeated evaluation of the priority functions. The construct, as we have proposed it, requires two context switches for each process that enters the monitor; one for priority evaluation, and a second at monitor entry. In certain situations even this may be deemed excessive overhead.

## Conclusion

A generalization of the monitor [H74] [B75] has been presented which permits a natural solution to scheduling problems. The construct is analogous to a priority wait mechanism [H74] for monitor entry. Present monitor implementations usually impose a first come first served discipline on monitor entry which precludes their use in certain situations and leads to awkward system constructions.

## Acknowledgements

The authors are grateful to A. Mahjoub, P. Harter, and K. Ekanadham for discussion about the material presented here.

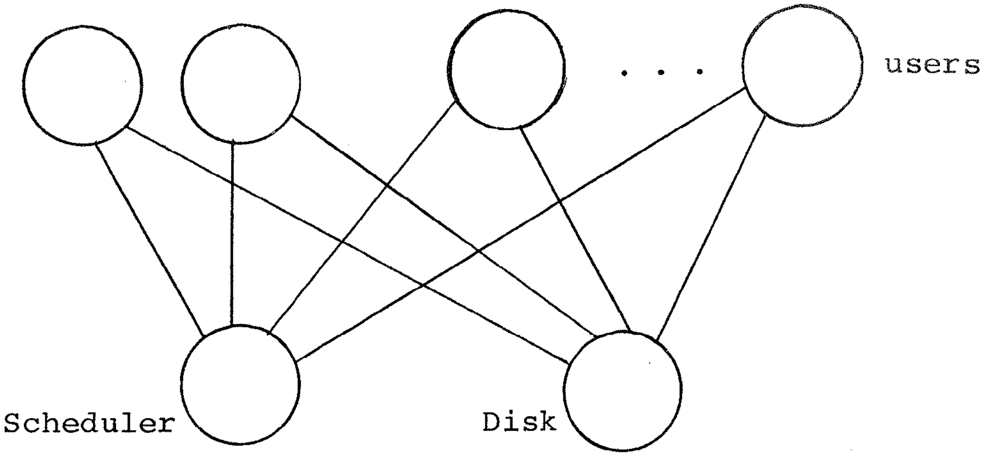

Figure 1

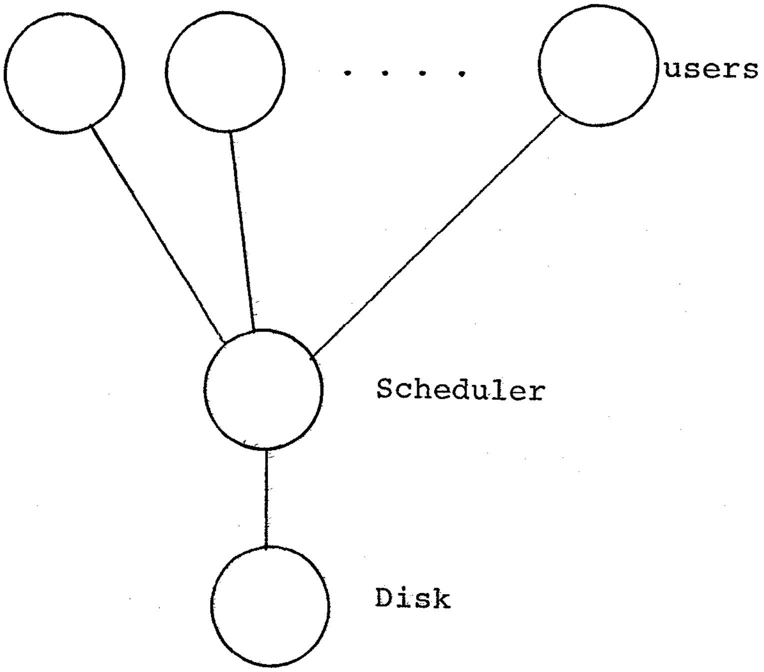

Figure 2

```
type  disksched = monitor  (diskdrive : disk ) ;
     const  disksize = D ; ( * number of tracks on disk *)
     var  incr, lastincr : boolean ;
          tracks_scanned, curaddr, lastaddr : integer ;

     function  priority  ( trkno : integer ) : integer ;
        begin
            if incr and  (trkno>curaddr )
               then  priority := trkno + tracks_scanned
               else if incr
                       then  priority := disksize - trkno + disksize
                                           + tracks_scanned
                       else if not incr and trkno<curaddr
                                   then priority := disksize - trkno
                                                    + tracks_scanned
                                   else priority := disksize + trkno
                                                    + tracks_scanned
        end ;  (* schedule discipline *)

     procedure entry accessdisk ( trkadder : integer ;
                                  var block : page ;
                                  iotype : (read,write ) )
               use priority (trkadder) ;
        begin
           lastaddr := curaddr ;
           curaddr := trkadder ;
           lastincr := incr ;
           if curaddr>lastaddr then incr := true
                               else incr := false ;
           if not (lastincr = incr) then tracks_scanned :=
                                tracks_scanned + disksize ;
           call diskdrive (block, iotype) ;
        end ;

     begin (* initialization *)
           incr := true ; curaddr := 0 ; tracks_scanned := 0
     end
```

Figure 3 - Example

# References

A78 Andrews, G.R., McGraw, J.R., "Language Features for Process Interaction", Operating Systems Review, Vol 11, No. 2 (April 1977) pp 114-127.

B72 Brinch Hansen, Per, Operating System Principles Prentice Hall, New Jersey, 1973.

B75 Brinch Hansen, Per, "The Programming Language Concurrent Pascal", IEEE Transactions on Software Engineering, SE-1,2 (June 1975) pp 199-206.

B76 Brinch Hansen, Per, "The Solo Operating System: A Concurrent Pascal Program", Software Practice and Experience, Vol 6 pp 141 - 149.

G77 Graf, N., Kretschmar, H., Lohr, L.P., Morawetz, B., "How to Design and Implement Small Time-Sharing Systems Using Concurrent Pascal", TR-77-09, Fachbereich Informatik, TU Berlin (1977).

H74 Hoare, C.A.R., "Monitors: An Operating System Structuring Concept", CACM 17,10 (Oct 1974), pp 549 - 557.

H77 Haddon, B.K., "Nested Monitor Calls", Operating Systems Review Vol 11, No. 4 (Oct 1977), pp 18 - 23.

K77 Kieburtz, R.B., Silberschatz, A., "Capability Managers" TR 71, Dept. of Computer Science, S.U.N.Y. at Stony Brook (May 1977).

L77 Lister, A., "The Problem of Nested Monitor Calls", Operating Systems Review, Vol 11, No. 2 (July 1977), pp 5 - 7.

P78 Parnas, D.L., "The Non-problem of Nested Monitor Calls", Operating Systems Review, Vol 12, No. 1 (Jan. 1978) pp 12 - 14.

S77 Silberschatz, A., Kieburtz, R.B., Berstein, A.J., "Extending Concurrent Pascal to Allow Dynamic Resource Management", IEEE Transactions on Software Engineering, SE-3, No. 3 (May 1977).

W78 Wettstein, H., "The Problem of Nested Monitor Calls Revisited", Operating Systems Review, Vol 12, No. 1 (Jan. 1978) pp 19 - 23.