PRIORITY IS A LIMITED PROPERTY*

Jeffrey A. Barnett
USC Information Sciences Institute

Alvin S. Cooperband
American Data Products Corp.

A system contains several processes with different priorities. All are in the wait queue for the same semaphore and each has the form

```
WHILE TRUE DO
BEGIN WAIT(S);
      . . .
      SIGNAL(S);
END;
```

What happens when the semaphore `S` is signalled? If you thought the highest priority process runs continuously to the exclusion of all others, then you made a bad guess. Consider this program written in UCSD PASCAL.

```
PROGRAM M;
VAR   S:SEMAPHORE;
      K:INTEGER;
      Q:PROCESSID;
PROCESS P(I:INTEGER);
BEGIN WHILE TRUE DO
      BEGIN WAIT(S)
            WRITE(I);
            SIGNAL(S);
      END;
END;
```

*This paper originally appeared in *Operating Systems Review* **17**(3), (1983).

```
BEGIN SEMINIT(S.0);
      FOR K:=1 TO 5 DO
          START(P(K),Q,300,200+K);
      SIGNAL(S);
END.
```

The main program starts processes 1 . . . 5 with respective increasing priorities 201 . . . 205. (The processes are named by the number they output.) All are in the wait queue of S after execution of the for loop. When the main program signals S, the output produced is

```
54535452545354515453
54525453545555......
```

Notice that the processes (printing) 1 . . . 4 run respectively 1, 2, 4, and 8 times before process 5 takes over the CPU.

To see why the highest priority process does not immediately seize complete control, three things must be understood. First, there are two process queues — RUN_QUE and S.QUE, the queue associated with semaphore S. Each queue is ordered so that processes with higher priorities come first. Second, when a semaphore is signalled, the highest priority process (if any) in the associated queue is moved to RUN_QUE. Third, the first process in RUN_QUE is the process that is executing.

Now consider the sample program. When the main program, M, signals S, RUN_QUE is [M] and S.QUE is [5 4 3 2 1]. Just after the signal, the queues are [5 M] and [4 3 2 1]. Process 5 runs and prints "5" then signals S. Since 5 is not now in S.QUE, it cannot consume the semaphore. Rather, 4 does and the resulting queues are [5 4 M] and [3 2 1]. Process 5 continues execution with its wait statement, but there is no way for it to proceed beyond that point because S.COUNT = 0. Therefore, the queues become [4 m] and [5 3 2 1], and process 4 is run.

Similar reasoning applies through execution of the first 30 write statements when the queues become [5 4 3 2 1 M] and [], i.e., all process are in RUN_QUE. Thereafter, the highest priority process,5, just continues because there is no other process in S.QUE to consume the semaphore when it signals.