Monitors and Semaphores

This handout describes how to express monitors in terms of semaphores. If an operating system provided semaphores as primitives, this is what a compiler would produce when presented with a monitor:

1 var mutex, urgent, xcond: semaphore;
   urgentcount, xcondcount: integer;

The body of each procedure in the monitor is set up like this:

3 down(xmutex);
4 (* procedure body*)
5 if urgentcount > 0 then
6   up(urgent)
7 else
8   up(mutex);

Each x.wait within the procedure is replaced by:

9 xcondcount := xcondcount + 1;
10 if urgentcount > 0 then
11   up(urgent)
12 else
13   up(mutex);
14 down(xcond);
15 xcondcount := xcondcount - 1;

Each x.signal within the procedure is replaced by:

16 urgentcount := urgentcount + 1;
17 if xcondcount > 0 then begin
18   up(xcond)
19   down(urgent);
20 end;
21 urgentcount := urgentcount - 1;

**line 1** The semaphore mutex is initialized to 1 and ensures that only one process at a time is executing within the monitor. The semaphore urgent enforces the requirement that processes that signal (and as a result are suspended) are to be restarted before any new process enters the monitor. The semaphore xcond is used to block processes doing waits on the condition variable x. If there is more than one such condition variable, a corresponding semaphore for each condition variable must be created. urgent and xcond are initialized to 0.

**line 2** The integer urgentcount indicates how many processes are suspended as a result of a signal operation (and so are waiting on the semaphore urgent); the counter xcondcount is associated with the condition variable x, and indicates how many processes are suspended on that condition (i.e., suspended on the semaphore xcond).

**lines 3-8** Since only one process at a time may be in the monitor, the process entering the monitor procedure must wait until no other process is using it (down(mutex)). On exit, the process signals others that they may attempt entry, using the following order: if any other process has issues a signal and been suspended (i.e., urgentcount ≠ 0), the exiting process indicates that one of those is to be continued (up(urgent)). Otherwise, one of the processes trying to enter the monitor may do so (up(mutex)).

**lines 9-15** The process indicates it will be executing an x.wait by adding 1 to xcondcount. It then signals some other process that that process may proceed (using the same priority as above). It suspends on the semaphore xcond. When restarted, it indicates it is done with the x.wait by subtracting 1 from xcondcount, and proceeds. Note that the down(xcond) will always suspend the process since, unlike semaphores, if no process is suspended on x.wait, then x.signal is ignored. So when this is executed, the value of the semaphore xcond is always 0.

**lines 16-21** First, the process indicates it will be executing an x.signal by adding 1 to urgentcount. It then checks if any process is waiting on condition variable x (xcondcount > 0), and if so signals any such process (up(xcondsem)) before suspending itself (down(urgent)). When restarted, the process indicates it is no longer suspended (by subtracting 1 from urgentcount).