Blocking Process

A process is described as blocking or waiting when it is set inactive until a specific event occurs.

Blocking Process

A blocking process is usually waiting for an event such as a semaphore being released or a message arriving in its message queue. In multitasking systems, such processes are expected to notify the scheduler with a system call that it is to wait, so that they can be removed from the active scheduling queue until the event occurs. A process that continues to run while waiting (i.e., continuously polling for the event in a tight loop) is said to be busy-waiting, which is undesirable as it wastes clock cycles which could be used for other processes.

Sleeping

A special case of waiting is sleeping, which is when a process is set inactive for a given period of time (e.g., 20 milliseconds). This is usually handled separately for ordinary waiting, primarily for efficiency reasons; since the clock interrupt is frequent, and keeping track of individual ticks is infeasible, the usual approach is to use a system of relative counters to mark when a given process will be awakened.

A commonly used data structure for tracking sleeping processes is the delta queue, an ordered list of sleeping processes in which each process has a counter which is offset relative to last process in the list before it; for example, if process A is sleeping for 3 ticks, process B for 5 ticks, process C for 5 ticks, and process D for 8 ticks, then the list would be

```
{A, 3} -> {B, 2} -> {C, 0} -> {D, 3}
```

At each clock tick, the counter for the topmost process is decremented, like so:

```
{A, 2} -> {B, 2} -> {C, 0} -> {D, 3}
```

If at this point process E is added which is to wait 6 ticks, then it would be inserted before D, and both E and D would be updated appropriately:

```
{A, 2} -> {B, 2} -> {C, 0} -> {E, 2} -> {D, 1}
```

If the topmost process reaches zero, then it -- and any processes immediately following it that are also zero -- are placed back into the active scheduling queue. Thus, after 2 more ticks, A is set as active; 2 ticks after that, both B and C are set as active; and so on. This approach has the advantage of minimizing the number of changes required in a given clock tick.

Interruptible and Uninterruptible

It is generally a good idea to differentiate between interruptible wait/sleep and uninterruptible wait/sleep. Depending on your system, you will probably want most wait states interruptible, though you almost definitely will need some uninterruptible states. You might want to consider building interruptible wait on top of uninterruptible wait, instead of using special flags.
