CMPS 111 Operating Systems
Winter 2018, Homework #2

Aaron Steele, atsteele@ucsc.edu

Question 1

The PCB stores data for each process. The data stored is everything the process needs to run and keep running. The PCB is required because we have to have some way of keeping the data in a block of memory and moving it around as the process is taken on and off the CPU.

1. Process ID (PID) - The ID the OS knows the process by
2. Program Counter - The location in the program the process is at
3. Register Files - The various register values for the process

Question 2

- Race condition - Processes that are faster to request the resources are the ones that gain control of said resources. They are all in a race, and the first one to get there is the winner. If this is the only method of requesting resources it is possible for a process to never get a resource, and thus violate bounded wait.
- Deadlock - A situation from which it is impossible to proceed. In terms of OS, this can happen when a process is waiting on a resource, and has one resource. If another process has the first resource and wants the second one, the processes will forever be waiting and thus deadlock.
- Starvation - When a thread or process is waiting forever.
- Deadlock & Starvation - If a process is starved, it is not necessarily deadlocked. But if it is deadlocked then it is necessarily starved.

Question 3

If we assume that when processes are interrupted they are placed in a queue containing all non-running processes not waiting for an I/O operation to complete, briefly describe two strategies the Operating System might adopt to service that queue. One-word answers will not suffice.

1. Round robin
   - Take the next process out of the queue and let it run for a little bit of time, then put it back in when it gets interrupted. Continue this, giving each process a slice of processing time.
2. Shortest job first
   - Take the job that has the lowest expected 'burst' time, or amount of time it is expected to need to be on the CPU. Once it is interrupted put it back on the queue and continue with the rest of the queue.

Question 4

If we talk about priority inversion also in terms of preemption, that if a low-priority process is in a critical section, then a higher-priority thread preempts it off, this is also priority inversion.
User-level threads disallow preemption, which means that priority inversion is not possible with them, when it is with kernel-level threads.

Question 5

A counting semaphore is, "a nonnegative integer count. Semaphores are typically used to coordinate access to resources, with the semaphore count initialized to the number of free resources. Threads then atomically increment the count when resources are added and atomically decrement the count when resources are removed." [1]

Making a counting semaphore S, using a binary semaphore, we need a value for S.

```
unsigned int val,
binary semaphore wait
```

Then, [1]

```
P(S) {
        if S.val = 0
        then { P( S.wait ) }
        else { S.val := S.val - 1 }
}
V(S) {
        if "processes are waiting on S.wait"
        then  {  V( S.wait ) }
        else { S.val := S.val + 1 }
}
```

The P function says to wait if the counting semaphore's value is 0, if it isn't then decrease the semaphore's value.
The V function checks if there are processes that are waiting, if so mark the mutex. If there aren't processes waiting then increase S's value.

[1]: http://www.cs.umd.edu/ shankar/412-Notes/10-BinarySemaphores.html