Alan Achtenberg

CSCE 410

Homework 2

11/4/14

1.

The fork() system call creates a new process and in the parent process returns the child pid, but in the child process returns 0. The getpid() function simply returns the pid of the current process. Therefore, In line A the outut will be “child: pid = 0”. In line B the output will be “child: pid1 = 2603”. In line C the output will be “parent: pid = 2603”. In line D the output will be “parent: pid1 = 2600”.

2.

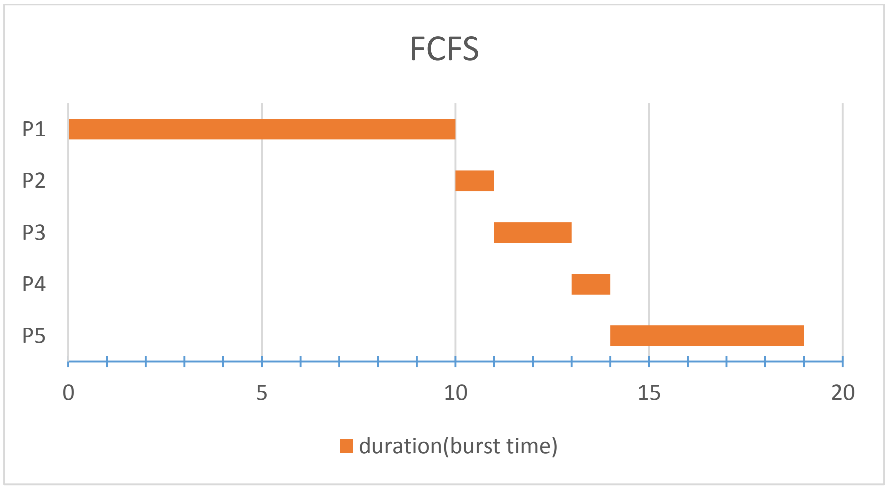
In the case the normal case user level threads will not perform better than a single threaded application, because without support from the kernel and kernel level threads, the user threads cannot be run on multiple processors at the same time. However in the case that the user level threads are backed by kernel level threads, it can definitely perform better in many cases. Some applications are not suited to being parallelized making it always possible that the single threaded version could be better.

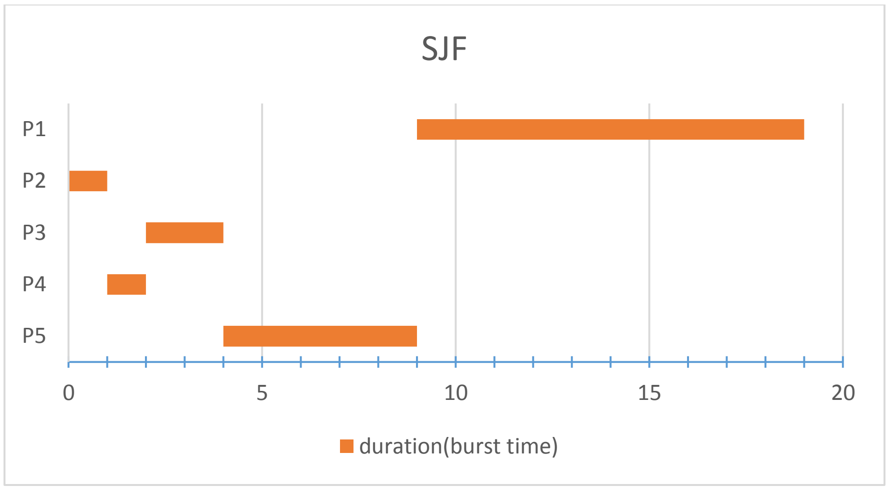
3.

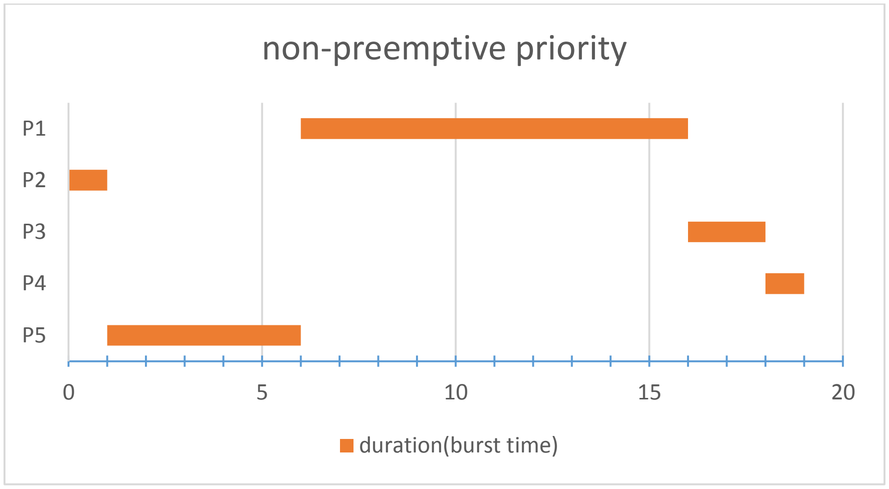
In line C the output will be “CHILD: value = 5”, because the thread created in the child process will modify the global variable value. In simple terms threads share the address space of their parent process. In line P the output will be “PARENT: value = 0” because as processes created with fork do not share the same address space. The child will modify its own variable value, while leaving the parents variable value alone.

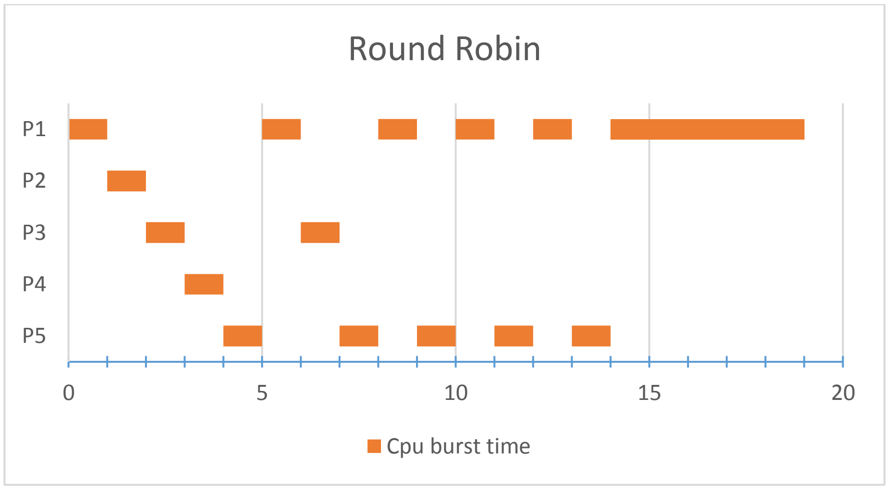
4.

a)




SJF
P1
P2
P3
P4
P5
0
5
10
15
20
duration(burst time)


non-preemptive priority
P1
P2
P3
P4
P5
0
5
10
15
20
duration(burst time)


Round Robin
P1
P2
P3
P4
P5
0
5
10
15
20
Cpu burst time

b)

	Waiting time			
Process	FCFS	SJF	Priority	RR
P1	0	9	6	9
P2	10	0	0	1
P3	11	2	16	5
P4	13	1	18	3
P5	14	4	1	9

c)

The shortest job first results in the smallest average waiting time. The reason why the shortest job first results in the shortest waiting time over round robin, is that the amount of time with the round robin you have to wait between process's is proportional to the number of process's being scheduled. Both the SJF and RR algorithms far outperform priority and FCFS algorithms in average waiting time.

5.

a) Spinlocks are not appropriate for single processor systems because they can cause deadlock, in a single processor system if you spin to wait for a resource you may possibly never release the processor for another thread to release said resource depending on the scheduling. Even if you had scheduling that would not cause deadlocks(with some quantum interrupt or decaying priority) the end result would be the same. You would want to switch to another thread that would release the resource.

The reason why it is used in multiprocessor systems, is because you can spinlock a thread on one processor while the resource is released on another. In many cases this can be faster than the overhead caused by the alternative of putting the thread to sleep and then awake on resource availability.

b) If a user level program is allowed to disable interrupts, then the timer interrupt can be disabled. If the timer interrupt is disabled context switches will also be disabled essentially allowing the user process to starve out the other processes.

6.

The wait() function decrements the semaphores value if the value is greater than 0 otherwise it waits. If wait function was not atomic in the case that you had two wait calls with an initial value of 1 it is possible that at the time of the condition both threads will see a value of 1 and both will decrement without waiting. If this happens the final value of the semaphore would be -1 and more than 1 thread will have acquired the lock.

7.

a)

Mutual exclusion, the intersections cannot be occupied by more than one vehicle at a time.

Hold and wait, the cars occupying one intersection are waiting for the next intersection to be clear before it will let the current intersection be unoccupied.

No preemption, the cars cannot be just removed by anyone.

Circular wait, the vehicle in the first intersection waits on the second to leave, the second waits on the third to leave, the third waits on the fourth to leave and the fourth waits on the first to leave. As you can see it completes a circle of waiting.

b)

Do not enter an intersection unless you can also leave it, ie. do not aquire a resource unless you can also release it.

8.

a)

Need Matrix				
	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

b)

It is in a safe state, P0 can run at any time, and P3 can run to completion freeing resources for P1 P2 and P4.

c)

Yes it can be granted immediately. The following sequence can run to completion, P0 P2 P3 P1 and P4