4.2 When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner.

4.4 No, because the operating system sees only a single process and will not schedule the different threads of the process on separate processors.

4.13

(a) The scheduler can only schedule user level processes to the kernel threads, and since some of the processes are not mapped to the kernel threads, they will be idle.

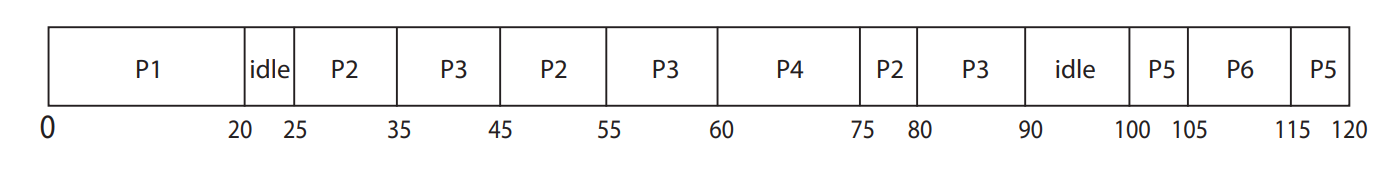
(b) All of the processes will be working simultaneously assuming there areenough user threads. If a kernel thread is blocked, it may be swappedout for one that isn’t blocked.

(c) When there are more kernel threads than processors, a blocked kernel thread could be swapped out in favor of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system.

5.6 Regressive round robin favours CPU-bound types of a process since those processes actively consume CPU time.

those types of processes will be rewarded up to 5 times by scheduler. The reasoning behind this is that if a CPU bound process consumes entire slice it had it was doing computation and processor was "active" the entire time. It is possible that such CPU bound process could finish soon so it is rewarded by scheduler.

5.8

(a) 

(b P1: 20-0 – 20

P2: 80-25 = 55

P3: 90-35 = 55

P4: 75-60 = 15

P5: 120-100 = 20

P6: 115-105 = 10

(c) P1: 0, p2: 40, P3: 35, P4: 0, P5: 10, P6:

(d) 105/120 = 87.5 percent

5.10 b, If a process with shorter burst-time is present in the queue, the CPU has to execute that particular process before moving on to other processes with larger Burst time. This may create a problem for processes with larger Burst time because a new process might come up when these processes are just about to go inside the Running State

d might seem a possible answer but Processes with same priorities are executed on a First-come first-serve basis which might solve the problem of starvation. Also, we can use Multi-level Feedback queues to solve the problem of Priority Scheduling.

5.15

(a) FCFS—discriminates against short jobs since any short jobs arriving after long jobs will have a longer waiting time.

(b) RR—treats all jobs equally (giving them equal bursts of CPU time) so short jobs will be able to leave the system faster since they will finish first.

(c) Multilevel feedback queues work similar to the RR algorithm— they discriminate favorably toward short jobs.

6.4 disabling interrupts only prevents other processes from executing on the processor in which interrupts were disabled; there are no limitations on what processes could be executing on other processors and therefore the process disabling interrupts cannot be used in user-level programs.

6.10 To prevent the diminishing efficiency of operating systems in such cases the concept of introducing spinlocks may be beneficial enough.

A lock that makes threads of a process enter into wait condition, while searching for availability of desired lock is known as a spinlock.

Spinlocks are taken for small durations and hence may not impact that much on the computer resources.

Also Introducing spinlocks may allow threads to divide processors actively according to their needs. Where the thread waiting for a spinlock may run on one processor and other threads can concurrently run over other processors without getting interrupted.

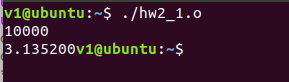
6.11

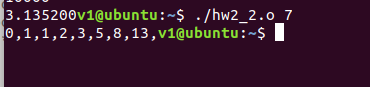
The lock is to be held for a short duration - It makes more sense to use a spinlock as it may in fact be faster than using a mutex lock which requires suspending –and awakening - the waiting process.

The lock is to be held for a long duration - a mutex lock is preferable as this allows the other processing core to schedule another process while the locked process waits.

A thread may be put to sleep while holding the lock - a mutex lock is definitely preferable as you wouldn’t want the waiting process to be spinning while waiting for the other process to wake up.

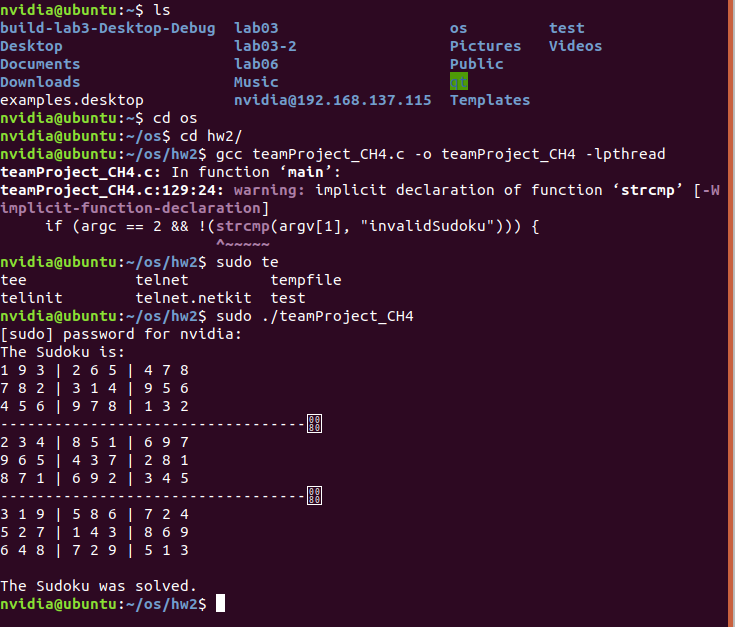
program

1. 

2. 

3. 

Team project:

Ch4: 

Ch6: 