資工三 110590029 陳思群

4.8 (1) Any kind of sequential program is not a good candidate to be thread.
An example is calculates an individual tax return.

(2) Another example is a shell program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open file, environment variables, and current working directory.

4.10 The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

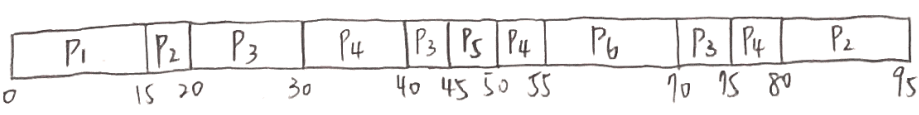
4.16
(1) Since the file must be accessed sequentially, the work of reading and writing it cannot reasonably be parallelized, and only a single thread should be used for each of these tasks.

(2) The CPU-intensive portion should be divided evenly among the four processors, so four threads should be created for this part of the program. Fewer than four would waste processor resources, while more threads would be unable to run simultaneously.

5.14
each processing core has its own run queue:
優：不易衝突，缺：管理不便

a single run queue is shared by all processing cores:
優：管理方便，缺：同步麻煩，可能會衝突

5.18
(a)



(b) P_1: 15-0=15, P_2: 95-15=80, P_3: 75-20=55, P_4: 80-30=50, P_5: 50-45=5, P_6: 70-55=15

(c) P_1: 0, P_2: 80-20=60, P_3: 70-30=40, P_4: 50-40=10, P_5: 0, P_6: 0

5.22

(a) Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context-switch. This results in a CPU utilization of $\frac{1}{1.1} \times 100 = 91\%$

(b) The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore $10 \times 1.1 + 10.1$ (I/O-bound + CPU-bound). The CPU utilization is therefore $\frac{20}{21.1} \times 100 = 94\%$

5.25

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

(b) 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.

6.7

(a) Top have a race condition.

(b) use Mutex Lock in push() and pop(), make sure its doesn't work at same time.

6.15 Synchronization primitives will make a program at user level to be able to switch interrupts, thereby disabling the timer interrupt. Once a program at user-level has the ability to disable interrupts it also disables context switching which is not recommended for a program at that level. Once the context switching is disabled, the program can now run on the processor without allowing other programs to execute, which is bad considering the fact that it is to be used in a single-processor system because single processor systems contain only one process, and only one program can run at a time.

6.18 Every mutex lock have a waiting queue, when process can't not use mutex lock, it will be putted in queue. when a process release mutex lock, it will remove and wake up first proces from waiting queue.