Hw #2

CH4

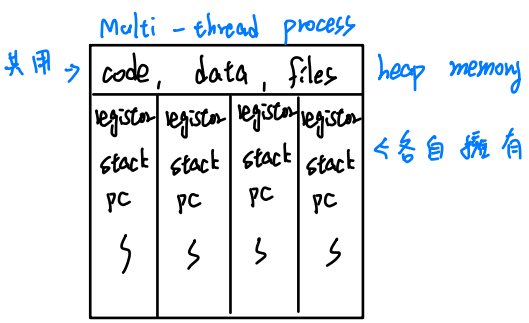
- 4.8 If we are running a constant time complexity program is separate thread, the overhead of creating the threads exceeds the tasks by them, thus decreasing performance when compared to single threaded alternative.

1. If the computer has only one physical processor core, then multi-threaded solution would not improve performance. Moreover, the additional over-head of context-switching may degrade performance.

2. Trivial operations on a list of numbers - Muti threading would not speed up the operations since the time taken by the operations is constant, and other element of the list may or may not wait for the previous to finish.

- 4.10

The threads of a multithread process share heap memory & global variable. Each thread has its separate set of register values and a separate stack.



- 4.16

(1) The work of reading and writing can not reasonably be parallelized because the file must be accessed sequentially; thus, only single thread should be used for each these tasks.

(2) The CPU-bound should be divided into 4 processors, so 4 thread should be used in these tasks. If threads are fewer than 4, it would waste precessor resources, and also more than 4 threads can not run correctly.

CH5

5.14 (1) Processing core

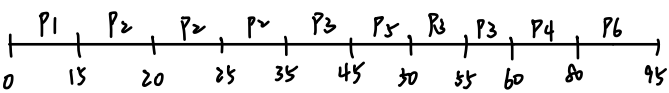
It has its own run queue; thus there is no contention over single run queue when scheduler is running on 2 or more processors. Also, the scheduler only need to look no further than its private run if scheduler must be made for a processing core.

(2) Single run

It must be protected with locks to prevent a race condition and a processing core may be available to run a thread, yet it must first require the lock to retrieve the thread from the single queue. When each processing core has its own run queue, it must has some sort of load balancing between the different run queue.

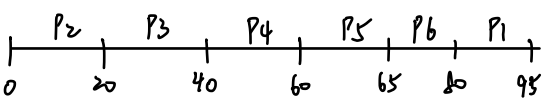
5.18 (a) Gantt chart

- Preemptive



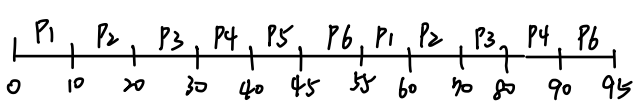
	Turnaround	waiting
P1	15	0
P2	35	15
P3	60	20
P4	80	35
P5	50	0
P6	95	25

- Priority – based



	Turnaround	waiting
P1	95	80
P2	20	0
P3	40	0
P4	60	15
P5	65	15
P6	80	10

- R-R (time quantum = 10)



	Turnaround	waiting
P1	60	45
P2	70	50
P3	80	40
P4	90	45
P5	45	-5
P6	95	25

5.22

(a) Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context switching cost for every context-switching. The result is $1/1.1 = 0.91 = 91\%$

(b) The IIo bound tasks incur a context switch after using up only 1 millionsecond of time quantum. Thus, it require to cycle through all process is $10 \times 1.1 + 10.1 = 21.1$. The result is $20/21.1 = 94\%$

5.25

(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 like RR algo, they discriminate favorably toward short jobs.

CH6

6.7

(a) If push( ) & pop( ) be used in the same time. Before stack[top] = item execute, pop( )'s top has already execute. 導致需要被推

6.15

If user-level program is given the ability to disable interrupts, then it can disable the timer interrupt and prevent context switching from taking place, thereby allowing it to use the processor without letting other processes to execute, which will led to process starvation.

6.18

The change would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue is Spinlocks Spinlock can be defined as the process of enabling a thread to wait while looking for lock that are available.
Spinlock is important as it enable thread to partition processor based on their needs so as to make use of one processor while other threads can as well run on another processor without hindering or interrupting ore another.