OSu HW2

110590004 資工三 林奕廷

Chap.4

4.8

1. If the program only contains a small workload, e.g. add 4 numbers together. There is no need to create multiple threads to deal with it since the thread creation overhead doesn't compare with the saved time.
2. When the tasks should be done sequentially, the threads created still need to wait its precedents to finish, which doesn't save any time at all.

4.10

(b) Heap memory and (c) Global variables are shared, (a) Register values and (d) Stack memory are not.

4.16

1. 1 Thread for I/O respectfully, since the I/O operation depends on the I/O device's ability, and usually 1 I/O device can handle 1 task at the same time, hence, using multithread to I/O won't enhance the performance.
2. For the CPU-intensive portion, we can assume the tasks are parallel computable, hence, I will use all 2*2 = 4 threads for computation.

5.14

(1)

Advantages: Reducing the overhead for data synchronization.

Disadvantageous: Hard to balance each processor's loading, if want to migrate processes to another processor to balance the load will be complex and have high overhead too.

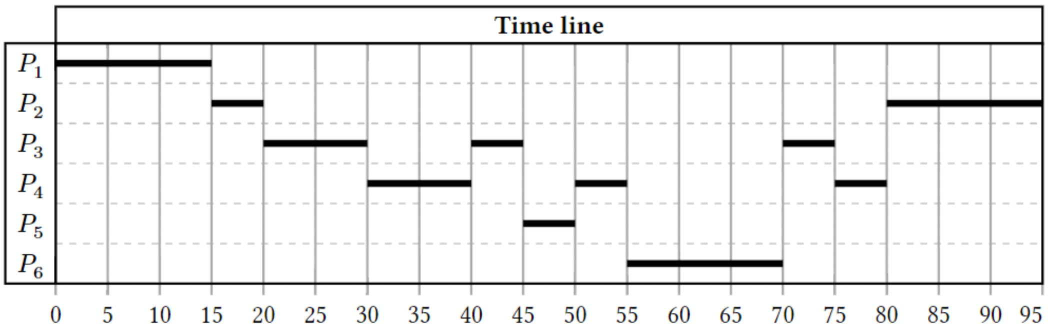
(2)

Advantages: Can better balance the load across processors.

Disadvantageous: It will increase the overhead for resource contention.

5.18

(a)



(b)

P_1: 15, P_2: 95, P_3: 55, P_4: 55, P_5: 5, P_6: 15

(c)

P_1: 0, P_2: 75, P_3: 35, P_4: 35, P_5: 0, P_6: 0

5.22

(a)

$$\frac{1+10}{(1+0.1)+(1+0.1)\cdot 10} = \frac{11}{12.1} = 90.9\%$$

(b)

$$\frac{10+10}{(10+0.1)+(1+0.1)\cdot 10} = \frac{20}{21.1} = 94.8\%$$

5.25

(a) No effect, since the process is being done in the coming order.

(b) Moderate, if the quantum time is set properly. The process with smaller bursts will be done earlier.

(c) Highest, By utilizing multiple queues with varying priorities and different quantum sizes for round-robin scheduling within each queue, the Multilevel feedback queue can significantly expedite the completion of shorter processes.

6.7

(a) The `top` pointer has a race condition when there is more than one process doing `push(item)`, which may cause two processes to push at the same position in the `stack`.

(b) Use semaphores to control the access of the stack. Use Mutex lock to lock the critical section of `push()` and `pop()`

6.15

When implementing synchronization primitives by disabling interrupts is not appropriate. Since the interrupts not only relate to internal events, the external events would be blocked too.

6.18

By implementing a waiting queue, and putting the task in when the lock is invalid. Every time the lock is free, pop a task from the queue to execute.