□Operating-system homework#2 □

Written exercises

• Chap.4

- 4.8: Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.
  - ex1 低負擔：
    如果只需要少量的運算，多執行序反而會有額外的開銷，導致效能下降 例如只需要 1+2+3+...+10 的總和，使用多執行序，會有額外的開銷。
  - ex2 串接性：
    如果需要等待前一個執行序的結果，才能繼續執行下一個執行序，使用多執行序，也不會較快，不僅需要等待前一個執行序的結果，還需要等待多執行序的開銷。

- 4.10: Which of the following components of program state are shared across threads in a multithreaded process?
  - (a) Register values
  - (b) Heap memory
  - (c) Global variables
  - (d) Stack memory

  b,c

- 4.16: A system with two dual-core processors has four processors available for scheduling – A CPU-intensive application is running on this system
  – All input is performed at program start-up, when a single file must be opened
  – Similarly, all output is performed just before the program terminates, when the program results must be written to a single file
  – Between start-up and termination, the program is entirely CPU-bound
  – Your task is to improve the performance of this application by multithreading it
  – The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread)
  - How many threads will you create to perform the input and output? Explain.
    2 threads, input and output 各一個 thread。
  - How many threads will you create for the CPU-intensive portion of the application? Explain
    4 threads, 每個核心一個 thread。

• Chap.5

- 5.14: Most scheduling algorithms maintain a run queue, which lists processes eligible to run on a processor. On multicore systems, there are two general options:
  – What are the advantages and disadvantages of each of these approaches? • (1) each processing core has its own run queue, or
  - advantages: 減少競爭且擴充性好
  - disadvantages: 需要較為複雜的管理

  • (2) a single run queue is shared by all processing cores.
  - advantages:
    方便管理

  - disadvantages:
    多個核心共用，可能會有競爭的問題

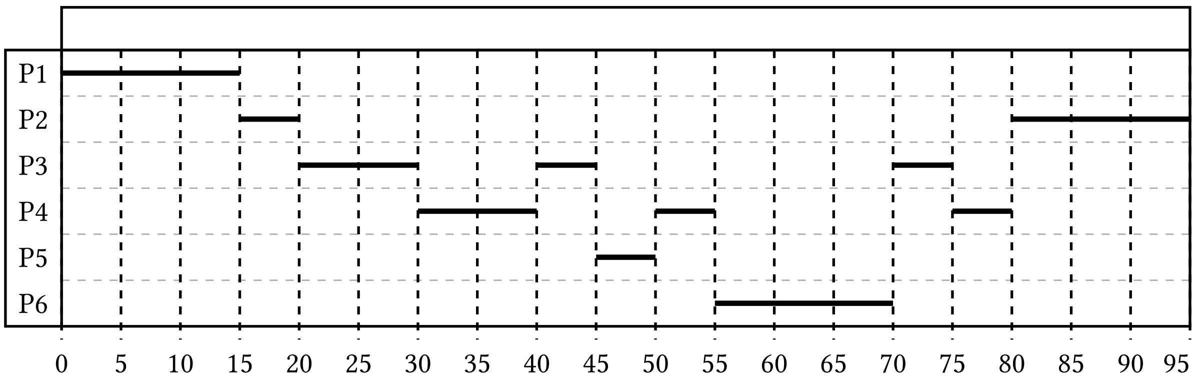
- 5.18: The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm.
  – Each process is assigned a numerical priority, with a higher number indicating a higher relative priority.
  - higher number = higher priority

  – For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units.
  - q = 10

  – If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

Thread	Priority	Burst	Arrival
P1	8	15	0
P2	3	20	0
P3	4	20	20
P4	4	20	25
P5	5	5	45
P6	5	15	55

  - (a) Show the scheduling order of the processes using a Gantt chart.



  - (b) What is the turnaround time for each process? 到達 結束時間
    P1: 15, P2: 95, P3:55, P4: 55, P5: 5, P6: 15
  - (c) What is the waiting time for each process?
    P1: 0, P2: 75, P3: 35, P4:35, P5: 0, P6: 0

- 5.22: Consider a system running ten I/O-bound tasks and one CPU-bound task.
  – Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete.
  – Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long-running tasks.
  – Describe the CPU utilization for a round-robin scheduler when:
  - (a) The time quantum is 1 millisecond
    - CPU time = 1m
    - CPU context = 0.1m
    - I/O time = 1m

    - I/O context = 0.1m

$$\frac{(\text{CPU time})\cdot 1+(\text{I/O time})\cdot 10}{(\text{CPU time + CPU context})\cdot 1+(\text{I/O time + I/O context})\cdot 10} = \frac{1\cdot 1+1\cdot 10}{(1+0.1)\cdot 1+(1+0.1)\cdot 10} = 90\%$$

  - (b) The time quantum is 10 millisecond
    - CPU time = 10m
    - CPU context = 0.1m
    - I/O time = 1m
    - I/O context = 0.1m

$$\frac{(\text{CPU time})\cdot 1+(\text{I/O time})\cdot 10}{(\text{CPU time + CPU context})\cdot 1+(\text{I/O time + I/O context})\cdot 10} = \frac{10\cdot 1+1\cdot 10}{(10+0.1)\cdot 1+(1+0.1)\cdot 10} = 94\%$$

- 5.25: Explain the differences in how much the following scheduling algorithms discriminate in favor of short processes:
  - (a) FCFS
    對於短進程不利，因為長進程會佔用 CPU，短進程需要等待。
  - (b) RR
    對於短進程有利，因為每個進程都有一定的時間片，短進程可以在時間片內完成。
  - (c) Multilevel feedback queues
    可以根據不同 queues 去切換 quantum，對應到不同程度的進程，可以慢慢提昇 quantum，去適應不同的進程。

- **Chap.6**

- 6.7: The pseudocode of Figure 6.15 illustrates the basic push() and pop() operations of an array-based stack. Assuming that this algorithm could be used in a concurrent environment, answer the following questions:
  - (a) What data have a race condition?
    如果同時 push 或同時 pop，可能會有 race condition
  - (b) How could the race condition be fixed
    使用 mutex lock，保證同一時間只有一個 thread 可以執行 push 或 pop

```
push(item) {
  if (top < SIZE) {
    stack[top] = item;
    top++;
  } else
    ERROR
}

pop() {
  if (!is_empty()) {
    top--;
    return stack[top];
  } else
    ERROR
}

is_empty() {
  if (top == 0){
    return TRUE;
  } else
    return FALSE;
}
```

- 6.15: Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.

在 single-processor 的情況下 disabling interrupts 不太適合，因為直接禁止使用 interrupts，代表著在執行結束前無法跳出，可能會導致程式無法結束、無法即時處理其他更重要的任務。

- 6.18: The implementation of mutex locks provided in Section 6.5 suffers from busy waiting.
  – Describe what changes would be necessary so that a process waiting to acquire a mutex lock would be blocked and placed into a waiting queue until the lock became available

```
Queue waitingQueue;

acquire() {
  if (!available) {
    waitingQueue.push(currentThread);
  }
  while (!available || waitingQueue[0] != currentThread)
    ; /* busy wait */
  available = false;
  waitingQueue.pop();
}
release() {
  available = true;
}
```