OS_110_CH5

Basic Concepts

- 為什麼要 scheduling，因為有 multiprogramming
- CPU-I/O burst cycle：一連串的執行是做什麼，就是甚麼 burst
  - 一連串長時間的 CPU 執行，就為 CPU burst
  - 通常會有
    - 大數量的短 CPU burst
    - 少數量的長 CPU burst

CPU - I/O Burst Cycle

- 大部分的 CPU burst 時長都非常短

CPU Scheduler

選擇 ready queue 裡面，誰先要執行

Preemptive vs. Non-preemptive

- 可以做 scheduler 的時間點
  - running to waiting ：在做 I/O，scheduler 就可以安排下一個
  - running to ready：Time sharing
  - waiting to ready：I/O 做完了，看要不要幫 process 插隊
  - Terminates
- Non-preemptive
  - 不打斷其他 process 的執行，要等別人做完，所以只有上述的 1 跟 4
- Preemptive
  - 會打斷別人，所以上述都會

Preemptive Issue

- 效能通常比較好
- 讓 CPU 的使用率比較高
- require process synchronization 需要同步進程
- Affect the design of OS kernel
  - 在 Kernel 裡面解決同步問題
  - 禁用 interrupt，就像是換成了 Non-preemptive 一樣，再也不會打斷其他 process

Dispatcher (知道就好瞜)

- 執行換人的動作
- example
  - switch context
  - jumping to the proper location in the selected program
- Dispatch latency 調度延遲
  - Scheduling time
  - Interrupt re-enabling time
  - Context switch time

Scheduling Algorithms

Scheduling Criteria

以系統的角度

- CPU utilization
  - 理論上： 0% ~ 100%
  - 實際上：40% ~ 90%
- Throughput
  + 每單位時間內平均完成 processes 的工作量

以單一 process 的角度

- Turnaround time
  - 一個 process 從進程到完成要多久的時間
  - submission ~ completion
- Waiting time
  - 在 ready queue 裡面要等多久時間結束
  - 完成時間 - 自己執行時間 - 抵達的時間
- Response time
  - submission ~ first reponse
  - 進去到開始執行的時間
  - 開始 - 抵達

FCFS Scheduling

- 先進先服務
- Convoy effect：有可能會有 short process 要等前面的 long process
  - 順序就是要看運氣

Shortest-Job-First (SJF) Scheduling

- 讓最短的先服務
- 可以能會有兩種可能
  - Non-preemptive
  - preemptive

Approxiamte SJF 近似 SJF

- SJF 很難實現
- 所以用預測的方式，exponential average
  - $T_{n+1} = a\,t_n (1-a)\,T_n$ ：t_n is new one , T_n is history
  - Commonly a = 1/2，兩個的平均

Priority Scheduling

- 靠著當下的優先度，決定誰要執行\

- 其實 SJF 也算是一個 proority scheduling

- 也有問題
  - Preemptive
  - Non-preemptive
- 會有 **Starvation**，就是優先度太低，可能永遠執行不到
  - Sol: **aging** 在一段時間，增加未執行的優先度

Round-Robin (RR) Scheduling

- 給定一段時間，每個 process 輪流，沒做完的就要重新排隊
- Performance
  - TQ large → **FIFO**
  - TQ small → (context switch) **overhead** increases
- 可以給你最好的 response time

Multilevel Queue Scheduling

- 每種 Queue 放不同類型的 process，就有不同種類的優先度
- 用機率的方式去控制每個種類被選中的機率，優先度越高，機率就越大
  - 可以解決 startvation

Multilevel Feedback Queue Scheduling

- 一樣是有好幾個 Queue，會將優先度最高做完，才會繼續往下
  - 需要有 aging
- Process 會在 Queue 裡面移動，在 run time 的時候可以用時間排程
  - 有人做太久，下次就會將他往後排，將 schedule information 記載 PCB
- 也有舉例是，在最後面放 FCFS

Exaluation Methods

- Deterministic modeling : 先確定注重的點是什麼，選擇哪一種的演算法
- Queueing model : 數理分析

- Simulation : 隨機數模擬
- Implementation : 實作之後，繼續觀察

Special Scheduling Issues

Multi-Processor Scheduling

Asymmetric multiprocessing 對稱

- 會有一個 process 管理，但該 process 也只能做管理，不能做其他事情
- 會比 SMP 還要簡單

Symmetric multiprocessing (SMP) 非對稱

- 所有 process 去競爭
- 需要有同步的機制
- 較常見

Processor affinity

- affinity 會將一個 process 綁在一個 CPU core 上
  - 好處：可以 reuse cache，會很快

NUMA and CPU

NUMA (non-uniform memory access)

Memory Access Architecture

Load-balancing

- Push migration : 將重負載的 processes 移到輕負載的 processor 上
- Pull migration : 輕負載的 processor 自己要工作做
- Load balancing often counteracts (抵銷) the benefits of processor affinity

Multi-Core Processor Scheduling

前情提要

- Multi-core Processor
  - 快且消耗低
  - memory stall : 當要訪問記憶體，會花很多時間載入資料
- Multi-threaded multi-core sysytems
  - 一個 core 會有多個線程

開始

- coarse-grained : 會把 pipe line 將 flushed 掉
- fine-grained

Real-Time Scheduling

- keeping deadlines，有期限的意思

- 有分軟體跟硬體

- Rate-Monotonic (RM) algorithm
  - Shorter period → higher priority 誰短誰優先
- Earliest-Deadline-First (EDF) algorithm
  - Earlier deadline → higher priority 誰先過期誰優先

Scheduling Case Study

Solaris Scheduler

- Priority-based
- real-time, system, time sharing, interactivem fair share, fixed priority

Windows XP Scheduler

- Priority-based
- RR

Linux Scheduler

- Static priorities