作業系統
Operating System

TZU-CHUN HSU[1]

[1]vm3y3rmp40719@gmail.com

[1]Department of Computer Science, Zhejiang University

2020 年 10 月 19 日
Version 1.0

Disclaimer

本文「作業系統」為台灣研究所考試入學的「作業系統」考科使用，內容主要參考洪逸先生的作業系統參考書 [1]，以及 wjungle 網友在 PTT 論壇上提供的資料結構筆記 [2]。本文作者為 TZU-CHUN HSU，本文及其 LaTeX 相關程式碼採用 **MIT 協議**，更多內容請訪問作者之 GitHub 分頁Oscarshu0719。

1 Overview

1. 本文頁碼標記依照實體書 [1] 的頁碼。

2. TKB 筆記 [2] 章節頁碼：

Chapter	Page No.	Importance
1	3	⋆ ⋆
2	15	⋆ ⋆
3	25	⋆ ⋆
4	34	⋆ ⋆ ⋆ ⋆ ⋆
5	99	⋆ ⋆ ⋆
6	119	⋆ ⋆ ⋆ ⋆ ⋆
7	175	⋆ ⋆ ⋆
8	197	⋆ ⋆ ⋆ ⋆ ⋆
9	221	⋆ ⋆ ⋆
10	221	⋆

2 Summary

1. **Theorem (7, 8, 10, 12)** System types：

   - Multiprogramming system：
     - Multiprogramming degree：存在系統內執行的 process 個數。
     - 執行方式：
       * Concurrent（並行）：CPU 在多個 processes 之間切換，同一時間只有一個 process 執行。
       * Parallel（平行）：同一時間多個 processes 在不同 CPU 上執行。
   - Time-sharing (Multitasking) system：
     - 屬於 Multiprogramming 的一種。
     - CPU 頻繁在任務間切換，response time 短，適合 interactive computing，經常採用 Round-Robin scheduling。
     - 採用 virtual memory。
     - 採用 spooling (Simultaneous Peripheral Operations On-Line processing)，將 disk 當作 buffer，將輸出檔先儲存在 disk，直到 device 取走，達到多個 I/O devices 的效果。
   - Multiprocessors (Tightly-coupled, Parallel) system：
     - Processors 溝通通常採用 shared memory。
     - Graceful degradation (Fail-soft)：系統不會因為某些軟硬體元件故障而停頓。具備此能力的系統稱為 Fault-tolerant。
     - Multicores，即一晶片多核，相較 multiprocessors 速度快且省電。
     - 類型：
       * SMP (Symmetric MultiProcessors)：所有 processors 能力與權利相同，trough-put 與 reliability 較 ASMP 好，但 OS 設計較 ASMP 複雜。
       * ASMP (ASymmetric MultiProcessors)：Master-slave 架構，master 負責工作與資源分配，其餘為 slave。
   - Distributed (Loosely-coupled) system：
     - Processors 的 clock 和 OS 未必相同。
     - Processors 溝通通常採用 message passing，透過 network 連接。
     - C/S model 達到 resources sharing；P2P (Peer-to-peer) 不區分 C/S。

- Real-time system：
  - 類型：
    - Hard real-time system：
      - 若工作未在 deadline 前完成，即為失敗。
      - 處理時間過長或無法預測的設備少用，disk 少用，virtual memory 不用。
      - 不與 time-sharing system 並存。
      - 減少 kernel 介入時間。
    - Soft real-time sharing：
      - real-time processes 必須持續保有最高 priority 直到結束。
      - 必須支援 preemptive priority scheduling。
      - 不提供 Aging。
      - 減少 kernel 的 dispatch latency。
      - 可與 virtual memory 共存，但是 real-time processes 不能被 swapped out 直到完工。
      - 可與 time-sharing 並存。
      - 現行 OS 皆支援。
- Batch system：非急迫或週期性的任務一起執行，提高資源使用率，不適用 real-time 及 user-interactive。

2. **Theorem (16, 17)** I/O 運作方式：
   - Polling (Busy-waiting, Programmed)：CPU 需不斷 polling I/O device controller 直到完成。
   - Interrupted：
     - I/O 完成時，I/O device controller 會 interrupt，使 CPU 暫停目前 process 放到 ready state queue。
     - CPU 親自監督 I/O 完成。
     - 若頻繁 interrupt CPU 使用率仍會很低，因此若 I/O 時間不長，polling 反而可能較有利。
     - 分類：
       - External interrupt：CPU 外的周邊設備發出，例如 device controller 發出 I/O-completed。

- Internal interrupt：CPU 執行 process 遭遇重大 error，例如 Divide-by-zero，執行 priviledged instruction in user mode。
- Software interrupt：Process 需要 OS 提供服務，呼叫 system call。

* Interrupt：Hardware-generated，例如 device controller 發出 I/O-completed。
- Trap：Software-generated。Catch arithmetic error，即 CPU 執行 process 遭遇重大 error，例如 Divide-by-zero。Process 需要 OS 提供服務，會先發 trap 通知 OS。

* Interrupt 要分 priority。
- Non-maskable interrupt：通常是重大 error 引起，需要立即處理，例如 internal interrupt。
- Maskable interrupt：可以忽略或是延後處理，例如 software interrupt。

- DMA (Direct Memory Access)：
  - DMA controller 代替 CPU 負責 I/O 與 memory 間的傳輸。
  - 適用 block-transfer。
  - interrupt 頻率較低，但設計較複雜。
  - 造成 resource contention（IF、MEM 和 WB 週期），因此採用 interleaving (cycle stealing)，與 CPU 輪流使用 memory 與 bus。
  - DMA 比 CPU 有更高 priority (SJF)。

3. **Theorem (18)**
   - Non-blocking I/O 與 Asynchronous I/O 差異：前者會有多少通知多少，後者會等 I/O 全部完成才通知 process。
   - I/O 種類：
     - Memory-mapped I/O：無專門 I/O 指令，但特別分一塊 memory 給 I/O，寫入該塊 memory 視為 I/O 操作。
     - Isolated I/O：有專門 I/O 指令。
4. **Theorem (22)** CPU protection：利用 timer，時間到 timer 發出 timer-out interrupt 通知 OS，讓 OS 強制取回 CPU。
5. **Theorem (34)** Virtual machine：
   - Host：Underlying hardware system.
   - Virtual Machine Manager (VMM, Hypervisor).

- Guest：Process provided by VM, for example, OS, applications.
- Implementations：
  - Type 0：Hardware.
  - Type 1：Kernel mode.
    - OS-like software：只提供 virtualization。
    - General-purpose OS 在 kernel mode 提供 VMM services。
  - Type 2：User mode：Applications that provide VMM services.
- Virtualization variations：
  - Paravirtualization：Present guest similar but NOT identical to host hardware. Guest should be modified to run on paravirtualized hardware.
  - Emulators：Applications to run on a different hardware environment.
  - Application containment (container)：Not virtualization at all, but provides segregating applications from the OS.
- Java Virtual Machine (JVM)：只提供規格（class loader，class verifier 和 Java interpreter），並非實現。
- 優點：
  - 提供 Cloud computing。
  - 測試中的 OS 掛了，視為一個 user process 掛掉，其他 user processes 以及 real system 不受影響。
- 缺點：
  - 不易開發 VMM，因為要複製與底層 host hardware 一模一樣的 VM 非常困難，例如 modes control and transition、資源調度和 I/O device and controller 之模擬。
  - 效能比 host hardware 差。

6. **Theorem (52)**
   - Software as a service（SaaS）：Applications, e.g. Dropbox, Gmail.
   - Platform as a service（PaaS）：Software stack, i.e. APIs for softwares, e.g. DB server.
   - Infrastructure as a service（IaaS）：Servers or storage, e.g. storage for backup.
7. **Theorem (61)** Process control block (PCB)：

- Process ID.
- Prcoess state.
- CPU register (stack top pointer, accumulator).
- Program Counter (PC).
- CPU scheduling info (PCB pointer, process priority).
- Memory management info (base/limit registers, page table).
- Accounting info (CPU time).
- I/O-status info (Uncompleted I/O request, allocated I/O devices).

8. **Theorem (60, 61)** Process life cycles (1)：
   - State：
     - New (Created)：Process is created and PCB is allocated in kernel, but memory space is NOT allocated.
     - Ready：Prcoess is allocated memory space, being put in ready queue. Ready for execution, being able to get CPU allocation.
     - Waiting (Block, Sleep in memory)：Waiting for some event to occur, being put in waiting queue, e.g. waiting for I/O-completed. Unable to get CPU allocation.
     - Terminated (Exit, Zombie)：Process is finished.
   - Transition：
     - Admitted：Prcoess is allocated memory space. In batch system, use long-term scheduler to decide which process to load into memory. In time-sharing and real-time systems, do NOT use long-term scheduler.
     - Dispatch：Short-term (CPU) scheduler decides which process to get CPU allocation and allocates CPU to execute.
     - Exit：Process is finished, releasing resources.
     - Interrupt (Time-out)：Process release CPU, putting on ready queue, e.g. time-out interrupt.
     - I/O event wait：Process waits for I/O-completed or event occurs.
     - I/O-completed or event occurs：I/O-completed or event occurs, putting in ready queue.
     - Only in Stalling version (2)：

* Swap out：Out of memory, and other process needs more space. Medium-term scheduler can choose some **Blocked** processes to swap out to disk with their process image being saved.
* Swap in：Sufficient memory space. Medium-term scheduler can choose some Suspended/Ready processed to swap in from disk to memory, putting in ready state.
* Suspend：Out of memory even all Blocked processes are swapped out or Blocked processes got higher priority than Ready processes. Swap out some **Ready** processes to get sufficient memory space.
* (Poor design) Running to Suspended/Ready：If a higher priority Suspended/Block process becomes Ready, kernel can force a lower priority Running process to release CPU and memory space, putting the former in running queue.
* (Poor design) Suspended/Block to Suspended/Ready：If as Suspended/Blocked got higher priority than a Running process, the latter release CPU and memory space, swapping in the former and get CPU allocation.

- Zombie state：Process is finished, but the parent process have NOT collected results of the children processes, or parent process have NOT executed wait() system call. Resources are released, but PCB have NOT been deleted, until parent process collects the results, then kernel delete PCB.

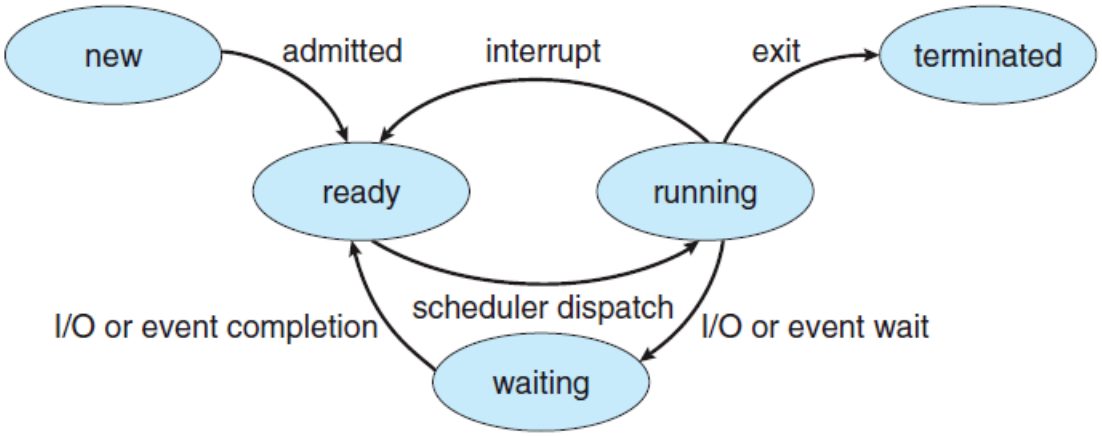

Figure 1: Process life cycles.

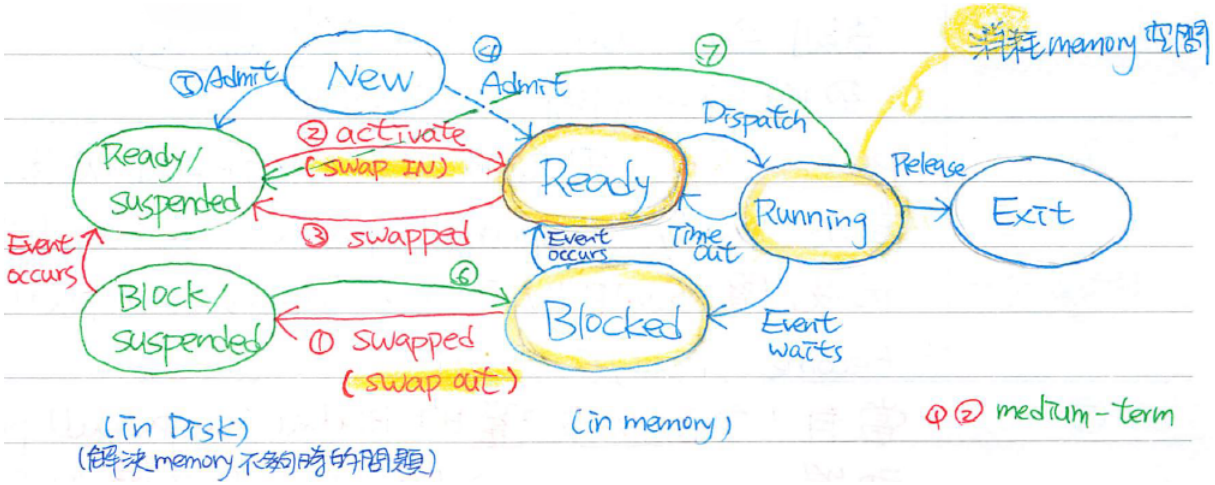

Figure 2: Process life cycles (Stalling version).

9. **Theorem (67)** Scheduler:

   - Long-term (Job) scheduler：通常 batch system 採用，time-sharing 與 real-time systems 不採用，從 job queue 中選 jobs 載入 memory。執行頻率最低，可以調控 multiprogramming degree 與 CPU-bound 與 I/O-bound jobs 的比例。
   - Short-term (CPU, process) scheduler：從 ready queue 選擇**一個** process 分派給 CPU 執行。所有系統都需要，執行頻率最高，**無法**調控 multiprogramming degree 與 CPU-bound 與 I/O-bound jobs 的比例。
   - Medium-term scheduler：Memory space 不足且有其他 processes 需要更多 memory 時執行，選擇 Blocked 或 lower priority process swap out to disk。Time-sharing system 採用，batch 和 real-time systems 不採用，可以調控 multiprogramming degree 與 CPU-bound 與 I/O-bound jobs 的比例。

10. **Theorem (69, 70)**

    - Context switch:
      - 執行期間無法執行 process，主要取決於硬體因素。
      - 降低負擔:
        * 提供 Multiple registers set：每個 process 有自己的 registers set，只需要切換 pointer 就能 context switch。
        * 使用 Multi-threading。
    - Dispatcher:

- 將 CPU 真正分配給 CPU scheduler 選擇的 process。
- Context switch.
- Switch mode to user mode.
- Jump to execution entry of process.

11. **Theorem (63)** Process control operation：
    - `fork()`：
      - child process 有與 parent process 不同的 memory space，而起始 code section 和 data section 皆來自 parent process 的複製。
      - 失敗：回傳負值；成功：回傳 0 給 child process，> 0 值即 child process PID 給 parent process。
    - `wait()`：若 child process 已終止，帶 parent process 還沒執行 $wait()$，但 kernel 含不能清除 child process PCB，直到 parent process 收集完 child process info，此段期間稱 zombie。
    - `execlp(dir, filename, args)`：載入特定工作執行，memory content 不再是 parent process 的複製，沒有參數填 `NULL`，e.g. `execlp("/bin/ls", "ls", NULL)`。
12. **Theorem (70)** 評估 scheduling performance：
    - CPU utilization：$\frac{CPU\ process\ execution\ time}{CPU\ total\ time}$。
    - Throughput：單位時間完成 process 數量。
    - Waiting time：Process 在 **ready queue** 的時間。
    - Turnaround：Process 進入系統到完成工作的時間。
    - Response time：user 輸入到系統產生第一個回應的時間。
13. **Theorem (75)**
    - Starvation：
      - Process 長期無法取得完成工作所需資源，因此遲遲無法完成，形成 indefinite blocking。
      - 容易發生在不公平或是 preemptive 的 scheduling。
      - 有機會完成但是機會很小。

  - 通過 Aging 解決：當 process 在系統中時間增加，逐漸提升 process 的 prioirty。但是 soft real-time systems 不能使用 Aging，因為會違反定義。
- Non-preemptive (Cooperative)：
  - 完成工作時間較可預期。
  - Context switch 次數較少。
  - 較少發生 Race condition，特別是 kernel。
  - Scheduling performance 較差，平均等待時間較長。
  - Real-time 和 time-sharing systems 不適合。
- Preemptive：
  - 大部分 OS 在 kernel mode 採用，防止 race condition。
  - 對 soft real-time systems 不利，例如 kernel 執行時

14. **Theorem (71, 72, 75, 76, 77, 78)** Scheduling algorithms：
    - FCFS (First-come-first-serve)：可能遭遇 Convoy effect，即許多 processes 都在等待一個需要很長 CPU time 的 process 完成工作。
    - SJF (Shortest-Job-First)：
      - Preemptive SJF 又稱 SRTF (Shortest-Remaining Time First)。
      - 效益最佳（包含 SRTF），即平均等待時間最短。
      - 對 long-burst time job，可能有 starvation。
      - 不適合用在 CPU (short-term) scheduling，因為不知道 process 的精確 next CPU burst time，短時間也難以精準預估。
      - Long-term job 可能可行。
      - Exponential Average：預估 next CPU burst time。

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \tag{1}$$

        其中，t_n 為本次 CPU 實際值，α 為機率。
    - Priority scheduling：FCFS, SJF, SRTF $\subset$ Priority scheduling
    - Round-Robin (RR) scheduling：
      - Time-sharing system 採用。
      - FCFS $\subset$ RR

  - 效能取決於 time quantum 大小，太小 context switch 太頻繁。
  - Time quantum 大小會影響 turnaround time，平均 turnaround time **未必**會隨著 quantum time 增加而下降。
- Multi-level queue：
  - Queue 之間採用 fixed priority preemptive scheduling 或 RR。
  - 每個 queue 有各自的 scheduling。
  - process 不能在 queues 之間移動，因此缺乏彈性。
  - 易 starvation，且無法通過類似 Aging 改善。
- Multi-level feedback queue：
  - 允許 process 在 queues 間移動，可降級增加彈性。
  - No starvation，可以採用 Aging 防止 starvation。
  - 所有 scheduling superset。
- Combination of RR and Priority scheduling：多個 processes 有相同 priority 時，採用 RR。

Scheduling	公平	Preemptive	Non-preemptive	Starvation
FCFS	√		√	
SJF			√	√
SJF		√		√
Priority		√	√	√
RR	√	√		
Multi-level queue		√		√
Multi-level feedback queue		√		

15. **Theorem (79, 80)** 特殊系統 scheduling：

- Multiprocessors system：
  - 包含多 CPUs，Multicores，NUMA system，Heterogeneous（手機上多核 CPU）system，Multithreaded（硬體）system。
  - 多 CPUs：
    * ASMP：類似 single-CPU scheduling。
    * SMP：分 2 approaches：
      · 所有 CPU 共用一條 ready queue，no load balancing problem，須防止 race condition。

- 每一個 CPU 有各自 ready queue，可能有 load balancing problem，可以通過 kernel 協調 imbalance CPUs 給其他 ideal CPUs processes。

– Processor affinity：
  * 一但 process 在某 CPU 上執行，盡量避免 migration。
  * 因為 migration from one CPU to another，first CPU cache should be invalidated and second CPU cache should be repopulated，it cost a lot。
  * Migrating a process may incur a penalty on NUMA systems, where a process may be moved to a CPU that requires longer memory access time。

– Multicores：有 Memory stall problem。
  * 通過 Multithreaoded processing cores 解決，2 or more hardware threads are assigned to each core。
  * 當一 thread memory stall，core can switch to another thread。
  * 每個 thread OS 視為 logical CPU，皆可執行 software thread (process)，稱為 Cheap Multithreading Technology (CMT)。

• Real-time system：
  – Hard：
    * Minimize latency：包含 interrupt latency 和 dispatch latency，real-time system 不適合 non-preemptive；若是 preemptive，須防止 race condition。
    * Prioity inversion：Higher priority process waits for lower priority process to release resources.

References

[1] 洪逸. 作業系統金寶書（含系統程式）. 鼎茂圖書出版股份有限公司, 4 edition, 2019.

[2] wjungle@ptt. Tkb 筆記. https://drive.google.com/file/d/0B8-2o6L73Q2VelFZaXpBVGx2aWM/view?usp=sharing, 2017.