2017 Operating System Final (CLD)

*Warning: This documentation is drafted with pure memory, which is often inaccurate.*

1. Please list 5 scheduling criteria.
2. Please draw the Gantt charts and calculate the average waiting time for the following process for the problems shown below.
   1. FCFS
   2. RR
   3. Shortest-Remaining-Time-First

|  | Arrival Time | Burst Time |
| --- | --- | --- |
| P1 | 0 | 6 |
| P2 | 3 | 4 |
| P3 | 2 | 3 |
| P4 | 5 | 5 |
| P5 | 8 | 2 |

1. Please define what is the condition of starvation in operating system and explain how to avoid this condition.
2. Please explain multiple feedback queue.
3. Please explain the critical-section.
4. Please explain how “disable interrupt” and “critical-section problem” solve race condition problem.
5. Please list and explain the 3 criteria of critical-section design (solution to critical-section problem).
6. Peterson’s solution is a classic software based solution to the critical-section problem. Please fill the blanks in the Peterson’s solution.

| do {  \_\_\_\_\_\_\_\_\_\_;  \_\_\_\_\_\_\_\_\_\_;  \_\_\_\_\_\_\_\_\_\_;  // critical section  flag[i] = FALSE;  // remainder section  } while (TRUE); |
| --- |

1. UNKNOWN
2. What is the difference between mutex and semaphore?
3. Please explain 4 necessary conditions for “deadlock”.
4. Please explain 3 solutions to “deadlock”.
5. Please explain 2 ways deadlock recovery works.
6. Consider a system with processes, through , and resource types, and . Suppose that at time the following snapshot of a system has been taken. According to the snapshot below. Please answer the following questions and explain why.
   1. The system is in a safe state. Does the sequence satisfy the safety criteria?
   2. The system is in a safe state. Does the sequence satisfy the safety criteria?
   3. If a request from process arrives for , can the request be granted immediately?
   4. If a request from process arrives for , can the request be granted immediately?

|  | Allocation | | | | Max | | | | Available | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 2 | 1 | 5 | 2 | 0 |
|  | 1 | 0 | 0 | 0 | 1 | 7 | 5 | 0 |  | | | |
|  | 1 | 3 | 5 | 4 | 2 | 3 | 5 | 6 |
|  | 0 | 6 | 3 | 2 | 0 | 6 | 5 | 2 |
|  | 0 | 0 | 1 | 4 | 0 | 6 | 5 | 6 |

1. Consider a system consisting of 3 processes, and , each accessing 3 semaphores set to the value . ***(incomplete question)***

|  |
| --- |

* 1. ( ) The system always has deadlock.
  2. ( ) The system has a race condition.
  3. ( ) The system is always in safe state.

# Answer

1. (Out of Testing Range)
2. (Out of Testing Range)
3. (Out of Testing Range)
4. (Out of Testing Range)
5. 對共享變數之存、取進行管制，當 取得共享變數存、取機制，在它尚未完成期間，任何其他 process 無法存取共享變數，即使它們取得 CPU。
   1. **Disable Interrupt:**Process 在對共享變數存取之前，先 disable interrupt，等到完成共享變數存取後，才 enable interrupt，如此可保證 process 在存取共享變數的期間，CPU 不會被 preempted，即此一存取是 atomically executed，所以可以防止 race condition。
   2. **Critical Section Problem:**  
      對共享變數之存、取進行管制，當 取得共享變數存、取機制，在某一 process 尚未完成期間，任何其他 process 無法存取共享變數，即使它們取得CPU。
   3. **Mutual Exclusive:**  
      在任何期間點，最多只允許一個 process進入它自己的 critical section，不可有多個 processes 分別進入各自的 critical section。
   4. **Progress:**  
      須滿足
      1. 不想進入 critical section 的 process 不參與進入 critical section 之決策。
      2. 從那些想進入 critical section 的 processes 中，決定誰可以進入 critical section 的決策時間是有限的 (不可以無窮)，即 no deadlock (不可以 waiting forever，不可以大家皆無法進入 critical section)。
   5. **Bounded Waiting:**自某 process 提出申請到核准進入critical section 的等待時間是有限的，即若有 個 processes 想進入 critical section，則任一 process 最多等 次後即可進入 critical section ，即 no starvation，須公平對待。
   6. flag[i] = TRUE;
   7. turn = j;
   8. while (flag[j] && turn == j);
6. UNKNOWN
7. 1. **Mutex:**

像是持有資源的鎖，當 process 持有 mutex，便可進入 critical section 中 access 資源，離開時再釋放mutex，讓其它 process 進入。

* 1. **Semaphore:**Generalized mutex，計算想進入到 critical section 的 processes 數量，當有 process 離開 critical section 時讓下個等待中的 process 進入 critical section。
* [Mutex vs. semaphore](https://www.geeksforgeeks.org/mutex-vs-semaphore/)
  1. **Mutual Exclusion:**  
     對 resource 而言，具有此性質的 resource，在任何時間點最多只允許一個 process 持有或使用，不可多個 processes 同時持有或使用。
  2. **Hold & Wait:**  
     Process 持有部分資源且又在等待其他 processes 所持有之資源。
  3. **No Preemption:**Process 不可任意剝奪其他 process 所持有之資源，必須等對方釋放資源後，才有機會取得資源。
  4. **Circuit Waiting:**  
     系統中存在一組 processes 形成循環等待之情況。
  5. **Deadlock Prevention**避免可能造成 deadlocks 的 request 被執行，防止四個必要條件之一成立即可。
  6. **Deadlock Avoidance**
     1. **For Single-instance**

利用RAG搭配claim edge，在claim edge轉為assignment edge後，只要沒有cycle產生，代表在safe state中

* + 1. **For Multiple-instance**假設系統核准此次 request，執行 Banker’s Algorithm，確認是否還仍在 safe state，若沒有就否決，要求 process 得等一陣子再 request。
  1. **Deadlock Detection & Recovery**用 detection algorithm 每隔一段時間偵測系統中有沒有 deadlock，如果有就要解開 deadlock(做recovery)。
  2. **Kill Process in the Deadlock:**
     1. (法一) Kill all processes in the deadlock → cost 太高，先前的工作成果全部作廢。
     2. (法二) Kill process one by one，kill 一個之後，需再跑 detection algorithm。若死結仍存在，再 repeat 上述步驟，cost亦高。
  3. **Resource Preemption:**
     1. 選擇 victim process。
     2. 剝奪他們身上的資源。
     3. 回復此 victim process 當初未取得此剝奪資源的狀態。

1. 1. Yes.  
      因為在此 sequence 中，後面的 processes 的需求可由 available 及前面的 process 的 resource 所滿足。
   2. Yes.
   3. Yes.  
      因為假設接受此 request 後，可以找到一組 safe sequence，因此不會進入 unsafe state，故此 request 可被接受。
   4. Yes.  
      因為假設接受此 request 後，可以找到一組 safe sequence (如：)，因此不會進入 unsafe state，故此 request 可被接受。
   5. False.
   6. Flase.
   7. False.