FIT2100
Semester 2 2019
Revision Questions
Suggested Solutions for selected questions

October 19, 2019

Revision Status:

`$Id: FIT2100-Sample-Questions.tex, Version 2.0 2018/10/25 13:05 Jojo $`

`Updated by Daniel Kos, Oct 2019`

Contents

1 Part I: Review Questions

Question 1

Describe the two general roles of an operating system, and elaborate why these roles are important.

Answer:

(a) high-level abstraction — more friendly to programmers; provides common core for all applications; hides details of hardware to make application code portable

(b) resource manager — divides resources amongst competing programs (processes) or users according to some system policy to ensure fairness and no starvation where possible.

Question 2

Describe the *three-state process model*, describe what transitions are valid between the three states, and describe an event that might cause such a transition.

Question 3

What is a *process*? What are the attributes of a process?

Answers:

- A program in execution; an instance of running program; an owner of the resource allocated to program execution; a task or a job; encompass one or more threads.
- Possible attributes of a process: Address Space; Process ID; Process Group; Parent Process; signals; etc.
- Possible attributes of threads: Scheduling Parameters; Registers; Program Counter; Program Status Word; Stack Pointer; Thread state.
- If the OS does not support multi-threaded applications, then thread attributes become process attributes.

Question 4

What is the function of the *ready queue*?

Question 5

What is the relationship (or differences) between threads and processes?

Question 6

Name the advantages and disadvantages of *user-level* and *kernel-level* threads.

Question 7

Describe the *process control block* and the details of information it maintains?

Question 8

What is a *race condition*? Give an example.

Answer:

- A race condition occurs when there is an uncoordinated access to resources (e.g. memory or device registers), which leads to an incorrect behavior of the program, deadlocks or lost work.
- An example would be two process updating a counter simultaneously. Say the counter has a value of 1. Process A reads the variable but before updating Process B reads and updates the counter to 2. Process A, having no knowledge of what Process B does, updates the counter to 2. The correct behavior of the program would have the counter being incremented to 3, not 2.

Question 9

What is a *critical section*? How do they relate to controlling access to shared resources?

Question 10

What is *deadlock*? What is *starvation*? How do they differ from each other?

Question 11

What are the four conditions required for deadlock to occur?

Question 12

Describe general strategies for dealing with deadlocks.

Question 13

For single unit resources, we can model resource allocation and requests as a *directed graph* connecting processes and resources. Given such a graph, what is involved in deadlock detection?

Answer:

- When given a resource graph, you simply have to look at the graph to see if there are loops in the graph.
- The system in a resource graph is not deadlocked as long as there is no loop between the graphed resources. One or more loops means the processes involved the loop are deadlocked.

Question 14

Assuming the operating system detects the system is deadlocked, what can the operating system do to recover from deadlock?

Question 15

What must the *banker's algorithm* know a priori in order to prevent deadlock?

Question 16

Describe the general strategy behind *deadlock prevention*, and give an example of a practical deadlock prevention method.

Answer:

- The general approach of prevention is to negate one of the conditions required for deadlock (careful allocation is deadlock avoidance).
- A practical method is ordered resource acquisition.

Question 17

Explain what is meant by *short term* and *medium term* scheduling policy.

Question 18

Define *turnaround time* and *normalised turnaround time*. Why are these useful for measuring the performance of a scheduling algorithm?

Answer:

turnaround time = finish time - arrival time (into the system)
normalised turnaround time = turnaround time / service time

On average, or in the worst case, we want these measurements to be minimised. The normalised turnaround time gives better information about short jobs. Each one measures how quickly jobs move through the system.

"How fast a process will run" is not a good answer. n.t.t. is not the average turnaround time over all the processes in the system.

Question 19

Describe the difference between *external* and *internal* fragmentation. Indicate which of the two are most likely to be an issue on: (a) a simple memory management machine using static partitioning; and (b) a similar machine using dynamic partitioning.

Answer:

- Internal fragmentation: the space wasted internal to the allocated region. Allocated memory might be slightly larger than requested memory.
- External fragmentation: the space wasted external to the allocated region. Memory exists to satisfy a request but it's unusable as it's not contiguous.
- Static partitioning is more likely to suffer from internal fragmentation. Dynamic partitioning is more likely to suffer from external fragmentation.

Question 20

What is *thrashing*? Provide an example of a situation in which thrashing could occur.

Answer:

Thrashing is a situation where the time spent swapping pages or segments exceeds the time spent executing useful instructions. Since secondary storage is far slower than main memory, such a situation can seriously degrate system performance. Thrashing may occur when too many processes have been admitted to the system, when the page replacement algorithm is not working efficiently, or (on a segmented system) when external fragmentation prevents effective utilisation of main memory.

Question 21

Enumerate some pros and cons for increasing the page size with regard to memory management.

Answer:

- Pros: decrease the page table size due to less pages; increase swapping I/O throughput
- Cons: increase internal fragmentation; increase page fault latency

Question 22

Describe two virtual memory *page replacement* policies. Which is less common in practice? Why?

Question 23

What is the maximum file size supported by an inode file system with 16 direct blocks, 1 single, 1 double, and 1 triple indirection block? The block size is 512 bytes. Disk block numbers can be stored in 4 bytes.

Answer:

Block size = 512
Number of block numbers in an indirection block = block size / 4 = 128

Number of blocks for file data in that file object
= 16 (direct pointers) + 128 (single indirection) + 128^2 (double) + 128^3 (triple indirection)

Maximum file size
= (number of blocks for file data that file object)*(block size)
= (16 + 128 + 128^2 + 128^3)*(512) bytes

Question 24

Explain the following basic algorithms that are used for the selection of a page as replacement algorithms: clock policy, first-in-first-out (FIFO), least recently used (LRU), and optimal policy?

2 Part II: Problem-Solving Questions

2.1 Task 1

Consider the following table, which shows when each of the processes arrives to the system and the CPU time (in seconds) required for its execution. Assume that no I/O operations are involved in these processes.

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Draw a chart (or sequence) of process execution under the following process scheduling:

(a) First-Come-First-Served (FCFS) or First-In-First-Out (FIFO)

(b) Round Robin with the CPU time slice quantum of 1 second

(c) Shortest Process Next (SPN)

(d) Shortest Remaining Time (SRT)

(e) Feedback Scheduling

2.2 Task 2

Is the following system of 4 processes with 2 resources deadlocked?

Process	R1	R2
P1	1	3
P2	4	1
P3	1	2
P4	2	0

Table 1: Current allocation matrix

Process	R1	R2
P1	1	2
P2	4	3
P3	1	7
P4	5	1

Table 2: Current request matrix

R1	R2
1	4

Table 3: Availablility vector

(a) If the availability vector is as below (Table 4), is the system above still in deadlock?

R1	R2
2	3

Table 4: Availablility vector

(b) Is the system deadlocked if the availability is as below (Table 5)?

R1	R2
2	4

Table 5: Availablility vector

Answer for Task 2:

When the availability vector (after the current allocation) is 1 4, P2 and P4 are deadlocked.

When the availability vector (after the current allocation) is 2 3, P2, P3, P4 are deadlocked.

When the availability vector (after the current allocation) is 2 4, then it the system is in a safe state.

General approach:

(a) Look for a process whose requests <= the availability.
We know we can then satisfy the requirements for this process, so we assume it gets its resources required and terminates. Thus, we can then:

(b) Release all resources *currently allocated* to that process, and add them to the pool.

(c) Repeat for all processes.

(d) If you eventually reach a point where you cannot do this, and there are still processes remaining, the system is deadlocked. If you release them all, then the current allocation will result in a safe state.

2.3 Task 3

Consider the following snapshot of a system:

R1	R2	R3	R4
3	4	5	5

Table 6: Available resources

Process	R1	R2	R3	R4
P1	1	1	2	2
P2	1	0	0	0
P3	0	0	1	2

Table 7: Current allocation matrix

Process	R1	R2	R3	R4
P1	2	4	5	2
P2	1	1	3	5
P3	1	0	2	2

Table 8: Maximum request matrix

(a) Prove that the system is in a *safe state* by showing a sequence of process completion even when all the processes eventually request for their maximum requirements.

(b) If a request from process `P2` arrives for `(0, 1, 0, 0)`, can the request be granted immediately? Justify your answer.

Answer for Task 3(a):

R1	R2	R3	R4
1	3	2	1

Table 9: Currently available resources (after the current allocation)

Process	R1	R2	R3	R4
P1	1	3	3	0
P2	0	1	3	5
P3	1	0	1	0

Table 10: Need matrix

Sequence of completion for this worst-case request matrix, taking processes in-turn:

- P1 cannot be completed at the moment
- P2 cannon be completed at the moment
- P3 can be completed as (1 0 1 0) <= (1 3 2 1)
  Available becomes (0 0 1 2) + (1 3 2 1) = (1 3 3 3)
- P1 can be completed as (1 3 3 0) <= (1 3 3 3)
  Available becomes (1 1 2 2) + (1 3 3 3) = (2 4 5 5)
- P2 can be completed as (0 1 3 5) <= (2 4 5 5)
  Available becomes (1 0 0 0) + (2 4 5 5) = (3 4 5 5)

So, the system is in a SAFE state.

Answer for Task 3(b):

Let us see whether this system is SAFE or UNSAFE. First let us grant the request to P2.

Process	R1	R2	R3	R4
P1	1	1	2	2
P2	1	1	0	0
P3	0	0	1	2

Table 11: Current allocation matrix

R1	R2	R3	R4
1	2	2	1

Table 12: Currently available resources (after the current allocation)

Process	R1	R2	R3	R4
P1	1	3	3	0
P2	0	0	3	5
P3	1	0	1	0

Table 13: Need matrix

- P1 cannot be completed at the moment
- P2 cannot be completed at the moment
- P3 can be completed as (1 0 1 0) <= (1 2 2 1)
  Available becomes (0 0 1 2) + (1 2 2 1) = (1 2 3 3)
- P1 cannot be completed
- P2 cannot be completed

So, the system will result in an UNSAFE state if the request for P2 is granted and hence the request should be denied at this time.