Carleton UNIVERSITY

Sample Solution to SAMPLE MIDTERM
EXAMINATION

DURATION: 1.5 HOURS **No. Of Students: 30**

Department Name & Course Number: Systems and Computer Engineering SYSC 4001

Course Instructor: Thomas Kunz

AUTHORIZED MEMORANDA:

William Stallings, *Operating Systems: Internals and Design Principles*, 8th edition, Pearson 2015, ISBN-13: 9780133805918 (as physical book, no ebook) or earlier versions of that same book

Students MUST count the number of pages in this examination question paper before beginning to write, and report any discrepancy to a proctor. This question paper has 5 pages + cover page = 6 pages in all.

This examination question paper MAY NOT be taken from the examination room.

In addition to this question paper, students require: an examination booklet: NO
Scantron Sheet: NO

Name: _____

Student Number: _____

Question 1: _____ /10

Question 2: _____ /10

Question 3: _____ /10

Question 4: _____ /10

Total: _____/40

Question 1: Process Description and Control (10 marks)

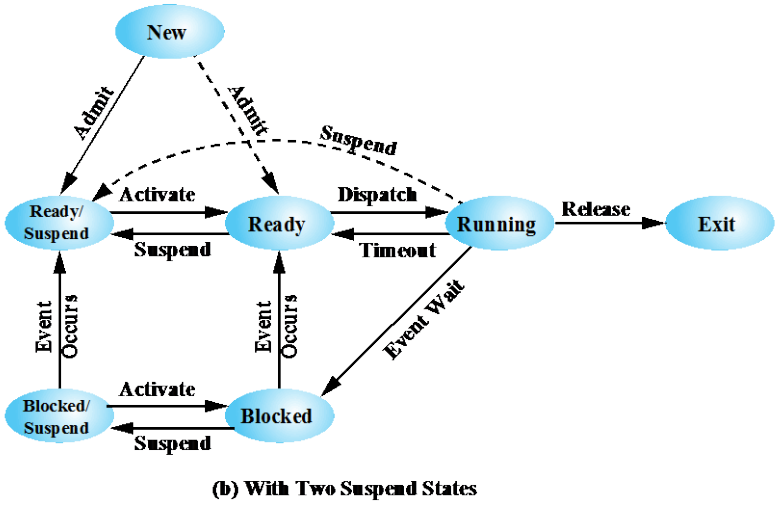

Figure 3.9 Process State Transition Diagram with Suspend States

The above figure, reproduced from the textbook, shows a process state diagram with seven states and all possible state transitions. The remaining potential state transitions (i.e., the arrows connecting two states that are NOT drawn in this diagram) are impossible. For each of these impossible transitions, briefly explain why.

Answer (10 marks):

New → Blocked, Blocked/Suspend, or Running: A newly created process remains in the new state until the processor is ready to take on an additional process, at which time it goes to one of the Ready states.

Ready → Blocked or Blocked/Suspend: Typically, a process that is ready cannot subsequently be blocked until it has run. Some systems may allow the OS to block a process that is currently ready, perhaps to free up resources committed to the ready process.

Ready/Suspend → Blocked or Blocked/Suspend: Same reasoning as preceding entry.

Ready/Suspend → Running: The OS first brings the process into memory, which puts it into the Ready state.

Blocked → Ready /Suspend: this transition would be done in 2 stages. A blocked process cannot at the same time be made ready and suspended, because these transitions are triggered by two different causes.

Blocked → Running: When a process is unblocked, it is put into the Ready state. The dispatcher will only choose a process from the Ready state to run

Blocked/Suspend → Ready: same reasoning as **Blocked → Ready/Suspend**

Blocked/Suspend → Running: same reasoning as **Blocked → Running**

Running → Blocked/Suspend: this transition would be done in 2 stages

Exit → Any State: Can't turn back the clock, once a process is terminated, it cannot be revived again, has to be started all over again.

Question 2: Concurrency: Mutual Exclusion and Synchronization (10 marks)

Consider the following program:

```
const n = 50;
int tally;
void total()
{
      int count;
      for (count = 1; count <= n; count++) {
            Tally++;
      }
}
void main()
{
      tally = 0;
      parbegin (total(), total());
      write(tally);
}
```

1. Determine the proper lower and upper bound on the final value of the shared variable `tally` output by this concurrent program. Assume processes can execute at any relative speed and that a value can only be incremented after it has been loaded into a register by a separate machine instruction.

   Answer (6 marks):

   On casual inspection, it appears that tally will fall in the range $50 \leq \text{tally} \leq 100$ since from 0 to 50 increments could go unrecorded due to the lack of mutual exclusion. The basic argument contends that by running these two processes concurrently we should not be able to derive a result lower than the result produced by executing just one of these processes sequentially. But consider the following interleaved sequence of the load, increment, and store operations performed by these two processes when altering the value of the shared variable:

   1. Process A loads the value of tally, increments tally, but then loses the processor (it has incremented its register to 1, but has not yet stored this value.
   2. Process B loads the value of tally (still zero) and performs fortynine complete increment operations, losing the processor after it has stored the value 49 into the shared variable tally.
   3. Process A regains control long enough to perform its first store operation (replacing the previous tally value of 49 with 1) but is then immediately forced to relinquish the processor.
   4. Process B resumes long enough to load 1 (the current value of tally) into its register, but then it too is forced to give up the processor (note that this was B's final load).
   5. Process A is rescheduled, but this time it is not interrupted and runs to completion, performing its remaining 49 load, increment, and store operations, which results in setting the value of tally to 50.
   6. Process B is reactivated with only one increment and store operation to perform before it terminates. It increments its register value to 2 and stores this value as the final value of the shared variable.

   Some thought will reveal that a value lower than 2 cannot occur. Thus, the proper range of final values is $2 \leq \text{tally} \leq 100$.

2. Suppose that an arbitrary number `N` of these processes are permitted to execute in parallel under the assumptions of part (1). What effect will this modification have on the range of final values of `tally`?

   Answer (4 marks):

   For the generalized case of `N` processes, the range of final values is $2 \leq \texttt{tally} \leq (\texttt{N} \times 50)$, since it is possible for all other processes to be initially scheduled and run to completion in step (5) before Process B would finally destroy their work by finishing last.

Question 3. Deadlocks (10 marks)

1. A computer has six tape drives, with n processes competing for them. Each process may need two drives. For which values of *n* is the system deadlock free?

   Answer (2 marks):
   Up to 5 processes can use this system without a deadlock. There is always at least one process that can acquire two disk drives, allowing it to complete, releasing its disk drives, which will allow other processes to acquire the necessary set of disk drives, etc. For higher values of *n*, there is always a chance that six processes have one disk drive and are blocked, waiting for one of the other (blocked) processes to release one drive.

2. A system has four processes and five allocatable resources. The current allocation and maximum needs are as follows:
   AVAIL = 0 0 *x* 1 1

		1	0	2	1	1		1	1	2	1	3
		2	0	1	1	0		2	2	2	1	0
HAVE	=	1	1	0	1	0	MAX =	2	1	3	1	0
		1	1	1	1	0		1	1	2	2	1

   What is the smallest value of *x* for which this is a safe state? Explain your answer!

   Answer (5 marks):
   The only process not potentially asking for more of resources 1 and 2 (which are both completely handed out) is process 4. This process can ask for at most one more unit of resource 3, and there are also enough units of resource 4 and 5 to allow the process to terminate. So assuming we set x to 1, process 4 can terminate. Once it terminates, AVAIL = 1 1 2 2 1. At this point, process one may ask for two more units of resource 5, process two may ask for two more units of resource 2, or process 3 may ask for three more units of resource three (for example). To satisfy the maximum request by process 3, we need to increase the number of units of resource 3 by one again, giving AVAIL = 1 1 3 2 1, allowing us to terminate process 3, with AVAIL = 2 2 3 3 1. We still end up with process potentially asking for two more units of resource 5 or process two potentially asking for more units of resource 2 etc., which can be granted, resulting in AVAIL = 4 3 4 4 1. At this point we are stuck, since process 1 can ask for two more units of resource 5, which we don't have (and given that resource 5 is a resource with a fixed number of units, this will not change). So no matter what value we pick for *x*, the system is in an unsafe state already. It involves a "deadlock" with a single process only, this process made an initial claim that the system knows it will not be able to honor, so the OS should have never started that process (process initiation denial would/should have dealt with this).

3. In a real computer system, neither the resources available nor the demands of processes for resources are consistent over long periods (months). Resources break or are replaced, new processes come and go, new resources are bought and added to the system. If deadlock is controlled by the banker's algorithm, which of the following changes can be made safely (without introducing the possibility of deadlock), and under what circumstances?
   - Increase Available (add new resources)
   - Decreases Available (remove resources permanently from system)
   - Increase Max for one process
   - Decrease Max for one process
   - Increase the number of processes
   - Decrease the number of processes

   Answer (3 marks):
   - Increase Available (add new resources): *no problem*
   - Decreases Available (remove resources permanently from system): *may lead to deadlock*
   - Increase Max for one process: *may lead to deadlock*
   - Decrease Max for one process: *no problem* (assuming that the process has not yet acquired all resources, i.e., HAVE = MAX)
   - Increase the number of processes: *no problem*, assuming processes start out with initially no resources
   - Decrease the number of processes: *no problem*, assuming the OS can correctly reclaim resources held by such processes

Question 4. Memory Management (10 marks)

A pure paging system (no segmentation) has a page size of 512 words, a virtual memory of 512 pages numbered 0 through 511, and a physical memory of 10 frames numbered 0 through 9. The current content of physical memory is as follows:

Physical Address	Content
0	……
…	……
1536	start of Page 34
2048	start of Page 9
…	…
3072	start of Page Table
3584	start of Page 65
…	……
4608	start of Page 10
…	……

a) Assuming that page tables contain frame numbers (rather than physical memory addresses), show the current content of the page table.

Answer (3 marks)

Virtual Page	Page Frame
.	
.	
.	
9	4
10	9
.	
.	
.	
34	3
.	
.	
.	
65	7
.	
.	
.	

b) Show the content of the page table after page 49 is loaded at location 0 and page 34 is replaced by page 12.

Answer (3 marks)

Virtual Page	Page Frame
.	
.	
.	
9	4
10	9
.	
.	
.	
12	3
.	
.	
.	
49	0
.	
.	
.	
65	7
.	
.	
.	

c) What physical address is referenced by the virtual address 4613?

Answer (4 marks)
Virtual location 4613 is virtual page 9, offset 5, which is at physical page frame 4, offset 5. The address is 2053.