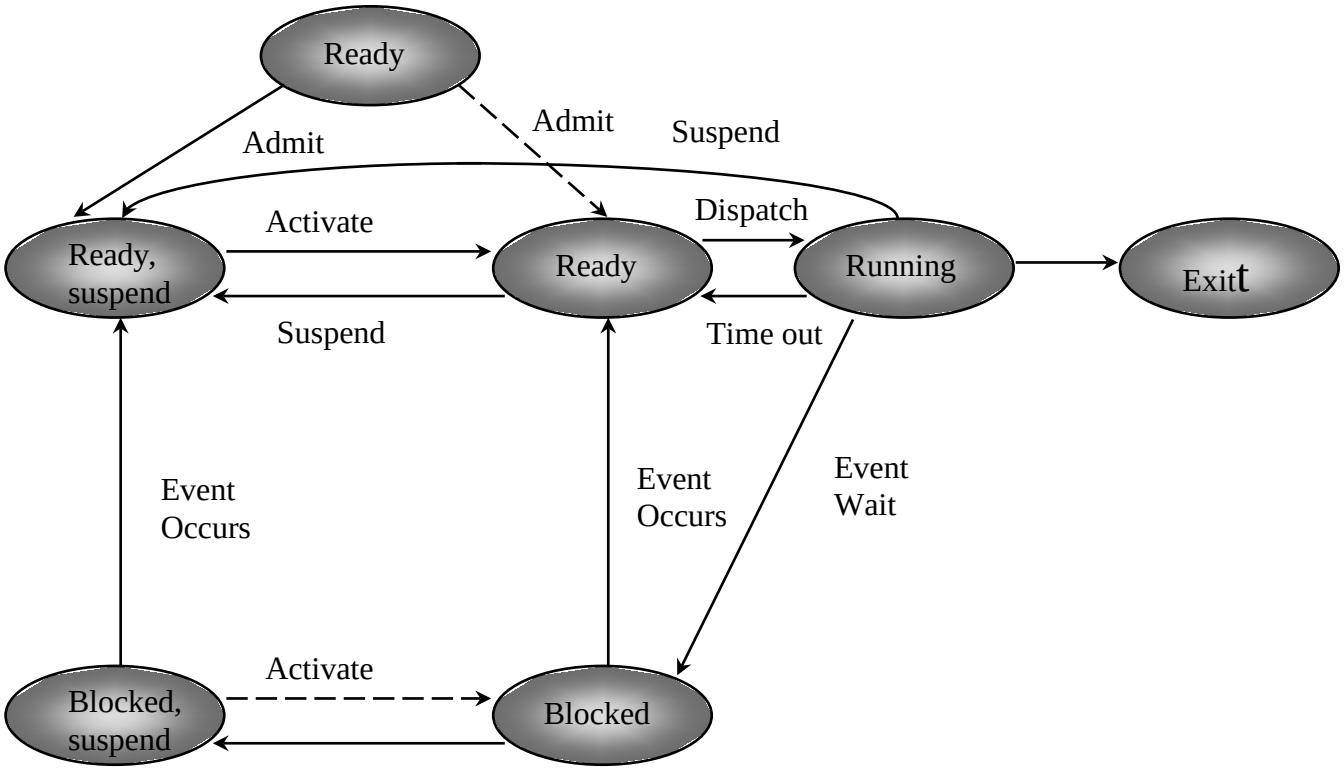
SYSC 4001 Operating Systems
Solutions to Sample Midterm 2

Question 1. Processes and Threads (10 marks)

a) Draw the process state model with 5 basic states plus support for swapping as discussed in class. Clearly label all relevant state transitions in the diagram.

Answer (3 marks):



b) Briefly describe the difference between a process and a thread.

Answer (3 marks):

- Process:
  - Has a virtual address space which holds the process image
  - Protected access to processors, other processes, files, and I/O resources
- Thread:
  - Has an execution state (running, ready, etc.)
  - Saves thread context when not running
  - Has an execution stack
  - Has some per-thread static storage for local variables
  - Has access to the memory and resources of its process, all threads of a process share this

c) What are the advantages of threads, compared to processes? What are the disadvantages?

Answer (2 marks):
Advantages:
- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Less time to switch between two threads within the same process
- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel

Disadvantage:
- Less protection from each other

d) What is the difference between kernel-level threads and user-level threads?

Answer (2 marks):
User-level threads:
- All thread management is done by the application
- The kernel is not aware of the existence of threads
- Thread switching does not require kernel mode privileges
- Scheduling is application specific

Kernel-level threads:
- Kernel maintains context information for the process and the threads
- Switching between threads requires the kernel

Question 2. Handling Deadlocks (10 marks)

Consider the following ways of handling deadlock: (1) banker's algorithm, (2) deadlock detect and kill thread, releasing all resources, (3) reserve all resources in advance, (4) restart thread and release all resources if thread needs to wait, (5) resource ordering, and (6) detect deadlock and roll back thread's application.

a) One criterion to use in evaluating different approaches to deadlock is as follows: Which permits the greater concurrency? In other words, which allows the most threads to make progress without waiting when there is no deadlock? Give a rank order from 1 to 6 for each of the listed ways of handling deadlock, where 1 allows for the greatest degree of concurrency. Comment on your ordering (e.g., why did you rank an approach at a certain level, compared to the other approaches).

Answer (5 marks):
In order from most-concurrent to least, there is a rough partial order on the deadlock-handling algorithms:

1. Detect deadlock and kill thread, releasing its resources
   Detect deadlock and roll back thread's actions
   Restart thread and release all resources if thread needs to wait

   None of these approaches limit concurrency before deadlock occurs, because they rely on runtime checks rather than static restrictions. Their effects after deadlock is detected are harder to characterize: they still allow lots of concurrency (in some cases they enhance it), but the computation may no longer be sensible or efficient. The third algorithm is the strangest, since so much of its concurrency will be useless repetition; because threads compete for execution time, this algorithm also prevents useful computation from advancing. Hence it is listed twice in this ordering, at both extremes.

2. banker's algorithm
   resource ordering

   These algorithms cause more unnecessary waiting that the previous ones by restricting the range of allowable computations. The banker's algorithm prevents unsafe allocations (a proper subset of deadlock producing allocations) and resource ordering restricts allocation sequences so that threads have fewer options as to whether they must wait or not.

3. reserve all resources in advance

   This algorithm allows less concurrency than the previous two categories, but is less pathological than the worst one. By reserving all resources in advance, threads have to wait longer and are more likely to block other threads while they work, so the system-wide execution is in effect more linear.

4. restart thread and release all resources if thread needs to wait

   As noted before, this algorithm has the dubious distinction of allowing both the most and the least amount of concurrency, depending on the definition of concurrency.

b) Another criterion is efficiency; in other words, which requires the least processor overhead? Rank order the preceding approaches from 1 to 6, with 1 being the most efficient, assuming that deadlock is a very rare event and comment on your ordering. Does your ordering change if deadlocks occur frequently?

Answer (5 marks):
In order from most-efficient to least, there is a rough partial order on the deadlock-handling algorithms:

1. reserve all resources in advance
   resource ordering

   These algorithms are most efficient because they involve no runtime overhead. Notice that this is a result of the same static restrictions which make these rank poorly in concurrency.

2. banker's algorithm
   detect deadlock and kill thread, releasing its resources

   These algorithms involve runtime checks on allocations which are roughly equivalent; the banker's algorithm performs a search to verify safety which is O(nm) in the number of threads n and the number of allocations m, and the deadlock detection performs a cycle-detection search which is O(n) in the length of resource-dependency chains. Resource-dependency chains are bounded by the number of threads, the number of resources, and the number of allocations.

3. detect deadlock and roll back thread's actions

   This algorithm performs the same runtime check discussed previously bu also entails a logging cost which is O(n) in the total number of memory writes performed.

4. restart thread and release all resources if thread needs to wait

   This algorithm is grossly inefficient for two reasons. First, because threads run the risk of restarting, they have a low probability of completing. Second, they are competing with other restarting threads for finite execution time, so the entire system advances towards completion slowly if at all.

   This ordering does not change when deadlock is more likely. The algorithms in the first group incur no additional runtime penalty because they statically disallow deadlock-producing execution. The second group incurs a minimal bounded penalty when a deadlock occurs. The algorithm in the third tier incurs the unrolling cost, which is O(n) in the number of memory writes performed between checkpoints. The status of the final algorithm is questionable because the algorithm does not allow deadlock to occur; it might be the case that unrolling becomes more expensive, but the behavior of this restart algorithm is so variable that accurate comparative analysis is nearly impossible.

Question 3. Memory Management (10 marks)

a. Why is the capability to relocate processes desirable?

Answer (2 marks):
Typically, it is not possible for the programmer to know in advance which other programs will be resident in main memory at the time of execution of his or her program. In addition, we would like to be able to swap active processes in and out of main memory to maximize processor utilization by providing a large pool of ready processes to execute. In both these cases, the specific location of the process in main memory is unpredictable.

b. In a fixed-partitioning scheme, what are the advantages of using unequal-sized partitions?

Answer (2 marks):
By using unequal-size fixed partitions: **1.** It is possible to provide one or two quite large partitions and still have a large number of partitions. The large partitions can allow the entire loading of large programs. **2.** Internal fragmentation is reduced because a small program can be put into a small partition.

c. What is the difference between internal and external fragmentation?

Answer (2 marks):
Internal fragmentation refers to the wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition. External fragmentation is a phenomenon associated with dynamic partitioning, and refers to the fact that a large number of small areas of main memory external to any partition accumulates.

d. Explain the difference between simple paging and virtual memory paging.

Answer (2 marks):
Simple paging: all the pages of a process must be in main memory for process to run, unless overlays are used.
Virtual memory paging: not all pages of a process need be in main memory frames for the process to run; pages may be read in as needed

e. What elements/entries are typically found in a page table entry for a virtual memory system based on paging? Briefly define each element.

Answer (2 marks):
Frame number: the sequential number that identifies a page in main memory; **present bit:** indicates whether this page is currently in main memory; **modify bit:** indicates whether this page has been modified since being brought into main memory. Also possible to have **other control information**, protection bits or page replacement algorithm data structures, for example.