CSCI 3327
Spring 2017
Exam 2

Last Name: ____________________

First Name: ____________________

There are 100 possible points. Write all answers on the answer sheets provided (unless otherwise specified).

1. (4 pts) List the four necessary conditions for deadlock.
2. (3 pts each) Briefly define each of the following terms in the context of synchronization and deadlock issues within operating systems.
   (a) atomic
   (b) race condition
3. (4 pts) What is the difference between the "reader preference" and "writer preference" versions of the Readers-Writers Problem?
4. (4 pts) What is the difference between deadlock avoidance and deadlock prevention?
5. (6 pts) Compare and contrast monitors and semaphores.
6. (4 pts) Name two limitations of Peterson's algorithm for enforcing mutual exclusion.
7. (6 pts) Below is given a *correct* solution to the "producer-consumer problem" (PCP) using semaphores.[1] How would the solution to this problem change if the buffer being shared by the producer and consumer was of unlimited size? Mark your changes to the code below.

```
#define N 100                        /* number of slots in buffer */

semaphore mutex = 1;
semaphore A = N;
semaphore B = 0;

void producer(void)
{
  int item;
  while (TRUE)
  {
    produce_item(item);
    P(A);
    P(mutex);
    enter_item(item);          /* put item into the shared buffer */
    V(mutex);
    V(B);
  }
}
```

[1]Taken from Tanenbaum, Andrew S., *Modern Operating Systems*, p. 43.

```
void consumer(void)
{
  int item;
  while (TRUE)
  {
    P(B);
    P(mutex);
    remove_item(item);          /* get item from the shared buffer */
    V(mutex);
    V(A);
    comsume_item(item);
  }
}
```

8. (4 pts) If you were writing your own operating system, which of the four methods for handling deadlock would you employ. Briefly defend your answer.

9. (3 pts each) Construct two scenarios in Dijkstra's Banker's algorithm that meet the criteria described below:

   (a) There are exactly 3 borrowers. Each borrower has a maximum requirement of exactly \$100. The bank has at most \$5 that has not yet been allocated. The state is safe.

   (b) There are exactly 3 borrowers. Each borrower has a maximum requirement of exactly \$100. The bank has exactly \$95 that has not yet been allocated. The state is unsafe. (Remember: for this to make sense the total amount of money the bank has to give away before it started lending has to be at least as much as the largest maximum request by a borrower).

10. Imagine an operating system with 5 processes (P0, P1, P2, P3, P4) and 6 resources (R0, R1, . . . , R5). There are 2, 3, 1, 2, 2, and 2 of the resources available, respectively.

P0	has	1	of R2	and wants	1	of R3
P1	has	2	of R0			
	has	1	of R1			
P2	has	1	of R4	and wants	1	of R4
	has	2	of R5			
P3	has	2	of R3	and wants	1	of R2
P4	has	1	of R4	and wants	2	of R5

   (a) (4 pts) Draw a resource graph of this problem.

   (b) (4 pts) Reduce the resource graph.

   (c) (4 pts) Interpret the reduced resource graph (i.e., explain the meaning of the reduced graph).

11. (12 pts) The program below simulates a multi-threaded application in which people at various locations are asked to rate a movie (-1, 0, or 1). The ratings as well as the total number of responses are stored for each movie. So, the "server" randomly picks a movie for which it wants to obtain a response and then farms the task of obtaining a response out to a thread.

It is possible, therefore, for two threads to be solicting responses for the same movie at the same time.

Show how you would modify this program (if at all) to ensure that the ratings and number of responses are correctly calculated given the fact that multiple threads are involved. You may use either monitors or semaphores to construct your solution.

```
import java.util.Scanner;
import java.io.File;

public class MovieRating
{
  public static void main(String [] args) throws Exception
  {
    Scanner kb= new Scanner(System.in);
    int i,j,whichOne,totalResponses;
    int numThreads= 10;

    Thread [] rateit= new Thread[numThreads];

    Movie m[]= new Movie[4];
    m[0]= new Movie("Gone with the Wind");
    m[1]= new Movie("Ice Age");
    m[2]= new Movie("SAW MLVXII");
    m[3]= new Movie("The Princess Bride");

    // each thread obtains 100 responses ...
    for (i=0; i<100; i++) {
      for (j=0; j<numThreads; j++) {
        whichOne= (int) (Math.random()*4.0);
        rateit[j]= new Thread(new RateThread(m[whichOne]));
      }

      for (j=0; j<numThreads; j++)
        rateit[j].start();

      for (j=0; j<numThreads; j++)
        rateit[j].join();
    }

    totalResponses= 0;
    System.out.printf("%-20s %5s %5s\n","Movie","Score","Responses");
    System.out.println("-----------------------------------------------------");
    for (i=0; i<4; i++) {
      System.out.printf("%-20s %5d %5d\n",m[i].name,m[i].score,m[i].numResponses);
      totalResponses+= m[i].numResponses;
    }
    System.out.println("-----------------------------------------------------");
    System.out.printf("%-20s %5s %5d\n","Total Responses","",totalResponses);
```

```
  }
}

class RateThread implements Runnable
{
  private Movie m;
  public RateThread(Movie m) { this.m= m; }
  public void run()
  {
    int response= getResponse();
    m.score+= response;
    m.numResponses++;
  }

  // find out from survey-taker how many thumbs-ups they give it
  // 1= thumbs up; 0=thumb up, thumb down; -1= thumbs down
  public int getResponse()
  {
    System.out.println("What did you think of the movie "+m.name);
    return (int) (Math.random()*3.0) - 1;
  }
}


class Movie
{
  public Movie(String name) { this.name= name; score= 0; numResponses= 0; }
  public int score,numResponses;
  public String name;
}
```