Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Operating Systems (CS550)**

**Exam 1**

**Part I (20 points).**

Circle either true or false to indicate that the following statements are true or false. Evaluate “truth” in terms of the material covered in this course. The statements below are intended to have an unambiguous answer.

1. **T** F MPI\_Comm\_rank retrieves a process’s rank.

2. **T** F Two threads in the same process can both access a static variable
 declared at global scope.

3. T **F** Processes may be in one of four states at any given time.

4. **T** F An IO request may cause a process to enter the blocked state.

5. **T** F The process control block (PCB) contains a process’s CPU register
 image.

6. T **F** Semaphores are always needed to solve embarrassingly parallel
 problems.

7. **T** F Batch operating systems/schedulers typically require jobs to wait

 in a queue before running.

8. **T** F An operating system is a resource manager.

9. T **F** The CPU scheduler controls the allocation and deallocation of

memory.

10. **T** F Simple load balancing is trying to evenly distribute work between

a set of processes or threads.

Part II (80 points).

Read instructions and questions carefully.

1. Using a diagram, show the three states in which a process or thread may be in, briefly describe each, and the transitions between them (10 points).

 Resource is available Process needs a resource that is unavailable

(Ready)<--------------------- (Blocked/Wait) <----------- (Run)

 ^ ^

 | <- Context Switch |

 +--------------------------------------------------------+

 Dispatch ->

Ready: process is ready to run, but is not yet able to access the run state (CPU is in use)

Run: process is currently executing (using CPU)

Blocked: process is waiting for a resource to become available

(Diagram 4 pts, Descriptions 4 pts, Transition descriptions 2pts)

1. Write two threads called compute\_average and find\_maximum for a C program. Assume both have access to a static double array arr that is dynamically allocated. Both also have access to the size of the array in a static variable called size. Both will print their results to the screen (standard out). Write a segment of code to start both threads concurrently and to wait for both to finish. You may assume this code segment is in a main function. Do not write the entire main function (15 points).

void \* compute\_average(void \* tid)

{

 int i;

 double avg = 0.0;

 for(i = 0; i < size; i++)

 avg += arr[i];

 printf(“%lf\n”, avg/size);

 pthread\_exit(NULL);

}

void \* find\_maximum(void \* tid)

{

 int i;

 double max = arr[0];

 for(i =1; i < size; i++)

 if(arr[i] > max)

 max = arr[i];

 printf(“%lf\n”, max);

 pthread\_exit(NULL);

}

int main(int argc, char \*\* argv)

{

 pthread\_t max\_t, avg\_t;

//Fill array here

 pthread\_create(&max\_t, NULL, find\_maximum, (void \*) 1);

 pthread\_create(&avg\_t, NULL, compute\_average, (void \*) 1);

 pthread\_join(max\_t, NULL);

 pthread\_join(avg\_t, NULL);

 return 0;

}

1. Explain if the previous program contains a race condition (5 points).

The previous program does not contain a race condition given the following conditions: 1) the array is filled before the threads are started; 2) the threads are joined in main to prevent the main function from finishing before the threads finish; and 3) data shared between threads is not modified by either thread.

1. Conceptually, explain the difference between blocking and non-blocking send/receive operations in MPI (5 points).

In blocking I/O, when a message is sent, a process waits until it has acknowledgement that the message has been received before it can continue processing. Similarly, when a message is requested (a receive method/function is called) the program waits until the message has been received before continuing processing. Non-blocking I/O allows for messages to be sent or requested for receipt without waiting for an acknowledgement that the message has been received. This means that programs may continue processing immediately after sending a message or after requesting that a message be received.

1. Explain mutual exclusion and the conditions processes need to meet when providing it. Describe how a semaphore can be used to provide mutual exclusion(10 points).

Mutual exclusion means only one process (or thread) may access a shared resource at a time. All others must wait.

Processes (or threads) need to meet the following conditions for mutual exclusion on critical sections.

1. Mutual exclusion by definition
2. Absence of starvation - processes (threads) wait a finite period before accessing/entering critical sections.
3. Absence of deadlock - processes (threads) should not block each other indefinitely.

A *binary* semaphore can provide mutual exclusion by allowing for exactly one process to access a critical section at a time. It does so by forcing all other processes to wait (i.e. to be queued) until the semaphore has been released.

1. Describe the three semaphores for the producer-consumer problem, as presented in class. Be sure to include the type of semaphore and to explain how each is used (10 points).

The producer-consumer problem requires:

a) one semaphore called *full* for the number of full slots (counting) to prevent the producer from putting items in the buffer/list when it is full and to avoid gaps in the buffer/list,

b) one semaphore called *empty* for the number of empty slots (counting) to prevent the consumer from removing items from the buffer/list when it is empty

, and

c) one binary semaphore *s* for mutual exclusion (mutually exclusive insertion or deletion) for the list or buffer of items.

1. List and describe five of the parts of the Process Control Block (PCB) (10 points).

Valid answers include:

PID – the process identifier - the identifier used by the operating system for this process

CPU Register Image – a copy of the current CPU register values used by this process

Process owner/user – the user or owner of this process – this could be root (the superuser), an administrative user, or a regular user

State – the state of this process – typically ready, run or blocked

List of threads – a list of the threads created and owned by this process. These may also be referred to as child threads.

List of resources – a list of the resources currently acquired (owned) by this process

List of child processes – a list of the processes created by this process. These include processes created using the fork() or exec() commands.

Address space – this indicates which portion of the system memory (RAM/main memory) is used by this process.

Privileges or permissions – this indicates the permissions or privileges of the process which might include access to certain system calls, files, or data depending upon the type of user that invoked this process. Such permissions may be read, write, and/or execute at various levels (user, group, and/or world).

1. Assume you are writing a program to compute the tuition for a large number of students, where S is the number of students. Assume that these computations are very expensive. You have a multiprocessor system or cluster with N CPU cores. How would you divide the work between each core if you were to use one process per core? Write MPI-style pseudocode to solve this problem (15 points).

Work for S students could be divided between N cores using the formula S/N (integer division).

Remaining work could be determined using the formula S % N.

Pseudocode follows:

int rank, npes, N, i;

MPI\_Init(&argc, &argv);

 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); //Get rank (local pid)

 MPI\_Comm\_size(MPI\_COMM\_WORLD, &npes); //Get num processes

 N = npes;

 for(i = 0; i < S/N; i++)

 //Compute tuition for student number (S/N)\*rank + i

 if(rank == npes – 1 && S % N != 0)

 for(i = (S/N)\*rank + S/N; i < S; i++)

 //Compute tuition for student i

MPI\_Finalize();