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

**Operating Systems (CS550)**

**Final Exam**

**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\_size retrieves a process’s rank.

2. **T** F One of the main functions of the CPU scheduler is to transition

processes between the ready and run states.

3. **T** F Mutual exclusion is typically not necessary when solving

embarrassingly parallel problems.

4. T **F** Hold and wait is the only condition necessary for deadlock to occur.

5. T **F** In round robin scheduling, process and thread priorities may change.

6. T **F** The least recently used page replacement strategy requires prior

knowledge of the entire page utilization list.

7. **T** F A cache hit happens when the processor immediately reads data from

the cache.

8. **T** F The process control block contains information such as the CPU

register image, process ID, parent process ID, memory space information, etc.

9. **T** F Monitors generally provide mutual exclusion at the method or

function level.

10. **T** F The access matrix is a general model of protection that represents the

operations each process executing in a particular domain may use on

a set of objects.

**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 describe 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)

2. Given 5 processes P1, P2, P3, P4, and P5 that start at the same time, having CPU burst times of 23, 12, 47, 58, and 35 nanoseconds, respectively, draw a Gantt chart for those processes using the Round Robin scheduling policy with a time slice of 20ns. Additionally, assume a process P6 arrives with CPU burst time 30ns arrives at time 100ns (10 points).

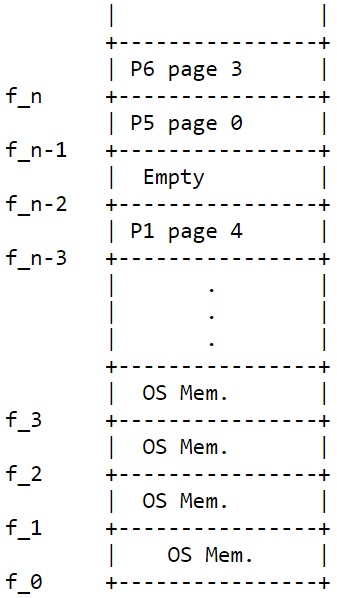
0 20 32 52 72 92 95 115 135 150 170 177 195 205  
+---+---+---+---+---+---+---+---+---+---+---+---+---+  
| P1| P2| P3| P4| P5| P1| P3| P4| P5| P6| P3| P4| P6|  
+---+---+---+---+---+---+---+---+---+---+---+---+---+

3. Given 5 processes P1, P2, P3, P4, and P5 that start at the same time, having CPU burst times of 32, 21, 70, 86, and 25 nanoseconds, respectively, draw a Gantt chart for those processes using the Longest Job First scheduling policy (5 points).

0 86 156 188 213 234  
+---+---+---+---+---+  
| P4| P3| P1| P5| P2|  
+---+---+---+---+---+

4. Briefly explain internal and external memory fragmentation, with respect to paged memory. (5 points).

Non-contiguous memory allocations, often make use of paged memory. This methodology uses a page table that corresponds with frames in memory, and a program may use any set of frames necessary to store its data in memory. There is no external fragmentation with this strategy because every frame in memory represents a page that may be allocated by a program, and portions of programs and their data may be stored in memory. Internal fragmentation may exist, as the last frame that is part of the program’s data/instructions may be partially empty. A diagram follows below.



5. Draw a resource allocation graph for the following situation:   
  
There are 4 processes, A, B, C, and D.  
There are 3 resources of type I.  
There are 3 resources of type II.  
There is one resource of type III.  
There are 2 resources of type IV.   
  
Process A owns two resources of type I and requests one resource of type II.  
Process B owns one resource of type II and requests one resource of type I.  
Process C owns one resource each of type II and III and requests two of type IV.  
Process D owns one resource each of type II and IV and requests one of type III.

Explain if deadlock exists, assuming mutual exclusion, hold and wait, and no preemption. Justify your answer (10 points).

\*\*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*\*\*\*

\*\* \*\* \*\* \*\* \*\* \*\* \*\* \*\*  
\* Process \* \* Process \* \* Process \* \* Process \*  
 \*\* A \*\* \*\* B \*\* \*\* C \*\* \*\* D \*\*  
 \*\*\*\*\*\*\* \*\*\*\*\*\*\* \*\*\*\*\*\*\*| \*\*\*\*\*\*\*  
 ^ ^ | | ^ ^ ^ || | ^ ^  
 | | | | | | | || | | |  
 | | +------+ | | | | +-))------+ | |  
 | | | | | +-------+ | | || | |

| | +------)-+ | | | | ++-----+ | |  
 | | V V | | | V V | |

+--------+ +--------+ +--------+ +--------+ |

|Resource| |Resource| |Resource| |Resource| |  
| I | | II | | III | | IV | |

| \*\*\* | | \*\*\* | | \* | | \*\* | |  
+--------+ +--------+ +--------+ +--------+ |

| |  
 +-------------------------------+

6. What is the difference between a segment, a page, and a frame? (5 points)

A segment is a logical representation of a sizable portion of a program or its data. A page is a logical representation of the smallest physically addressable portion of memory – a frame. In paged segmented memory, a segment may represent a list of pages and each page corresponds to a physical frame in memory.

7. Apply the Banker’s algorithm to the following situation. There are four resource types with the total number of system resources expressed as the tuple (2, 1, 3, 3). There are three processes with current allocations of (1, 1, 0, 1), (0, 0, 1, 1), and (0, 0, 1, 1). The maximum claim of each process is (1, 1, 0, 2), (0, 0, 2, 1), and (2, 1, 3, 1). Assume each process can complete its work once it receives its maximum claim. Determine if a safe state can be achieved, being sure to show all work (10 points).

Total Resources: (2, 1, 3, 3)

Used Resources: (1, 1, 2, 3)

Free Resources: (1, 0, 1, 0)

Resources Max Claim

P1 – (1, 1, 0, 1) (1, 1, 0, 2)

P2 – (0, 0, 1, 1) (0, 0, 2, 1)

P3 – (0, 0, 1, 1) (2, 1, 3, 1)

Give P2 1 unit of R3, so P2 can run to completion and release its resources.

Now (1, 0, 2, 1) is free.

Give P1 1 unit of R4, so P1 can run to completion and release its resources.

Now (2, 1, 2, 2) is free.

Finally, give P3 2 units of R1, 1 unit of R2, and 2 units of R3.

Now P3 can run to completion and release its resources.

Thus, the system is in a safe state.

8. Using the page replacement policies listed below, indicate how the pages would be replaced, for each page, for the following page utilization list for both parts a) and b):  
  
<3, 1, 9, 4, 2, 8, 6, 5, 2, 3>

Assume only 4 pages may be in memory at any given time (10 points total, 5 points each):

a) Least Recently Used

<3, , , >

<3, 1, , >

<3, 1, 9, >

<3, 1, 9, 4>

<1, 9, 4, 2>

<9, 4, 2, 8>

<4, 2, 8, 6>  
<2, 8, 6, 5>

<2, 8, 6, 5>  
<2, 3, 6, 5>

b) First In First Out

<3, , , >  
<3, 1, , >  
<3, 1, 4, >

<3, 1, 9, 4>

<1, 9, 4, 2>  
<9, 4, 2, 8>  
<4, 2, 8, 6>  
<2, 8, 6, 5>  
<2, 8, 6, 5>

<8, 6, 5, 3>

9. Assume you are writing a program to sort the jobs submitted based upon their integer priority within a batch scheduler. Since this program may handle a large number of jobs, you would like to write the program to perform parallel sorting. Luckily, you have an in-place sorting algorithm called qsort that takes an array, the number of elements to sort, the size of the data type to sort, and the name of the comparison function as parameters. Write complete C-style pseudocode including both a main function and a thread function to solve this problem (10 points).

The comparison function is provided as follows:

int comparePriority(const void \* p1, const void \* p2){

return ( \*(int \*)p1 - \*(int \*)p2 );

}

And would be called to sort an array of integer priorities as follows:

qsort(intArrayName, arrayLength, sizeof(int), comparePriority);

**You must use only one type of thread that will sort approximately equal sized parts of the array.** Within main, you are given a number of threads to use, and you must divide the data as equally as possible between threads. You may perform some preprocessing of the data before passing it to the threads. For example, for two threads, it is recommended that you provide all elements less than a particular value to the first thread and remaining values to a second thread.

int size;  
int \* arrayOfInts;  
  
struct ArrayInfo  
{  
 int startingLocation;  
 int numInts;  
};  
  
typedef struct ArrayInfo ArrayInfo;  
  
void \* sorter(void \* threadData)

{

ArrayInfo \* tData = (ArrayInfo \*) threadData;  
 int startLoc = tData->startingLocation;

int numInts = tData->numInts  
 qsort(&arrayOfInts[startLoc], startLoc, sizeof(int),

comparePriority);  
}  
  
int main(int argc, char \*\* argv)  
{  
 //Determine array size.  
 //Allocate arrayOfInts and store data in the array.  
 //Store the number of threads in numThreads.  
 //Allocate an array of threads of size “size”.  
  
 //Preprocessing for this problem is accomplished by moving the   
 //smallest size/numThreads values to the first partition of the   
 //array, and so on.  
  
 int i;  
 for(i = 0; i < numThreads; i++) {

ArrayInfo \* tInfo = (ArrayInfo \*) malloc(sizeof(ArrayInfo));  
 tInfo->startingLocation = (size / numThreads)\*i;  
 tInfo->numInts = size / numThreads +   
 ((i!=numThreads-1)?0:(size%numThreads));  
 pthread\_create(&thread[i], NULL, sorter, (void \*) tInfo);  
 }

for(i = 0; i < numThreads; i++)   
 pthread\_join(thread[i], NULL);  
}

10. Given the following program, indicate what problems exist within the program and show how to fix the problem (5 points).

static int x;

void \* incrementer(void \* tid) {

int i = 0;

for(i = 0; i < 1000; i++)

x++;

pthread\_exit(NULL);

}

void \* decrementer(void \* tid){

int i = 0;

for(i = 0; i < 1000; i++)

x--;

pthread\_exit(NULL);

}

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

pthread\_t incr, decr;

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

pthread\_attr\_setschedpolicy(&attr, SCHED\_RR);

pthread\_create(&incr, &attr, incrementer, (void \*) 1);

pthread\_create(&decr, &attr, decrementer, (void \*) 2);

pthread\_join(incr, NULL);

pthread\_join(decr, NULL);

printf("x is %d\n", x);

return 0;

}

A race condition exists in the incrementer and decrementer threads with respect to the shared variable x. This can be fixed by using a binary semaphore declared globally as follows:

static sem\_t s;

And initialized in main as follows:

sem\_init(&s, 0, 1);

And modified in the incrementer thread as follows

for(i = 0; i < 1000; i++)  
{  
 sem\_wait(&s);

x++;  
 sem\_post(&s);  
}

And finally in the decrementer thread as follows:

for(i = 0; i < 1000; i++)  
{  
 sem\_wait(&s);

x--;  
 sem\_post(&s);  
}

Bonus problem (5 points)

On the back of this page, complete problem 9 using MPI instead of threads. You will only receive bonus credit if your answer is correct and complete.