CSci 4061: Introduction to Operating Systems (Spring 2013)
Final Exam

May 14, 2013 (4:00 – 6:00 pm)

Open Book and Lecture Notes

(Bring Your U Photo Id to the Exam)

This exam paper contains 8 questions (12 pages)

Total 100 points.

Please put your official name and NOT your assumed name.

First Name:

Last Name:

Student ID:

Please indicate your Section Number:

Section 2: 1:25 – 2:15 pm
Section 3 2:30 – 3:20 pm
Section 4: 3:35 – 4:25 pm
Section 5: 4:40 – 5:30 pm
Section 6: 5:45 - 6:35 pm

Q #	1	2	3	4	5	6	7	8
Max Points	10	10	10	20	15	10	10	15
Points Scored								

Question 1 (10 points):

Indicate by circling YES or NO if the given statement is correct or incorrect, or indicate your answer to a given question.

(a) In the UNIX file system, can the inode for a file have hard-link count zero? **YES NO**

(b) In the UNIX file system, can the inode for a file have hard-link count negative? **YES NO**

(c) Threads in a process share all open file descriptors. **YES NO**

(d) Threads in a process can set up and use different handlers for a given signal type. **YES NO**

(e) A thread-safe function is always async signal-safe. **YES NO**

(f) Is the C library function `strtok` async signal-safe? **YES NO**

(g) Is it possible for a POSIX thread to lock a mutex object multiple times without unlocking it. **YES NO**

(h) Does a client process need to execute the `bind` function for connecting to an Internet server? **YES NO**

(i) Is it possible to make the accept function call non-blocking? **YES NO**

(j) Can a process in the zombie state become active again? **YES NO**

Question 2: (10 points)
Consider the following system state in which 5 processes use multiple units of one resource type. There are total 10 units of this resource type.

Currently Available = 2

Process ID	Max Need	Current Allocation	Remaining need
P1	5	2	3
P2	10	1	9
P3	7	2	5
P4	9	2	7
P5	3	1	2

(a) Using the Banker's Algorithm determine if the following system state is safe or unsafe. If safe, then show a sequence for completing all processes. (4 points)

(b) In the above state of the system, suppose that process P3 makes a request for one more resource. Should a free resource be given to process P3? Justify your answer. (3 points)

(c) In the above state of the system, suppose that process P5 makes a request for one more resource. Should a free resource be given to process P5? Justify your answer. (3 points)

Question 3: (10 points)
Part A (5 points): Assume that we have a demand-paged virtual memory system. The memory access time on this system is 100 nanoseconds. It takes 10 milliseconds to read or write a page on disk storage when a page-fault occurs. Assume that each page-fault requires some page to be removed from the memory. On this system 40% of the times the selected page to be removed is found to be dirty.

What is the maximum acceptable page-fault rate for the effective memory access time of no more than 200 nanoseconds? <u>You only need to show the formula. No need to show the exact number.</u>

Part B (5 points) Suppose that a system has two processes, each of which needs 10 seconds of CPU time and 5 seconds for I/O.

(a) How long will it take to complete both these processes if they run sequentially?

(b) What are the <u>best case</u> and <u>worst case</u> completion times for these processes if they run concurrently, using multiprogramming?

Question 4: (20 points) Consider a system in which a page can store 200 integers. On this system a small program that operates on a two-dimensional matrix *A* is executed. The program code resides in page 0, which corresponds to addresses 0 through 199. This page is always kept in the physical memory.

A is defined as: int A[ ] [ ] = int [10][100];

where A[0][0] is at the logical address 200 and the matrix is stored in the memory in the row-major form.

Part A (10 points): Consider the following two ways to initialize this matrix. For each of these two cases answer the following:

Identify the pattern in *page reference strings* these two initialization routines would generate for accessing matrix A. You may just identify the pattern that gets repeated because there may be several hundred page references.

Initialization 1:

```
for (int k =0; k < 100;  k++)
    for (int i =0; i< 10;  i++)
        A[ i ] [ k ] = 0;
```

Initialization 2:

```
for (int i =0; i< 10;  i++)
    for (int k =0; k < 100;  k++)
        A[ i ] [ k ] = 0;
```

Part B (10 points):
(i) Suppose that this program is executed on a virtual memory system which uses the LRU replacement policy. This program is given 3 page-frames. One of them is always occupied by page 0 which contains the code. What will be the number of page-faults?

For Initialization 1:

For Initialization 2:

(ii) If 6 page-frames are given and one of them is always occupied by page 0 which contains the code, what will be the number of page-faults?

For Initialization 1:

For Initialization 2:

Question 5 (15 points): Consider a system with three processes, named P, Q, and R. Each process continuously prints its color. We now require that the printing by the processes be coordinated such that the output produced would be

P:RED Q:WHITE R:BLUE P:RED Q:WHITE R:BLUE P:RED Q:WHITE R:BLUE …

Using semaphores, synchronize these three process such that processes P, Q, and R execute their print statements in the above order. Insert appropriate synchronization code before and after the each print statement:

Fill in here the declaration of semaphores with appropriate initialization ….

```
Process P   {
        while ( true ) {  // Insert synchronization code in space given below



              print("P:RED");



       }
  }  //end of process P
Process Q  {
         while ( true ) {  // Insert synchronization code in space given below



              print("Q:WHITE");



      }
}  //end of process Q
Process R  {
         while ( true ) {  // Insert synchronization code in space given below



                print("R:BLUE");



       }
}  //end of process R
```

Question 6 (10 points):
Part (A) (5 points): Suppose that a disk drive has 500 cylinders, numbered 0 through 499. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests is shown below in the FIFO order:
54, 144, 86, 250, 350, 249
Starting from the current head position, what is the order in which the pending requests in the queue will be handled by the following two algorithms?

(a) Shortest Seek Time First (SSTF)
(b) SCAN

Part (B) (5 points): Consider a Unix style file allocation on disk with total 13 storage pointer entries in the inode. In the inode, the first 10 storage pointers point directly to file data blocks, the other three are pointers to indirect pointer blocks, doubly-indirect pointer blocks, triple-indirect pointer blocks. Assume that the file data block size is 1000 bytes, and an indirect block contains 100 file data block addresses.
<u>What's the maximum file size which this system can store?</u>

Question 7: (10 points)
Consider the following multi-threaded program which has some potential problems.
The global variables:

```
int  Threshold  = 1000;
int  count = 0;
pthread_t tid[2];
pthread_mutex_t mutex  = PTHREAD_MUTEX_INITIALIZER;

void *  threadFunction ( void *arg ) {
   while ( 1 ) {
       pthread_mutex_lock( &mutex );
           count += rand() % 3;
           if  ( count > Threshold ) {
              pthread_cancel ( tid[ 0 ] );
              pthread_cancel ( tid[ 1 ] );
           }
       pthread_mutex_unlock( &mutex );
      sleep ( rand( ) % 3  );
   }
}

void main ( )  {
   pthread_create( &tid[0], NULL, (void *) threadFunction, NULL);
   pthread_create( &tid[1], NULL, (void *) threadFunction, NULL);
   pthread_join( tid[ 0 ], NULL );
   pthread_join( tid[ 1 ], NULL );
}
```

(a) What is the potential problem with this code?

(b) Briefly outline how you will modify this code eliminate the potential problem?

Question 8 (15 points):
Solve this problem using SEMAPHORES.
Across a canyon a rope is fastened so that baboons can cross from one side to the other. The two sides of the canyons are designated as “east” and “west”. Several baboons can cross at the same time, provided they are all going in the same direction. Assuming that baboons are modeled as processes. Write code to properly synchronize the baboons so that we never have situation where two or more baboons are at the same time on the rope and trying to cross in the opposite directions. The requirements are as follows:

- When east-side baboons are crossing on the rope, no west-side baboons are on the rope.
- When west-side baboons are crossing on the rope, no east-side baboons are on the rope.

The synchronization functions are the following four:

1. function Request_to_go_East_to_West ( )
2. function Request_to_go_West_to_East ( )
3. function Arrive_at_East_side ( )
4. function Arrive_at_West_side ( )

Example of a baboon process on east wanting to go west is shown below:

```
Process baboon ( ) {
    Play for a while;
    Request_to_go_East_to_West ();            /* executed when east side baboon wants to go west */
    Use the rope to go from east to west;     /* this will take some time to cross canyon */
    Arrive_at_West_side ( );                  /* executed when the baboon arrives on the other side */
}
```

Similar kind of code with be executed by the baboons on the west side wanting to go to east side.

Complete the pseudo-code given on the next page for functions (1) and (4).
<u>(Only for </u>Request_to_go_East_to_West and Arrive_at_West_side.)

Data Structures:

```
int   east_bound =  0          /* Number of baboons on the rope going east */
int  west_bound =  0           /* Number of baboons on the rope going west */
int east_side_waiting = 0;   /* Number of baboons on the east side waiting  */
int west_side_waiting = 0;  /* Number of baboons on the west side waiting  */

semaphore mutex  (initial value 1);

semaphore  east_side_waiting_queue (initial value 0)
        /* Baboons on east side waiting in the semaphore queue to go west */

semaphore  west_side_waiting_queue (initial value 0)
        /* Baboons on west  side waiting in the semaphore queue to go east */

direction: (null, eastward, westward) /* indicates the direction in which baboons are going currently */
            /* null means that no baboon is on  the rope.   */
Initial value of direction = null;
```

DO NOT ADD ANY ADDITIONAL DATA STRUCTURES.

```
Function Request_to_go_East_to_West  () {
    wait(mutex);
    If ( direction == null )  {
         direction = westward;       // set current direction to westward
    }
    If ( direction == westward ) {
           west_bound++;
           signal(mutex);
    }
    else {   // current direction is eastward
   /* Fill in the missing code  */

   }
}

function Arrive_at_West_side ( ) {
    wait(mutex);
    west_bound--;
    if (west_bound == 0)  {    // Last baboon going westward arrives on the west side
        /* Fill in the missing code  */

     }
     signal(mutex);
}
```

Blank page