York University
Department of Electrical Engineering and Computer Science
EECS3221 Operating Systems Fundamentals
Section E
Final Examination

Important: ***Please note that your exam answers will not be accepted for grading if you answer a version of the exam that is different from the version that is assigned to you***

December 13, 2020

Duration of Exam: 3 hours

Please check that your exam has a total of 17 pages, including this front page.

Last Name _____First Name_____

Student Number_____

For Marking Only

Q.1. _____/9

Q.2. _____/9

Q.3. _____/9

Q.4. _____/9

Q.5. _____/9

Q.6. _____/9

Q.7. _____/9

Q.8. _____/9

Q.9. _____/9

Q.10. _____/9

Total _____

1. (9 marks) For each of the statements below, circle the "T" letter following the statement if the statement is true; circle the "F" letter following the statement if the statement is false.

(a) When the number of *in-memory bits* used to keep track of the *Working Set* is increased, the *cost to service timer interrupts* will be lowered. T F

(b) When *execution-time address binding* is used, two processes running the same program will always generate the same sequence of *logical addresses* and the same sequence of *physical addresses*. T F

(c) When the system uses *contiguous, dynamic storage-allocation*, *compaction* is mainly used to reduce *internal fragmentation*. T F

(d) *Dynamic loading* is useful when large amounts of code are needed to handle *infrequently occurring cases*. T F

(e) When *paging* is used, a smaller *page size* results in less *translation overhead*. T F

(f) When *Virtual Memory* is used, decreasing the *page size* generally decreases the number of *page faults*. T F

(g) One must generate *relocatable code* for a routine if the final *binding of instructions and data to memory* for that routine is delayed until *load time*. T F

(h) *Logical and physical addresses* are the same in *load-time address-binding* schemes. T F

(i) When used for the allocation of frames among processes, *Local Replacement* generally results in greater *throughput* compared with *Global Replacement*. T F

2. (9 marks) For each of the statements below, circle the "T" letter following the statement if the statement is true; circle the "F" letter following the statement if the statement is false.

(a)) *Associative registers* can be used to improve the performance of memory management systems that use the *Inverted Page Table Architecture*.

T F

(b) When *Virtual Memory* is used, increasing the *page size* generally increases the *I/O overhead*.

T F

(c) One advantage of using *Virtual Memory* is that it increases *CPU utilization*.

T F

(d) With *Global Replacement*, a process cannot control its own *fault-rate*.

T F

(e) When *Thrashing* occurs, the operating system has a tendency to increase the *degree of multiprogramming*.

T F

(f) *Dynamic loading* requires special support from the *operating system*.

T F

(g) A process's *working set* size will change depending on the number of processes currently running.

T F

(h) When used for the allocation of frames among processes, *Local Replacement* generally results in greater *throughput* compared with *Global Replacement*.

T F

(i) Compared with the most commonly used type of page tables, one of the advantages of the *Inverted Page Table Architecture* is that it decreases the time that is needed to *search the table* when a page reference occurs.

T F

3. (9 marks) Answer the following questions.
3.1. Assume that BUFFER_SIZE = 9 in the Bounded Buffer – Shared Memory Solution *that does not require the use of semaphores* for the Producer-Consumer Problem. Assume the following sequence of executions:

(a) The Producer process performs seven (7) executions of *all* the code in the body of the while loop for the Producer process;

(b) After (a), the Consumer process performs three (3) executions of *all* the code in the body of the while loop for the Consumer process;

(c) After (b), the Producer process attempts to execute ***as many times as possible*** *all* the code in the body of the while loop for the Producer process while the Consumer process does not execute any further.

You are required to fill in the answer for each of the following questions:

(1) In (c) above, *the total number of times* that the Producer process will be able to execute *all* the code in the body of the while loop for the Producer process is _____.

(2) After (c), the value for the integer variable "*in*" will be _____.

(3) After (c), the value for the integer variable "*out*" will be _____.

3.2. Assume that BUFFER_SIZE = 9 in the Bounded Buffer – Shared Memory Solution *that does not require the use of semaphores* for the Producer-Consumer Problem. Assume the following sequence of executions:

(a) The Producer process performs seven (7) executions of *all* the code in the body of the while loop for the Producer process;

(b) After (a), the Consumer process performs three (3) executions of *all* the code in the body of the while loop for the Consumer process;

(c) After (b), the Producer process attempts to execute ***as many times as possible*** *all* the code in the body of the while loop for the Producer process while the Consumer process does not execute any further.

(d) After (c), the Consumer process attempts to execute as many times as possible *all* the code in the body of the while loop for the Consumer process while the Producer process does not execute any further.

You are required to fill in the answer for each of the following questions:

(1) In (d) above, the *total number of times* that the Consumer process will be able to execute *all* the code in the body of the while loop for the Consumer process is _____.

(2) After (d), the value for the integer variable "*in*" will be _____.

(3) After (d), the value for the integer variable "*out*" will be _____.

4. (9 marks) Suppose that in a system where Virtual Memory is used, the system has a total of four (4) memory frames, frame[0], frame[1], frame[2], frame[3],which are initially empty. Assume that the following sequence of memory page references occur:

3 6 7 3 5 4 1 2 3 7 4 3 6 3 7

What will be the (1) ***final contents of the four (4) memory frames***, frame[0], frame[1], frame[2], frame[3]; and (2) ***how many page faults would occur***, for each of the following page replacement algorithms, immediately after the last memory page reference in the above sequence has been processed by the algorithm?

(a) The FIFO Page Replacement algorithm;
(b) The Optimal Page Replacement algorithm;
(c) The LRU Page Replacement algorithm.

5. (9 marks) Answer the following questions.
Calculate the (a) *average waiting time*; and (b) *average turnaround time*, respectively, when the following scheduling algorithms are used to schedule the set of processes with corresponding arrival times and burst times below:

(1) Round Robin (RR) Scheduling (you may assume that the time quantum is 4, and the context switch time is 0 for Round Robin Scheduling)
(2) SJF Nonpreemptive Scheduling
(3) SJF Preemptive Scheduling

Process	Arrival Time	Burst Time
P_1	0.0	11
P_2	2.0	9
P_3	7.0	3

6. (9 marks) You had been introduced to the following program in the textbook in Chapter 3 in which a producer process produces items and puts them into a shared buffer in shared memory, while a consumer process consumes items in the shared buffer in shared memory:

```
#define BUFFER_SIZE 10
typedef struct {
        . . .
};
item buffer[BUFFERSIZE];
int in = 0;
int out = 0;
```

```
Producer Process:
item next_produced;
while (true) {
  /* produce an item  in next_produced */
  while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */
  buffer[in] = next_produced;
  in = (in + 1) % BUFFER_SIZE;
}
```

```
Consumer Process:
item next_consumed;
while (true) {
  while (in == out)
        ; /* do nothing */
  next_consumed = buffer[out];
  out = (out + 1) % BUFFER_SIZE;
  /* consume the item in next_consumed */
}
```


Does *the program in the textbook in Chapter 7 that uses **semaphores** to solve the bounded-buffer producer-consumer problem* have any advantages or disadvantages when compared with the program in the textbook in Chapter 3 above? If the program in the textbook in Chapter 7 that uses semaphores to solve the bounded-buffer producer-consumer problem has any advantages or disadvantages when compared with the program in Chapter 3 above, then list and explain the advantages and disadvantages.

7. (9 marks) Assume that in a system in which the *Hierarchical Paging Architecture* is used:
(a) Each logical address consists of 30 bits;
(b) The page size is 1024 bytes;
(c) The maximum size of physical-address space is 64MB;
(d) The space taken up by every page table must not exceed one frame;
(e) All page tables must start on a page boundary;
(f) The size of each page table entry (in bits) must be a power of 2;
(g) The number of paging levels must be the minimum number that is necessary to satisfy all the above requirements.

You are required to do the following:

7.1. You are required to explain *in as much detail as possible* how to determine all the elements in the *Hierarchical Paging Architecture* based on all the above requirements. For example, how do you determine the number of entries in each page table and the size of each entry (in bits) based on all the requirements above?

7.2. You are required to explain in detail how a logical address is translated into a physical address using all the elements in the *Hierarchical Paging Architecture* in 7.1. Draw a diagram to illustrate how a logical address is translated into a physical address using all the elements in the *Hierarchical Paging Architecture* in 7.1.

(Cont.)

8. (9 marks) Assume that in a system in which the *Inverted Page Table Architecture* is used:
(a) The total number of processes in the system is 64;
(b) Each process in the system has exactly 1024 pages (or frames);
(c) The size of each page (or frame) in the system is 4.
(d) The size of the inverted page table is 128K bytes. (1K = 1024)
(e) The order of the entries in the inverted page table in the system satisfies the following properties:.

(e1) for any pair of processes with process id *pid1,* and *pid2* respectively, if *pid1 < pid2* then all the entries for all the pages for the process with process id *pid1,* will be ordered *after* all the entries for all the pages for the process with process id *pid2* in the inverted page table.

(e2) for any pair of page numbers *p1*, and *p2,* in any pair of virtual addresses for a same process, if *p1 < p2* then the entry for the page *p1,* will be ordered *after* the entry for the page *p2* in the inverted page table.

You are required to do the following:

8.1. You are required to explain how the system which uses the inverted page table architecture above translates the following *virtual address*:

1010101011110110111

into a *physical memory address*.

8.2. You are required to provide the *exact physical memory address in decimal*.

8.3. You are required to *explain, in detail,* the *reasons* for which you believe that your answer is correct.

(Cont.)

9. (9 marks) The multithreaded C program below is to be run on a single processor system. What are the possible values of the variable ***result*** that could be printed out by this program? For each possible value of the variable ***result*** that could be printed out by this program, explain in detail why that value could be printed out. (***Marks will be deducted for wrong answers, so do not write answers that are only guesses***.)

```
#include <pthread.h>
#include <stdio.h>

int result, turn, lock; /* this data is shared by the thread(s) */

pthread_t tid1, tid2, tid3, tid4; /* the thread identifiers */
pthread_attr_t attr1, attr2, attr3, attr4; /* set of thread attributes */

void *runner1(void *param); /* thread runner1 calls this function */
void *runner2(void *param); /* thread runner2 calls this function */
void *runner3(void *param); /* thread runner3 calls this function */
void *runner4(void *param); /* thread runner4 calls this function */

int main(int argc, char *argv[])
{
 result = 2;
 turn = 0;
 lock = 0;

 /* get the default attributes */
 pthread_attr_init(&attr1);
 /* create the thread runner1*/
 pthread_create(&tid1,&attr1,runner1,NULL);

 while((turn != 0) && (lock != 0));
 lock = 1;
 result = result + 1;
 lock = 0;
 turn = 1;

 /* get the default attributes */
 pthread_attr_init(&attr3);
 /* create the thread runner2*/
 pthread_create(&tid2,&attr2,runner2,NULL);

 /* wait for the threads to exit */
 pthread_join(tid1,NULL);
 pthread_join(tid2,NULL);

 printf("result = %d\n", result);
```

```
}

/* The thread runner1 will begin control in this function */
void *runner1(void *param)
{
  while((turn != 1) && (lock != 0));
  lock = 1;
  result = result*(-1);
  lock = 0;
  turn = 2;

  pthread_exit(0);
}

/* The thread runner2 will begin control in this function */
void *runner2(void *param)
{
  while((turn != 2) && (lock != 0));
  lock = 1;
  result = result - 7;
  lock = 0;
  turn = 3;

  /* get the default attributes */
  pthread_attr_init(&attr3);
  /* create the thread runner3 */
  pthread_create(&tid3,&attr3,runner3,NULL);

/* wait for the thread3 to exit */
  pthread_join(tid3,NULL);

  pthread_exit(0);
}

/* The thread runner3 will begin control in this function */
void *runner3(void *param)
{
  while((turn != 3) && (lock != 0));
  lock = 1;
  result = result * 4;
  lock = 0;
  turn = 4;

  /* get the default attributes */
  pthread_attr_init(&attr4);
  /* create the thread runner4 */
```

```
  pthread_create(&tid4,&attr4,runner4,NULL);

/* wait for the thread4 to exit */
  pthread_join(tid4,NULL);

  pthread_exit(0);
}

/* The thread runner4 will begin control in this function */
void *runner4(void *param)
{
  while((turn != 4) && (lock != 0));
  lock = 1;
  result = result - 11;
  lock = 0;
  turn = 0;

  pthread_exit(0);
}
```

(Cont.)

10. (9 marks) Suppose that in a real-time system: there exists a total of ***n*** periodic real-time processes: P0, P1, P2, …, Pn-1;
For each periodic real-time process Pi, i = 0, 1, 2, …, n:
(1) the ***processing time*** of periodic real-time process Pi is denoted as Ti;
(2) the ***deadline*** of periodic real-time process Pi is denoted as Di;
(3) the ***period*** of periodic real-time process Pi is denoted as PRDi;

(a) Assume that the following is satisfied for each periodic real-time process Pi, when i is an ***even*** number, that is, i = 0, 2, …,:

(a1) Di = PRDi;
(a2) Ti = PRDi / (n + 1)

(b) Assume that the following is satisfied for each periodic real-time process Pi, when i is an ***odd*** number, that is, i = 1, 3, …,:

(b1) Di = PRDi;
(b2) Ti = (n*PRDi) / (3((n**2) + 3))

You are required to provide a ***step-by-step detailed mathematical analysis*** to determine whether ***rate-monotonic scheduling*** can guarantee that the above ***n*** periodic real-time processes: P0, P1, P2, …, Pn-1, can be scheduled so that they meet their deadlines.