CSC-246: Concepts and Facilities of Operating Systems for Computer Scientists (Section 2, Fall 2023)

It is a sample for the mid-term exam

Please Read All Questions Carefully!
There are 10 total numbered pages, and you can keep the last sheet numbered 9 and 10.
You have 75 minutes.

Please put your FULL NAME and Unity ID on THIS page only.
Please put your Student ID on the right corner of each page with odd page numbers.

Name: _______________________
Unity ID: _______________________

Grading Table

	Points	Total Possible
Q1		16
Q2		12
Q3		5
Q4		8
Q5		12
Q6		12
Q7		12
Q8		16
Q9		7
Total		100

1. Please circle "TRUE" if the statement is true, and "FALSE" if it is, well, false. (3 pts each)

   (a) One of the roles of the OS is to make hardware easier for applications to use. **TRUE** / FALSE

   (b) The primary performance goal of a modern OS to maximize the utilization of expensive resources. TRUE / **FALSE**

   (c) The scheduler will be invoked whenever an application makes a system call. TRUE / **FALSE**

   (d) A process is equivalent to a program. TRUE / **FALSE**

   (e) A process is equivalent to a thread. TRUE / **FALSE**

   (f) Hardware support is required to implement true multi-tasking. **TRUE** / FALSE

   (g) In Unix, the fork() system call creates a new process. **TRUE** / FALSE

   (h) Within multithreaded programs, every access to shared variables needs to be put into a critical section. TRUE / **FALSE**

2. Choose the best answer to each of the following questions. (3 pts each)

   (a) In a modern operating system, the kernel is:
   (A) the collection of library code that's stored in a process' memory
   (B) the layer of software that can access hardware directly
   (C) the program that runs at boot time to load the rest of the OS into memory
   (D) the portion of the OS that's stored in ROM or EEPROM Your choice ____B____

   (b) On an interactive computer system, a modern operating system would normally depend on __________ to ensure that one process doesn't use all the CPU time.
   (A) a timer interrupt
   (B) periodic calls to sched yield()
   (C) user-level threads
   (D) availability of multiple cores Your choice ____A____

   (c) A system might use a multilevel queue with feedback in order to:
   (A) prevent starvation
   (B) schedule processes differently based on their behavior
   (C) keep multiple CPU cores busy
   (D) be able to quickly select a runnable process Your choice ____B____

   (d) The contention scope for a thread identifies:
   (A) when other threads can terminate it
   (B) how its stack space is related to that of other threads in the same process
   (C) what portions of memory may be accessed by other threads at the same time
   (D) what threads it competes with for CPU time Your choice ____D____

3. The shortest-job-first (SJF) and shortest-time-to-completion-first (STCF) policies are both unrealistic policies to implement in a general-purpose operating system. Why? (4 pts)

   Because the length of jobs is difficult to estimate.

4. List four fields that would normally appear in a Process Control Block (PCB). Mark at least one of these with a star to indicate a field that would not be in the PCB if the OS supported multi-threaded processes. (6 pts)

   *** Saved CPU Registers**
   *** State**
   Files and other OS resources
   Accounting information
   And much more

5. The following code is shown to you:

```
int main(int argc, char *argv[]) {
   printf("a");
   fork();
   printf("b");
   return 0;
}
```

   Assuming `fork()` might fail (by returning an error code and not creating a new process) and `printf()` prints its outputs immediately (no buffering occurs), what are possible outputs of the program above? Please write "possible" or "impossible" for each of the following outputs. (9 pts)

   ab ____possible________ abb ____possible________

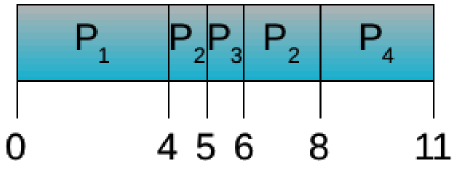
   bab ____ impossible_____ bba ____ impossible_____

   a ____ impossible_____ abbbb ____ impossible_____

6. Pretend the following table gives CPU burst time, arrival times in the ready queue and priority values (smaller values represent higher priority) for four processes. (10 pts)

Process	Arrival Time	Priority	Burst Time
P1	0	7	4
P2	3	6	3
P3	5	8	1
P4	6	4	3

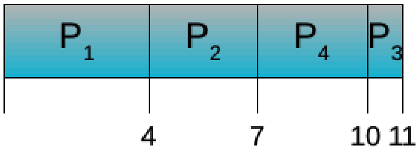
(a) Pretend that these processes are executed under SRTF. Draw a Gantt chart illustrating the execution schedule for these processes.



(b) Give the average turnaround time under SRTF. Don't simplify your answer, just leave it as a fraction.

(4 + 5 + 1 + 5)/4

(c) Pretend that these processes are executed under non-preemptive priority. Draw a Gantt chart illustrating the execution schedule.



(d) Give the average waiting time under non-preemptive priority. Don't simplify your answer, just leave it as a fraction.

(0 + 1 + 5 + 1)/4

7. A multi-threaded program looks like the following:

```
volatile int counter = 500;
void *worker(void *arg) {
   counter--;
   return NULL;
}
int main(int argc, char *argv[]) {
   pthread_t p1, p2, p3;
   pthread_create(&p1, NULL, worker, NULL);
   pthread_create(&p2, NULL, worker, NULL);
   pthread_create(&p3, NULL, worker, NULL);
   pthread_join(p1, NULL);
   pthread_join(p2, NULL);
   pthread_join(p3, NULL);
   printf("%d\n", counter);
   return 0;
}
```

Assuming `pthread_create()` and `pthread_join()` do not return an error, which outputs are possible? Please write "possible" or "impossible" for each of the following values. (9pts)

502 ____ impossible_____ 501 ____ impossible_____

500 ____ impossible_____ 499 ____ possible_____

498 ____ possible_____ 0 ____ impossible_____

8. You are given multi-threaded code and the initial state (the "inputs") of the program. Your task is to figure out the possible outputs of the code. (16 pts)

Here is the initial state of some variables in the first program we examine:

```
int g = 10;
int *f = NULL;
```

Here are the code snippets for each of two threads:

```
Thread 1                           Thread 2
-----------                        ------------
lock(m);                           lock(m);
f = &g;                            f = NULL;
... // do some other stuff         unlock(m);
printf("%d\n", *f);
unlock(m);
```

(a) What are all the possible outputs of this program, given an arbitrary interleaving of Threads 1 and 2?

10

The code gets rewritten as follows, to make the lock more “fine grained” by moving the “other stuff” out of the critical section:

```
Thread 1                            Thread 2
-----------                         ------------
lock(m);                            lock(m);
f = &g;                             f = NULL;
unlock(m);                          unlock(m);
... // do some other stuff
Lock(m);
printf("%d\n", *f);
unlock(m);
```

(b) What are all the possible outputs of the program now?

10 or segmentation fault

(c) You’ve been taught that locks around critical sections prevent “bad things” from happening. Is that true in part (b)? If so, why? If not, why not?

No. The two critical sections should really be one

Let’s examine another program, again with two threads:

```
Thread 1                            Thread 2
-----------                         ------------
pending = 1;                        pending = 0;
while (pending) {
  printf("hello\n");
}
```

(d) What are the possible outputs of this program, given an arbitrary interleaving of Threads 1 and 2?

0 to arbitrary number lines of hello

(e) How could we re-write the code such that Thread 2 would only run after “hello” has been printed at least one time?

One of many possible solutions:

```
int iOnce = 0;

Thread 1                Thread 2
------------            ------------
pending = 1;            while (iOnce == 0)
while (pending) {           ;
    printf(“hello\n”);  pending = 0;
    iOnce = 1;
}
```

9. Assume you are implementing a producer-consumer shared buffer (which can be used by producer threads to pass data to consumer threads), but that the buffer is *unbounded*; in other words, it does not have a limit as to how big it can get. Can we implement this producer-consumer buffer with only one condition variable? If yes, please write down your implementation based on the following code we discussed in class. If not, please explain why. (10 pts)

```
void *producer(int loops) {
  for (int i=0; i < loops; i++){
    Mutex_lock(&m);         //p1
    while (count == MAX)    //p2
      Cond_wait(&E, &m);    //p3
    put(i);                 //p4
    Cond_signal(&F);        //p5
    Mutex_unlock(&m);       //p6
  }
}
```

```
void *consumer(int loops) {
for (int i=0; i < loops; i++){
    Mutex_lock(&m);         //c1
    while (count == 0)      //c2
      Cond_wait(&F, &m);    //c3
    int tmp = get();        //c4
    Cond_signal(&E);        //c5
    Mutex_unlock(&m);       //c6
    printf("%d\n", tmp);
  }
}
```

A: For an unbounded buffer, only one condition variable will be needed:

P2, P3, and C5 are not needed any more.

```
void *producer(int loops) {
  for (int i=0; i < 1000000; i++){
    Mutex_lock(&m);         //p1
    ~~while (count == MAX)    //p2~~
      ~~Cond_wait(&E, &m);    //p3~~
    put(i);                 //p4
    Cond_signal(&F);        //p5
    Mutex_unlock(&m);       //p6
  }
}
```

```
void *consumer(int loops) {
  for (int i=0; i < 1000000; i++){
    Mutex_lock(&m);         //c1
    while (count == 0)      //c2
      Cond_wait(&F, &m);    //c3
    int tmp = get();        //c4
    ~~Cond_signal(&E);        //c5~~
    Mutex_unlock(&m);       //c6
    printf("%d\n", tmp);
  }
}
```

Student ID: ___________________

This is one scratch page.