Please print in pen:

Waterloo Student ID Number:

WatIAM/Quest Login UserID:

UNIVERSITY OF WATERLOO

Final Exam - Spring 2024 - ECE 252

1. Before you begin, make certain that you have one **2-sided booklet with 10 pages**. You have **120 minutes** to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question.

2. Please read all questions before starting the exam as some of the questions are substantially more time consuming. Read each question carefully. Make your answers as concise as possible. **If there is something in a question that you believe is open to interpretation, then please write your interpretation and assumptions!**

3. All solutions must be placed in this booklet. If you need more space to complete an answer, you may be writing too much. However, if you need extra space, use the blank space on the last page of the exam clearly labeling the question and indicate that you have done so in the original question.

Good Luck!

Question	Points Assigned	Points Obtained
1	**40**	
2	**28**	
3	**12**	
4	**20**	
Total	**100**	

1. (40 points) True-False with explanation.

For each question:

- Circle your answer and write your explanation below each question.
- Explanations should not exceed 3 sentences.
- One point for correct true-false.
- One point for correct explanation.
- No points for any explanation if true-false is incorrect.

1. The state of a thread could change from "ready" to "terminated."

True False

2. The overhead of creating a new thread in a running process is greater than that of creating a new process.

True False

3. Within the same process, any thread can access another thread's stack.

True False

4. With two modes of operation (i.e., kernel mode and user mode), a single bit is sufficient to track whether execution is in kernel mode or user mode.

True False

5. Communication between two threads within a process does not require kernel involvement.

True False

6. When a process is blocked, it is always swapped out of DRAM to disk.

True False

7. If multiple wait operations are performed on a semaphore, the value of the semaphore can become negative.

True False

8. In x86, a normal read is atomic for an integer.
True False

9. A monitor is one mutex for mutual exclusion with zero or more condition variables.
True False

10. The signal and broadcast operators on a condition variable do not change its state if the wait queue of the condition variable is empty.
True False

11. Using condition variables, a thread can safely block within a critical section without causing a deadlock.
True False

12. Deadlock prevention methods are commonly implemented in most modern operating systems.
True False

13. For single-item resources, deadlock has already occurred if there is a cycle in the resource allocation graph.
True False

14. In a multiprocessor system, a thread can ensure mutual exclusion by disabling interrupts on the processor it is running on.
True False

15. Implementing mutual exclusion is only achieved using atomic read-modify-write instructions, such as `test_and_set`.
True False

16. Synchronous send, asynchronous receive is the most common message-passing configuration.
True False

17. IPC through file system requires more system calls per communication than shared memory.
True False

18. In client/server networking, the 'send' system call in the client program returns success only after the data has been successfully received in the server program by the 'receive' system call.
True False

19. UNIX pips are unidirectional, which means either the child can send the parent a message through the pipe or the parent to child, but not both.
True False

20. The `inotify` API does not report file accesses and modifications that may occur because of `mmap`.
True False

2. (28 points) Short answers.

Consider the following pseudocode (it will be used for 1-4):

Producer

```
1. [produce item]
2. lock( mutex )
3. while count == BUFF_SIZE
4.    cond_wait( full, mutex )
5. end while
6. [add item to buffer]
7. count++
8. cond_signal( empty )
9. unlock( mutex )
```

Consumer

```
1. lock( mutex )
2. while count == 0
3.    cond_wait( empty, mutex )
4. end while
5. [remove item from buffer]
6. count--
7. cond_signal( full )
8. unlock( mutex )
9. [consume item]
```

The buffer is initially empty and BUFF_SIZE = 20.

1. (3 points) Suppose we create 50 threads: the first 25 are producers, and the remaining 25 are consumers. If thread A is the first producer to lock the mutex, is it always the one that inserts the first item into the buffer? Explain.

2. (3 points) Suppose we create 50 threads: the first 25 are producers, and the remaining 25 are consumers. If thread B is the first consumer to lock the mutex, is it always the one that removes the first item inserted into the buffer? Explain.

3. (3 points) Suppose we first create 20 producer threads, which all run and terminate. Next, we create two more producer threads, A and B. Since the buffer is full, A and B become blocked. Then, we create two consumer threads, C and D, which both run and terminate. Is it possible for A to finish while B starves forever? Explain.

4. (3 points) Suppose we first create 20 producer threads, which all run and terminate. Next, we create two additional producer threads, A and B. Since the buffer is full, A and B become blocked. Then, we create two consumer threads, C and D, which both run and terminate. After this, we enter an infinite loop where we continuously create two threads—one producer and one consumer. Is it possible for thread A to be signaled infinitely many times and still starve forever? Explain.

5. (4 points) Consider a multi-threaded program that takes 8 seconds to execute on a single core and 1 second to execute on 10 cores. Let S denote the portion of the application that must be performed serially. Can we provide an upper bound, a lower bound, or both for S? If so, what are those bounds? Please provide your answer as a rational number (i.e., a fraction x/y of two integers). Show your work.

6. (2 points) What is async-signal safe?

7. Consider the following implementation for NTFS journaling:

```
(1) Record the change(s) in the log file in the cache.
(2) Modify the volume in the cache.
(3) The cache manager flushes the log file to disk.
(4) Only after the log file is flushed to disk, the cache manager flushes the
volume changes.
```

   (1 point) What is the recovery process if the system shuts down while doing (1) and before starting (2)?

   (1 point) What is the recovery process if the system shuts down while doing (2) and before starting (3)?

   (1 point) What is the recovery process if the system shuts down while doing (3) and before starting (4)?

   (2 points) What is the recovery process if the system shuts down after starting (4) and before finishing it?

8. (3 points) In the Byzantine Generals problem, if there are *d* disloyal participants, how many total participants are needed for the loyal lieutenants to reach consensus? How does this number change if the general is known to be loyal or disloyal?

9. (2 points) Explain a common method through which two processes running on separate machines can synchronize.

3. (12 points) Deadlock.

1. (2 points) Suppose we use the Banker's algorithm to decide whether to grant resource requests to threads. The algorithm aims to keep the system in a 'SAFE' state by denying resource requests and putting the requesting thread to sleep if granting the request would result in an 'UNSAFE' state and waking it only when the request can be granted safely. What constitutes a SAFE state? [Define in no more than two sentences.]

2. (6 points) Suppose that we have the following resources: R1, R2, and R3 and threads T1, T2, T3, and T4. The total number of each resource is:

R1	R2	R3
12	12	9

Further, assume that the current and maximum allocations of threads are as follows:

Thread ID	Current allocations			Maximum		
	R1	R2	R3	R1	R2	R3
T1	2	3	1	4	4	9
T2	5	3	4	6	3	4
T3	1	3	2	5	3	3
T4	2	2	1	4	2	8

Is the system in a safe state? If "yes," show a non-blocking sequence of thread executions. Otherwise, provide a proof that the system is unsafe. Show your work and justify each step of your answer.

3. (4 points) State the four conditions necessary for deadlock to occur.

4. (20 points) What the fork!

We want to create a multi-threaded program that can be used to tell everyone how much we love our favorite ECE 252 course. So, in our first attempt, we get the following program as a starting point:

```
1. void* func1(void* args) {
2.    printf("is\n");
3.    return NULL;
4. }
5. void* func2(void* args) {
6.    printf("ECE252\n");
7.    return NULL;
8. }
9. int main(void) {
10.   pid_t pid;
11.   pthread_t pthread;
12.   int *ret = (int*) malloc(sizeof(int));
13.   int status;
14.   pid = fork();
15.   if (!pid) {
16.         pthread_create(&pthread, NULL, func2, (void*) ret);
17.         pthread_join(pthread, NULL);
18.   } else {
19.         printf("the\n");
20.   }
21.   printf("best!\n");
22.   return 0;
23.}
```

1. (10 points) List all the possible outputs that this program could produce when run. Assume that calls to `fork` and `pthread_create` always succeed. Add more columns if you need.

2. (10 points) Modify the program such that its output is always the following:

```
ECE252
is
the
best!
```

Fill in the blanks (next page) to show your changes to the original program.
- You must use `pthread_create` at least once.
- You may not call `printf` or the helper functions (`func1`/`func2`) directly.
- Write at most one statement per line. You may not need all lines.

```
1. void* func1(void* args) {
2.    printf("is\n");
3.    return NULL;
4. }
5. void* func2(void* args) {
6.    printf("ECE252\n");
7.    return NULL;
8. }
9. int main(void) {
10.   pid_t pid;
11.   pthread_t pthread;
12.   int *ret = (int*) malloc(sizeof(int));
13.   int status;

      ----------------------------------------

14.   pid = fork();
15.   if (!pid) {

            ----------------------------------------

16.         pthread_create(&pthread, NULL, func2, (void*) ret);
17          pthread_join(pthread, NULL);

            ----------------------------------------


            ----------------------------------------


            ----------------------------------------


18.   } else {

            ----------------------------------------

19.         printf("the\n");

            ----------------------------------------

20.   }

----------------------------------------

21. printf("best!\n");
22. return 0;
23.}
```