CSPB 3753 - Fall 2024 - Knox - Operating Systems

Started on	Friday, 8 November 2024, 8:48 PM
State	Finished
Completed on	Friday, 8 November 2024, 10:21 PM
Time taken	1 hour 33 mins
Grade	**75.92** out of 100.00

Question **1**

Partially correct

Mark 18.00 out of 20.00

Term Definitions:

1. Performing more than one task simultaneously.
2. Used to control access to a single or multiple resources by multiple processes.
3. An instruction (or set of instructions) indivisible or uninterruptible and appears to occur without being interrupted.
4. A sequence of time where process does not perform IO.
5. Kernel support for thread scheduling and management.
6. The sequence of time to complete execution of a process.
7. Time slices are assigned to each process in equal portions and in circular order.
8. Assigns a proportion of CPU processing time to each task based on the priority and amount of time previously assigned.
9. An API that allows a program to control multiple different threads of work that overlap in time.
10. A boolean lock variable used to synchronize data access.
11. Rapidly switching between processes, thereby allowing each process to make progress.
12. The sequence of time to begin executing a process.
13. System is analyzed to determine if the system is in an unsafe state.
14. State where there is no sequence of task execution that allows all resources to be allocated to all tasks.
15. A simple loop that checks to see if resource is available.
16. State where a set of processes are each waiting for an event that can be caused only by another process in the set.
17. An area within the code in which the process may be changing shared variables.
18. Situation where the outcome of the execution depends on the particular order in which the access to shared data takes place.
19. A sequence of time where process is performing IO.
20. System is analyzed to determine if a request can be granted without entering an unsafe state.

Place the number of the best definition next to each term

atomic instruction	3 ✔	mutex	10 ✔
CPU burst	4 ✔	parallelism	1 ✔
critical section	17 ✔	Round Robin	7 ✔
deadlock avoidance	20 ✔	turnaround time	6 ✔
kernel threads	5 ✔	unsafe state	16 ✖

Your answer is partially correct.

9 of your answers are correct.

Question **2**

Correct

Mark 2.00 out of 2.00

_____ is the simulation of running in _____.

Select one:

- a. A CPU, a GPU
- b. A process, threads
- c. Asynchronous, serial order
- d. Concurrency, parallel ✔

Your answer is correct.

Question **3**

Correct

Mark 2.00 out of 2.00

Why would a multiple process solution be better than a multiple thread solution?

Select one:

- a. processes are scheduled by OS
- b. processes are isolated/protected from other processes ✔
- c. threads have higher context switching costs
- d. threads share data and other resources

Your answer is correct.

Question **4**

Correct

Mark 2.00 out of 2.00

Please selected the **best** answer for the question.

When a process is forced to give up the processor before its completion, this is an example of which behavior?

Select one:

- a. preemptive ✔
- b. multi-programming
- c. multi-tasking
- d. cooperative

Your answer is correct.

Question **5**

Correct

Mark 2.00 out of 2.00

What is the name of a situation where several processes access and manipulate the same data at the same time causing different outcomes?

Select one:

- a. race condition ✔
- b. interrupt
- c. I/O blocking
- d. concurrency

Your answer is correct.

Question **6**

Correct

Mark 2.00 out of 2.00

What is the name for code that can safely be called again after being interrupted?

Select one:

- a. an interrupt handler
- b. kernel thread
- c. thread safe
- d. reentrant ✔

Your answer is correct.

Question **7**

Correct

Mark 2.00 out of 2.00

What is a Spin Lock?

Select one:

- a. trying to acquire a lock by repeatedly checking if the lock is available. ✔
- b. a lock policy for flipping the current state of the lock
- c. a lock that changes from locked to unlocked
- d. a random access method for obtaining a lock

Your answer is correct.

Question **8**

Correct

Mark 2.00 out of 2.00

What problem can arise when implementing synchronization with Semaphores?

Select one:

- a. Mutual Exclusion
- b. Race conditions
- c. Deadlock ✔
- d. Memory leaks

Your answer is correct.

Question **9**

Correct

Mark 2.00 out of 2.00

A semaphores are accessed through two atomic operations P() and V(). This correspond to what two other names for the functions?

Select one:

- a. lock() and unlock()
- b. wait() and signal() ✔
- c. unlock() and lock()
- d. signal() and wait()

Your answer is correct.

Question **10**

Correct

Mark 2.00 out of 2.00

Given the following code with critical section of task 1 and 2 (C1 and C2):

```
/* variables shared between 2 threads */
Lock mutex;
Condition CV;
int state=0;
```

Thread T1:

```
C1;  // code C1
lock(mutex);
state=1;
unlock(mutex);

signal(CV);
```

Thread T2:

```
lock(mutex)
while(!state) {
  unlock(mutex);
  wait(CV);
  lock(mutex);
}
unlock(mutex);
C2;   // code C2
```

Select one:

- C1 always execute followed by C2 executing
- C2 always executes followed by C1 executing
- C2 never executes
- C2 might not execute ✔

Your answer is correct.

Remember that a signal may occur when no process is waiting. If T2 does not reach the wait() before T1 signals, T2 will be stuck waiting for a signal that never occurs.

Question **11**

Correct

Mark 5.00 out of 5.00

Which of the following are shared by threads of a process?

Select one or more:

- [x] a. data ✔
- [x] b. code ✔
- [x] c. heap ✔
- [ ] d. stack
- [ ] e. CPU context
- [ ] f. program counter

Your answer is correct.

Question **12**

Partially correct

Mark 1.67 out of 5.00

Which of the following are benefits of using a Multi-Tasking?

Select one or more:

- [x] a. supports interactive applications ✔
- [ ] b. makes applications cooperative
- [ ] c. fault isolation
- [x] d. forces the CPU to not be idle ✖ CPU can still be idle if all processes are in the wait queue
- [x] e. efficient use of CPU ✔

Your answer is partially correct.

You have correctly selected 2.

Question **13**

Correct

Mark 5.00 out of 5.00

Which of the following are the parameters to the POSIX thread create function?

Select one or more:

- [x] a. routine to have the thread execute ✔
- [x] b. pointer to data to pass to thread ✔
- [x] c. pointer of where to store the thread id ✔
- [ ] d. program filename to load into thread
- [ ] e. integer value to pass to thread
- [ ] f. thread id to use for new thread

Your answer is correct.

Question **14**

Correct

Mark 4.00 out of 4.00

Which of the following conditions can you guarantee from the three processes below?

(Assume s1 and s2 are semaphores initialized to 0)

```
P0:  wait(&s1);  x1;  x2;  wait(&s2);  x3;
P1:  wait(&s1);  y1;  signal(&s2);  y2;
P2:  z0;  signal(&s1);  z1;  signal(&s1);
```

Select one or more:

- [ ] a. x1 executes before y1
- [x] b. y1 executes before x3 ✔
- [ ] c. z1 executes before x1
- [ ] d. x1 executes before z1

Your answer is correct.

Question **15**

Partially correct

Mark 11.25 out of 15.00

Process	Arrival Time
P1	0
P2	20
P3	10

Given the data and Gantt chart for execution of an unknown policy,

Calculate the requested performance measurements.

Assume that processes are always in the ready queue.

Time: 0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160
CPU	P1	P1	P1	P2	P2	P1	P3	P3	P1	P1	P3	P1	P1	P1	P1	IDLE

All processes have completed by the 150th tick.

Round performance values to nearest integer:

What is the turnaround time for P1? 150 ✔

What is the turnaround time for P2? 30 ✔

What is the turnaround time for P3? 100 ✔

What is the average response time? 23 ✖

Your answer is partially correct.

3 of your answers are correct.

Question **16**

Complete

Mark 0.00 out of 10.00

Adding Semaphores for Synchronization

There are 3 processes (X, Y, Z) that each have 4 code blocks. The code blocks for a process are executed in sequence (1, 2, 3, 4). However, there are constraints between the processes. Below you must add the appropriate semaphore operations, `signal(...)` and `wait(...)` before, between, and/or after each of the process code blocks. Place the signal/wait calls in the correct column(s) for the process.

Hint: keep it simple by:

- using semaphore *i* to code for constraint *i* (without assumptions about other signaling).
- place all signal/wait calls for the first constraint using semaphore s1, then add signal/wait(s) for each of the other constraints.
- use `signal(s1)` and `wait(s1)` when using semaphore 1
- initialize value for each semaphore (i.e. s# = 3 to initialize semaphore to 3)

Code block execution ordering must follow these synchronization constraints:

1. X1 must be the first statement to execute
2. Y3 must execute after Z2
3. Z3 must execute after Y4 AND X3
4. X4 must execute after EITHER Y4 OR Z4, or both
5. process X should be the last to exit

```
// Initialization of semaphores (one for each constraint):
Semaphore:
               s1 = ____0____
               s2 = ____0____
               s3 = ____0____
               s4 = ____0____
               s5 = ____0____

// Place signal(<id>) and wait(<id>) calls before and/or after the process code blocks
Process X              Process Y              Process Z

wait(s1)               -                      -

<X1>                    <Y1>                  <Z1>

signal(s1)             wait(s2)               signal(s2)

<X2>                    <Y2>                  <Z2>

wait(s3)               signal(s3)             signal(s4)

<X3>                    <Y3>                  <Z3>

signal(s4)             signal(s5)             -

<X4>                    <Y4>                  <Z4>

wait(s5)               -                      -
```

Comment:

Question **17**

Complete

Mark 8.00 out of 10.00

The code below produces a race condition.

```
 1. int available_resources = 5;
 2. int count_increased = 0;
 3. int count_decreased = 0;
 4.
 5. // Decrease available_resources by count
 6. // return 0 if no resources are avail,
 7. //              1 if successful
 8. int decrease_count(int count)
 9. {
10.     if (available_resources >= count)
11.     {
12.         available_resources -= count;
13.         count_decreased += count;
14.         return 1;
15.     }
16.
17.     return 0;
18. }
19.
20. // Increases available_resources by count
21. void increase_count(int count)
22. {
23.     available_resources += count;
24.     count_increased += count;
25. }
```

Identify the data involved in the race condition. Identify all the locations (include line numbers) in the code where the race condition occurs.

The race condition that occurs is from the **available_resources** integer that is originally defined on **line 1** with an initial value of 5. Both the **decrease_count** and **increase_count** functions access this shared variable that is responsible for the race condition in this code.

The lines of code where this shared variable is accessed by the functions that cause the race condition are:

- **Line 10** - Initial if statement of checking if the available resources is greater than the count
- **Line 12** - Decrementing the available resources variable by count
- **Line 23** - Incrementing the available resources variable by count

Comment: Lines 10, 12, 13, 23, 24 are all part of the race condition

Question **18**

Complete

Mark 4.00 out of 7.00

Explain the key differences between a mutex and a semaphore in the context of process synchronization. Additionally, provide a scenario from your labs or programming assignments where using a semaphore would be more appropriate than using a mutex.

The easiest way to think about the difference between a mutex and a semaphore in general is a mutex is like having a singular key to open a lock where as a semaphore is like having a code that will open a lock that can be given to other people. Its like the difference between a safe that opens with a code and one that opens with a key.

In the context of process synchronization specifically, mutexes are really good for if you are requiring that only one thread has access to a specific resource at a time. On the contrary, semaphores are good if you want multiple threads to have access to multiple resources all at the same time.

It is important to use each one in the context of where it is best. If you need multiple threads to have access to a specific resource at the same time, this would be a case where using semaphores would be a lot more beneficial than mutexes.

Comment: expected a reference to a lab or programming assignment that can use a semaphore.

Question **19**

Complete

Mark 1.00 out of 1.00

Write some comments about the exam or write the justification on your answers to some difficult questions (may allow for partial credit).
You should NOT write comments for every question, just a few that you felt were the **most** difficult for you.
You should **write "no comments"** in the answer if you have none.

To make a comment,
To not make a comment.

Thank you for the hints.

Comment: