CPSC 3220 – Exam #2 March 28, 2018 Name:_____

1.(3 points) True / False - If a multi-threaded program runs correctly in all cases on a single time-sliced processor, then it will run correctly if each thread is run on a separate processor of a shared-memory multiprocessor. Justify your answer.

True
To run correctly in all cases on a single processor implies that the program's threads can execute in any interleaving — that is, they are independent of the order of instruction execution.

```
1:      void RWLock::donwRead() {
2:              lock.acquire();
3:              activeReaders--;
4:              if (activeReaders == 0 && waitingWriters > 0) {
5:                      writeGo.signal();
6:              }
7:              lock.release();
8:      }
```

2. (3 points) In the readers / writers lock example for the function RWLock:doneRead (above), why do we use writeGo.Signal() rather than writeGo.Broadast() at line 5?

When a read function finishes, at most only one writer can make progress and any one of the writers can make progress. So, a broadcast is wasteful and unnecessary.

3. (3 points) How can a semaphore be used as a mutual exclusion lock?

Create/Initialize the semaphore to 1.

4. (3 points) How do semaphores differ from locks & condition variables in synchronization?

A semaphore has memory and condition variables have no memory
A semaphore can have any non-zero positive integer value and locks are binary.

Consider the following simple implementation of a hybrid user-level/kernel-level lock.

```
1:  class TooSimpleFutexLock {
2:      private :
3:          int val;
4:      public :
5:          TooSimpleMutex() : val (0) { } // Constructor

6:          void acquire () {
7:              int c;
8:              // atomic_inc returns ↢old↢ value
9:              while ((c = atomic_inc (val)) != 0) {
10:                 futex_wait (&val , c + 1);
11:             }
12:         }

13:         void release () {
14:             val = 0;
15:             futex_wake (&val , 1);
16:         }
17: };
```

5. (3 points) The goal of this code is to avoid making expensive system calls in the uncontested case of an acquire on a FREE lock or a release of a lock with no other waiting threads. This code fails to meet this goal at LINE 15. Why?

Release always makes a system call, even if there are no waiting threads.

6. (3 points) A corner case (Lines 9-10) can occur when multiple threads try at acquire the lock at the same time. It can show up as occasional slowdowns and bursts of CPU usage. What is the problem?

2+ threads get into a "livelock" loop. A first thread reads and increments the val, then a second thread reads and increments the val before the first thread finishes the update. The 2+ threads begin to fight each other until one is able to read, increment and update successfully.

7. (3 points) A corner case (Lines 9-10) can cause the mutual exclusion correctness condition to be violated, allowing two threads to both believe they hold the lock. What is the problem?

val can be repeatedly incremented by each thread, so it is possible for it to wrap around to 0, allowing multiple threads to simultaneously believe they have acquired the lock.

8. (3 points) What is a race condition among threads?

A race condition occurs when the behavior of a program depends on the interleaving of operations of different threads. The output is determined by the random order of execution of independent operations.

9. (3 points) How do locks and conditional variables prevent a race condition among threads?

A lock add synchronizations (or mutual exclusion). Only one thread can execute a critical section of code at a time.

10. (3 points) How do conditions variables work with locks to stop busy waiting?

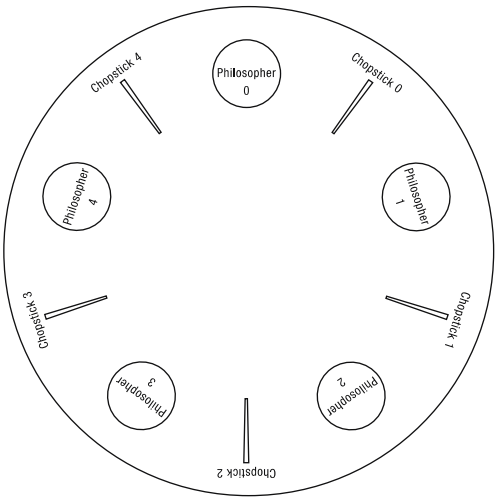
A conditional variable is a synchronization object that allows a thread to wait for a change to a shared state. A thread in busy wait keeps polling a shared state looking for a change, but a conditional variable will signal a waiting thread when the change occurs.

11. (8 points) List four of the six best practices for coding with shared objects and locks.

Any four

1. Consistent structure / Always do things the same way
2. Synchronize with locks and condition variables
3. Acquire the lock at the beginning of a method and release at the end of a method
4. Hold the lock before using a condition variable
5. Wait in a while() loop
6. Do not sleep while holding a lock

12.(3 points) Consider the variation of the Dining Philosophers problem shown to the right, where all unused chopsticks are placed in the center of the table and any philosopher can eat with any two chopsticks.
One way to prevent deadlock in this system is to provide sufficient resources. For a system with n philosophers, what is the minimum number of chopsticks that ensures deadlock freedom? Why?



n+1 suffices and is the minimum.
It suffices because in any state, if any philosopher wishes to eat, some philosopher can finish eating, returning chopsticks to the pool, and allowing any other philosopher to eat

13. (4 points) List the four conditions necessary for deadlock to occur:

Limited Resources - Resources are exclusively held
No Preemption - Resources cannot be taken away
Multiple Independent Requests - Wait while holding
Circular Waiting – Processes can wait while holding resources on processes holding other resources.

14. (3 points) Suppose you build a system using a staged architecture with some fixed number of threads operating in each stage. Assuming each stage is individually deadlock free, describe a way to guarantee that your system as a whole cannot deadlock. You only need to break one of the four necessary conditions for deadlock.

(1) no limited resources -
Make the queues infinitely large so that a stage never blocks when sending a message.
(2) preemption is allowed -
When a pipe fills up, a stage holding a lock is restated.
(3) no multiple requests -
Each stage can only request one lock.
(4) no circular waiting -
Arrange the stages into a directed graph so that there cannot be a cycle of stages waiting for each other.

Banker's Algorithm Usage Table

Process	Max Needed	Allocated	Remaining Need
A	5	3	2
B	3	2	1
C	2	0	2

Total Units Available = 6

This is a safe state.

15. (1 point) How many resources are left?

1

16. (1 point) Which Process(es) cannot be given any more resources?

A and C

17. (1 point) Which Process(es) can be given more resources?

B

18. (2 points) Give a sequence of requests and releases that recover all resources.

B requests, B releases, A requests, A releases, C requests, C releases

or B requests, B releases, C requests, C releases, A requests, A releases

19. (2 point) Will the sequence of requests work?

B Request (1), B Releases (3), A Request (1), C Request (2)

Yes

Banker's Algorithm Usage Table for two resources (A, B)

Process	Max Needed	Allocated	Remaining Need
A	(3, 2)	(1, 1)	(2, 1)
B	(3, 2)	(2, 1)	(1, 1)
C	(2, 1)	(1, 1)	(1, 0)

Total Units Available = (5, 3)
This is a safe state.

20. (1 point) How many resources are left?

(1,0)

21. (1 point) Which Process(es) cannot be given any more resources?

A and B

22. (1 point) Which Process(es) can be given more resources?

C

23. (2 points) Give a sequence of requests and releases that recover all resources.

C requests, C releases, A requests, A releases, B requests, B releases

or C requests, C releases, B requests, B releases, A requests, A releases

24. (2 points) Will the sequence of requests work?
C Request (1,0), C Releases (2, 1), B Requests (0, 1), A Requests (2, 0)

No

25.(3 points) For SJF, if the scheduler assigns a task to the processor, and no other task becomes schedulable in the meantime, will the scheduler ever preempt the current task? Why or why not?

No, if a job is scheduled, then it must be the shortest one. Unless an even shorter on arrives, it will remain the shortest one until it finishes.

26. (3 points) Suppose you do your homework assignments in SJF-order. After all, you feel like you are making a lot of progress. Why will this not work?

Shortest job first does not account for deadlines. A long job will be "starved" until the deadline is passed.

27. (3 points) Most round-robin schedulers use a fixed size quantum. Give an argument against a small quantum.

A small time quantum will increase overhead due to the cost of switching contexts and cache interference.

28. (3 points) How could a Multi-level Feedback Queue (MFQ) of 3+ levels cause starvation?

A workload of high priority jobs run at Level 1 and Level 2. The jobs run to completion and are never preempted into the lower levels. Jobs in the lower Levels are never allowed to run.

29. (4 points) Given the following mix of tasks, task lengths, and arrival times, compute the completion and response time for each task, along with the average response time for the FIFO algorithm.

Task	Length	Arrival Time	Completion Time	Response Time
0	35	0	**35**	**35**
1	15	5	**50**	**45**
2	20	15	**70**	**55**
			Average Response Time:	**45**

30. (4 points) Given the following mix of tasks, task lengths, and arrival times, compute the completion and response time for each task, along with the average response time for the SJF algorithm. Assume a time slice of 10 time units.

Task	Length	Arrival Time	Completion Time	Response Time
0	35	0	**70**	**70**
1	15	5	**25**	**20**
2	20	15	**45**	**30**
			Average Response Time:	**40**

31. (4 points) Given the following mix of tasks, task lengths, and arrival times, compute the completion and response time for each task, along with the average response time for the RR algorithm. Assume a time slice of 10 time units.

Task	Length	Arrival Time	Completion Time	Response Time
0	35	0	**70**	**70**
1	15	5	**45**	**40**
2	20	15	**65**	**50**
			Average Response Time:	**53**

Processor utilization	20.0%
Disk	99.7%
Network	5.0%

Measured utilizations of a computer system.

32. Consider a computer system running a general-purpose workload. Measured utilizations are given in the figure above. For each of the following changes, say what its likely impact will be on **PROCESSOR UTILIZATION**, and explain why.
Is it likely to significantly increase, marginally increase, significantly decrease, marginally decrease, or have no effect on the processor utilization?

a) (2 points) Get a faster CPU

A faster CPU will spend less time working and more time waiting for the disk. This will SIGNIFICANTLY DECREASE CPU utilization.

b) (2 points) Get a faster disk

This is the bottleneck. A faster disk will reduce the time spent waiting for disk use. This will SIGNIFICANTLY INCREASE CPU utilization

c) (2 points) Get a faster network

Little time is spent on the network. A faster network will have NO EFFECT on CPU utilization.