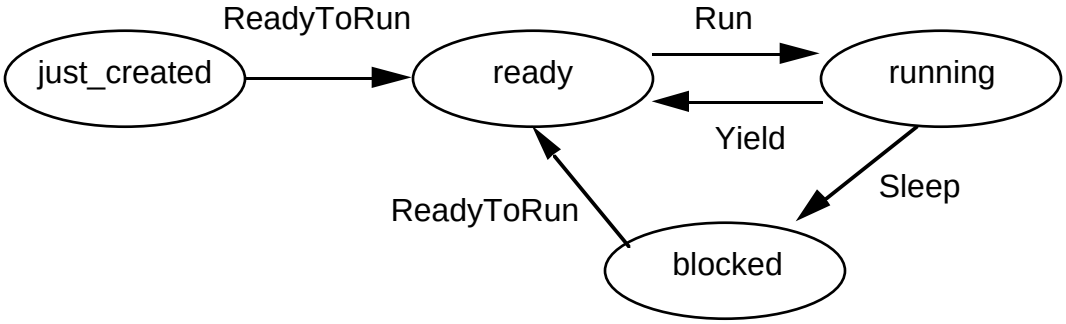
Name: _____

CS 6/73201 **Exam #2** **Advanced OS**

Wednesday 15 March 2000

1. **Draw a diagram showing the 4 states that a Nachos thread can be in, with labels and a brief explanation of the allowable transitions between states. (15 points)**



Scheduler::ReadyToRun puts a thread on the ready list — either a newly created thread or one that is blocked (e.g., corresponding to a thread "wait"ing for a semaphore or condition variable signal)

Scheduler::Run dispatches a thread onto the CPU

Thread::Yield is called to cause a thread to voluntarily give up the CPU — if other threads are on the ready list, this thread is moved onto the ready list and the next thread on the ready list is dispatched onto the CPU

Thread::Sleep causes the current thread to relinquish the CPU and block, dispatching a thread from the ready list onto the CPU (if there is one on the ready list)

2. **Consider the distinction between semaphores, locks, and condition variables. (5 points each = 15 points)**

   a. **Is it possible to use locks <u>without</u> condition variables? Explain.**

      Yes, locks can simply be used to provide mutual exclusion when there is not need for condition variables to provide synchronization.

   b. **Is it possible to use condition variables <u>without</u> locks? Explain.**

      No, condition variables are designed to work only within locked regions, and part of the operation of a condition variable wait is that it releases the associated lock while it is sleeping.

   c. **Is it possible to use semaphores <u>with</u> locks? Explain.**

Yes, the lock can provide mutual exclusion, and a semaphore can provide synchronization, but only outside the critical region protected by the lock. Semaphores can not be used for synchronization inside a locked region; condition variables must be used for that purpose.

3. Explain how each of the following two algorithms for synchronizing physical clocks accounts for network delay during the course of the algorithm. (Note that I am not asking you to summarize each algorithm for me, only how they account for network delay.) (20 points)

a. Christian's algorithm

When a node sends a request to the time server, it measures the time required to receive a reply, and then assumes that half of that time is the network delay. (It then adds that network delay to the time sent by the time server to compute the time to which it should set its own local clock.)

b. The Berkeley algorithm

The master polls all the slaves, and computes their local times using the same idea as in Christian's algorithm to account for the network delay. Then it computes a fault-tolerant average of those clock values, and sends back to each slave the amount by which it should adjust its local clock; since it sends an adjustment rather than an absolute time, the network delay of this last transmission is irrelevant.

4. With Lamport's logical clocks, it would be desirable to be able to compare two timestamps, and if the timestamp on event A is less than the timestamp on event B, declare that A happened before B. Unfortunately, this is not the case. Explain why it is not, perhaps by drawing and explaining an example. (15 points)

See slide 10 ("Limitation of Logical Clocks") in Lecture 12 ("Logical Clocks").

5. A variety of algorithms have been designed to provide mutual exclusion in a distributed environment. For each of the following algorithms, how does a thread know when it can enter the critical section? (Don't explain the entire algorithm, just this part.) (5 points each = 20 points)

a. Central coordinator algorithm

A thread can enter the critical section when it receives a reply message from the coordinator (the "reply" message grants permission).

b. Lamport's algorithm

A thread can enter the critical section when (1) its request is at the top of its own request queue, and (2) it has received a reply message with a timestamp larger than its request from all other threads in the request set.

c. Ricart and Agrawala's algorithm

A thread can enter the critical section when it has received a reply message from all other threads in the request set.

d. Suzuki and Kasimi's broadcast algorithm

A thread can enter the critical section when it has acquired the token.

6. Suzuki and Kasimi's broadcast algorithm uses two vectors, LN and RN. Briefly explain what each of these vectors represents, and how it is updated. (15 points)

The request vector RN contains the largest sequence number received from each thread in a request message. When a thread wants to enter the critical section it increments its sequence number, updates its own RN, and sends the new sequence number to the other threads so they can also update their RNs.

The latest executed vector LN contains the sequence number of the latest executed request from each thread. When a thread releases the critical section it updates its value in LN to indicate that its request has been satisfied.