Engineering and Computer Science 150
Spring Quarter 2022
Midterm Exam
Wednesday, April 27, 2022

NAME PHU LE

A Taken at SBC, no TA or Professor to ask for Clarification

Short Answer

1. (*5 points*) A synchronization mechanism that has two components, one that blocks if its value is 0 and continues if not, and one that increments the value, is a Semaphore. Mutex lock?

2. (*5 points*) A file that contains only the name of another file that the operating system uses to locate the file is a Symbolic link

3. (*5 points*) The difference between deadlock and starvation is that deadlock occurs when multiple processes block, each waiting on another, and Starvation occurs when a process is waiting for a resource that occasionally becomes available but never acquires it.

4. (*5 points*) Which of the following is *not* required for deadlock to occur:

   (a) Hold and wait

   (b) Mutual exclusion

   (c) Unbounded wait

   (d) No preemption

   (e) Circular wait

Is unbounded wait the same (similar) as circular wait? According to lecture slides they were grouped into one. If Unbounded wait is considered to be circular wait then all would be required.
Else
Unbounded wait

Matching

5. (*30 points*) Choose the phrase in the right column that best describes each of the terms in the left column.

a.	E Context switching	A. Entry points to the operating system so a process can use OS services
b.	I Critical section	B. A process that is not running but is waiting for some event
c.	K Multiprogramming	C. Unused memory space when processes that fit into the total unused memory could run
d.	A System calls	D. Scheduling algorithm that executes processes in the order of arrival
e.	J Fence address	E. Changing the executing process from one to another
f.	C Fragmentation	F. Scheduling algorithm that runs a program by a specified time or rejects the process
g.	D First come, first serve	G. The number of processes run in a given period of time
h.	F Deadline scheduling	H. Ability to remove currently running processes from CPU and start another
i.	L Synchronous send	I. Block of code only one process at a time can execute
j.	H Pre-emptive	J. Separates operating system from processes
		K. Many processes run interleaved on one system
		L. Process blocks until it can send a message

Longer answer

6. (*15 points*) A distributed system using mailboxes has two IPC primitives `send` and `receive`. The latter primitive specifies a process to receive from, and blocks if no message from that process is available, even though messages may be waiting from other processes. There are no shared resources, but processes need to communicate frequently about other matters. Is deadlock possible? If so, how? If not, show why it cannot occur.

Yes, a deadlock is possible. Given two processes, P_1 and P_2, where the system chooses to receive from P_1 initially. P_1 sends a message to P_2 and blocks until it receives from P_2. After some time has passed, the system chooses to receive from P_1 and blocks until it does. P_1 sits waiting on P_2, P_2 is sending a message to P_1 but it is not received because the system is waiting on P_1. This causes a circular wait. The mailbox system has P_2's resources and holds it while waiting on P_1, P_1 holds the resources needed for the system while waiting on P_2. There is no preemption or system that prevents this from happening. It has been stated that there are no shared resources so mutual exclusion has been met. Each process/system is holding and waiting. Therefore, the requirements of a deadlock have been met.

Longer answer

7. (*15 points*) The semaphore operations *wait* and *signal* are defined to be atomic. Why? Give an example to show the problem if they are not atomic.

Wait and Signal are defined as atomic to provide mutual exclusion. If they are not atomic, several scenarios could occur ranging from no processes being ran, up to n processes running at the same time. ~~If three processes were to call on wait with no atomic guarantee, the wait could become either -3, -2, -1, 0 and when they lock~~

~~Signal, if the value is something like -1, it could. If three processes were to enter, it could set the count for a bounded semaphore to be above zero on exit, locking out any other processes from entering as the value stays above 0.~~ value

One scenario is having three processes all call wait at the same time, decrementing the value by only one. If it has all three leave at separate times, it increases the value by 3, allowing 2 more processes in than should be allowed. If 3 processes were to signal at some point at the same time but their waits were separated, it would leave a deficit of 2, never allowing another process in.

Longer answer

8. (*20 points*) In class, we showed how to use semaphores to define monitors. This problem reverses this — show how to use monitors to implement general semaphores.

typedef monitor {
Condition x=0;
int count=0;
void entry() {
while(x<0){
x.wait;
/* do something */
}
void exit() {
x.signal;
}
}

Condition x;
int count=1;

void entry() {
count--;
while(count<0) {
x.wait
}
}

void exit() {
if(count<0) {
count++;
}
x.signal;
}

} general semaphore

typedef monitor {
Condition x;
int count=1;
void entry() {
count--;
while(count<0){
x.wait;
}
}
Void exit() {
if(count<0){
count++;
}
x.signal;
} general_semaphore