Midterm

• Grading
  – Problem 5: Ambiguous – everyone got 10 pts.
  – Exam too long: Everyone got 10 points for two lowest problems
  – Average: 75 points
  – Scale: 90, 80, 70, 60, 50

• If you got < 60, you should reevaluate how you are studying the material.
Question 1

In class we discussed how operating systems provide abstractions to the user. One of the most important abstractions is that of a process. Define a process and briefly explain why it is useful to provide the illusion to every program that it is the only one running on the computer.
Process

- A process is an active execution of a program. It consists of an address space, a copy of the executable program, data associated with the running program, and program status contained in CPU registers, along with other status contained in various OS tables.
Process Illusion

• It is useful to provide the illusion that every program has the machine to itself so that the programs can be written without any *a priori* knowledge about what else is going to be running, and without knowing about the actual amounts and types of resources in the system.

• It is easier to write user programs – the OS manages everything for us.

• It also allows for smart centralized resource management.
Common Mistakes

• No mention of address space
• Not mentioning that the illusion makes programming applications significantly easier.
Question 2

We have discussed several situations in which processor architectures have been developed to directly support operating systems needs. Give two examples of this and very briefly describe how they work.
Architectural Features

• Interrupts - separate hardware line(s) for signalling the completion of device operations.
• Processor modes - distinct operating modes of the CPU for protecting certain privileged instructions and interfaces.
• Test-and-set instruction - special instruction for enabling mutual exclusion without disabling interrupts.
• DMA – allowing for memory access without involving the CPU (less of a processor arch.)
Common Mistakes

• Discussing OS architectures instead of processor architectures.
• Discussion types of systems (real-time, general-purpose) instead of processor architectures.
Question 3

Briefly describe some factors that might differentiate between the amount of time it takes to do a normal procedure call in an application program to one of its own procedures compared with the time it might take to perform a system call to an OS procedure.
Procedure Calls within an Application

- Push the parameters onto the stack
- Push the current location onto the stack
- Jump to the procedure
- Read the parameters from the stack
- Execute the procedure
- Return from the procedure
System Calls

• Call a procedure to do the system call
• Put the parameters into registers
• Execute a trap instruction
• Context switch to kernel
• Disable interrupts!
• Execute kernel code
• Possibly wait on I/O operations
• Return from system call
Time Issues

• Procedure call within a program:
  – Only known code gets executed
  – Could get interrupted by other processes

• System calls to OS procedures
  – Extra overhead associated with trap & context switch
  – Unknown code gets executed
  – Possible I/O waiting
  – But, interrupts disabled
Common Mistakes

- Forgetting how procedure calls work
- Forgetting how system calls work
- General confusion
Question 4

A context-switch is when the processor (CPU) stops executing one process and begins executing another. List the necessary steps for the operating system to perform a context-switch, including what information must (typically) be saved.
Steps

1. Save program counter of interrupted process
2. Save registers of interrupted process
   - IR?, SP, Data Registers
3. Switch to dispatcher/scheduler
4. Load registers of new process
5. Load program counter of new process
Common Mistakes

• Forgetting that the scheduler has to run at some point

• Not mentioning that the PC is the first thing to be saved (for the old process) and the last thing to be loaded (for the new process)
In most computers a clock device periodically raises an interrupt to signal the passage of a predetermined amount of time. Describe how one could take advantage of this to implement an operating system facility to periodically run various programs (chosen at run-time).
My Answer

• The OS can keep a queue of sleeping processes with times indicating when to wake them up.
• The list is sorted by time.
• Every clock interrupt, the interrupt handler can check to see if it is time to wake up the process at the head of the list.
Common Mistake

• Writing a poor midterm question (my mistake)

• 80% of the people interpreted this as a scheduling question, which isn’t what I meant.
Tape drives store information much like disk drives. However, the information stored on a tape must be physically accessed by moving the tape past the read/write heads. Because the tape is stored on reels, one must stream the tape forward or backward to get the appropriate data under the read/write head. Furthermore, repositioning the tape for random access is very slow, so the information is almost always accessed sequentially.
Question 6 (cont.)

Discuss how well or poorly FCFS, SSTF, Look, and Scan would work with tape drives. What is likely to be the best strategy for optimizing requests to tape drives? As a result, for what kind of storage are tape drives likely to be most useful?
Tape Drives

• Tapes should be accessed sequentially. That means that FCFS will do poorly, and SSTF may do poorly, unless the data happens to be ordered just right.
• Scan and Look are likely to do a lot better since they sequentialize the accesses.
• Best: Look, since it doesn’t go to the ends.
Tape Drives

• Any reordering that sequentializes the access will generally do better than one that does not.
• Tapes are better for data that is written and/or read sequentially.
• Archiving is a good example, since it is written sequentially, but extracting some but not all files is very slow.
Common Mistakes

• Forgetting how FCFS, SSTF, Look, and Scan work.

• Incorrectly applying these to tape drives.
Question 7

Discuss the differences between *preemptive* and *non-preemptive* scheduling, citing advantages and disadvantages of each one.
Scheduling Types

• Nonpreemptive – Currently scheduled applications keeps the CPU until it finishes, blocks, or voluntarily yields.

• Preemptive – Currently scheduled application can be involuntarily interrupted to allow other applications to run.
Advantages and Disadvantages

• Nonpreemptive
  – Simpler to implement, little overhead
  – Poor response time for interactive applications

• Preemptive
  – Better wait time behavior
  – Better interactive response times
  – More complicated to implement, some overhead
Common Mistakes

• Forgetting the difference between preemptive and nonpreemptive scheduling.
• Thinking that round-robin is the only form of preemptive scheduling.
• Forgetting to mention interrupts, which play a large role in preemption.
• Incorrectly stating advantages and disadvantages.
Question 8

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>50</td>
</tr>
</tbody>
</table>
Schedule with FCFS

<table>
<thead>
<tr>
<th>Job</th>
<th>Start</th>
<th>End</th>
<th>Wait</th>
<th>Turnaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>80</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>80</td>
<td>105</td>
<td>80</td>
<td>105</td>
</tr>
<tr>
<td>2</td>
<td>105</td>
<td>142</td>
<td>95</td>
<td>132</td>
</tr>
<tr>
<td>3</td>
<td>142</td>
<td>156</td>
<td>132</td>
<td>146</td>
</tr>
<tr>
<td>4</td>
<td>156</td>
<td>206</td>
<td>136</td>
<td>186</td>
</tr>
</tbody>
</table>
## Schedule under Non-Preemptive SJN

<table>
<thead>
<tr>
<th>Job</th>
<th>Start</th>
<th>End</th>
<th>Wait</th>
<th>Turnaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>39</td>
<td>15</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>76</td>
<td>29</td>
<td>66</td>
</tr>
<tr>
<td>4</td>
<td>76</td>
<td>126</td>
<td>56</td>
<td>106</td>
</tr>
<tr>
<td>0</td>
<td>126</td>
<td>206</td>
<td>126</td>
<td>206</td>
</tr>
<tr>
<td>0</td>
<td>126</td>
<td>206</td>
<td>126</td>
<td>206</td>
</tr>
</tbody>
</table>
## Schedule under Preemptive SJN

<table>
<thead>
<tr>
<th>Job</th>
<th>Start</th>
<th>End</th>
<th>Wait</th>
<th>Turnaround</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>--   (15 remaining)</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>24</td>
<td>0</td>
<td>14   (turnaround time)</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
<td>39</td>
<td>14</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>39</td>
<td>76</td>
<td>29</td>
<td>66</td>
</tr>
<tr>
<td>4</td>
<td>76</td>
<td>126</td>
<td>56</td>
<td>106</td>
</tr>
<tr>
<td>0</td>
<td>126</td>
<td>206</td>
<td>126</td>
<td>206</td>
</tr>
</tbody>
</table>
Average Wait Time

• Sum wait times and divide by 5
• Mistakes:
  – Forgetting that wait time begins when the process enters the system, not always at time 0
  – Summing service time
Common Mistakes

• Forgetting the scheduling algorithms
• Thinking that preemptive SJN is Round-Robin
• Forgetting that wait time doesn’t include time before the job arrived
• Thinking that process 3 can run at time 0
Question 9

Dijkstra posed each of the following solutions as a potential software solution to the critical section problem and then explained why they fail. Provide your explanation about why each one fails by describing how it violates one of the following rules for solutions to the critical section problem:
Rules

- **Mutual Exclusion** - Only one process at a time should be allowed in its critical section.

- **Isolation** - If a set of processes tries to enter the critical section, only the processes currently competing for the critical section participate in the selection of which process gets to enter the critical section.
Rules

• **Starvation** - Once a process attempts to enter its critical section, it cannot be postponed indefinitely.

• **Bounded Wait** - After a process requests entry into its critical section, only a bounded number of other process may be allowed to enter their related critical sections before the original process enters its critical section
Proposed Solution 1

```c
int turn;
void proc(int i) {
    while(TRUE) {
        <compute>
        while(turn != i);
        <critical section>
        turn = (i+1) % 2;
    }
}

turn = 1;
fork(proc, 0);
fork(proc, 1);
```
How It Fails

• If turn = 1, process 1 is computing and process 0 is at "while(turn != i)" then process 0 can’t get into its critical section even though process 1 isn’t in its critical section and isn’t trying to get in, violating Isolation.
boolean flag[2];
void proc(int i) {
    while(TRUE) {
        <compute>
        while(flag[(i+1) mod 2]);
        flag[i] = TRUE;
        <critical section>
        flag[i] = FALSE;
    }
}
flag[0] = flag[1] = false;
fork(proc, 0);
fork(proc, 1);
How It Fails

• If one process does
"while(flag[(i+1)mod2])", gets interrupted immediately afterwards, then the other process does the same thing, then they both go on to do “flag[i]=TRUE” and enter their critical sections at the same time, violating Mutual Exclusion.
Proposed Solution 3

```java
boolean flag[2];
void proc(int i) {
    while (TRUE) {
        <compute>
        flag[i] = TRUE;
        while (flag[(i+1) mod 2]);
        <critical section>
        flag[i] = FALSE;
    }
}
flag[0] = flag[1] = FALSE;
fork(proc, 0);
fork(proc, 1);
```
How It Fails

• If one process does "flag[i] = TRUE", then gets interrupted immediately afterwards and the other one does the same thing, then both of them will block at "while(flag[(i+1) mod 2]", violating Starvation.
Common Mistakes

• General confusion
Question 10

Suppose that Nachos had Locks and Condition Variables, but not Semaphores. Show how you could implement Semaphores in user code using Locks and Condition Variables (as needed). Show the actual class definition and code.
class Semaphore {
    public:
        Semaphore(int initval);
        ~Semaphore();
        P();
        V();

    private:
        int value;
        Lock *semlock;
        Condition *semcondition;
};
Constructor and Destructor

Semaphore::Semaphore(int initval)
{
    value = initval;
    semlock = new Lock;
    semcond = new Condition;
}

Semaphore::~Semaphore()
{
    delete semlock;
    delete semcond;
}
P()

Semaphore::P()
{
    semlock->Acquire();

    value--;  

    if(value < 0)
        semcond->Wait(semlock);

    semlock->Release();
}

Semaphore::V()
{
    semlock->Acquire();
    value++;
    semcond->Signal(semlock);
    semlock->Release();
}
Common Mistakes

• Forgetting how Semaphores work
• Forgetting how Locks and Condition Variables work
• General confusion