Short Answer Questions (15 pts)

a. What is the fetch-execute cycle? How do interrupt break this cycle and how does control pass to the OS?

The fetch-execute cycle is the normal hardware cycle by which instructions are fetched from main memory to the CPU’s instruction register and executed through interpretation by the hardware. The address of the instruction to be fetched is contained in the PC (program counter) which is automatically incremented to the next instruction in the user’s program. Interrupts break this cycle by jamming the PC with the address of the corresponding interrupt, thus allowing the CPU to start executing code within the OS’s interrupt handling code. (Without this mechanism, the CPU would continue to execute user code without OS intervention – until the user’s program exited.

b. What are the strengths and weaknesses of Shortest Job First Scheduling?

Strengths: theoretically optimal with respect to average turnaround time. Small jobs get finished quickly and are not held up by big jobs. Good performance in practice. Straightforward, intuitive algorithm (not highly complex).

Weaknesses: large jobs can starve. SJF has overhead to predict the next CPU burst for use in selecting the ‘smallest’ job. Overhead for scheduler to search through ready queue to pick smallest job – O(n) where n is size of ready queue.

c. Suppose the SJF scheduler accidentally uses Process B’s actual CPU bursts to predict the future bursts of Process A (using exponential averaging). What would be the impact on who the scheduler chooses from the Ready Queue? What is the impact on overall system performance if this happens throughout the system for all processes (i.e. predictions always based on the wrong process’ actual bursts

Obviously SJF order would be violated. A bad situation would occur if B is actually a very long running job and A is short. Then, the scheduler would select B thinking its next CPU burst was the shortest, and performance would suffer (smaller jobs would pile up behind B). Conversely, if B is actually very short but scheduled according to long job A’s predicted CPU burst, it would have very long turnaround time (unfair to job B but not harmful to other jobs.)

If this mixup happened to all jobs, the scheduling algorithm would no longer be SJF and would be more like random scheduling which does not have very good performance.
2. Frequency of Events (20 pts)
For the following pairs of events or situations, write ‘less than,’ ‘greater than,’
‘approximately equal,’ or ‘cannot predict’ to describe the relative frequency (values) of
the two events. You can use the abbreviations <, >, =, and ? respectively. Explain each
answer in ONE sentence.

a. # of processes in ready queue _____?_____ # of processes in blocked queue
depends on job mix (CPU bound v. I/O bound mix)

b. Length of Round Robin time quantum ____<______ Human response time
RR quantum must be chosen to be small enough so that the scheduler can schedule many
processes and get back to the first to give it service within human response time so
interactive users get good service.

c. Process X is in ready state _____>______ Process X is in blocked state
Processes go from blocked to ready or ready to ready.

d. The scheduler is called _____>______ An interrupt occurred
Scheduler is called after processing each interrupt AND after processing kernel calls.

e. A clock interrupt occurred ______?______ An I/O interrupt occurred
Depends on job mix (CPU bound v. I/O bound mix)

f. P(mutex) is executed ____=______ V(mutex) is executed
Usually P(mutex) critical section V(mutex)

g. Length of time busywaiting in a test-and-set instruction _____>______
   Length of time busywaiting on a semaphore
test-and-set has busywaiting but semaphores block
Also acceptable: = because semaphores are implemented with test-and-set !!

h. # of processes in scheduler’s blocked queue _____>______
   # of processes in semaphore S1’s blocked queue
The former includes the latter because the blocked queue is used across many
semaphores.

i. # times Process X requests I/O ____=______ # times an I/O interrupt is for Process X
Request initiates the I/O and later when I/O completes an interrupt occurs.

j. # times you have watched the Superbowl _____>______ # of times Ginnie Lo has
watched the Superbowl _____>______ # times Edsgar Dijkstra has watched the
Superbowl Any answers correct. Thanks for several funny answers.
3. Exponential Averaging (15 pts)

Two rival factions designing the scheduler for the Linux Red Shoe operating system are arguing about what value of alpha to use in their exponential averaging formula for CPU burst prediction.

Team A wants to use alpha = 0.75 while the Team Z want to use alpha = 0.25.

To make peace, Professor Lo tells them to compromise and use a value of alpha that is the average of these two values, i.e. alpha = 0.5. Which team (A or Z) will be happier with this compromise? In other words, are the weights on each actual burst t(i) using alpha = 0.5 closer to those with alpha = 0.75 or those with alpha = 0.25???

a. First, fill in the table below with the weights each scheme will give to each successive term in the prediction Tau(n+1).

<table>
<thead>
<tr>
<th>Which scheme?</th>
<th>Weight on t(n)</th>
<th>Weight on t(n-1)</th>
<th>Weight on t(n-j)</th>
<th>Weight on t(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (alpha = 0.75)</td>
<td>0.75</td>
<td>(0.75)(0.25)</td>
<td>(0.25) * (0.75)</td>
<td>(0.25)(^{n-10}) * (.75)</td>
</tr>
<tr>
<td>Z (alpha = 0.25)</td>
<td>0.25</td>
<td>(0.25)(0.75)</td>
<td>2(0.75)(^{j}) * (0.75)</td>
<td>(0.75)(^{n-10}) * (.25)</td>
</tr>
<tr>
<td>Ginnie Lo (alpha= 0.5)</td>
<td>0.5</td>
<td>(0.5)(0.5)</td>
<td>(0.5)(^{j+1})</td>
<td>(0.5)(^{n-9})</td>
</tr>
</tbody>
</table>

b. Which team will be happier? Explain briefly.
Team A will be happier because its weights are closer to those of Ginnie Lo’s compromise solution.

c. Which team expects process performance to be more erratic (as reflected in their choice of alpha)? Erratic means widely varying values for CPU bursts over time. Explain briefly.
Team Z expected performance to be erratic so they gave less emphasis to the most recent actual burst value. They give more weight to past history in order to damp the extremes of the erratic bursts.
4. General Semaphores (10 pts)

Suppose a general semaphore S is initialized to the value 10. This semaphore is shared among 100 processes.

a. What is the maximum number of processes that can be in critical section at the same time? _____10_____

b. What is the maximum number of processes that could be blocked on S? ___90_____

c. What is the minimum number of processes that could be in critical section at the same time? ______0_____

d. True or False (Circle one)
The number of processes in critical section + the number of processes blocked on S = 100. Explain. Processes could be doing other things. Number in c.s. could be from 0 to 100. Number blocked could be 0 to 90.

e. When a process executes signal(S), how does it determine whether there is a process waiting to enter critical section? Checks the value of the semaphore’s counter to see if it is <= than 0.

5. Dining Philosophers Problem (10 pts)

a. Does the correct solution for the Dining Philosophers use a general semaphore?

Circle one: Yes No

If yes, explain how that semaphore is used.
The table semaphore is a counting semaphore that only allows |philosophers – 1| (four in our demonstration) philosophers to compete for the chopsticks concurrently.

b. The Dining Philosophers Problem is an abstract representation of a synchronization problem. Describe a realistic problem in the OS management of resources that is parallel to the Dining Philosophers abstraction. Any situation in which a process needs to acquire two resources before it goes into critical section, e.g. a process that wants a read buffer and a write buffer.
6. **Scheduling and Synchronization (15 pts)**

Suppose we wish to design an OS scheduler that chooses the process to become the Running Process based on how long it has been blocked on a semaphore in the past. In particular, it searches the Ready Queue to find that process whose time blocked on a semaphore is the longest. If there are no processes that were blocked on any semaphores, it picks the process whose PCB is at the head of the Ready Queue.

Define any new fields in the PCB, new data structures, new methods that you need here. Those defined earlier are on the pink sheet – you can redefine any of those if you wish.

- PCB.startblock – field in PCB to record beginning of blocked time
- PCB.blockedtime – field in PCB to record total blocked time
- SelectLongestBlocked (Q) - method to select the process whose time blocked on a semaphore is the longest of all processes in that queue. Returns its PCB.
- Find( Q, sema) – method to search queue and find the first process blocked on that sema

```plaintext
Scheduler() {
    if (called by wait(SEMA)) { /* block current Running Process on this semaphore */
        /* Write code here to handle the Running Process that gets blocked */
        RP.startblock = readtime(); /* record start time just before blocking */
        Append(RP, BlockedQ); /* block the former running process */
    }
    /* Write code here to select the next process to run */
    RP = SelectLongestBlocked (ReadyQ); /* choose longest blked on semaphore */
    if (RP == NIL) RP = head( ReadyQ); /* if none blked on sema, choose head */
    if (RP == NIL) idle();
    else /* context switch */
        ContextSwitch(RP, TimeQuantum);
}
```
if (called by signal(SEMA, S-PCB))  { /* unblock process S-PCB on semaphore SEMA */

    /* Write code here to handle the Running Process and to handle the process S-PCB
    that was blocked on SEMA */

    Append(RP, ReadyQ);

    P = Find (BlockedQ, S_PCB);  /* find process blocked on this sema

    P.blockedtime = runtime() – P.startblock;    /* compute total blocked time

    Append(P, ReadyQ);         /* it is ready to run

    /* Write code here to select the next process to run */

    identical to previous page

    ContextSwitch(RP, TimeQuantum);

}
7. **Fork() and Exec()** (15 points)

a. Write pseudo code that causes Process A to fork off two children. These processes must then print messages giving their own PIDs (process IDs) and those of their relatives.

Use the Unix kernel calls: 
- getpid() which returns a process’ own PID
- getppid() which returns a process’ parent’s PID

The parent process should print the message “I am a parent” followed by its own PID and then the PIDs of each of its children.

Each child process should print its own PID followed by the PID of its parent.

```c
main () {
    pid1 = fork();
    if (pid1 == 0) { /* first child */
        me = getpid();
        my-mom = getppid();
        printf("I am child", me, ", and my parent is ", my-mom);
    } else { /* parent */
        pid2 = fork();
        if (pid1 == 0) /* second child */
            me = getpid();
            my-mom = getppid();
            printf("I am child", me, ", and my parent is child", my-mom);
        } else { /* parent */
            me = getpid();
            printf("I am the parent", me, ", and my kids are", pid1, ", and ", pid2);
        }
    }
}
```

b. If these processes coordinate using semaphore SEM initialized to the value 1, will the parent process enter critical section before both of its children do? Why or why not?

No way to predict. The order they enter c.s. depends on the scheduler, not the order in which processes are created.

c. If SEM is initialized to the value 2, what combinations of two processes can be in critical section at the same time? Circle all that can.

<table>
<thead>
<tr>
<th>One parent and</th>
<th>Two children</th>
<th>One child and some other process</th>
</tr>
</thead>
<tbody>
<tr>
<td>One child</td>
<td>YES</td>
<td>Synchronized on a different semaphore</td>
</tr>
<tr>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>