1. Basic Concepts I (15 pts)  Give brief answers. 

Note: the answers below are longer than necessary for full points on the exam. The long answers are here so you can learn from studying this exam.

   a. In a multiprogrammed uniprocessor system, how does control pass from an executing user process to the operating system?

**System call (kernel call):** user process voluntarily asks for service from the OS

**Interrupt:** user process involuntarily gives us control because some event has occurred that needs action (I/O interrupt, timer interrupt,

   b. How does control pass from the operating system to a user process?

**Scheduler context switches to the user process** it has selected to be the running process.

   c. The Process Control Block (PCB) is a key operating system data structure. Why is the PCB so important?

**The PCB keeps track of all information about each process, the unit of work in the OS.**

d. The Program Counter (PC) is a key CPU register. What happens to the PC when an I/O interrupt occurs?

**The hardware causes the PC of the running process to be saved and loads the PC with the address of the I/O interrupt handler.** Thus, the hardware fetch/execute cycles move seamlessly from the user process to the I/O interrupt handler code.

e. List one advantage and one disadvantage of the Shortest Job First scheduling algorithm.

**Advantages:** theoretically optimal for turnaround time, has been shown to have good performance in real systems.

**Disadvantages:** can cause starvation of long jobs
List one advantage of semaphores and one disadvantage of semaphores.

**Advantages:** simple to use; little or no busywait overhead; able to solve a wide variety of synchronization problems.
**Disadvantages:** can be programmed incorrectly; can be hard to keep track of many semaphores

2. **More Basic Concepts II: Frequency of Events (20 pts)**

For the following pairs of events or situations, use ‘less than,’ ‘greater than,’ ‘approximately equal,’ or ‘cannot predict’ to describe the relative frequency (values) of the two events. Use the abbreviations <, >, =, and ? respectively. Explain each answer briefly but clearly.

a. # of processes in ready queue ___?_______ # of processes in blocked queue
   **Depends on the mix of CPU-bound and I/O bound jobs in the system.**

b. Length of Round Robin time quantum ____?_______ next predicted CPU burst using exponential averaging.
   **Next predicted burst can be high or low, depending on process behavior.**

c. Process X is in ready state _____ >= _or > ____ Process X is in blocked state
   **Each time process X is in blocked state, it will go to ready state after it's I/O is done. It may also cycle through ready state several times without going through blocked state (during CPU bursts).**

d. The scheduler is called ____ >= _or > ____ An interrupt occurred
   **The scheduler is called after every interrupt but also after user process has made a kernel call.**
e. A clock interrupt occurred _____?_______ An I/O interrupt occurred
Depends on length of I/O bursts. They could be so short and frequent that
the timer rarely goes off.

f. P(mutex) is executed _____ =______ V(mutex) is executed to protect critical
section code
P(mutex) and V(mutex) are executed before and after the critical section
code.

g. # of processes in scheduler’s blocked queue _____>= or >______ # of
processes in semaphore S1’s blocked queue
When a process blocks for a semaphore, it must also be put into the
blocked queue. However, if the process also does I/O, it will be in the
blocked queue waiting for I/O.

h. In the Dining Philosophers Problem,
# philosophers _____ =______ # chopsticks
One chopstick lies between each pair of philosophers in the circle of
philosophers.

i. Over a 24 hour period, total time CPU is executing user code _see
below______ total time CPU is executing OS code.
> since CPU should be spending most of its time doing user work rather
than OS management overhead
? is acceptable because it is unknown how often the user processes are
requesting OS services

j. # times you have watched the Avatar _____>_________ # of times Ginnie Lo has
watched Avatar _____>_________ # times Edsgar Dijkstra has watched Avatar.
Anyway answer OK here, but Dijkstra passed away in 2002.
3. Synchronizing Siblings in a Tree of Processes (15 pts)

Write pseudocode using fork and exec in which the parent forks off four children, and each of these children fork off four children to yield a tree of processes. All processes are trying to enter critical sections. Your code must ensure that the (four) direct descendants of a given parent node cannot critical section concurrently. Other combinations of processes are allowed to freely execute without synchronization.

How many semaphores do you need? five

main() {

    Int pid1, pid2;
    Semaphore sema1 = new Semaphore(1); //initialized to one
    Semaphore sema2 = new Semaphore[4]; //initialize to one
    for(int i = 0; i < 4; i++){
        sema2[i] = new Semaphore(1);
    }
    for(int i = 0; i < 4; i++){
        pid1 = fork();
        if(pid1 == 0){ //child process
            for(int j = 0; j < 4; j++){
                pid2 = fork();
                if(pid2 == 0){ //child process
                    p(sema2[i]); //critical section
                    v(sema2[i]);
                    break;
                } else if(pid2 > 0){ //parent process
                    p(sema1); //critical section
                    v(sema1);
                }
            }
        } else if(pid > 0){ //parent process
            //critical section
        }
    }
}
3. Exponential Averaging (15 pts)

(a) What is the purpose of exponential averaging and why does it work pretty well?

Exponential averaging predicts the next CPU burst based on past history of actual CPU bursts. It works well because the behavior of user processes is such that past history is a good predictor of future behavior; also, it provides a tunable parameter alpha that can be adjusted to account for erratic v. stable patterns of CPU burst.

(b) Suppose alpha = 1/4. In the estimate of TAU(n+1) what is the ratio of the weights given to two actual bursts that are 4 iterations apart, such as t(8) and t(4), or t(7) and t(3). Show your work for partial credit.

\[
\frac{\text{weight on } t(k)}{\text{weight on } t(k-4)} = ???
\]

\[
= \frac{\alpha \cdot (1 - \alpha)^{(n-k)}}{\alpha \cdot (1 - \alpha)^{(n-(k-4))}}
\]

\[
= \frac{1}{(1 - \alpha)^4} = \frac{1}{(3/4)^4} = \frac{256}{81}
\]

This answer comes by unrolling the recursive formula and finding the weights on t(k) and t(k-4) in TAU(n+1).

(c) What metric would you propose for evaluating the accuracy of exponential averaging predictions? You can use the one from your group solution in HW2 or make up a new one. Give the definition here.

Mean error = \[ \frac{\sum | \text{TAU(i)} - t(i) |}{n} \] where n is number of iterations measured/predicted

Mean square error = \[ \frac{\sum (\text{TAU(i)} - t(i))^2}{n} \] where n is the number of iterations measured/predicted

Others that take into account the difference between measured and predicted values as well as the number of iterations
5. Semaphores (15 pts)

a. Below is the code for a blocking implementation of P() and V() with semaphores.

```c
typedef struct { int value; /* counter */
    struct process list; /* list of processes */
} semaphore;

P(semaphore *S) {
    S->value--;
    if (S->value < 0) {
        Add this process to S-> list;
        block();
    }
}

V(semaphore *S) {
    S->value++;
    if (S->value <=0) {
        remove a process P from S->list;
        unblock(P);
    }
}
```

For P() and V() to function correctly, they must execute atomically. This means that the P() and V() code are themselves critical sections relative to each other.

(i) Why must they execute atomically?

**P() and V() must execute atomically because a “race condition” exists when they increment and decrement the semaphore counter S->value and semaphore list S->list. (Race condition means that unpredictable interleaving of the code can yield undesired/wrong results.)**

(ii) In the space below, write code to ensure that P() and V() execute atomically. You can write “code for P” and “code for V” as shorthand (need not copy all the code above).

```c
while (test-and-set(flag)) /* test-and-set is an atomic hardware instruction */
    code for P or V
flag = false;
```

b. If semaphore S is initialized to 5, and 17 processes have executed P(S) but no processes have executed V(S), what is the value of S->value? ____-12_______ and how many processes are in the list S->list? ____12_________
c. When a process in the semaphore blocked queue is signaled that it can now enter critical section, to where does the OS next move the PCB? Circle all that apply.

Scheduler blocked queue  **Scheduler ready queue**  Running Process

Semaphore ready queue  Semaphore busywait queue  Semaphore hotel

6. Scheduler Pseudo-code (20 pts)

Write pseudo-code for a Scheduler that computes normalized turnaround time for each process. Recall that normalized turnaround time is defined as

\[ NTT = \frac{\text{turnaround time}}{\text{runtime}} \]

Your code should accomplish the following:

- In each case append the running process to the appropriate queue.
- Compute process runtime = total time that the process is executing as the running process.
- Compute process turnaround time = total time from when the process is created to when it exits and becomes a zombie.
- When the process exits (becomes a zombie) compute and print out the process ID and the normalized turnaround time for that process.

The basic code has been provided for selecting next process (round robin) and context switch. You are to add your new code in the spaces provided. You need not use all the lines provided (the purpose of the lines is to keep your answers neat so I can grade them more easily). Add comments next to your code to help explain your solution.

You are free to define any new fields in the PCB, any new global variables or any new procedures.. Define these here, give initial values, give short explanation:

To compute process runtime, must add up the individual bursts it accumulates whenever it is the running process.

To compute process turnaround time, the easiest way to to record its start time when it is created, and subtract that from its completion time, when it becomes a zombie!!

**PCB-> starttime; /* process start time**

**PCB-> runtime; /* cumulative process runtime**

**PCB-> turnaroundtime; /* completion time – start time;**
Scheduler () {

If (called by fork() ) { /* This call was made by the parent process, the current running process. It passes a pointer to CHILD_PCB */

Running_Process --> runtime += runtime() – start_time;
/* parent accumulates more runtime – must recent burst */

Append(Running_Process, Ready_Queue); /* parent gets put in Ready_Queue */

Child_PCB \rightarrow start_time = runtime(); /* newly created process start_time */

Append(Child_PCB, Ready_Queue); /* child gets put in the Ready_Queue */

Running_Process = Head(Ready_Queue); /* get next process */
start_time = read_time(); /* record its start time */
if (Running_Process != NIL) Context_Switch (Running_Process, DELTA);
else IDLE(); /* start next process or idle until next interrupt */

else if (called by I/O routine) { /* running process requested I/O */

Running_Process --> runtime += runtime() – start_time;

Append(Running_Process, Wait_Queue); /* blocks for I/O */

Running_Process = Head(Ready_Queue); /* get next process */
start_time = read_time(); /* record its start time */
if (Running_Process != NIL) Context_Switch (Running_Process, DELTA);
else IDLE(); /* start next process or idle until next interrupt */
} else if (called by I/O interrupt handler {

    Running_Process -->runtime += readtime() – start_time;
    Append(Running_Process, Ready_Queue);

    Append (Select(Wait_Queue, IO_done_PID) , Ready_Queue);
    /* find process whose I/O just completed and move to Ready_Queue */

    Running_Process = Head(Ready_Queue); /* get next process
    start_time = read_time();   /* record its start time
    if (Running_Process != NIL) Context_Switch (Running_Process, DELTA);
    else IDLE(); /* start next process or idle until next interrupt

} else if called by timer interrupt handler {

    Running_Process -->runtime += readtime() – start_time;

    Append(Running_Process, Ready_Queue);

    Running_Process = Head(Ready_Queue); /* get next process
    start_time = read_time(); /* record its start time
    if (Running_Process != NIL) Context_Switch (Running_Process, DELTA);
    else IDLE(); /* start next process or idle until next interrupt
} else if called by exit() { /* zombie */

    Running_Process -> runtime += realtime() – starttime;

    Running_Process -> turnaroundtime = realtime() - Running_Process -> starttime;

    Print (Running_Process -> PID; Running_Process-> turnaroundtime / Running_Process -> runtime);

    Append (Running_Process, Zombie_Queue);

    Running_Process = Head(Ready_Queue); /* get next process start_time = read_time(); /* record its start time
    if (Running_Process != NIL) Context_Switch (Running_Process, DELTA);
    else IDLE(); /* start next process or idle until next interrupt

    }

/* end scheduler pseudo code */

7. Feedback

a. This exam was: _____EASY  _____SO-SO  _____HARD

b. How much did you study for this exam?
   _____ < 2 hours  _____ 2-4 hours  _____ 4-8 hours  _____ > 8 hours

c. Would you prefer to have two shorter midterms in weeks 4 and 7, rather than one midterm in Week 5 or 6. _____ YES  _____ NO  _____ DON’T CARE