Sample Midterm

1. What are the values of the integer variables `x` and `y` when the following program completes? (If either variable could have more than one value, say why.)

```
y = 2;
parbegin
        x = y * 2;
        y = 6;
parend;
```

Answer: By the Bernstein conditions, the value of `x` will be undefined, since it uses a variable that is both read and written concurrently. The value of `y` will be 6, since it is written to only at one place in the concurrent statements.

2. Is the following true or false? Justify your answer. "When several processes access shared information in primary storage, mutual exclusion must be enforced to prevent the production of indeterminant results."

Answer: This statement is false because it is too general. If many processes are reading shared data, and not writing it, mutual exclusion is unnecessary. If however any process does alter that data, then mutual exclusion becomes necessary.

3. Process A should finish before process B starts, and process B should finish before either of processes C or D start. Show how these processes may use two semaphores to provide the necessary synchronization.

Answer: Define two semaphores `Adone` and `Bdone`, both of which are initialized to 0. Have B, C, and D execute as their first instructions *down*(`Adone`), *down*(`Bdone`), and *down*(`Bdone`), respectively. Have A execute as its last instruction *up*(`Adone`); this enables B to proceed past its first instruction. Have B execute two *up*(`Bdone`)s as its last two instructions; this enables C and D to proceed past their first instructions. This arrangement provides the necessary synchronization.

4. A *bounded semaphore s* is a counting semaphore that cannot exceed a given value *smax* > 0. The corresponding *signal* and *wait* operations are:

```
signal(s):  wait until s < smax; then increment s by 1
wait(s):  wait until s > 0; then decrement s by 1
```

Write a monitor to implement bounded semaphores. (*Hint*: assume the semaphore is to be initialized to the constant **SINIT** and the maximum value is **SMAX**.)

Answer:

```
typedef monitor {
        /* scount is the integer value of the semaphore; if this is too big
           and the process tries to up it, the process will block on toobig;
           similarly, if thevalue is too small and the process tries to down
            it, the process will block on toosmall. */
        int scount = 0;
        condition toobig, toosmall;

        /* straighforward implementation of up */
        void entry up(void)
        {
                /* if too big, wait until ok */
                while (scount >= SMAX)
                        toobig.wait;
                /* increment and notify someone
```

```
            waiting for the semaphore to be
            nonzero that it is */
        scount += 1;
        toosmall.signal;
}

/* straighforward implementation of down */
void entry down(void);
begin
        /* if too small, wait until ok */
        while (scount <= 0)
                toosmall.wait;
        /* decrement and notify someone
            waiting for the semaphore to be
            less than SMAX that it is */
        scount -= 1;
        toobig.signal;
} boundedsemaphore;
```

5. Suppose a scheduling algorithm (at the level of short-term scheduling) favors those programs which have used little processor time in the recent past. Why will this algorithm favor I/O bound programs and yet not permanently starve CPU bound programs?

   Answer: Since I/O bound jobs use little processor time due to their blocking for I/O, they will be favored over CPU bound jobs, which use large amounts of CPU time. However, if I/O bound jobs repeatedly use the processor and thereby prevent CPU bound jobs from acquiring it, then the amount of processor time used by CPU bound programs in the recent past drops; eventually it will be low enough so the CPU bound process gets the CPU. Hence this algorithm will not starve CPU bound programs.

6. Assume you have been given the following jobs with the indicated arrival and service times:

name	arrival time	service time
A	0	3
B	2	5
C	4	2
D	6	1
E	8	4

   (a) When, and in what order, would these jobs run if the scheduling algorithm were first come first serve?

   (b) When, and in what order, would these jobs run if the scheduling algorithm were shortest job next?

   (c) When, and in what order, would these jobs run if the scheduling algorithm were round robin with a quantum of 2? Assume that if events are scheduled to happen at exactly the same time, that new arrivals precede terminations, which precede quantum expirations.

   Answer:

   (a) A (from 0 to 3), B (from 3 to 8), C (from 8 to 10), D (from 10 to 11), E (from 11 to 15)

   (b) A (from 0 to 3), B (from 3 to 8), D (from 8 to 9), C (from 9 to 11), E (from 11 to 15)

   (c) A (from 0 to 2), B (from 2 to 4), A (from 4 to 5), C (from 5 to 7), B (from 7 to 9), D (from 9 to 10), E (from 10 to 12), B (from 12 to 13), E (from 13 to 15)