Department of Electrical Engineering, IIT Delhi

EEL358 Operating Systems: Minor I Examination

(Closed book/Closed Notes) Time: 1 hour Maximum Marks: 25

"Thou shalt not covet thy neighbour's answers"

1. **`grabbing()` opportunities and `processing()` them**
Consider the problem of using two buffers `FB0` and `FB1` to continuously grab and process video/audio frames. This solution uses four mutexes, initialised as below:

```
grabbing_mutex_0 = 1; grabbing_mutex_1 = 0;
processing_mutex_0 = 0; processing_mutex_1 = 0;
```

When grabbing of a frame takes place in one frame buffer, processing of the previously grabbed frame takes place in the other frame buffer - this can take place concurrently. The role of the frame buffers is then reversed: the processing happens on the just grabbed frame, and the grabber thread grabs a new frame.

```
/* grabbing thread */
statement_A;

/* grab in FB0 */
statement_B;
statement_C;

/* grab in FB1 */
statement_D;
```

```
/* processing thread */
statement_E;

/* processing in FB1 */
statement_F;
statement_G;

/* processing in FB0 */
statement_H;
```

(a) Each of `statement_X` statements above is one involving a `wait()` or `signal()` command. Please state the *complete command* corresponding to each of the above 8 statements.

(b) Suppose that both the time taken to grab a frame, and to process a frame, are constants. Will the correctness of the above solution depend on the relationship between these two times? Explain.

(c) Comment about the first frame processed. *(8+2+2 marks)*

2. **Do not Bank on this...**

(a) For m resource types, and n processes, show that the time complexity of the Safety part in the deadlock avoidance is $O(mn^2)$.

(b) What is the main philosophical difference between the Banker's algorithm for deadlock avoidance and deadlock detection, in terms of the algorithm requirements? *(2+2 marks)*

3. **Producing Consumerism!** Consider the following solution to the Producer-Consumer problem that uses no semaphores, mutexes, or any special concurrency control measures. `in`, `out` and `counter` are integer variables, and the variable `counter` is shared between the producer and the consumer processes (along with the shared array `buffer`, of course). All other variables and constants are self-explanatory.

```
/* === Producer process === */
while(TRUE)
{
while(counter==BUFFER_SIZE)
      ;
buffer[in] = nextProduced;
in = (in+1) % BUFFER_SIZE;
counter++;
}
```

```
/* === Consumer process === */
while(TRUE)
{
while(counter==0)
      ;
nextConsumed = buffer[out];
out = (out+1) % BUFFER_SIZE;
counter--;
}
```

Suppose that `counter` is initially 5, and that the producer and consumer process execute concurrently. Illustrate three cases as to how the value of `counter` can be either 4, 5 or 6, respectively. *(2+2+2 marks)*

4. If a semaphore on a uniprocessor (one single core processor CPU) were not implemented in conjunction with the OS's CPU scheduler (the block and wake-up operation associated with a `wait()` and `signal()` operation), what would be the main problem associated with the following plain-vanilla implementation of semaphores? *(3 marks)*

```
wait(S) {while(S<=0); S--;}
```

```
signal(S) {S++;}
```