Section 1: MCQ (2 marks each)

MCQ 1 and 2 based on the following code fragment:

```
int main( int argc, char** argv )
{
    int N, i;
    char newArgStr[12];

    printf("Oh no!");

    N = atoi(argv[1]);  //convert cmd line arg to number
    sprintf(newArgStr, "%d", N-1); //convert N-1 to string

    for (i = 0; i < N; i++){
        fork();
        execl("./a.out", "./a.out", newArgStr, NULL);
    }

    return 0;
}
```

If the program above is compiled as "`a.out`", answer the following two MCQs:

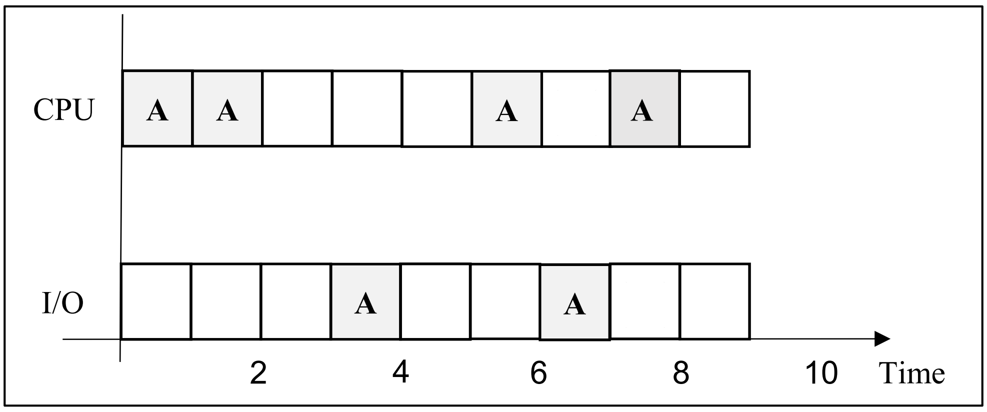
1. How many **processes** in total will be created if we execute "`a.out 3`"? Remember to count the first "`a.out`".

   a. 3
   b. 8
   c. 15
   d. 31
   e. None of the above

2. How many "`Oh no!`" messages are printed in total if we execute "`a.out 3`"?

   a. 3
   b. 8
   c. 15
   d. 31
   e. None of the above

3. Suppose process A has the following execution schedule:



What is the **most accurate process state** changes for A according to the generic 5-state process model?

a. New→Ready→Running→Blocked→Ready→Running→ Blocked→ Running→Terminated
b. New→Ready→Running→Blocked→Running→Blocked→ Ready→Running→Terminated
c. New→Ready→Running→Blocked→Ready→Running→ Blocked→Ready→Running→Terminated
d. New→ Running→Blocked→Running→Blocked→Running→Terminated
e. None of the above

[redacted]

4. Suppose we have the following multi-threaded processes on a **hybrid thread** system:

Process	**Number of User Threads**	**Number of Kernel Threads**
P1	4	1
P2	1	1
P3	5	3

If all threads are CPU intensive that runs forever, what is the approximate proportion of CPU time utilized by the processes after a long time? You can assume a preemptive fair scheduler.

a. P1 = 20%; P2 = 20%; P3 = 60%
b. P1 = 33%; P2 = 33%; P3 = 33%
c. P1 = 40%; P2 = 10%; P3 = 50%
d. P1 = 25%; P2 = 10%; P3 = 65%
e. None of the above

[redacted]

[redacted]

5. OS provides **system call** interface for the following reason(s):
   i. Faster execution time for the user program.
   ii. Hide the low level hardware details from user program.
   iii. Protect system integrity from user programs.

a. (i) only.
b. (i) and (ii) only.
c. (ii) and (iii) only.
d. (i), (ii) and (iii).
e. None of the above.

[redacted]

[redacted]

6. On an interactive processing enviroment with round-robin preemptive scheduler, shortening the interval between timer interrupt (i.e. more timer interrupts in the same time duration) can have the following effect(s):
    i. User programs can be more responsive.
    ii. User program can have a shorter turn around time.
    iii. User program can utilize the CPU more.

    a. (i) only.
    b. (i) and (ii) only.
    c. (ii) and (iii) only.
    d. (i), (ii) and (iii).
    e. None of the above.

[redacted]

[redacted]

Section 2: Bonus Question (1 mark)

7. Which of the following fantastical creatures have **not been** mentioned in the lectures so far?

    a. Vampire
    b. Zombie
    c. Man-eating Titan
    d. Undead
    e. All of the above have been mentioned in the course

[redacted]

Section 3: Short Questions (28 marks)

Question 8 (6 marks)

Consider the 5-threads global sum program:

5 threads execute the following code, where **`globalVar`** is shared

```
for ( int i = 0; i < 50000; i++ )

        globalVar++;
```

a. [2 marks] What is the **smallest possible value** for **`globalVar`** at the end of execution (i.e. after all threads terminated)?

b. [2 marks] Briefly describe the interleaving pattern which gives you the answer in (a).

c. [2 marks] If the loop counter "**`i`**" is changed to a shared global variable among the 5 threads, what is the smallest and largest possible value for **`globalVar`** at the end of execution?

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Question 9 (6 marks)

Consider the standard 3 levels MLFQ scheduling algorithm with the following parameters:

- Time quantum for all priority levels is 2 time units (TUs).
- Interval between timer interrupt is 1 TU.

a. [4 marks] Give the **CPU schedule** for the following 2 tasks using notation similar to MCQ3. The first time unit has been filled as an example in the answer sheet.

Task A
Behavior: **CPU 3TUs**, *I/O **1TU***, **CPU 3TUs**

Task B
Behavior: **CPU 1TU**, *I/O **1TU***, **CPU 1TU**, *I/O **1TU***, **CPU 1TU**

Note that we only ask for the CPU schedule, you will have to keep track of the priority level of the tasks separately on your own.

b. [2 marks] Using the tasks A and B above to illustrate one advantage and one disadvantage for the MLFQ scheduling algorithm.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

Question 10 (6 marks)

Clever compiler can transform a recursive function into an equivalent **tail-recursive** function. The key characteristic of a tail recursive function: the partially computed result is passed to the callee so that the last callee (i.e. the base case) has the final result without further calculation. Below is an example of factorial (what else ☺) and its tail-recursive version:

Factorial – Recursive Example call: `fac(5)`	**Factorial – Tail Recursive** Example call: `facTail(5, 1)`
`int fac( int N )`	`int facTail( int N, int Res )`
`{`	`{`
`  if (N == 0) return 1;`	`  if (N == 0) return Res;`
`  return N * fac(N-1);`	`  return facTail(N-1, Res * N);`
`}`	`}`

a. [2 marks] Assuming no special optimization (i.e. treating **`facTail()`** as a normal recursive function), what is the maximum number of stack frames of **`facTail()`** function during the function call **`facTail(5, 1)`**?

b. [2 marks] Using your understanding of function call and stack frame, briefly describe how we can exploit the behavior of tail-recursive function call to **reduce the usage of stack memory**.

c. [2 marks] If your optimization in (b) is used, what is the maximum number of stack frames of **`facTail()`** during the function call **`facTail(13, 1)`**?

[redacted]

[redacted]

[redacted]

[redacted]

Question 11 (10 marks)

Below is the blocking solution for producer-consumer problem:

```
while (TRUE) {
        Produce Item;

        wait( notFull );
        wait( mutex );
        buffer[in] = item;
        in = (in+1) % 5;
        count++;
        signal( mutex );
        signal( notEmpty );
}
                          Producer Process
```

```
while (TRUE) {

        wait( notEmpty );
        wait( mutex );
        item = buffer[out];
        out = (out+1) % 5;
        count--;
        signal( mutex );
        signal( notFull );

        Consume Item;
}
                          Consumer Process
```

Suppose we initialized the semaphores as follows (note that the size of the shared buffer is **5** and pay attention to the two incorrect initializations):

- **`notFull = Semaphore(3);`** //incorrect!
- **`notEmpty = Semaphore(1);`** //incorrect!
- **`mutex = Semaphore(1);`** //correct

a. [2 marks] Briefly describe the effect of the incorrect **`notFull`** semaphore initialization. You should ignore the effect of the wrong **`notEmpty`** in this question.

b. [2 marks] Briefly describe the effect of the incorrect **`notEmpty`** semaphore initialization. You should ignore the effect of the wrong **`notFull`** in this question.

c. [3 marks] Suppose we have 5 producers tasks (P1 to P5) and 3 consumers tasks (C1 to C3) using the incorrect initialization as stated. Give a snapshot of the semaphores (the value and the blocked tasks on them) at **any point during the execution**. **The only restriction is that the status of all 8 tasks must be clearly stated or implied in the snapshot.**

d. [3 marks] If you were to mark your friend's answer on part (c) above, give the rules that you use to determine correctness. Your rule should be concise and has clear outcome (i.e. if *XXXX*, then it is **wrong / correct** or similar). You can also consider using formula to express relationship.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]