Indian Statistical Institute
M.Tech.(CS) Semester-II 2024-2025
Mid-semester Examination (26 February, 2025)
Subject: Operating Systems

Total: 35 marks **Maximum marks: 30** **Duration: 2.5 hrs.**

1. (11 marks) Consider the following scheduling algorithms: (a) FCFS (b) Round-robin (c) Priority-based scheduling. For each algorithm, state (and briefly explain/justify)

   (i) what data structure would be most appropriate for storing the processes in the ready queue;

   (ii) the (asymptotic) time complexity of adding a process to the ready queue;

   (iii) the (asymptotic) time complexity of selecting a process during scheduling;

   (iv) whether starvation is possible.

   Assume that the number of ready processes is n. [3 + 2 + 4 = 9]

   In case starvation is possible, briefly describe a method by which it can be prevented. [2]

2. (5 marks) Consider a system that uses round-robin scheduling with a time quantum of t units. N processes, $P_1, P_2, \ldots, P_N$ are created in that order at roughly the same time. Each process has a single CPU burst of duration $T_1 = kt$ units (where $k \in \mathbb{N}$ and $k \gg 1$), and no I/O bursts. The kernel takes $T_0 > 0$ units of time to switch from one process to another. Draw the Gantt chart for the above scenario, and compute the CPU utilisation. [3 + 2 = 5]

3. (6 marks) Suppose that the following is an excerpt from the definition of the *proc* structure in recent Linux kernels.

```
typedef struct task_struct {
        long            state;
        pid_t           pid;
        ...
        struct task_struct   *parent, *eldest_child, *prev_sibling, *next_sibling;
        ...
} proc;
```

   Further suppose that a process P has called `fork()`, and let C denote the child process that is being created. Write C code (or pseudo-code for partial credit) to show how the kernel updates / initialises all the relevant fields shown in the definition of `task_struct` given above for P and C. [6]

   You do not need to consider any other field of the *proc* structure apart from the **six** fields listed above.

   - You may assume that
     - C has already been assigned a `pid`, and memory for C's `proc` structures has already been allocated;
     - `p` and `c` are, respectively, pointers to P and C's `proc` structures (i.e., `p` and `c` are both valid pointers of type `struct task_struct *`);
     - the *proc* structures of a process' children are stored in a **non-circular** doubly-linked list.
   - Do **not** make any assumptions about whether P has other child processes.

4. (10 marks) Consider the following code excerpt from a program. Assume that `buffer` is a global variable.

```
#define BUF_LEN 1024
...
char buffer[BUF_LEN] = "abc\n";
...
fork();
for (i = 0; i < 4; i++)
    fprintf(stderr, "%c", buffer[i]);
```

(a) If `BUF_LEN` is `#define`-d to be `10240` instead of `1024`, will there be any change in the size of the executable file generated by compiling the program? Justify your answer. [2]

(b) Assuming that all possible interleaves between any 2 processes happen in practice, compute the total number of distinct output(s) that the program can generate. [4]

(c) Explain how your answer will change if `stderr` is replaced by `stdout`. [4]

5. (3 marks) Give one example of a function from the standard C library that (a) definitely does, (b) definitely does not, (c) may or may not involve a system call. Briefly justify your answer in each case. [1 + 1 + 1 = 3]