# Q1- Shortest Job First (SJF) Scheduler

- Draw a scheduling graph for the SJF scheduler without preemption for the following jobs. Draw vertical lines for key events and be sure to label the X-axis times as in the example.

| Job | Arrival Time | Job Length |
| --- | --- | --- |
| A | T=0 | 25 |
| B | T=5 | 10 |
| C | T=10 | 15 |

# Q1 – SJF - 2

What is the response time (RT) and turnaround time (TT) for jobs A, B, and C?

**RT Job A:** ____________________ **TT Job A:** ____________________

**RT Job B:** ____________________ **TT Job B:** ____________________

**RT Job C:** ____________________ **TT Job C:** ____________________

What is the average response time for all jobs? ______________

What is the average turnaround time for all jobs? ___________

# Q2 – Shortest Time to Completion First (STCF) Scheduler

Draw a scheduling graph for the STCF scheduler with preemption for the following jobs.

Draw vertical lines for key events and be sure to label the X-axis times as in the example.

| Job | Arrival Time | Job Length |
|---|---|---|
| A | T=0 | 25 |
| B | T=5 | 10 |
| C | T=10 | 15 |

```
        |
        |
CPU     |
        |
        |______________________________________________
        0
```

# Q2 – STCF - 2

- What is the response time (RT) and turnaround time (TT) for jobs A, B, and C?

**RT Job A:** ________________ **TT Job A:** ________________

**RT Job B:** ________________ **TT Job B:** ________________

**RT Job C:** ________________ **TT Job C:** ________________

- What is the average response time for all jobs? ___________

- What is the average turnaround time for all jobs? __________

# Q3 - Operating System APIs

1. Provide a definition for what is a blocking API call

2. Provide a definition for a non-blocking API call

3. Provide an example of a blocking API call.
Consider APIs used to manage processes and/or threads.

4. Provide an example of a non-blocking API call.
Consider APIs used to manage processes and/or threads.

# Q4 – Operating System APIs - II

1. When implementing memory synchronization for a multi-threaded program list one **advantage** of combining the use of a condition variable with a lock variable via the Linux C thread API calls: `pthread_mutex_lock()` and `pthread_cond_wait()`

2. When implementing memory synchronization for a multi-threaded program using locks, list one **disadvantage** of using blocking thread API calls such as the Linux C thread API calls for: `pthread_mutex_lock()` and `pthread_cond_wait()`

3. List (2) factors that cause Linux blocking API calls to introduce **overhead** into programs:

# Q5 – Perfect Multitasking Operating System

In a perfect-multi-tasking operating system, every process of the same priority will always receive exactly $1/n^{th}$ of the available CPU time. Important CPU improvements for multi-tasking include: (1) fast context switching to enable jobs to be swapped in-and-out of the CPU very quickly, and (2) the use of a timer interrupt to preempt running jobs without the user voluntarily yielding the CPU. These innovations have enabled major improvements towards achieving a coveted "Perfect Multi-Tasking System".

List and describe two challenges that remain complicating the full realization of a Perfect Multi-Tasking Operating System. In other words, what makes it very difficult for all jobs (for example, 10 jobs) of the same priority to receive **EXACTLY** the same runtime on the CPU? Your description must explain why the challenge is a problem for achieving perfect multi-tasking.

# Q6 – Round-robin Scheduler

Show a scheduling graph for a Round-Robin (RR) scheduler with job preemption where newly arriving jobs will immediately run. Assume a time slice of 3 timer units.  Draw vertical lines for key events and be sure to label the X-axis times as in the example.

| Job | Arrival Time | Job Length |
| --- | --- | --- |
| A | T=0 | 25 |
| B | T=5 | 10 |
| C | T=10 | 15 |

```
            |
            |
RR          |
            |
            |_____________________________________________
            0
```

# Q6 – RR scheduler - 2

Using the graph, from time t=10 until all jobs complete at t=50, evaluate Jain's Fairness Index:

Jain's fairness index is exp

$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

Where n is the number of jobs, and $x_i$ is the time share of each process Jain's fairness index=1 for best case fairness, and 1/n for worst case fairness.

For the time window from t=10 to t=50, what percentage of the CPU time is allocated to each of the jobs A, B, and C?

Job A: _________ Job B: _________ Job C: _________

With these values, calculate Jain's fairness index from t=10 to t=50.

# Q7 – Approximate (sloppy) counter

Below is a tradeoff space graph similar to those we've shown in class. Based on the sloppy counter threshold (S), add numbers on the **left** or **right** side of the graph for each of the following tradeoffs:

1. High number of Global Updates
2. High Performance
3. High Overhead
4. High Accuracy
5. Low number of Global Updates
6. Low Performance
7. Low Overhead
8. Low Accuracy

**Low sloppy threshold (S)** **High sloppy threshold (S)**