FIFO EXAMPLE

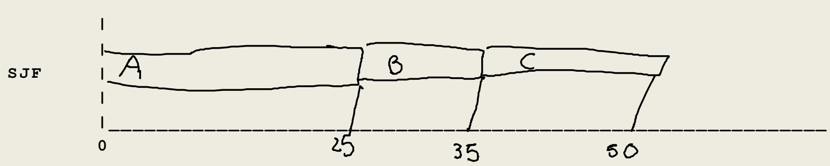
- Operation of CPU schedulers can be visualized with timing graphs.
- The graph below depicts a FIFO scheduler where three jobs arrive in the sequence A, B, C, where job A runs for 10 time slices, job B for 5 time slices, and job C for 10 time slices.

```
       |
FIFO   |AAAAAAAAAABBBBBCCCCCCCCCC
       |________________________________
       0         10   15       25
```


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: 0

TT Job A: 25

RT Job B: 25 - 5 = 20

TT Job B: 35 - 5 = 30

RT Job C: 35 - 10 = 25

TT Job C: 50 - 10 = 40

What is the average response time for all jobs? $\frac{0+20+25}{3} = \frac{45}{3} = 15$

What is the average turnaround time for all jobs? $\frac{25+30+40}{3} = \frac{95}{3}$ = 31.66

57

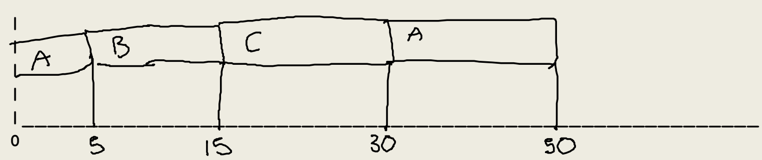
Q2 – SHORTEST TIME TO COMPLETION FIRST (STCF) SCHEDULER

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~~ 20
B	T=5	~~10~~ ~~5~~ 0
C	T=10	15

CPU



58

Q2 – STCF - 2

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

RT Job A: 0 | TT Job A: 50

RT Job B: 0 | TT Job B: 15 – 5 = 10

RT Job C: 15 – 10 = 5 | TT Job C: 30 – 10 = 20

- What is the average response time for all jobs? $\frac{0+0+5}{3} = \frac{5}{3} = 1.\bar{66}$

- What is the average turnaround time for all jobs? $\frac{50+10+20}{3} = \frac{80}{3}$ → $26.\bar{66}$

Q3 - OPERATING SYSTEM APIs

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

AN API Call that suspends the calling thread to wait for a system interrupt to fire when an event occurs. Calling thread goes to sleep. (RUNNING → BLOCKED)

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

AN API CALL that DOES NOT SUSPEND the calling thread, but returns quickly AND DOES NOT WAIT FOR AN INTERRUPT TO OCCUR (OR EVENT)

3. Provide an example of a blocking API call.

Consider APIs used to manage processes and/or threads. Others?

pthread_mutex_lock() wait() waitpid()

4. Provide an example of a non-blocking API call. Others?

Consider APIs used to manage processes and/or threads.

pthread_mutex_trylock() fork()

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()`

The combination ensures the order that blocked threads waiting for the lock will be woken up and given access to the lock. Threads wait in FIFO order.

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()`

- w/ pthread_mutex_lock the lock may never become available resulting in deadlock
- Detail: must introduce and check state variable - more API calls

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

- With fine-grained locking, many calls to lock APIs to synchronize critical sections will increase overhead
- blocking APIs must trap + context switch and be run in kernel mode introducing more overhead

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.61

61

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.

November 2, 2021	TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma	L10.62

62

2 challenges for perfect multitasking

- Jobs arrive at different times and run for different lengths making it more difficult to perfectly balance runtime for jobs in the same priority queue
- Job accounting (tracking time) involves overhead – measurements may not be precise (vruntime)
- context switching: # of context switches + overhead of a c/s may lead to inconsistencies in job runtime

63

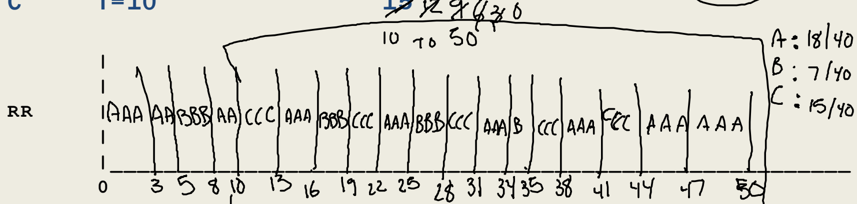
Q6 – ROUND-ROBIN SCHEDULER

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.

ARRIVING JOBS ARE ADDED TO THE BACK OF THE RUNQUEUE AND THE JOB PTR WILL JUMP TO THE MOST RECENTLY ADDED JOB – AND CONTINUES IN RR FASHION

Job	Arrival Time	Job Length
A	T=0	25 20 18 15 12 9 6
B	T=5	10 7 4 1 0
C	T=10	15 12 9 6 3 0

RUNQUEUE: ABC



64

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 expressed as:

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

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: 18/40 = .45 Job B: 7/40 = .175 Job C: 15/40 = .375

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

Q6 - II

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

worst case $\frac{1}{3} = .333$

perfect 1

$(.45 + .175 + .375) = (1)^2 = 1$

$n \cdot \sum_{i=1}^{n} x_i^2 \rightarrow 3 \cdot ((.45)^2 + (.175)^2 + (.375)^2)$

$3 \cdot (.2025 + .030625 + .140625)$

$3 \cdot (.37375)$

1.12125

$\rightarrow \frac{1}{1.12125}$

$\rightarrow .8918617$

$\sim 89.2\%$

Q7 – 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) — 1364

High sloppy threshold (S) — 5728

|____|