CPU Scheduling

CSD2180 Operating Systems
BSc in Computer Science (IMGD / RTIS)
Singapore Institute of Technology / DigiPen Institute of Technology
September 2021

SINGAPORE INSTITUTE OF TECHNOLOGY

Attendance Taking

https://forms.office.com/r/jAE1Pw5sey

- Log in to your SIT account to submit the form
- You can only submit the form once
- The codeword is **balloon**

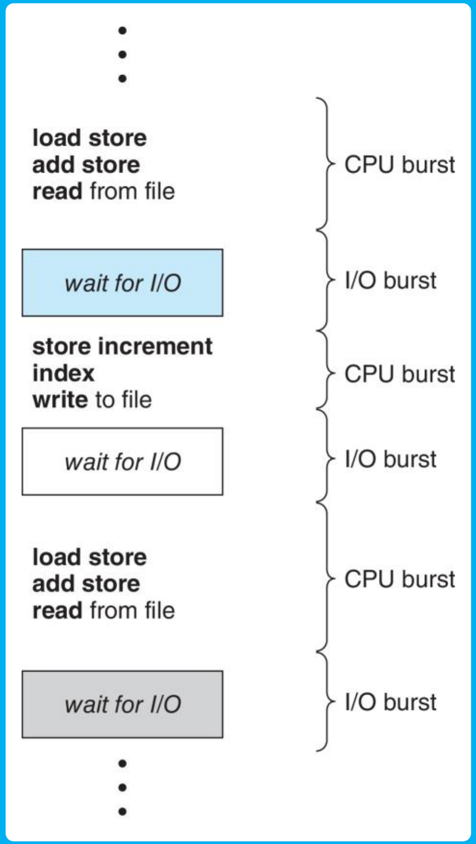
Basic Concepts

Basic Concepts

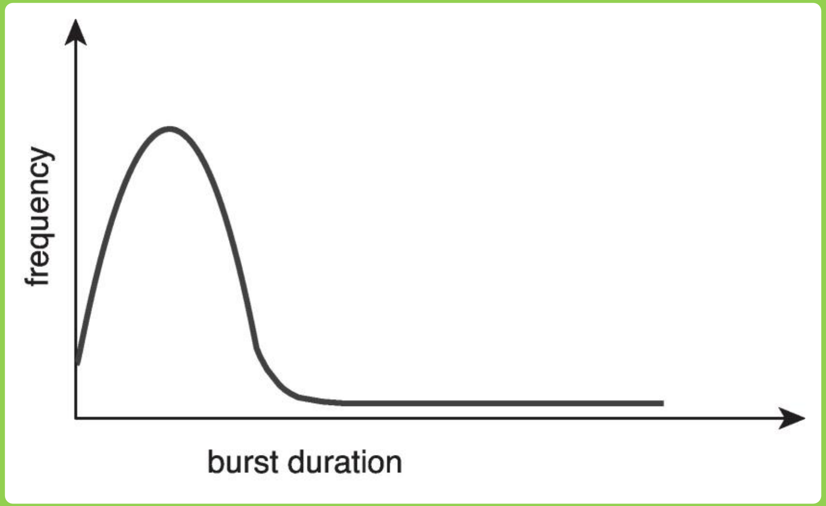
Maximum CPU utilization obtained with multiprogramming

CPU-I/O Burst Cycle: Process execution consists of a cycle of CPU execution and I/O wait

- CPU burst followed by I/O burst
- CPU burst distribution is of main concern



Characteristics of CPU Bursts



Large number of short bursts

Small number of longer bursts

CPU Scheduler

The CPU scheduler selects from processes in ready queue and allocates a CPU core to one of them

CPU scheduling decisions may take place when a process

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready
4. Terminates

For situations 1 and 4, there is **no choice** in terms of scheduling, i.e., a new process must be selected for execution

For situations 2 and 3, there is a choice

Preemptive and Non-Preemptive Scheduling

Preemptive scheduling: The CPU is allocated to a process for a limited amount of time, and the process is taken away and placed back in the ready queue after the amount of time lapses

Non-preemptive scheduling: Once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state

Virtually all modern operating systems including Windows, macOS, Linux, and Unix use preemptive scheduling algorithms

If scheduling takes place only under 1 and 4, the scheduling scheme is non-preemptive (why?)

Otherwise, it is preemptive (why?)

What are the possible issues with preemptive scheduling?

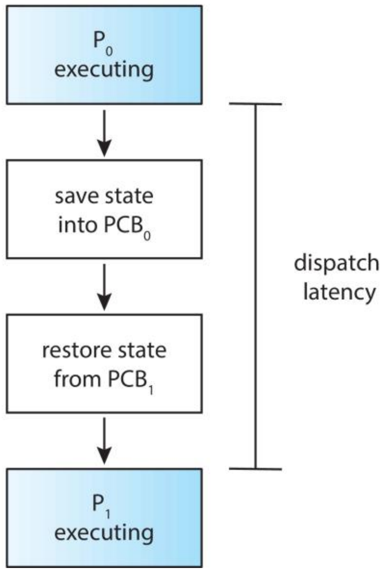
Dispatcher

Dispatcher module gives control of the CPU to the process selected by the scheduler

This involves

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to execute that program

Dispatch latency is the time it takes for the dispatcher to stop one process and start another running



How often do context switches occur?

vmstat command

```
$ sudo vmstat -w 1 14
```

```
--procs-- -----------------------memory---------------------- ---swap-- -----io---- -system-- --------cpu--------
   r    b         swpd         free         buff        cache   si   so    bi    bo   in   cs  us  sy  id  wa  st
   0    0            0     63111372        71972      1237952    0    0     1     4    2   10   0   0 100   0   0
   0    0            0     63111592        71972      1237952    0    0     0     0   62  297   0   0 100   0   0
   0    0            0     63111592        71972      1237952    0    0     0     0   56  266   0   0 100   0   0
   0    0            0     63111592        71972      1237952    0    0     0     0   53  262   0   0 100   0   0
   0    0            0     63111592        71972      1237952    0    0     0     0   60  280   0   0 100   0   0
   0    0            0     63111592        71972      1237952    0    0     0     0   71  352   0   0 100   0   0
   0    0            0     63111592        71972      1237952    0    0     0     0   77  328   0   0 100   0   0
   0    0            0     63111600        71972      1237952    0    0     0     0   68  288   0   0 100   0   0
   0    0            0     63111600        71972      1237952    0    0     0     0   55  262   0   0 100   0   0
   0    0            0     63111600        71972      1237952    0    0     0     0   67  289   0   0 100   0   0
   0    0            0     63111600        71972      1237952    0    0     0     0  162  545   0   0 100   0   0
   0    0            0     63111600        71972      1237952    0    0     0     0   66  305   0   0 100   0   0
   0    0            0     63111600        71972      1237952    0    0     0     0   58  286   0   0 100   0   0
   0    0            0     63111600        71972      1237952    0    0     0     0   53  263   0   0 100   0   0
```

/proc filesystem

```
$ sudo cat /proc/151/status
```

```
Name:   bash
Umask:  0022
State:  S (sleeping)
Tgid:   151
Ngid:   0
Pid:    151
PPid:   150
...
Seccomp_filters:        0
Speculation_Store_Bypass:       thread vulnerable
Cpus_allowed:   ffffffff
Cpus_allowed_list:      0-31
Mems_allowed:   1
Mems_allowed_list:      0
voluntary_ctxt_switches:        314
nonvoluntary_ctxt_switches:     0
```

Scheduling Criteria

Scheduling Criteria

CPU utilization: Keep the CPU as busy as possible

Throughput: Number of processes that complete their execution per time unit

Turnaround time: Amount of time to execute a particular process

Waiting time: Amount of time a process has been waiting in the ready queue

Response time: Amount of time it takes from when a request was submitted until the first response is produced

Scheduling Algorithms

First-Come, First-Serve (FCFS) Scheduling

The process that requests the CPU first is allocated the CPU first

Pros

- Simple to write and understand
- Easy to manage using a **FIFO queue**

Cons

- Average waiting time is often long
- Algorithm is non-preemptive, and may not suit interactive systems
- **Convoy effect**, where short processes wait for long process

First-Come, First-Serve (FCFS) Scheduling Example

Process	Burst time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order P1, P2, P3

Waiting time: $P1 = 0; P2 = 24; P3 = 27$

Average waiting time: $\frac{(0+24+27)}{3} = 17$

First-Come, First-Serve (FCFS) Scheduling Example

Process	Burst time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order P2, P3, P1

Waiting time: $P1 = 6; P2 = 0; P3 = 3$

Average waiting time: $\frac{(6+0+3)}{3} = 3$

Much better than previous case ☺

Shortest-Job-First (SJF) Scheduling

Use the length of each process' next CPU burst to schedule the process with the shortest

Pros

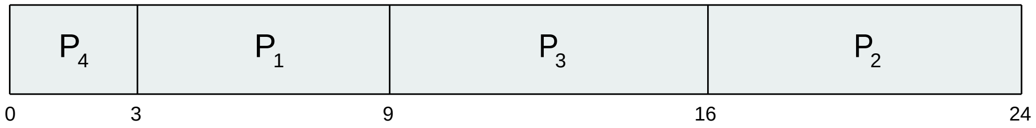
- Optimal, gives minimum average waiting time for a given set of processes

Cons

- Difficult to know the length of the next CPU burst

Shortest-Job-First (SJF) Scheduling Example

Process	Burst time
P1	6
P2	8
P3	7
P4	3



Waiting time: $P1 = 3; P2 = 16; P3 = 9; \mathrm{P4} = 0$

Average waiting time: $\frac{(3+16+9+0)}{4} = 7$

How do we know the length of the next CPU burst?

Ask the user? Estimate? 🤔

Determining Length of Next CPU Burst

Expect that the next CPU burst will be similar in length to the previous ones

Predict next CPU burst as an exponential average of the measured lengths of previous CPU bursts

How? Exponential averaging!

Let

t_n be the length of the n-th CPU burst

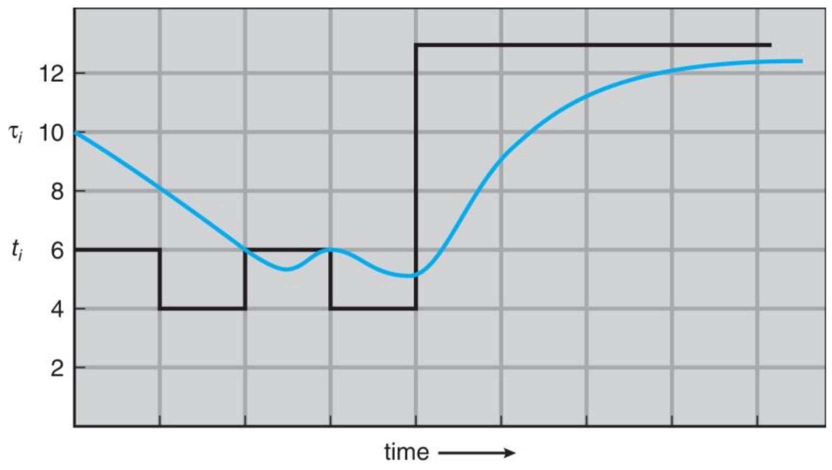
τ_{n+1} be the predicted next CPU burst length

For

α, $0 \leq \alpha \leq 1$, define $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$

Predicting Length of Next CPU Burst

$$\alpha = \frac{1}{2};\ \tau_0 = 10$$



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Examples of Exponential Averaging

For $\alpha = 0$, $\tau_{n+1} = \tau_n$

- Recent history does not count

For $\alpha = 1$, $\tau_{n+1} = t_n$

- Only the actual last CPU burst counts

Expanding the formula, we get

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \cdots + (1-\alpha)^j \alpha t_{n-j} + \cdots + (1-\alpha)^{n+1}\tau_0$$

Since both α and (1 - α) are ≤1, each successive term has less weight than its predecessor

Shortest-Remaining-Time-First (SRTF) Scheduling

Essentially a preemptive shortest-job-first (SJF) scheduling algorithm

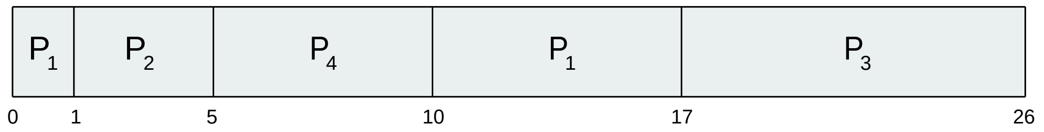
Difference between SJF and SRTF arises when a new process arrives at the ready queue while a previous process is still executing

The next CPU burst of the newly arrived process may be shorter than what is left of the currently executing process

SRTF will **preempt the currently executing process**, whereas SJF will allow the currently running process to finish its CPU burst

Shortest-Remaining-Time-First (SRTF) Scheduling Example

Process	Arrival Time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5



Waiting time:

$$P1 = 0 + (10 - 1) = 9$$
$$P2 = 1 - 1 = 0$$
$$P3 = 17 - 2 = 15$$
$$P4 = 5 - 3 = 2$$

Average waiting time: $\frac{(9+0+15+2)}{4} = 6.5$

Round Robin (RR) Scheduling

Each process gets a small unit of CPU time (called **time quantum**)

After the time quantum has elapsed, the process is preempted and added to the end of the ready queue

If there are n processes in the ready queue and the time quantum is q, then each process gets $\frac{1}{n}$ of the CPU time in chunks of at most q time units at once

No process waits more than $(n-1)q$ time units

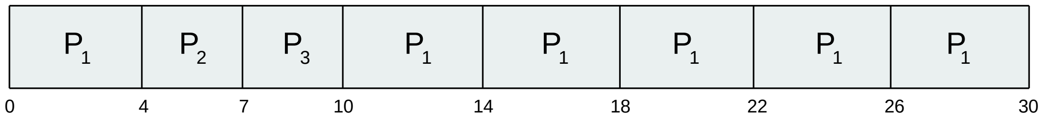
Timer interrupts at every quantum to schedule next process

Performance

- If q is large, performance tends towards FCFS
- If q is small, then q must be large with respect to context switch, otherwise overhead from context switching will be too high

Round Robin (RR) Scheduling Example with Time Quantum = 4

Process	Burst time
P1	24
P2	3
P3	3

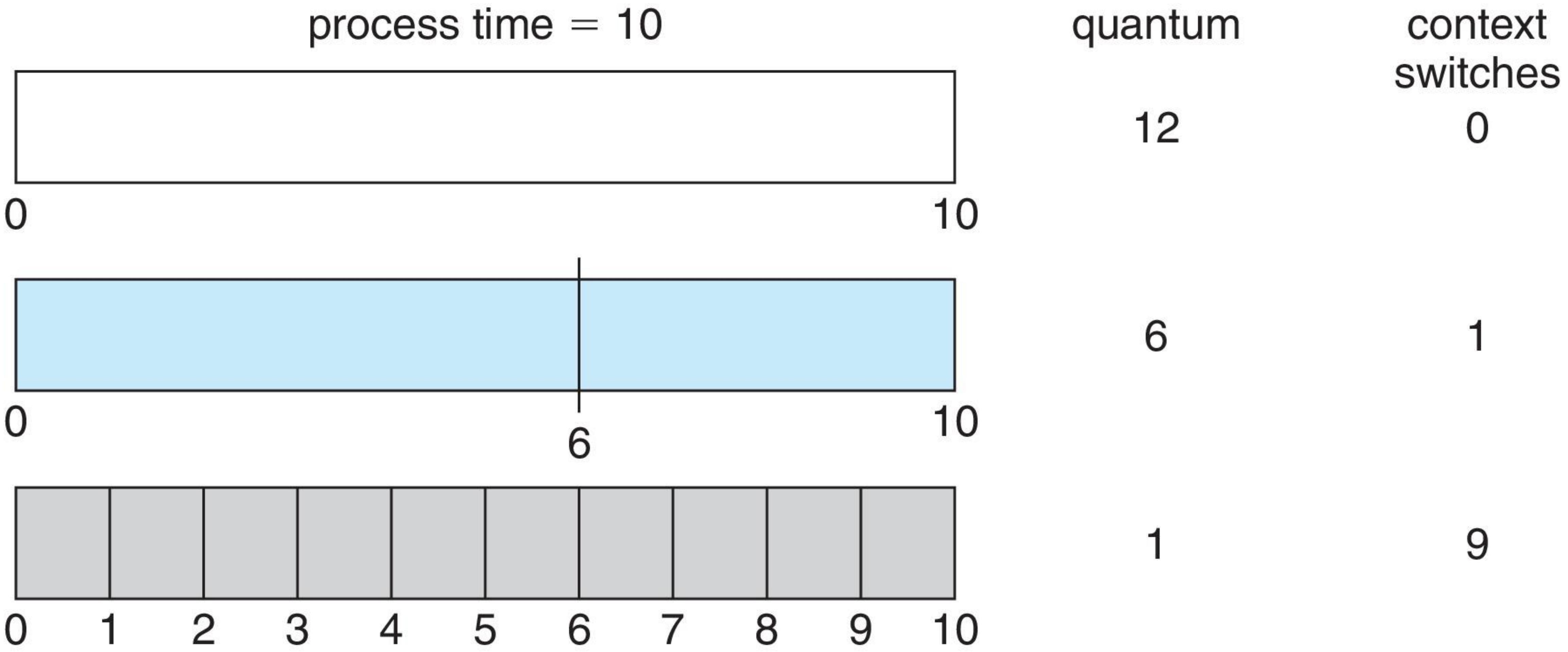


Typically, higher average turnaround time than SJF, but better response time

Time quantum q should be **large** compared to context switch time

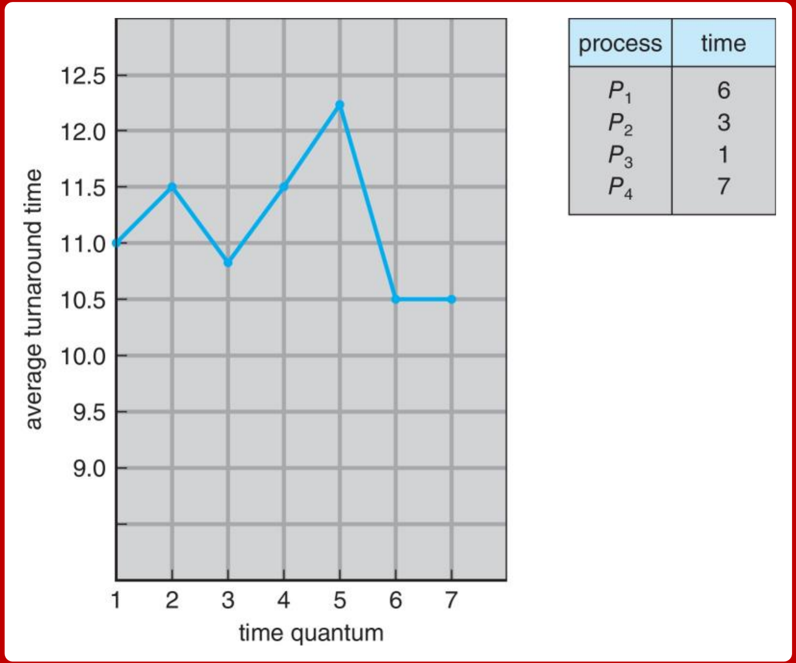
- q usually ~10ms to ~100ms
- Context switch <10µs

Time Quantum and Context Switch Time



Turnaround Time vs Time Quantum

Process	Burst time
P1	6
P2	3
P3	1
P4	7



Priority Scheduling

A priority number (integer) is associated with each process

Starvation: Low priority processes may never execute

- **Solution:** Aging, i.e., as time progresses increase the priority of the process

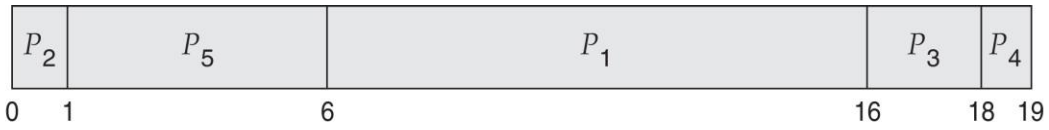
The CPU is allocated to process with highest priority

- **Preemptive:** Preempt CPU if newly arrived process has higher priority than the current running process

- **Nonpreemptive:** Put the new process at the head of the ready queue if newly arrived process has higher priority than the current running process

Priority Scheduling Example

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2



Average waiting time: $\frac{(6+0+16+18+1)}{5} = 8.2$

SINGAPORE INSTITUTE OF TECHNOLOGY

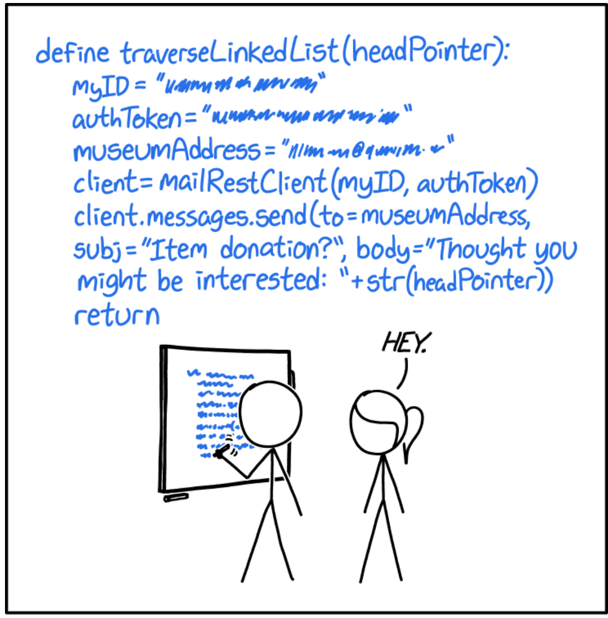
Questions?

Thank You!

Weihan.Goh {at} Singaporetech.edu.sg

https://www.singaporetech.edu.sg/directory/faculty/weihan-goh

https://sg.linkedin.com/in/weihan-goh



CODING INTERVIEW TIP: INTERVIEWERS GET REALLY MAD WHEN YOU TRY TO DONATE THEIR LINKED LISTS TO A TECHNOLOGY MUSEUM.