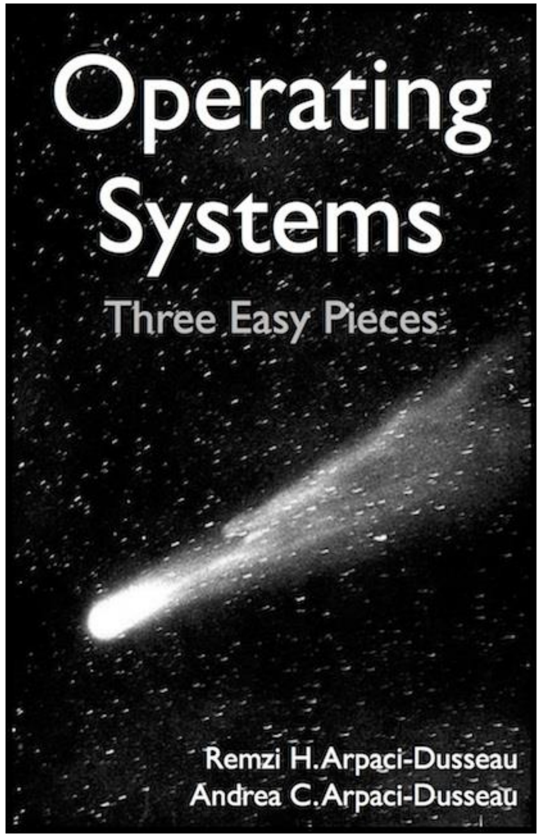
CMSC 125: Operating Systems

- Instructor: **Joseph Anthony C. Hermocilla**
- Email: jchermocilla@up.edu.ph
- Web: https://jachermocilla.org

Resources

Book: https://pages.cs.wisc.edu/~remzi/OSTEP/

Slides Template:
https://pages.cs.wisc.edu/~remzi/OSTEP/Educators-Slides/Youjip/



Acknowledgement

- This lecture slide set was initially developed for Operating System course in Computer Science Dept. at Hanyang University. This lecture slide set is for OSTEP book written by Remzi and Andrea at University of Wisconsin.

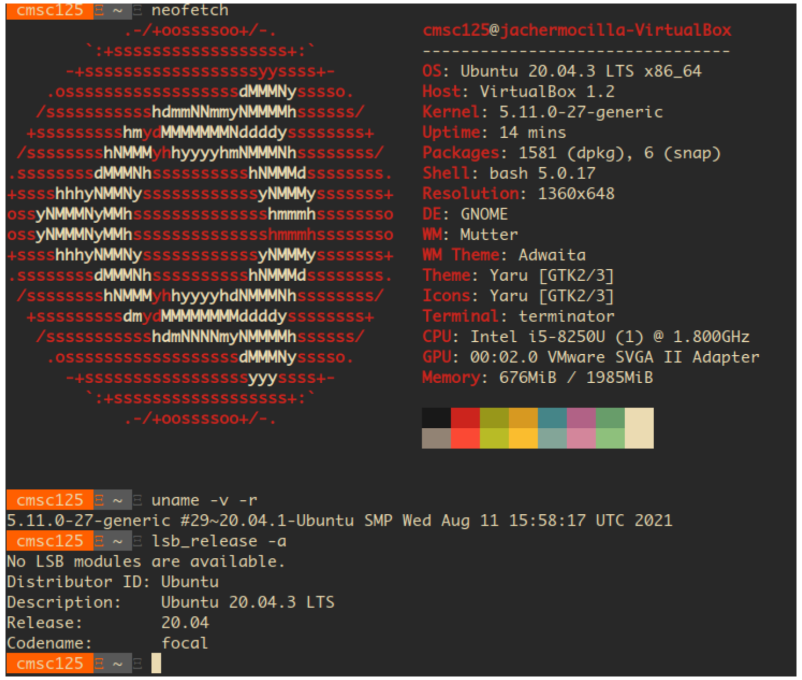
7. Scheduling: Introduction

Operating System: Three Easy Pieces

What constitutes an OS? Kernel + other components

- A **privileged/kernel process** running in kernel mode(as controlled by the hardware)
- Loaded by bootloader(GRUB or custom) at boot time
- Will run until the device is turned off
- Enables a **user process** to run on the CPU in a **limited direct execution** fashion via a **context switch**
- Performs privileged operations on behalf of user processes through **system calls** invoked via a **trap** instruction

https://www.techrepublic.com



```
cmsc125 ~ neofetch
            .-/+oossssoo+/-.               cmsc125@jachermocilla-VirtualBox
        `:+ssssssssssssssssss+:`           --------------------------------
      -+ssssssssssssssssssyyssss+-         OS: Ubuntu 20.04.3 LTS x86_64
    .ossssssssssssssssssdMMMNysssso.       Host: VirtualBox 1.2
   /ssssssssssshdmmNNmmyNMMMMhssssss/      Kernel: 5.11.0-27-generic
  +ssssssssshmydMMMMMMMNddddyssssssss+     Uptime: 14 mins
 /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/    Packages: 1581 (dpkg), 6 (snap)
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Shell: bash 5.0.17
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   Resolution: 1360x648
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   DE: GNOME
ossyNMMMNyMMhsssssssssssssshmmmhssssssso   WM: Mutter
+sssshhhyNMMNyssssssssssssyNMMMysssssss+   WM Theme: Adwaita
.ssssssssdMMMNhsssssssssshNMMMdssssssss.   Theme: Yaru [GTK2/3]
 /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/    Icons: Yaru [GTK2/3]
  +sssssssssdmydMMMMMMMMddddyssssssss+     Terminal: terminator
   /ssssssssssshdmNNNNmyNMMMMhssssss/      CPU: Intel i5-8250U (1) @ 1.800GHz
    .ossssssssssssssssssdMMMNysssso.       GPU: 00:02.0 VMware SVGA II Adapter
      -+sssssssssssssssssyyyssss+-         Memory: 676MiB / 1985MiB
        `:+ssssssssssssssssss+:`
            .-/+oossssoo+/-.


cmsc125 ~ uname -v -r
5.11.0-27-generic #29~20.04.1-Ubuntu SMP Wed Aug 11 15:58:17 UTC 2021
cmsc125 ~ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.3 LTS
Release:        20.04
Codename:       focal
cmsc125 ~
```

Developing a Scheduling Policy

- How do we select which user process to run on the CPU?
- What are key assumptions? – about the workload, etc.
- What metrics are important?
- What are some historical approaches?

https://www.jollibee.com.ph/

Workload assumptions for now (simplified, unrealistic)

1. Each process runs for the **same amount of time**
2. All processes **arrive** at the same time
3. Once started, each process **runs to completion**
4. All processes only use the **CPU** (i.e., they perform no I/O)
5. The **run-time** of each process is known

We will relax these assumptions as we go along.

Scheduling Metrics (for Comparing Policies)

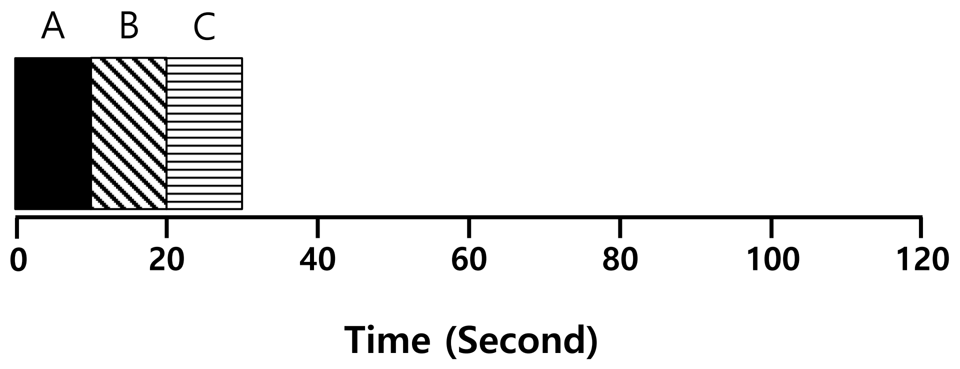
- Turnaround Time – a performance metric
  - The time at which **the process completes** minus the time at which **the process arrived** in the system
    - For now, $T_{arrival} = 0$, since we assumed that all processes arrived at the same time

$$\boxed{T_{turnaround} = T_{completion} - T_{arrival}}$$

  - We are usually interested in the **Average Turnaround Time(ATT)** for a given set of processes
- Fairness
  - Each process has "fair chance" to run on the CPU
  - Performance and fairness are often at odds in scheduling

Policy #1: First In, First Out (FIFO)

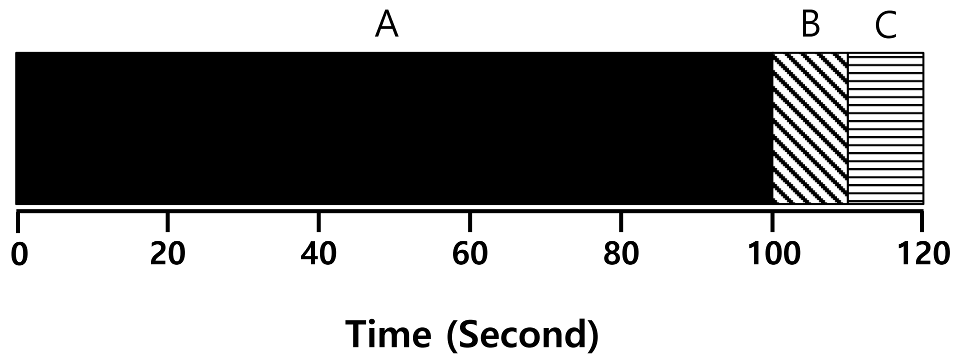
- **First Come, First Served (FCFS)**
  - Very simple and easy to implement
- Example:
  - A arrived just before B which arrived just before C.
  - Each job runs for 10 seconds.



$$\textit{Average turnaround time} = \frac{10+20+30}{3} = 20\ \textit{sec}$$

Policy #1: First In, First Out (FIFO) (Cont.)

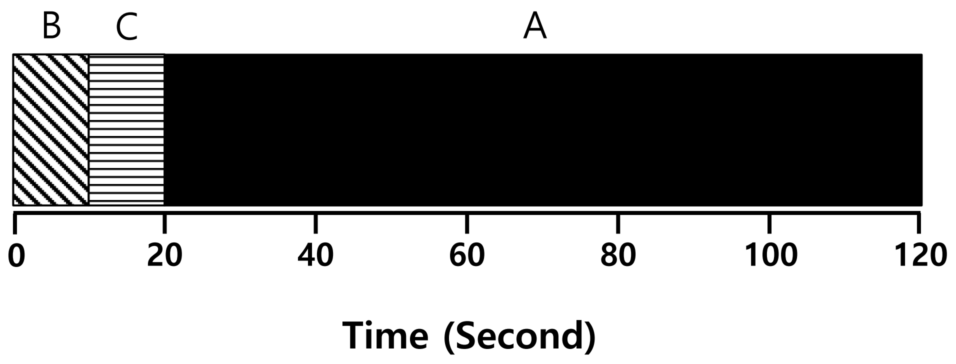
- Why FIFO is not that great?
- Let's relax assumption 1: Each process now **no longer** runs for the same amount of time.
- It suffers from the **Convoy Effect** – short-burst processes wait for long-burst processes, drastically affecting the ATT
- Example:
  - A arrived just before B which arrived just before C.
  - A runs for 100 seconds, B and C run for 10 each.



$$\textit{Average turnaround time} = \frac{100 + 110 + 120}{3} = 110\ sec$$

Policy #2: Shortest Job First (SJF)

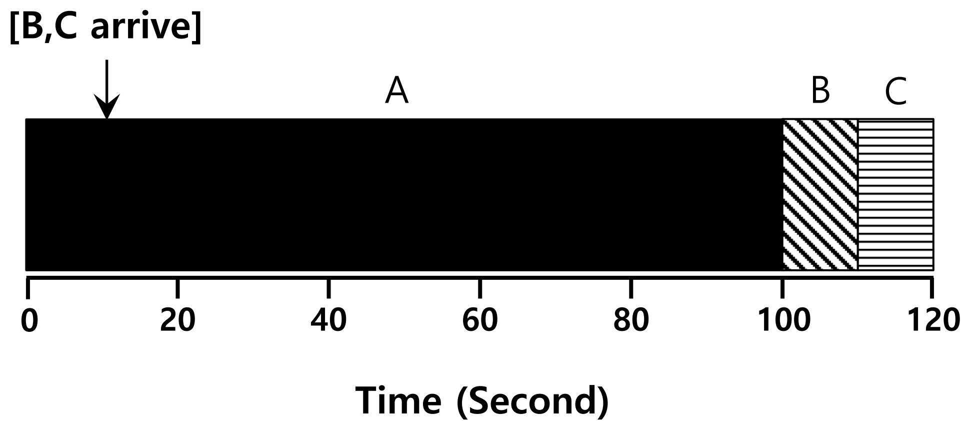
- Run the shortest process first, then the next shortest, and so on
  - A **non-preemptive scheduling** approach – runs processes to completion without interruption (opposite is called **preemptive** which is used by modern systems)
- Example 1:
  - A arrived just before B which arrived just before C.
  - A runs for 100 seconds, B and C run for 10 each.



$$\textit{Average turnaround time} = \frac{10+20+120}{3} = 50\ sec$$

Policy #2: Shortest Job First (SJF) (Cont.)

- Let's relax assumption 2: Processes can now arrive **at any time.**
- Results to a Convoy Effect due to non-preemption
- Example (SJF with Late Arrivals):
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds



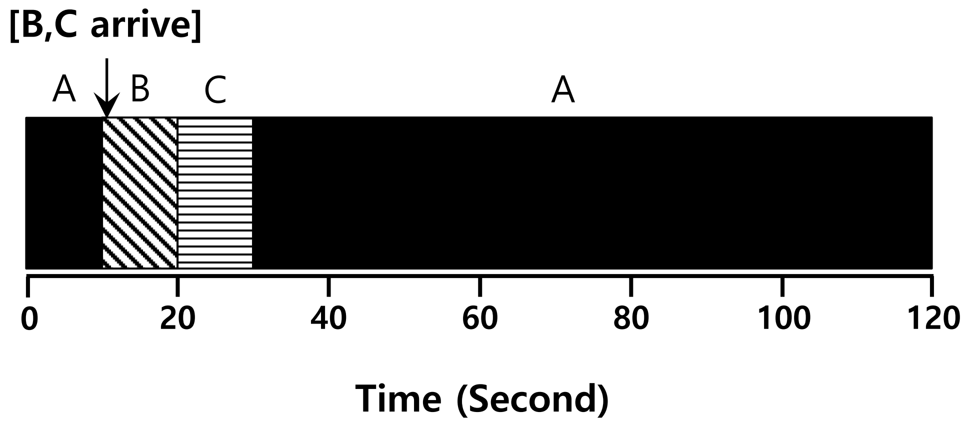
$$Average\ turnaround\ time = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33\ sec$$

Policy #3: Shortest Time-to-Completion First (STCF)

- Let's relax assumption 3: Processes now **need not run to completion**
  - Adds preemption to SJF
  - Also knows as **Preemptive Shortest Job First (PSJF)**
- A new process *arrives* to the system:
  - Determine of the *remaining times* of existing processes and the new process
  - Schedule the process which has the least time left
- Note that the decision point happens upon a <u>new process arrival</u> or <u>the timer interrupts</u>

Policy #3: Shortest Time-to-Completion First (STCF) (Cont.)

- Example:
  - A arrives at t=0 and needs to run for 100 seconds.
  - B and C arrive at t=10 and each need to run for 10 seconds



$$\textit{Average turnaround time} = \frac{(120-0)+(20-10)+(30-10)}{3} = 50\ \textit{sec}$$

A New Scheduling Metric

- Response Time – introduced in **timesharing** and **interactive** systems
  - Policies described previously mostly apply to **batch processing** systems
- The time from **when the process arrives** to the **first time it is scheduled** (or produce the first output).

$$T_{response} = T_{firstrun} - T_{arrival}$$

  - STCF and related policies are not particularly good for response time.
  - We are again interested in the **Average Response Time(ART)** given a set of processes

How can we build a scheduler that is sensitive to response time?

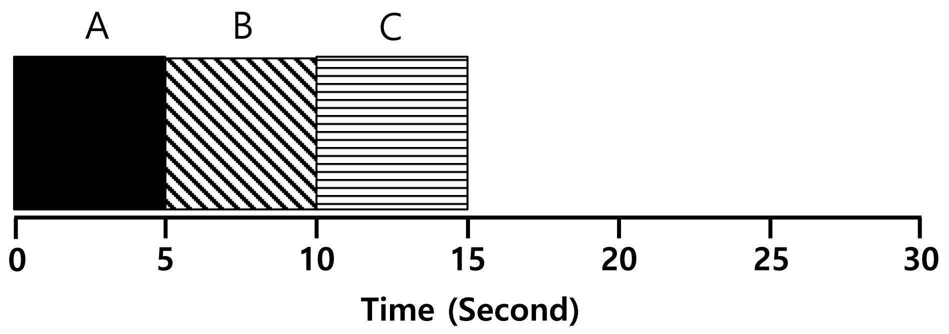
Policy #4: Round Robin (RR)

- Aka **Time-Slicing** Scheduling
  - Run a process for a time slice (or quantum denoted by **q**) and then switch to the next process in the **run queue**
  - It repeatedly does so until the processed exited
  - The time slice must be *a multiple of* the timer-interrupt period.
    - Example: If timer interrupts every 10ms, q must be 10ms, 20ms, etc.

RR is fair, but performs poorly on metrics such as turnaround time

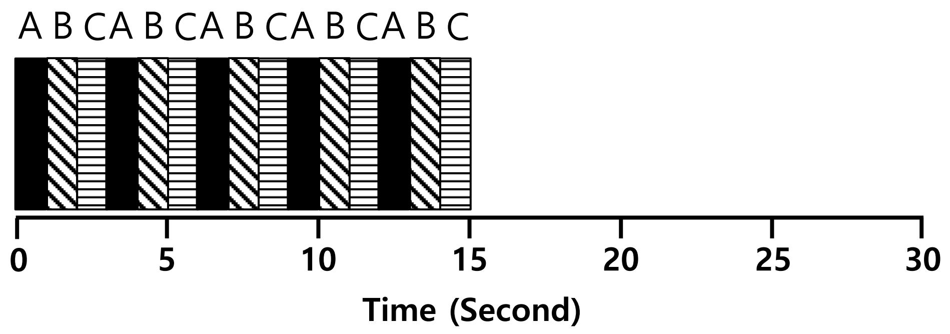
Policy #4: Round Robin (RR) (Cont.)

- Example:
  - A, B and C arrive at the same time
  - They each wish to run for 5 seconds



SJF (Bad for Response Time)

$$T_{average\ response} = \frac{0+5+10}{3} = 5sec$$



RR with a time slice of 1sec (Good for Response Time)

$$T_{average\ response} = \frac{0+1+2}{3} = 1sec$$

Policy #4: Round Robin (RR) (Cont.)

- The length of the time slice is critical
- **Shorter time slice**
  - Better response time
  - Cost of context switching will dominate overall performance
- **Longer time slice**
  - Worsens response time
  - **Amortizes** the cost of context switching – less context switching
- Note: Cost of context switching is not just due to the copying of registers but also to other architectural aspects like caches, TLBs, etc.

Deciding on the length of the time slice presents a trade-off to a system designer.
How about the ATT in RR?

Incorporating I/O

- Let’s relax assumption 4: All processes can now **perform I/O**

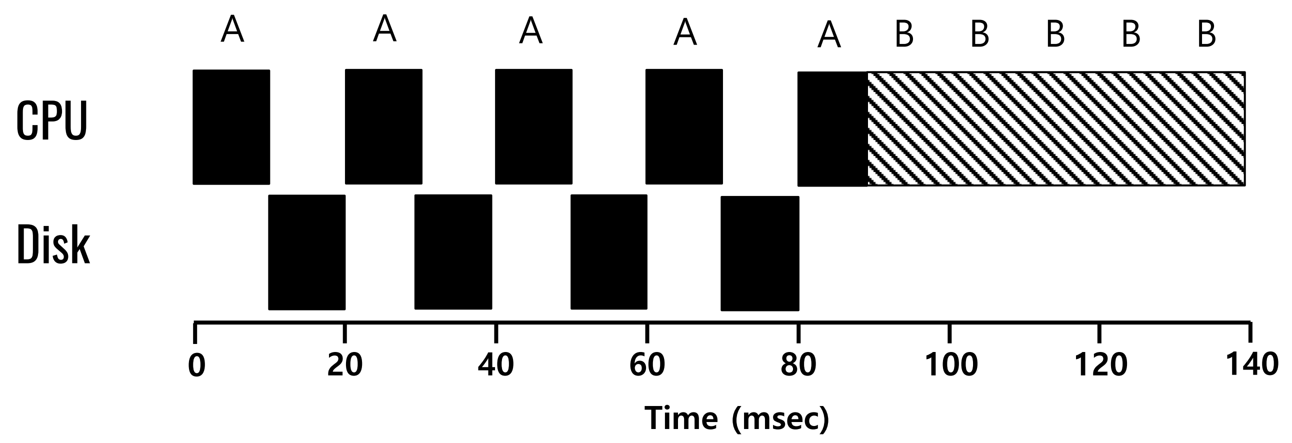
- When a process initiates an I/O request
  - The process state is set to **blocked**/**waiting** (process waiting for I/O completion)
  - The scheduler should schedule another process on the CPU, otherwise the CPU will be idle

- When the I/O completes
  - An interrupt is generated and control is transferred to the kernel
  - The kernel changes the state of the process that requested the I/O back to **ready** state
  - The scheduler *may* n schedule this process on the CPU now

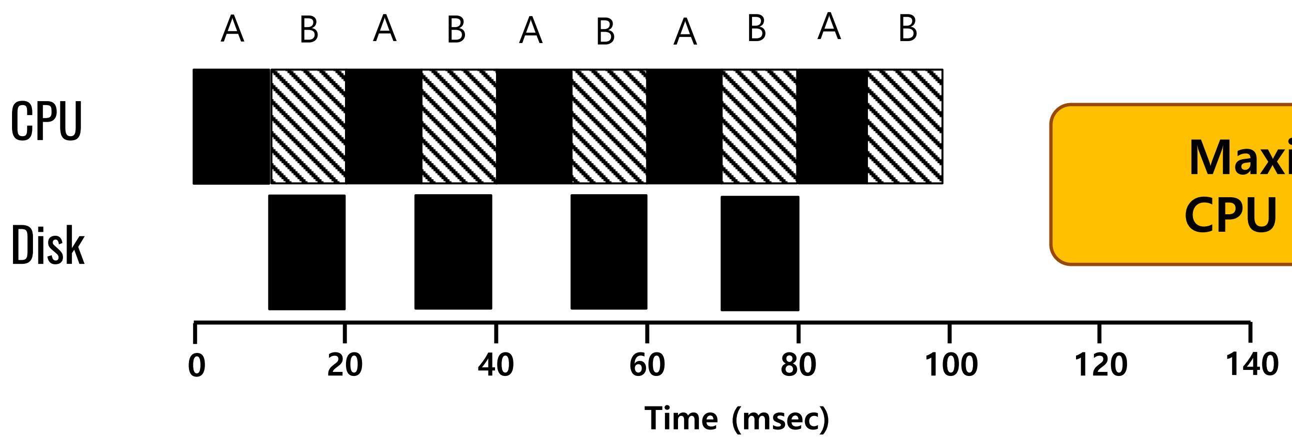
Incorporating I/O (Cont.)

- Example:
  - A and B need 50ms of CPU time each
  - A runs for 10ms and then issues an I/O request
    - I/Os each take 10ms
  - B simply uses the CPU for 50ms and performs no I/O
  - The scheduler runs A first, then B after

Incorporating I/O (Cont.)



Poor Use of Resources



Overlap Allows Better Use of Resources

Maximizes the CPU utilization

No More Oracle

- Let’s relax assumption 5: The scheduler now **does not know the run-time** of the processes
- More realistic but how can this be achieved? – **MLFQ**