DSC-204A: Scalable Data Systems, Winter 2024 **Basics: OS-2**

Lecture Title: Scalable Data Systems, Winter 2024

Lecturer: Hao Zhang *Scribe: Ruinan Ma, Somansh Budhwar*

1 Operating System Goals

OS: basically, a software between apps and hardware

- Goal 1: Provide convenience to users
- Goal 2: Efficiency – Manage compute, memory, storage resources
  - Goal 2.1: Running N processes Not N times slower
    - Given limited resources As fast as possible (Process Management)
  - Goal 2.2: Running N apps (Memory Management)
    - Even when their total memory $\gg$ physical memory cap
- Goal 3: Provide protection
  - One process won't mess up the entire computer
  - One process won't mess up with other processes

2 Process Management

Question: How would you build an Operating System?

2.1 A strawman solution

- Assign individual memory (say 1/3) to each APP
- Assign CPU to work on an APP until completion, then next

This is sort of "scheduling".

- **Does this algorithm provide convenience?** *Yes.*
- **Does this algorithm provide protection?** *Yes.*
- **What about efficiency?** We can't run N processes but not N times slower...

Problem: resource is not fully utilized.

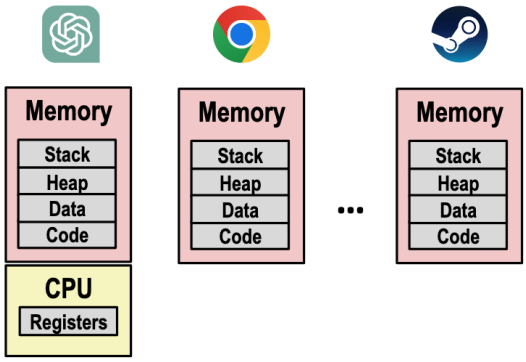


Figure 1: Example: Assign CPU to ChatGPT app, upon completion, grant it to Chrome, then Steam...

2.2 Time sharing of processors

- Idea: Virtualize the CPU time as time slices
- Assign time slices to different processes

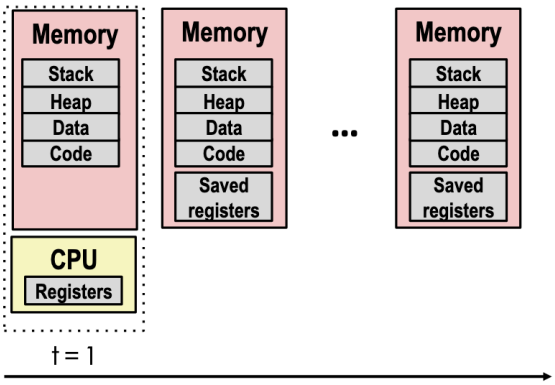


Figure 2: At t=1, assign CPU to work on the first process for 1s

If the first process has used CPU for 1s but still running:

- Save current registers in memory
- So later the process could recovery the previous state
- CPU register is small and fast

Then, move to the next process:

- Assign time slice t = 2 to the next process
- Resume progress: Move Saved registers from memory to CPU

After the second time slice:

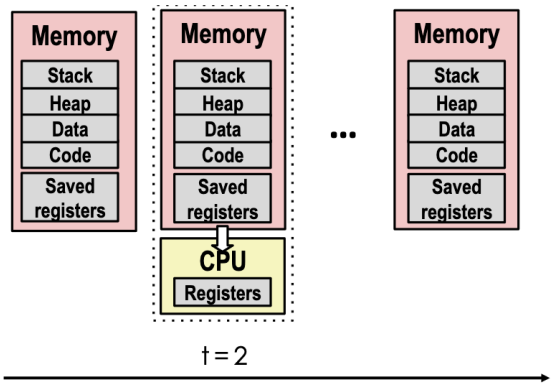


Figure 3: At t=2, assign CPU to work on the second process for 1s

- Then we repeat
- This is called **context switch**

Even in today's system, people do a lot of context switch

- For example, you can do context switch for **GPU**s.
- Copy the content of GPU to somewhere, move to the next machine learning model to do some prediction.

2.3 Time sharing for multiple CPU cores

Two ways of doing it:

- 1. All processors sweep from left (1st process) to right (last process).
- 2. Each process accounts for 1/2 of the processes

Question: If completion of each process requires roughly the same number of time slices, which is faster?

Answer: Equally fast.

Question: If the first process requires 5 times slices, and the last process requires 15 time slices, and they are grouped in different groups. Which is faster?

Answer: Likely the first one.

3 Implementing Multiprocessing

We begin by creating a representation for virtualizing the CPU resources temporally and spatially. Which means we need to divide CPU time into time slices which we can then allocate to processes. Spatially, we represent each core of the CPU as a resource that we assign to a process. Let's begin by creating a Process ID (PID) for each process.

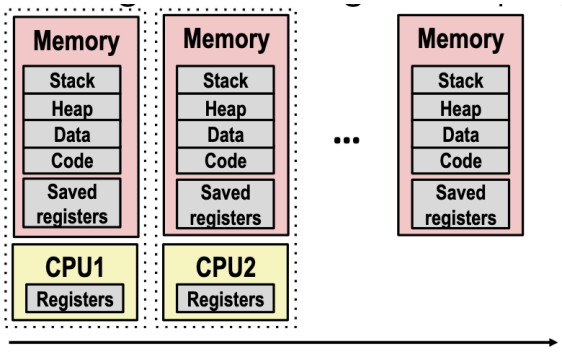


Figure 4: Multiple CPU cores

3.1 Temporal Abstraction

To switch between processes at different times, he CPU must be able to put a process in different states as per the requirement.

Three states of a process are:

- 1. Running - A process is running during its allocated time slice.
- 2. Ready - CPU will run this process as soon as it has a time slice allocated to it.
- 3. Blocked - Process has to wait for other process to finish (say I/O) before it is ready to be run. No time slice is allocated to this process in this state.

Represent time allocation as Gantt chart. Divide CPU time to processes. Then it becomes an optimisation problem where objective may be to minimize the idle time, finish most processes etc.

Figure 5: Gantt chart representation of temporal abstraction

Time sharing is implemented via a scheduling policy. A **schedule** is a record of what process runs when on each CPU.

Key terms of scheduling a process (job):

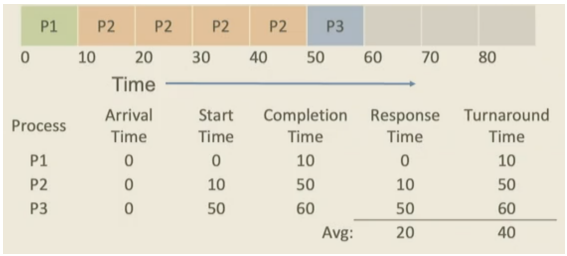
- 1. Arrival time: Time when the process is created
- 2. Job length: Time required to finish a job. Note that this may not always be known.
- 3. Start time: Time when a process actually runs on the CPU. It is different from arrival time. Two processes may arrive at the same time but on one CPU only one process can run at a time slice. So the start time of the other process would be different from its arrival time.

- 4. Completion time: Time point at which the process finishes.
- 5. Response time: (Arrival time − Start time) Denotes how long the process waited to get started.
- 6. Turnaround time: (Completion time − Arrival time)

Workload is the set processes, arrival times, and job lengths the CPU has to handle. For some jobs we don't know the workloads since we don't know how losing will that process will be in use. But for others we know the workload, thus we can optimise them.

3.2 Scheduling Policy

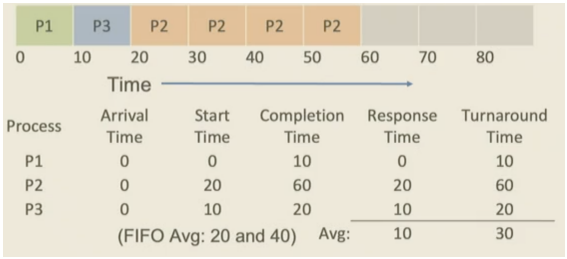
- 1. FIFO: First in first out. Whichever process comes first, will be allocated the CPU. Only after that process is done, the CPU moves to other processes. Disadvantage is that the response time and turnaround times are high.



Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	10	50	10	50
P3	0	50	60	50	60
			Avg:	20	40

Figure 6: FIFO Example

- 2. SJF. Shortest Job First. Sort the processes as per their job length and allocate based on the shortest jobs first. Disadvantage is that we don't know the job length always. So it is only useful when we know the specific job lengths.



Process	Arrival Time	Start Time	Completion Time	Response Time	Turnaround Time
P1	0	0	10	0	10
P2	0	20	60	20	60
P3	0	10	20	10	20
	(FIFO Avg: 20 and 40)		Avg:	10	30

Figure 7: SJF Example

- 3. Round Robin. Allocates same quantum of time to each process. It is fair to each process but average time goes up.
- 3. SCTF. Shortest Completion Time First. It is applicable when processes arrive at different times. Preemption is possible. Preemption is when OS can start or stop a process in favor of a high priority task.

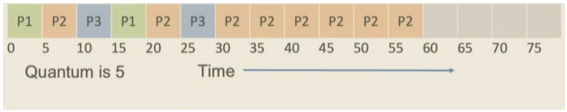


Figure 8: Round Robin Example

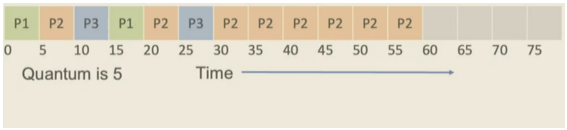


Figure 9: SCTF Example

Essentially scheduling faces two key challenges, maximize workload performance while maintaining allocation fairness for a process.

Perfomance Metric is usually Average Turnaround Time ad **Fairness Metric** can be Jain's Fairness Index.

There are many other Scheduling poilcies, today ML schedulers are gaining ground.

In ML models we also observe scheduling. In ChatGPT, response time is difference between the time we send the input and the time we recieve the first token. Turnaround time of the language model request is the difference between the time we send the input and the time we recieve the last token. Here the job length is not fixed since the output length is unknown, although the total token length of input is known. There can be a maximum output length but length of all outputs is not the same. ChatGPT is also not fair to everyone, the paid members get preference.

Another interesting part is how to do batching of requests and scheduling them. The recent paper "Orca: A Distributed Serving System for Transformer-Based Generative Models" addresses these issues.