time sharing problems
1. performance (no excessive overhead for switching/scheduling)
2. control: run process efficiently while retaining control over CPU (program could run forever or access machinery it shouldn't)

user mode: code running in this mode is restricted in what it can do (violations cause hardware interrupts, causing OS to kill it)
kernel mode: OS runs in this mode. does what it likes.

system call: when a user program wishes to perform some kind of privileged operation

indirection to serve as protection:
"trap" instruction: executed by program to jump into the kernel and raise privilege level
"return-from-trap" instruction: OS calls when finished
"kernel stack": a per process stack that saves process context, to correctly return from system call
"trap table": kernel sets up at boot time to inform hardware of the locations of these trap handlers
"system call number': assigned to each system call. user code places it in a register in the stack

Fig 6.2

scheduling:
if a process is running, OS is not running
how can OS regain control of the CPU so it can switch between processes

Cooperative Approach: Wait for system calls, inc yield

Non-cooperative Approach: Timer interrupt, OS takes control

Scheduler / context switch

Fig 6.3

Operating System (OS) CS232

Scheduling Algorithms

Dr. Muhammad Mobeen Movania

Outlines

- Introduction of a few key terms
- Policy vs Mechanism
- Scheduling Assumptions
- Scheduling Algorithms and their Issues
  - First In First Out (FIFO)/First Come First Serve(FCFS)
  - Shortest Job First (SJF)
  - Shortest Time-to-Completion First (STCF)
  - Round Robin (RR)
- Scheduling with I/O
- Summary

Few Key Terms

- Scheduler
  - Part of OS that selects the next process to run
- Scheduling policy
  - Policy/algorithm according to which the scheduler selects the next process to be run on the CPU
- Workload
  - The set of process that the OS has to run
  - Knowing it helps in developing better policies
- Metric
  - Criterion to measure the performance of a scheduler
  - No. of jobs/sec
  - $T_{turnaround} = T_{completion} - T_{arrival}$
  - $T_{response} = T_{first\_run} - T_{arrival}$

Policy vs Mechanism

- A *policy* helps pick an option if there is a choice
  - For e.g. given two programs which should run first?
  - Usually answers which, what, when
  - Based on some algorithm to pick a choice
- A *mechanism* is how a policy would be implemented
  - For e.g. given two processes how a process will be picked
  - Usually answers how
  - Details implementation

Scheduling Assumptions

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

Scheduling Algorithm: FIFO

- Key Idea: Schedule processes in order of arrival, first process first and so on
- Example
  - Processes A, B, C each take 10 secs to execute

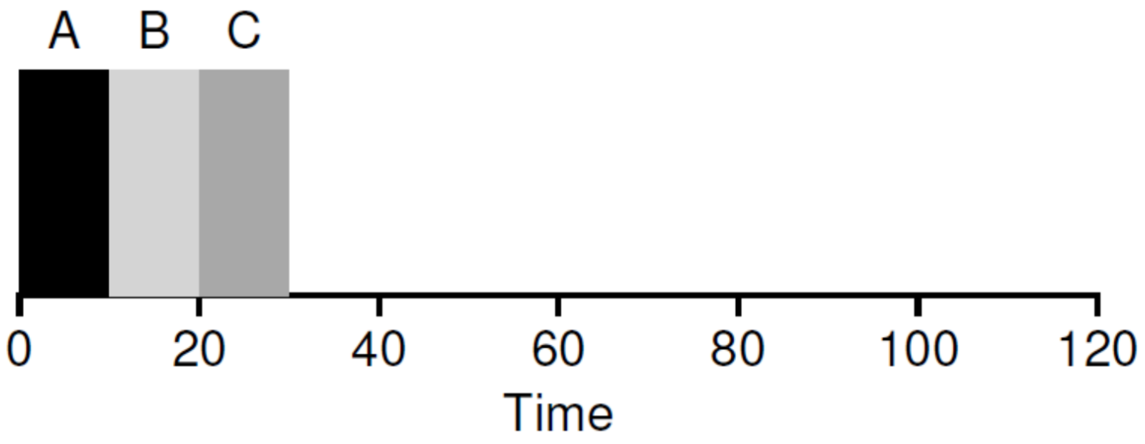

Figure 7.1: **FIFO Simple Example**

Calculating our metric

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

- As per assumption 2, $T_{arrival} = 0$ so,

$$T_{turnaround} = T_{completion}$$

- Average turnaround time
  - (10+20+30)/3 = 20 secs

Scheduling Algorithm: FIFO

- What if, assumption 1 is relaxed, now we may have processes with different times
  - FIFO will give largely varying turnaround times depending on which process is scheduled first
- Example
  - Process A takes 100 secs, B and C each take 10 secs to execute

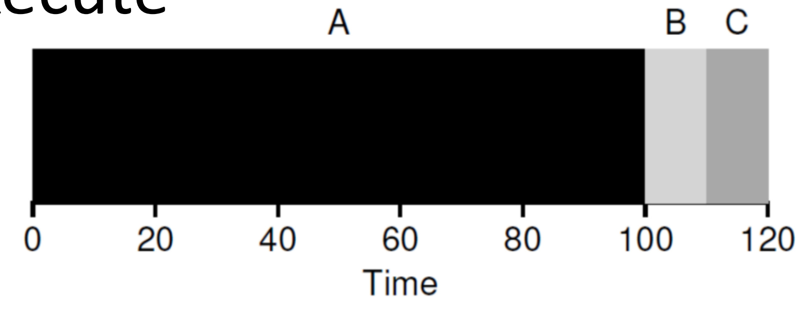

Figure 7.2: **Why FIFO Is Not That Great**

Calculating our metric

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

- As per assumption 2, $T_{arrival} = 0$ so,

$$T_{turnaround} = T_{completion}$$

- Average turnaround time
  - (100+110+120)/3 = 110 secs

Issues with FIFO

- Average turnaround time varies greatly depending in which order the processes are scheduled
- **Convoy Effect:** A process with large completion time prevents processes with small completion time to execute

Scheduling Algorithm : SJF

- Key Idea
  - Schedule shortest process first
- Example
  - Process A takes 100 secs, B and C each take 10 secs to execute

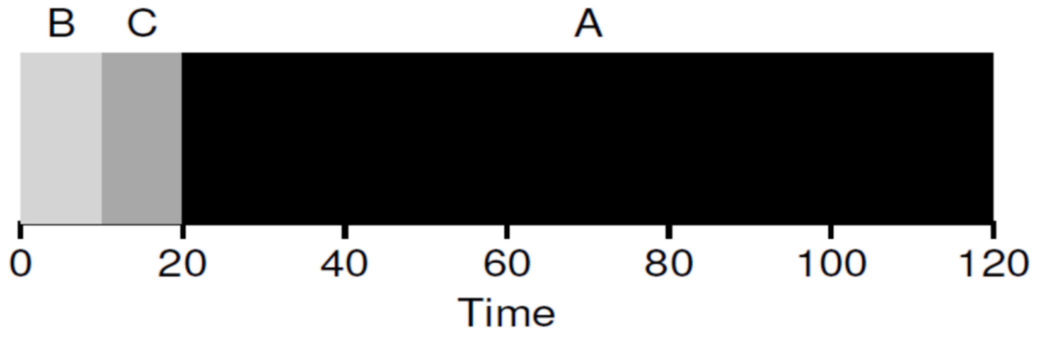

Figure 7.3: **SJF Simple Example**

- Average turnaround time
  - (10+20+120)/3 = 50 secs

Scheduling Algorithm : SJF

- If assumption 2 is relaxed that is processes have varying arrival times
  - Process A arrives at time 0 secs, B and C arrive at time 10 secs

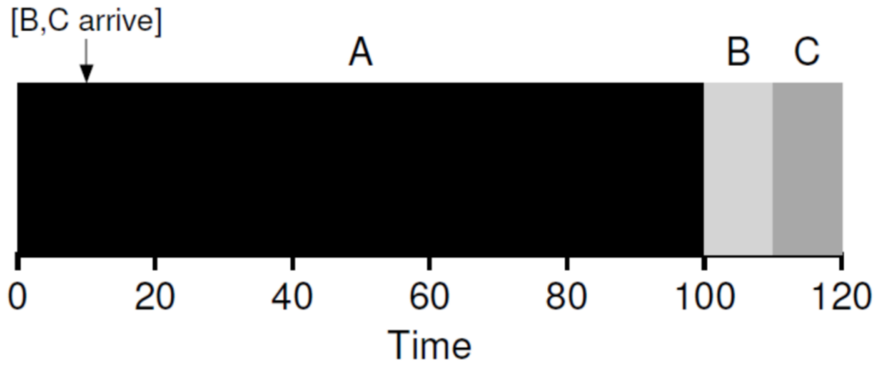

Figure 7.4: **SJF With Late Arrivals From B and C**

- Average turnaround time
  - ((100-0)+(110-10)+(120-10))/3 = 103.33 secs

Issues with SJF

- With varying arrival times, we suffer from the same **convoy effect**
  - a process with larger completion time prevents shorter processes from being scheduled
- Average turnaround time varies depending on the arrival time and completion time of first scheduled process

Scheduling Algorithm : STCF

- Shortest Time to Completion First is also known as Preemptive SJF
- Key Idea:
  - Relax assumption 3 i.e. a process may be pre-empted (halted during execution)
  - When a new job comes, the STCF scheduler determines which of the remaining jobs (including the new job) has the least time left, and schedules that, preempting the currently running process

Example

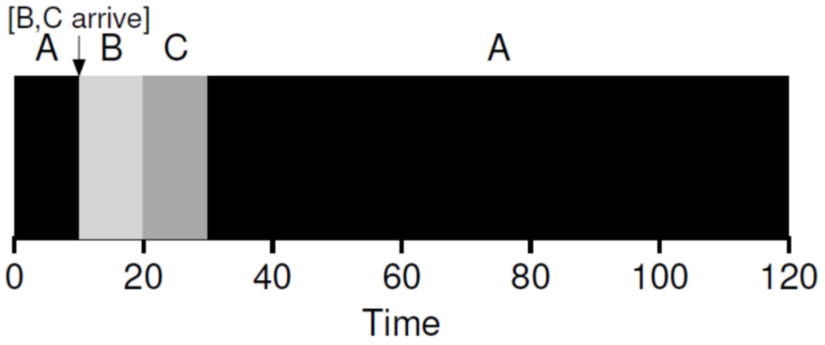

Figure 7.5: **STCF Simple Example**

- Process A takes 100 secs, B and C each take 10 secs to execute
- Average turnaround time
  - ((120-0)+(20-10)+(30-10))/3 = 50 secs

Issues with STCF

- STCF is great if
  - We already know process's total execution time
  - Process only uses CPU (no I/O)
  - Only turnaround time metric is used
- However
  - Modern processors are time shared and so users expect to have an interactive response
  - Turnaround time metric is thus not useful
- New Metric (Response Time)
  - $T_{response} = T_{first\_run} - T_{arrival}$

Example (Calculating response time)

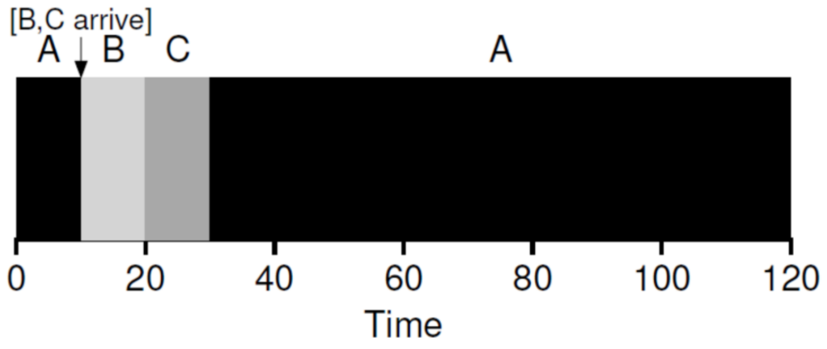

Figure 7.5: **STCF Simple Example**

- *$T_{arrival(A)}$ = 0, $T_{arrival(B)}$ = 10, $T_{arrival(C)}$ = 10*
- *$T_{first\_run(A)}$ = 0, $T_{first\_run(B)}$ = 10, $T_{first\_run(C)}$ = 20*
- *$T_{response(A)}$ = (0-0=0), $T_{response(B)}$ = (10-10=0), $T_{response(C)}$ = (20-10=10)*
- Average response time
  - (0+0+10)/3 = 3.33 secs
- Crux
  - The last process has to wait till all previous processes have finished completely therefore we do not get interactivity

Scheduling Algorithm : RR

- Round Robin (RR) Scheduling
- Key Idea:
  - Make a scheduler that is sensitive to response time
  - Instead of running jobs to completion, RR runs job for a time slice (scheduling quantum)
  - If a process finishes its time slice, another process is scheduled from the ready queue

Example (SJF and response time)

- Processes A, B, C arrive at the same time and each runs for 5 secs.

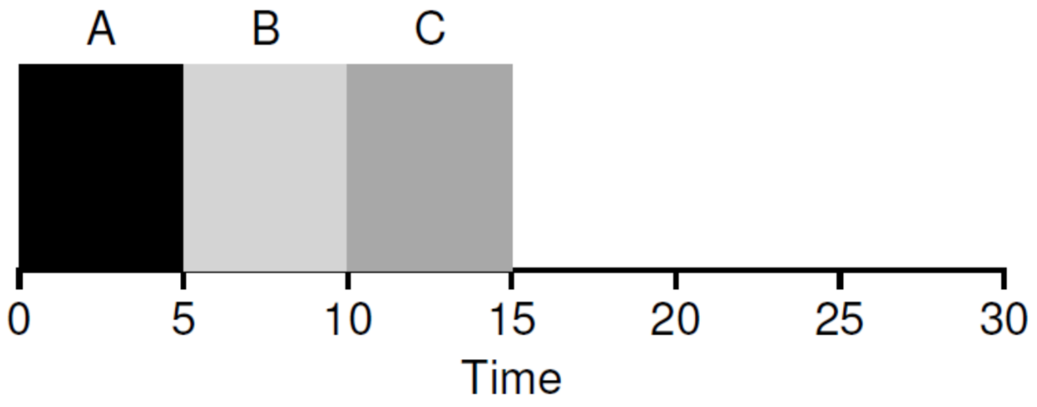

Figure 7.6: **SJF Again (Bad for Response Time)**

- Response Time with SJF
  - ((0-0)+(5-0)+(10-0))/3 = 5 secs

Example (RR and response time)

- Processes A, B, C arrive at the same time and each runs for 5 secs. Time slice: 1 sec

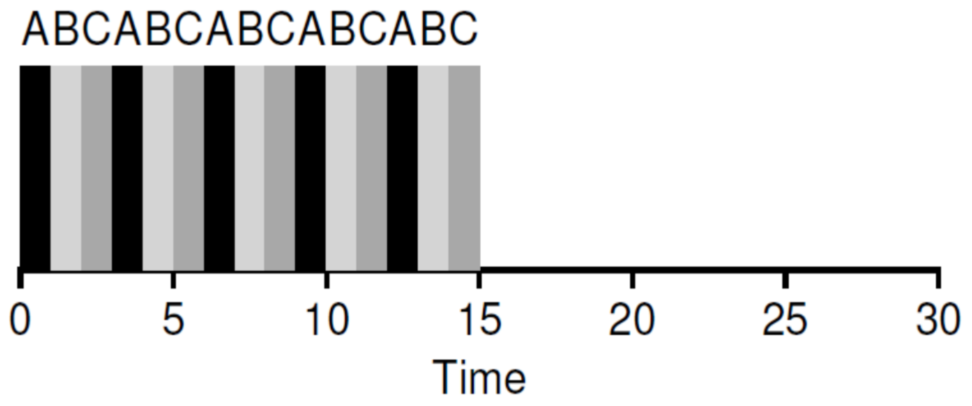

Figure 7.7: **Round Robin (Good For Response Time)**

- Response Time with RR
  - ((0-0)+(1-0)+(2-0))/3 = 1 sec

Issues with RR

- Length of the time slice is critical
  - The shorter it is, the better the performance of RR under the response-time metric
  - But making the time slice too short is problematic: the cost of context switching will dominate overall performance
- Deciding on the length of the time slice presents a trade-off to a system designer,
  - making it long enough to **amortize** the cost of context switching without making it so long that the system is no longer responsive
- RR has a large turnaround time
  - Worse algorithm to use if turnaround time is the metric

Scheduling with I/O

- Relaxing assumption 4
  - Scheduler must decide what to do for the time when a process is blocked waiting for I/O completion
- Example
  - Two processes A and B both run for 50 ms but process A initiates I/O after every 10 ms

Example (Scheduling with I/O)

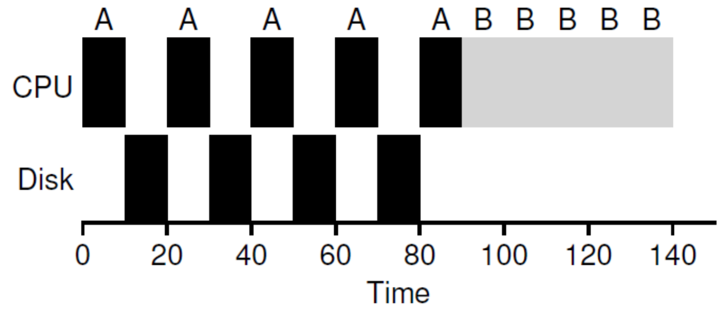

Figure 7.8: **Poor Use Of Resources**

- Naïve STCF without overlap considers each 10 ms sub-job of A as an independent job
  - Scheduler picks 10 ms of A then initiates I/O for 10 ms, then 10 ms of A and so on until process A finishes its 50 ms execution and then schedules process B for 50 ms

Example (Scheduling with I/O)

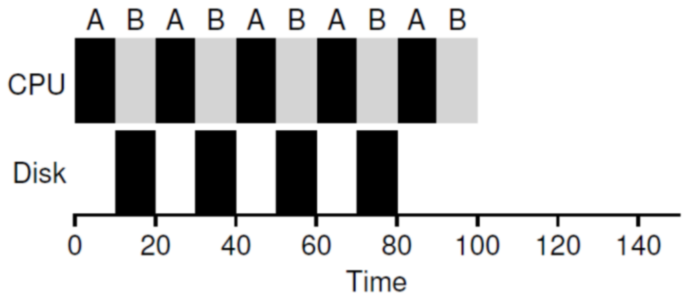

Figure 7.9: **Overlap Allows Better Use Of Resources**

- Optimal STCF scheduler would overlap I/O duration with execution of 10 ms of process B
  - Scheduler picks 10 ms of A then initiates I/O for 10 ms
  - With overlap, the I/O duration is overlapped by 10 ms execution of process B.
  - When I/O finishes after 10 ms, the execution of process B is preempted to allow 10 ms of process A and so on.

Summary

- We discussed basic ideas behind scheduling
- Two families of approaches were discussed
  - Algorithms that run the shortest job remaining thus optimize turnaround time (SJF, STCF)
  - Algorithms that alternate between all jobs thus optimize response time (RR)