Recap – Direct Execution

- *direct execution* means user processes run directly on the CPU
  - in *user mode*, where privileged ops are not allowed
- control is transferred to the OS in the case of –
  - *system calls* – containing a trap instruction
  - *faults* – problems executing an instruction such as a divide-by-zero error
  - *interrupts* – due to an external condition such as a timer or completed I/O request complete
- a *trap table* set up on boot provides lookup for the handler code to ensure safe jumps when switching to instructions run in *kernel mode*

Recap – Context Switches

- each process (and the kernel) have their own address space in memory but anything running on the CPU uses the CPU registers (PC, etc) so these must be saved –
  - when going from user mode to kernel mode
  - during a context switch from one user process to another

Scheduling

- returning from a trap or interrupt handler returns to some process, but which one?
  - context switch is the mechanism
  - the scheduling policy determines which of the ready/runnable processes will run next
- real schedulers are complex and typically involve many heuristics
  - simple scheduling policies have good theoretical guarantees but involve too many (unrealistic) simplifying assumptions
- modern systems use preemptive schedulers
  - process can be switched out at any time, instead of only when the process is no longer ready/runnable (terminated, blocking syscall)
  - timer interrupts allow the OS to get control periodically

Scheduling Policy Goals

- performance metrics
  - minimize *turnaround* (time from process creation to completion)
  - minimize *response time* (time from process creation to the first time it is run)
- system efficiency metrics
  - maximize *utilization* (percentage of time CPU is doing useful work)
    - want efficient use of CPU hardware
  - low scheduler *overhead*
    - scheduler shouldn't take too long to choose the next job
    - scheduler shouldn't switch running jobs too often
- fairness metrics
  - all processes get a *fair share* of the CPU
    - including accounting for priorities

First In First Out (FIFO)

- also known as FCFS (first come, first served)
- newly created processes are put in a queue
  - next thing scheduled is the job at the head of the queue
- non-preemptive
  - each process runs until it blocks or terminates
  - when a process unblocks, it is re-added to the end of the queue
- example

Process	CPU time needed (units)	Arrives at end of time unit	Execution time slots
P1	5	0	1-5
P2	3	1	6-8
P3	2	3	9-10

  - average turnaround time? (5+7+7)/3 = 6.33
    - P1 arrives at the end of time unit 0 and finishes at the end of time unit 5 → turnaround time is (5-0) = 5
  - average response time? (0+4+5)/3 = 3
    - P2 arrives at the end of time unit 1 and starts at the beginning of time unit 6 → response time is (6-2) = 4

https://www.cse.iitb.ac.in/~mythili/os/iitb_slides/scheduling.pdf

Analyzing FIFO

can be hard to analyze in general – make simplifying assumptions to facilitate analysis

Assume –

- all jobs arrive at the same time
- all jobs have the same length (10s)
  - average turnaround time? (10+20+30)/3 = 20
  - average response time? (0+10+20)/3 = 10
  - can a different order of jobs be any better? no, they are all the same length

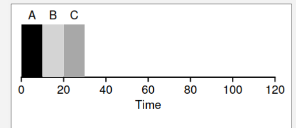

Figure 7.1: **FIFO Simple Example**

- all jobs arrive at the same time
- job A runs for 100s, B and C run for 10s
  - average turnaround time? (100+110+120)/3 = 110
  - average response time? (0+100+110)/3 = 70
  - this is worse than the previous case due to the *convoy effect* – short jobs can get stuck behind long ones

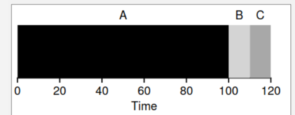

Figure 7.2: **Why FIFO Is Not That Great**

https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf

Shortest Job First (SJF)

try to fix the convoy effect...

- newly created processes are put in a priority queue ordered by *CPU burst time*
  - process with smallest burst time runs next
    - CPU burst time = amount of CPU time the process needs before blocking or termination
- non-preemptive
  - each process runs until it blocks or terminates
  - when a process unblocks, it is re-added to the priority queue
- example

Process	CPU burst	Arrival time	Execution time slots
P1	5	0	1-5
P2	3	1	8-10
P3	2	3	6-7

  - average turnaround time? (5+9+4)/3 = 6
    - P1 arrives at the end of time unit 0 and finishes at the end of time unit 5 → turnaround time is (5-0) = 5
  - average response time? (0+6+2)/3 = 2.66
    - P2 arrives at the end of time unit 1 and starts at the beginning of time unit 8 → response time is (8-2) = 6

https://www.cse.iitb.ac.in/~mythili/os/iitb_slides/scheduling.pdf

Analyzing SJF

can be hard to analyze in general – make simplifying assumptions to facilitate analysis

- all jobs arrive at the same time
- job A runs for 100s, B and C run for 10s
  - average turnaround time? (120+10+20)/3 = 50
  - average response time? (20+0+10)/3 = 10

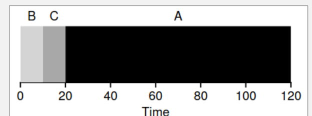

Figure 7.3: **SJF Simple Example**

SJF is provably optimal (lowest average turnaround) in this scenario (non-preemptive, known job lengths, all jobs arrive at the same time)

- jobs B, C arrive at time 10
- job A runs for 100s, B and C run for 10s
  - average turnaround time? (100+110+120)/3 = 110
  - average response time? (0+100+110)/3 = 70
  - without preemption, short jobs can still get stuck behind long ones

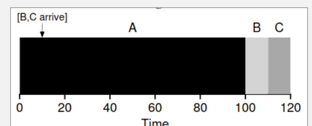

Figure 7.4: **SJF With Late Arrivals From B and C**

https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf

Shortest Time-to-Completion First (STCF)

- also called Shortest Time Remaining First (STRF) and Preemptive Shortest Job First (PSJF)
- newly created processes are put in a priority queue ordered by *CPU burst time*
  - process with smallest burst time runs next
    - CPU burst time = amount of CPU time the process needs before blocking or termination
- preemptive
  - a newly arrived process can preempt a running process if its CPU burst time is shorter than the remaining time of the currently running process

Shortest Time-to-Completion First (STCF)

Process	CPU burst	Arrival time	Execution time slots
P1	5	0	1, preempted, then 7-10
P2	3	1	2-4
P3	2	3	5-6

- example
  - average turnaround time? (10+3+3)/3 = 5.33
    - P1 arrives at the end of time unit 0 and finishes at the end of time unit 10 → turnaround time is (10-0) = 10
  - average response time? (0+0+1)/3 = 0.33
    - P3 arrives at the end of time unit 3 and starts at the beginning of time unit 5 → response time is (5-4) = 1

Analyzing STCF

can be hard to analyze in general – make simplifying assumptions to facilitate analysis

- jobs B, C arrive at time 10
- job A runs for 100s, B and C run for 10s
  - average turnaround time? (120+10+20)/3 = 50
  - average response time? (0+10+20)/3 = 10

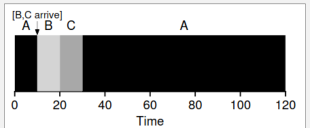

Figure 7.5: **STCF Simple Example**

- STCF is good for turnaround, but can be poor for response time
  - later-arriving short jobs no longer get stuck behind earlier-arriving long jobs but if multiple jobs arrive at once, some won't start until the other jobs complete

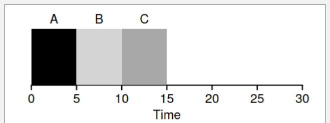

Figure 7.6: **SJF Again (Bad for Response Time)**

Round-Robin (RR)

try to fix the response time...

- also known as *time slicing*
- every process runs for a fixed slice of time
  - slice should not be too small – to amortize the cost of the context switch
  - slice should not be too big – to provide good responsiveness
- preemptive
  - processes are preempted when their time slice is up
  - requires timer interrupt

Process	CPU burst	Arrival time
P1	5	0
P2	3	1
P3	2	3

- example
  - average turnaround time?
  - average response time?

Analyzing RR

can be hard to analyze in general – make simplifying assumptions to facilitate analysis

- all jobs arrive at the same time
- all jobs are the same length (5s)

- STCF when all jobs arrive at the same time = SJF
  - average turnaround time? (5+10+20)/3 = 11.66
  - average response time? (0+5+10)/3 = 5

- RR with 1-unit time slices
  - average turnaround time? (13+14+15)/3 = 14
  - average response time? (0+1+2)/3 = 1

- RR is good for response time, fairness but bad for turnaround time

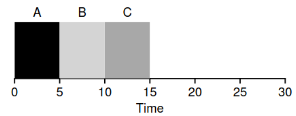

Figure 7.6: **SJF Again (Bad for Response Time)**

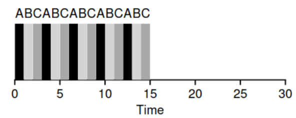

Figure 7.7: **Round Robin (Good For Response Time)**