Systems and Networking I

Applied Computer Science and Artificial Intelligence

2025-2026

SAPIENZA
UNIVERSITÀ DI ROMA

Gabriele Tolomei

Computer Science Department

Sapienza Università di Roma

tolomei@di.uniroma1.it

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling

TODAY

- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MQ)
- Multilevel Feedback-Queue (MFQ)

NEXT TIME

Scheduling Algorithms: An Overview

- **First-Come-First-Serve (FCFS)**
- Round Robin (RR)
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MLQ)
- Multilevel Feedback-Queue (MLFQ)

First-Come-First-Serve (FCFS)

- Very simple! Just a FIFO queue, like customers waiting in line at the post office
- The scheduler executes jobs to completion in arrival order
- The scheduler takes over only when the currently running job asks for an I/O operation (or finishes its execution)
- A job may keep using the CPU indefinitely (i.e., until it blocks)

First-Come-First-Serve (FCFS)

- Very simple! Just a FIFO queue, like customers waiting in line at the post office
- The scheduler executes jobs to completion in arrival order
- The scheduler takes over only when the currently running job asks for an I/O operation (or finishes its execution)
- A job may keep using the CPU indefinitely (i.e., until it blocks)

Non-preemptive

First-Come-First-Serve (FCFS): Scenario I

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario I

New | A | B | C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario I

New	A	B	C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Arrival time = 0 for all*

No I/O burst

* Actually, in this example, A arrives first, then B, and finally C comes: arrival time differences are considered negligible

First-Come-First-Serve (FCFS): Scenario I

New	A	B	C
Ready	A	B	C
Waiting			
Running			

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario I

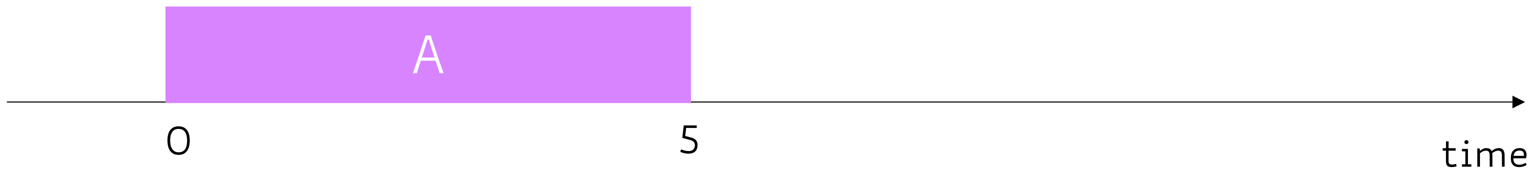
New A B C

Ready B C

Waiting

Runnin A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



First-Come-First-Serve (FCFS): Scenario I

New A B C

Ready C

Waiting

Runnin B

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario I

New A B C

Ready

Waiting

Runnin C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Average Waiting Time

$$N = \text{number of jobs}$$

$$T_i^{arrival} = \text{arrival time of job } i$$
$$T_i^{completion} = \text{completion time of job } i$$
$$T_i^{burst} = \text{burst time of job } i$$
$$T_i^{turnaround} = \text{tournaround time of job } i = T_i^{completion} - T_i^{arrival}$$

$$\overline{T}^{waiting} = \text{avg. waiting time} \qquad = \frac{1}{N}\sum_{i=1}^{N}(T_i^{turnaround} - T_i^{burst})$$

Unless otherwise specified, we will assume all jobs arrive at the same time, i.e.,

$$T_i^{arrival} = 0 \; \forall i \in \{1, \ldots, N\}$$

First-Come-First-Serve (FCFS): Scenario I

New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

avg. waiting time =

First-Come-First-Serve (FCFS): Scenario I

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

avg. waiting time = (0 + 5 + 7)/3 = 4

First-Come-First-Serve (FCFS): Scenario II

Order	Job	CPU burst (time units)
1	B	2
2	C	3
3	A	5

Arrival time = 0 for all

No I/O burst

avg. waiting time =

First-Come-First-Serve (FCFS): Scenario II

New | B | C | A

Order	Job	CPU burst (time units)
1	B	2
2	C	3
3	A	5

avg. waiting time = (5 + 0 + 2)/3 ~ 2.3

First-Come-First-Serve (FCFS): Scenario III

New	A	B	C
Ready	A	B	C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

First-Come-First-Serve (FCFS): Scenario III

New	A	B	C
Ready	A	B	C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

First-Come-First-Serve (FCFS): Scenario III

New A B C

Ready B C

Waiting

Running A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

First-Come-First-Serve (FCFS): Scenario III

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

First-Come-First-Serve (FCFS): Scenario III

New: A B C

Ready: C

Waitin[g]: A

Runnin[g]: B

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

I/O task requested by A might have finished before B finished

Non-preemptive scheduler!

First-Come-First-Serve (FCFS): Scenario III

New A B C

Ready C

Waiting

Runnin A

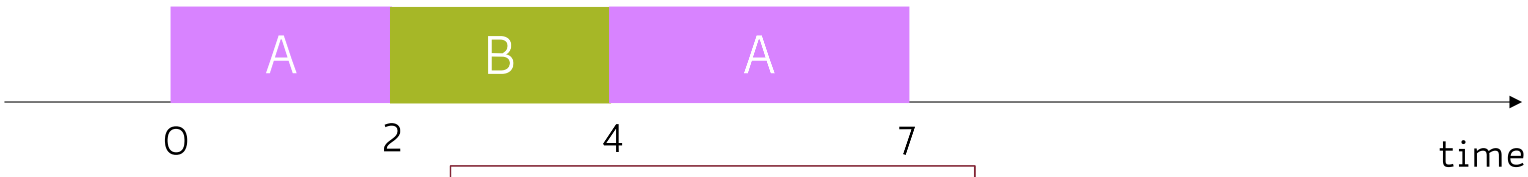
Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

First-Come-First-Serve (FCFS): Scenario III

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



Why A and not C?

First-Come-First-Serve (FCFS): Scenario III

New A B C

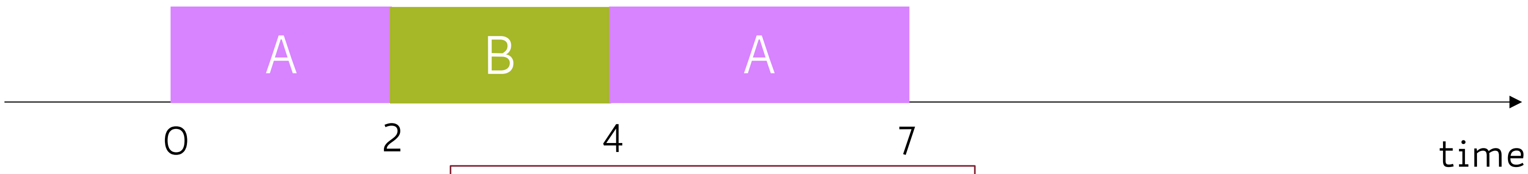
Ready C

Waiting

Runnin A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O



Why A and not C?

Because the FCFS scheduler cares only about the arrival time on the ready queue

First-Come-First-Serve (FCFS): Scenario III

New A B C

Ready

Waiting

Runnin C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

First-Come-First-Serve (FCFS): Scenario III

New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

avg. waiting time =

First-Come-First-Serve (FCFS): Scenario III

New | A | B | C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

avg. waiting time = (2 + 2 + 7)/3 ~ 3.7

First-Come-First-Serve (FCFS): Scenario III

New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

A does also I/O

NOTE:
We should remove from A's waiting time the time it spent doing I/O

FCFS: PROs and CONs

- PRO:
  - very simple!

FCFS: PROs and CONs

- PRO:
  - very simple!
- CONs:
  - (average) waiting time is highly variable as short CPU-burst jobs may sit behind very long ones

FCFS: PROs and CONs

- PRO:
  - very simple!
- CONs:
  - (average) waiting time is highly variable as short CPU-burst jobs may sit behind very long ones
  - **convoy effect** → poor overlap between CPU and I/O since CPU-bound jobs will force I/O bound jobs to wait

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- **Round Robin (RR)**
- Shortest-Job-First (SJF)
- Priority Scheduling
- Multilevel Queue (MLQ)
- Multilevel Feedback-Queue (MLFQ)

Round Robin (RR)

- Similar to FCFS, except that CPU bursts are assigned with limits called **time quantum** or (**time slice**)
- When a job is given the CPU, a timer is set for a certain value:
  - If the job finishes before the time quantum expires, then it is swapped out of the CPU just like the normal FCFS algorithm
  - If the timer goes off first, then the job is swapped out of the CPU and moved to the back end of the ready queue
- Used in many time-sharing systems in combination with timer interrupts

Round Robin (RR)

- Similar to FCFS, except that CPU bursts are assigned with limits called **time quantum** or (**time slice**)
- When a job is given the CPU, a timer is set for a certain value:
  - If the job finishes before the time quantum expires, then it is swapped out of the CPU just like the normal FCFS algorithm
  - If the timer goes off first, then the job is swapped out of the CPU and moved to the back end of the ready queue
- Used in many time-sharing systems in combination with timer interrupts

Preemptive

Round Robin (RR)

- The ready queue is maintained as a **circular queue**
- When all jobs have had a turn, the scheduler gives the first job another turn, and so on…
- RR is fair as it shares the CPU equally among all the jobs
- The average waiting time can be longer than with other scheduling algorithms

Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum

Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum
- Too large time quantum degenerates to FCFS, as jobs are never preempted from the CPU (high average waiting time)

Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum
- Too large time quantum degenerates to FCFS, as jobs are never preempted from the CPU (high average waiting time)
- Too small time quantum implies more context switches, which eventually dominate over the actual CPU utilization (low throughput)

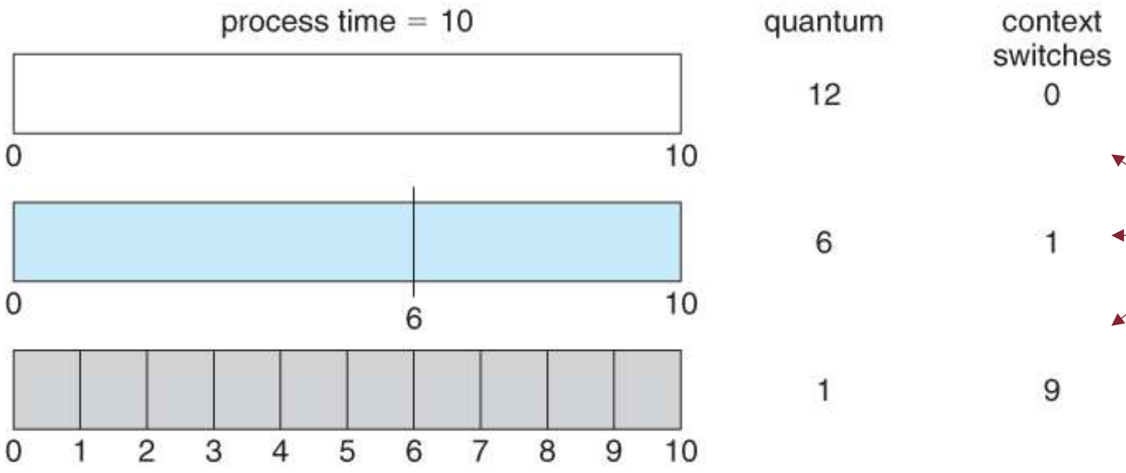
Round Robin (RR): The Time Quantum

- The performance of RR is sensitive to the time quantum
- Too large time quantum degenerates to FCFS, as jobs are never preempted from the CPU (high average waiting time)
- Too small time quantum implies more context switches, which eventually dominate over the actual CPU utilization (low throughput)

Trade-off:
Overhead for context switching should be relatively small compared to time slice

Example: time slice = 10÷100 msec. and context switch = 0.01÷0.1 msec.

Round Robin (RR): The Time Quantum



By decreasing the time quantum the number of context switches increase

Round Robin (RR): The Time Quantum

$$N = \text{number of jobs}$$

$$\delta = \text{time slice}$$

$$\sup\{T_i^{start}\} = \delta * (i - 1), \ \forall i \in \{1, \ldots, N\}$$

upper-bound on the
time a job is scheduled for the first time

worst-case scenario:
all job in front of the queue will use the whole time slice

Round Robin (RR): Example

New | A | B | C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Round Robin (RR): Example

New A B C

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

No I/O burst

Time quantum = 2

Context switch = 0

Round Robin (RR): Example

New	A	B	C
Ready	A	B	C
Waiting			
Running			

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Round Robin (RR): Example

New A B C

Ready B C

Waiting

Running A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

Round Robin (RR): Example

New A B C

Ready B C

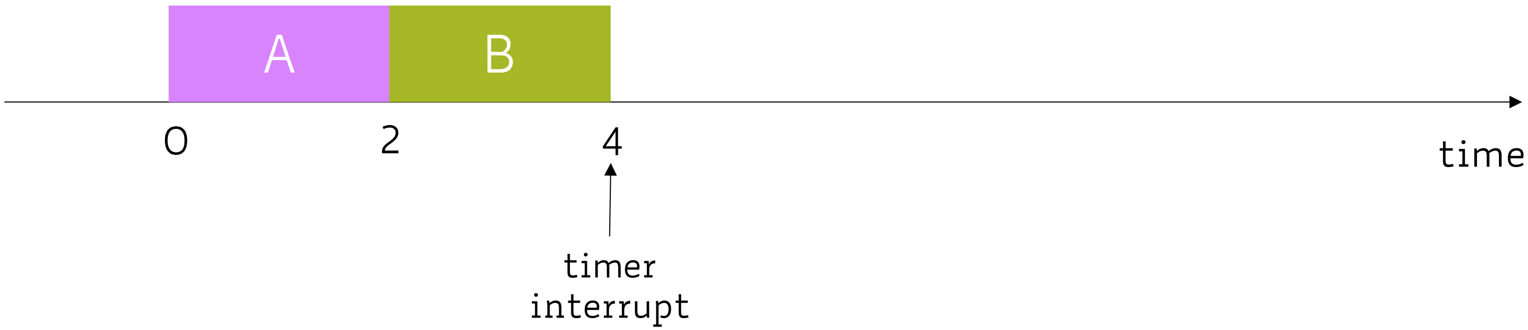
Waiting

Runnin A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

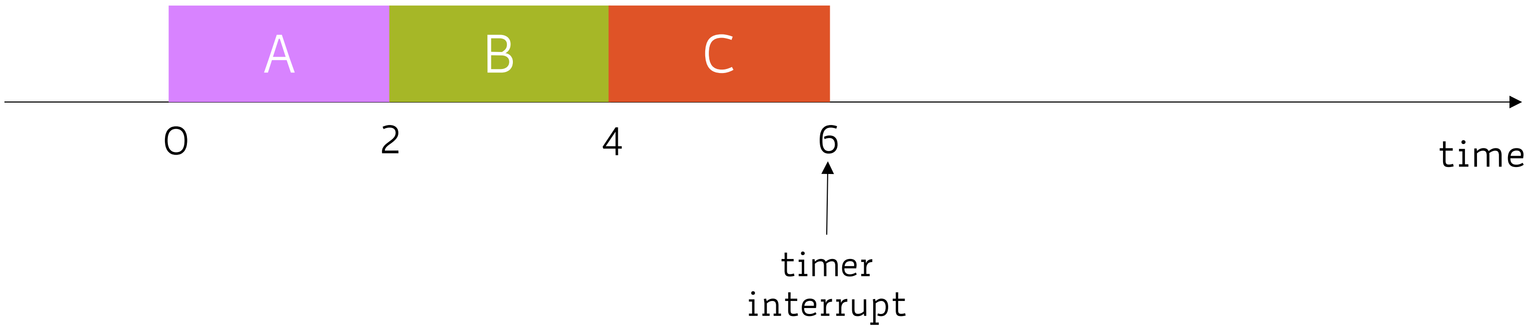
Round Robin (RR): Example

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

New A B C

Ready C

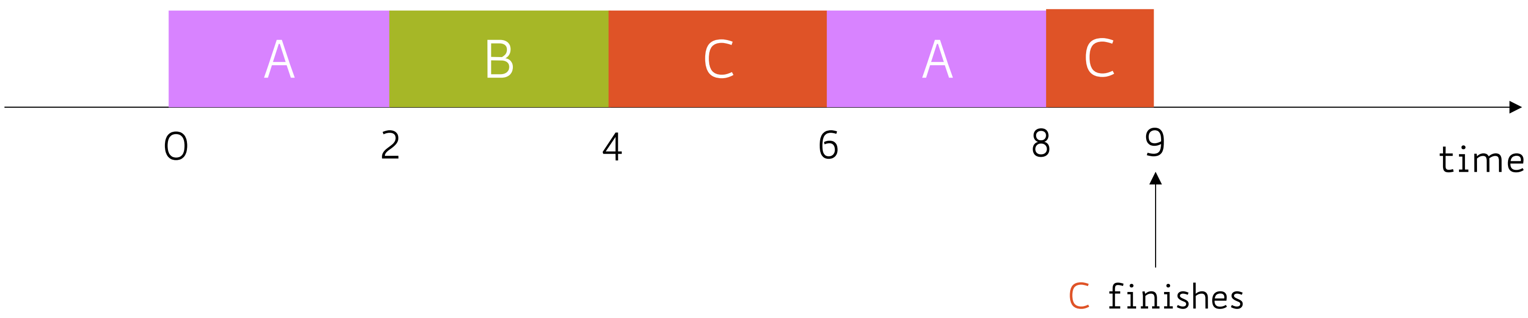
Waiting

Running A

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3

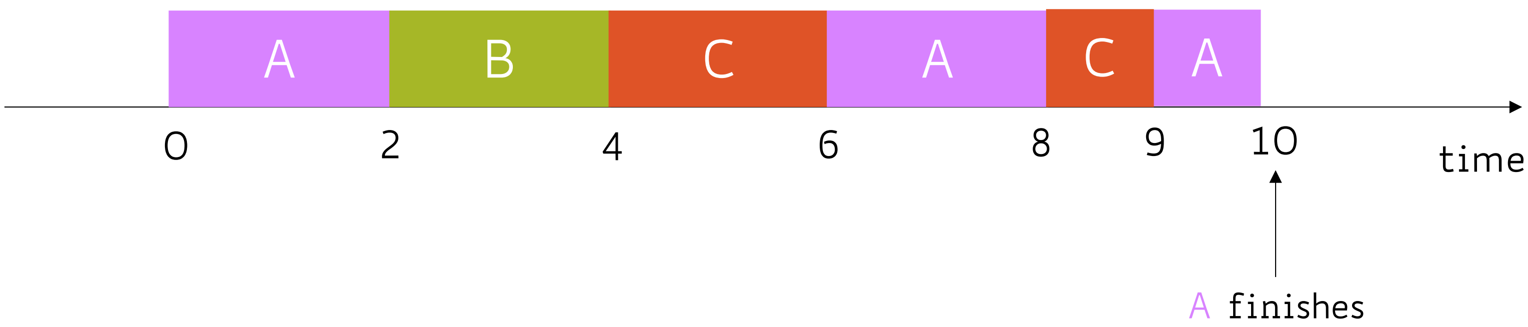
Round Robin (RR): Example

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



Round Robin (RR): Example

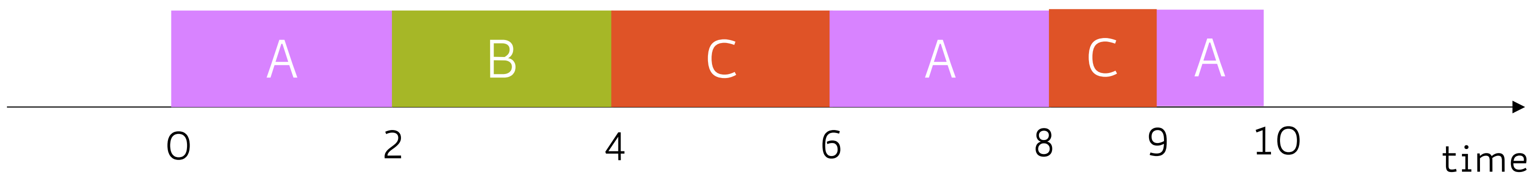
New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



avg. waiting time =

Round Robin (RR): Example

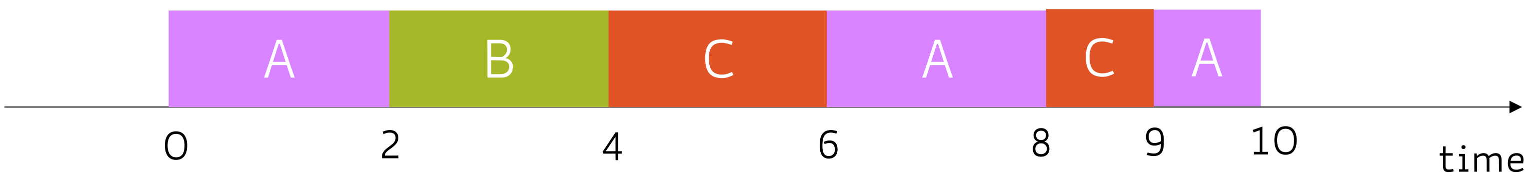
New A B C

Ready

Waiting

Running

Order	Job	CPU burst (time units)
1	A	5
2	B	2
3	C	3



avg. waiting time = (5 + 2 + 6)/3 ~ 4.3

FCFS vs. RR

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	100				
B	100				
C	100				
D	100				
E	100				
Avg.					

FCFS vs. RR

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	100	100			
B	100	200			
C	100	300			
D	100	400			
E	100	500			
Avg.		300			

FCFS vs. RR

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	100	100	496		
B	100	200	497		
C	100	300	498		
D	100	400	499		
E	100	500	500		
Avg.		300	498		

FCFS vs. RR

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	100	100	496	0	
B	100	200	497	100	
C	100	300	498	200	
D	100	400	499	300	
E	100	500	500	400	
Avg.		300	498	200	

FCFS vs. RR

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	100	100	496	0	396
B	100	200	497	100	397
C	100	300	498	200	398
D	100	400	499	300	399
E	100	500	500	400	400
Avg.		300	498	200	398

FCFS vs. RR

Assumptions:

5 jobs, 100 time units of CPU burst each

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	100	100	496	0	396
B	100	200	497	100	397
C	100	300	498	200	398
D	100	400	499	300	399
E	100	500	500	400	400
Avg.		300	498	200	398

FCFS seems to outperform RR in both metrics but... is it fair?

Look at the variance rather than the average!

FCFS vs. RR

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	50				
B	40				
C	30				
D	20				
E	10				
Avg.					

FCFS vs. RR

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	50	50			
B	40	90			
C	30	120			
D	20	140			
E	10	150			
Avg.		110			

FCFS vs. RR

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	50	50	150		
B	40	90	140		
C	30	120	120		
D	20	140	90		
E	10	150	50		
Avg.		110	110		

FCFS vs. RR

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	50	50	150	0	
B	40	90	140	50	
C	30	120	120	90	
D	20	140	90	120	
E	10	150	50	140	
Avg.		110	110	80	

FCFS vs. RR

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time		waiting time	
Job	CPU burst	FCFS	RR	FCFS	RR
A	50	50	150	0	100
B	40	90	140	50	100
C	30	120	120	90	90
D	20	140	90	120	70
E	10	150	50	140	40
Avg.		110	110	80	80

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- **Shortest-Job-First (SJF)**
- Priority Scheduling
- Multilevel Queue (MLQ)
- Multilevel Feedback-Queue (MLFQ)

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination

SJF: Idea

- Schedule the job that has the least *expected* amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

Assuming all jobs arrive at the same time (arrival time = 0)

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

avg. waiting time =

SJF: Idea

- Schedule the job that has the least ***expected*** amount of work to do until its next I/O operation or termination
- "Amount of work" means CPU burst

Job	CPU burst (time units)
A	6
B	8
C	7
D	3

avg. waiting time = (3 + 16 + 9 + 0)/4 = 7

SJF: PROs and CONs

- PROs:
  - Provably optimal when the goal is to minimize the avg. waiting time

SJF: PROs and CONs

- **PROs:**
  - Provably optimal when the goal is to minimize the avg. waiting time
  - Works both with preemptive and non-preemptive schedulers (preemptive SJF is called SRTF or Shortest Remaining Time First)

SJF: PROs and CONs

- PROs:
  - Provably optimal when the goal is to minimize the avg. waiting time
  - Works both with preemptive and non-preemptive schedulers (preemptive SJF is called SRTF or Shortest Remaining Time First)
- CONs:
  - Almost impossible to know the (next) CPU burst time of a job

SJF: PROs and CONs

- PROs:
  - Provably optimal when the goal is to minimize the avg. waiting time
  - Works both with preemptive and non-preemptive schedulers
    (preemptive SJF is called SRTF or Shortest Remaining Time First)
- CONs:
  - Almost impossible to know the (next) CPU burst time of a job
  - Long running CPU-bound jobs can *starve* (as I/O-bound ones have implicitly higher priority over them)

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)
- One simple, fast, and quite accurate method is the **exponential smoothing**

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)
- One simple, fast, and quite accurate method is the **exponential smoothing**

$$x_t = \textit{actual} \text{ length of the } t\text{-th CPU burst}$$

$$s_{t+1} = \textit{predicted} \text{ length of the } (t{+}1)\text{-th CPU burst}$$

$$\alpha \in \mathbb{R},\ 0 \leq \alpha \leq 1$$

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1 - \alpha) s_t$$

SJF: Estimating CPU Time of a Job

- Predict the length of the next CPU burst, based on some historical measurement of recent burst times (for this process)
- One simple, fast, and quite accurate method is the **exponential smoothing**

$$x_t = \textit{actual} \text{ length of the } t\text{-th CPU burst}$$

$$s_{t+1} = \textit{predicted} \text{ length of the } (t{+}1)\text{-th CPU burst}$$

$$\alpha \in \mathbb{R},\ 0 \leq \alpha \leq 1$$

$$\boxed{\begin{aligned} s_1 &= x_0 \\ s_{t+1} &= \alpha \boxed{x_t} + (1-\alpha)\boxed{s_t} \end{aligned}}$$

weighted average between previous observation and previous prediction

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1 - \alpha)s_t$$

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1 - \alpha) s_t$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \alpha \cancel{x}_t + \boxed{(1-\alpha)s_t}$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \alpha x_t + (1-\alpha)s_t$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Case 2: $\alpha = 1 \Rightarrow s_{t+1} = x_t$

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \boxed{\alpha x_t} \cancel{+ (1 - \alpha) s_t}$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Case 2: $\alpha = 1 \Rightarrow s_{t+1} = x_t$

The next burst is assumed to be the same as the last actual CPU burst observed

Exponential Smoothing

$$s_1 = x_0$$
$$s_{t+1} = \boxed{\alpha x_t} + \cancel{(1-\alpha)s_t}$$

Case 1: $\alpha = 0 \Rightarrow s_{t+1} = s_t$

Observed bursts are ignored and constant burst is assumed

Case 2: $\alpha = 1 \Rightarrow s_{t+1} = x_t$

The next burst is assumed to be the same as the last actual CPU burst observed

Recent history does not count

Exponential Smoothing

t	0	1	2	3	4	5	6	7	...

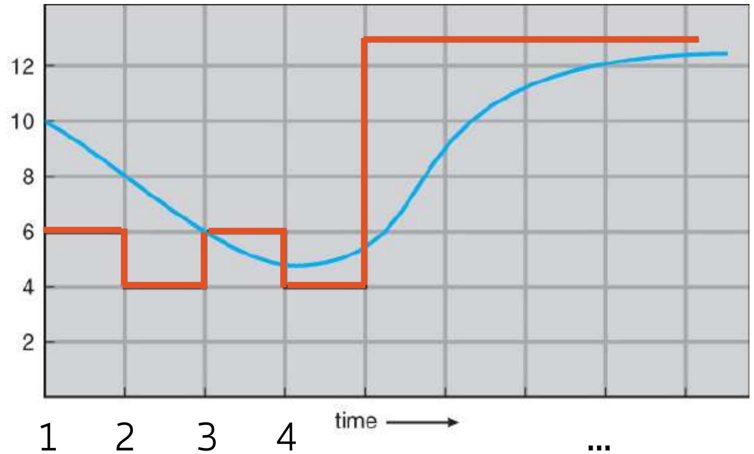
Exponential Smoothing

	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...

Exponential Smoothing

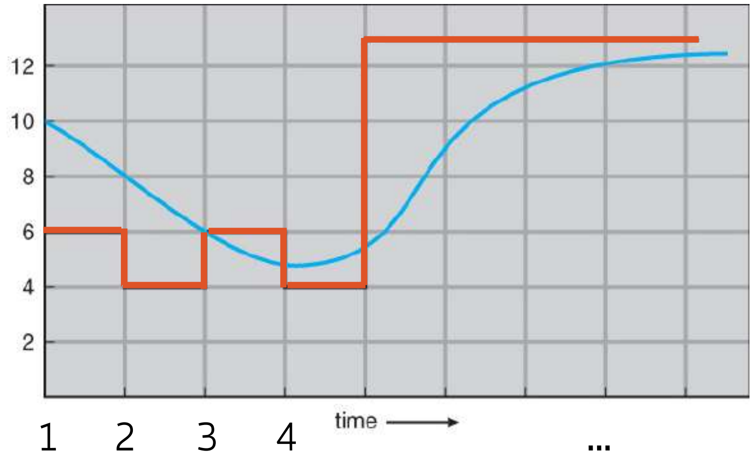
	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...
predictions	s_{t+1}	10	8	6	6	5	9	11	12	...

Exponential Smoothing



	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...
predictions	s_{t+1}	10	8	6	6	5	9	11	12	...

Exponential Smoothing

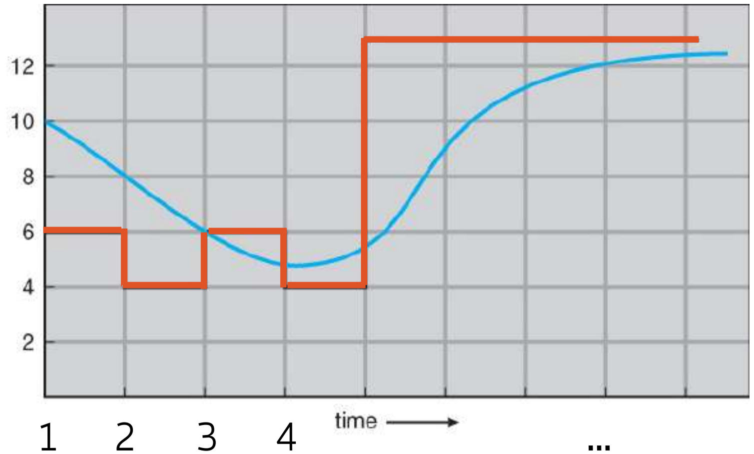


	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...
predictions	s_{t+1}	10	8	6	6	5	9	11	12	...

$s_1 = x_0$

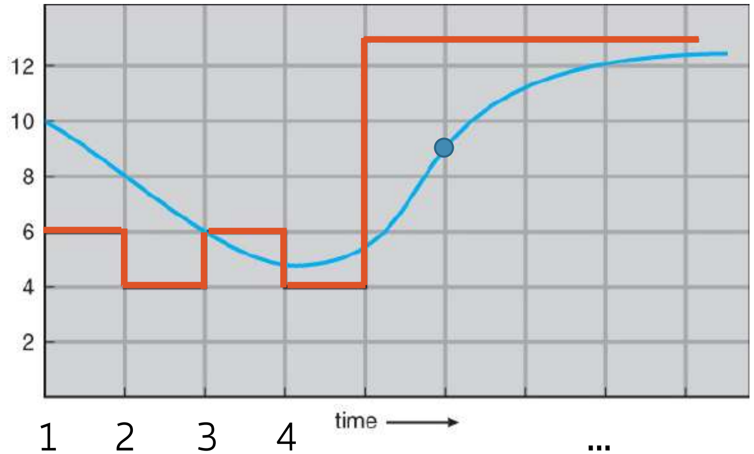
bootstrap

Exponential Smoothing



	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...
predictions	s_{t+1}	10	8	6	6	5	9	11	12	...

Exponential Smoothing

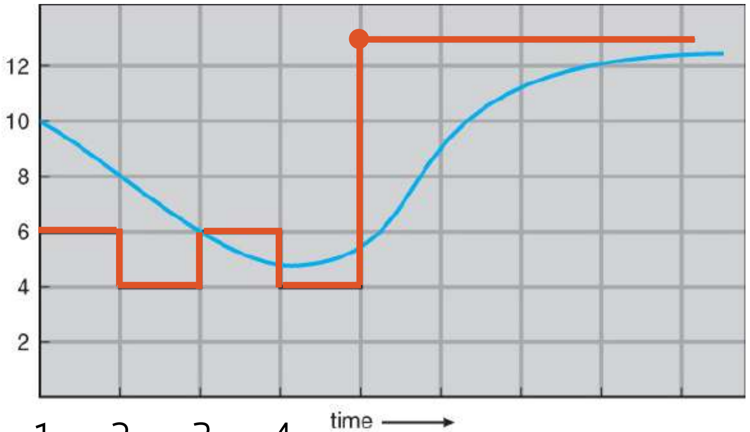


	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...
predictions	s_{t+1}	10	8	6	6	5	$\boxed{9}$	11	12	...

9 =

$s_6 =$

Exponential Smoothing



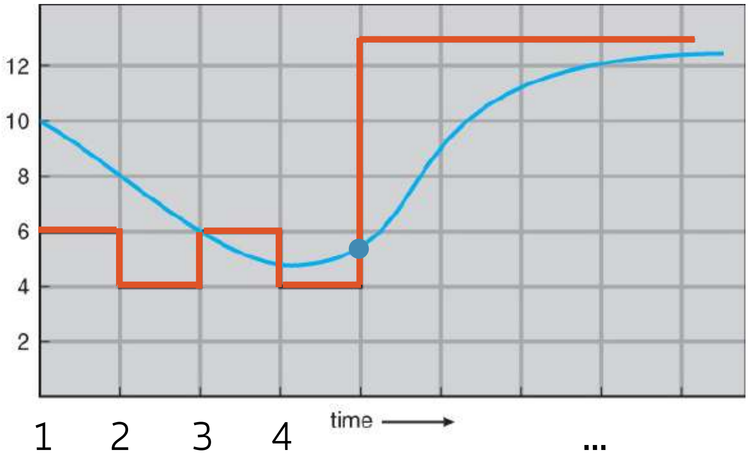
Usually, alpha is set to 0.5

	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...
predictions	s_{t+1}	10	8	6	6	5	9	11	12	...

9 = 0.5 * 13

$s_6 = \quad \alpha x_5$

Exponential Smoothing



Usually, alpha is set to 0.5

	t	0	1	2	3	4	5	6	7	...
observations	x_t	10	6	4	6	4	13	13	13	...
predictions	s_{t+1}	10	8	6	6	$\boxed{5}$	9	11	12	...

9 = 0.5 * 13 + 0.5 * 5

$$s_6 = \alpha x_5 + (1-\alpha) s_5$$

SJF vs. SRTF: Non-preemptive vs. Preemptive

- SJF (`non-preemptive`) → Once the CPU is given to a process this will execute until it completes its CPU burst

SJF vs. SRTF: Non-preemptive vs. Preemptive

- SJF (non-preemptive) → Once the CPU is given to a process this will execute until it completes its CPU burst
- SRTF (preemptive) → Preemption occurs whenever a new process arrives in the ready queue and its predicted CPU burst is shorter than the one remaining of the current executing process

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

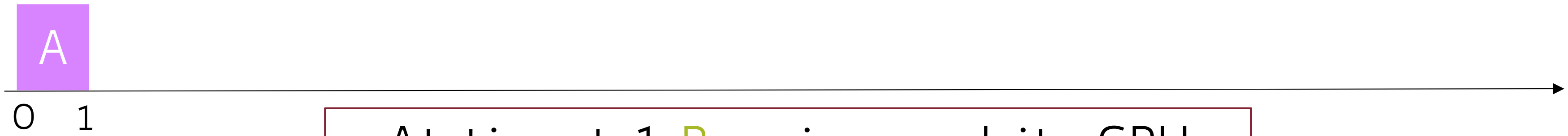
0

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5



At time t=1 B arrives and its CPU burst (4) is less than the remaining CPU burst of A (8-1=7)

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

B is scheduled and will execute until it finishes its 4 CPU burst units

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

Both C and D are arrived with 9 and 5 CPU burst units, respectively
A has still 7 CPU burst units left...

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

D is scheduled and will execute until it finishes its 5 CPU burst units
(no more jobs arrived in the meantime)

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

A has still 7 CPU burst units left and C has 9…

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

A is scheduled again until it finishes

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

Eventually, C is scheduled as well

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

avg. waiting time =

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

avg. waiting time = [(17-0-8)+]

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

avg. waiting time = $\left[(17-0-8)+(5-1-4)+ \qquad\qquad \right]$

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

avg. waiting time = $\left[(17-0-8)+(5-1-4)+(26-2-9)+ \qquad \right]$

SRTF: Example

Job	Arrival time	CPU burst (time units)
A	0	8
B	1	4
C	2	9
D	3	5

avg. waiting time = [(17-0-8)+(5-1-4)+(26-2-9)+(10-3-5)] /4 = 26/4 = 6.5

FCFS vs. RR vs. SJF

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time			waiting time		
Job	CPU burst	FCFS	RR	SJF	FCFS	RR	SJF
A	50	50	150		0	100	
B	40	90	140		50	100	
C	30	120	120		90	90	
D	20	140	90		120	70	
E	10	150	50		140	40	
Avg.		110	110		80	80	

FCFS vs. RR vs. SJF

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time			waiting time		
Job	CPU burst	FCFS	RR	SJF	FCFS	RR	SJF
A	50	50	150	150	0	100	
B	40	90	140	100	50	100	
C	30	120	120	60	90	90	
D	20	140	90	30	120	70	
E	10	150	50	10	140	40	
Avg.		110	110	70	80	80	

FCFS vs. RR vs. SJF

Assumptions:

5 jobs, different CPU burst

Time quantum = 1

Context switch = 0

Arrival time = 0 (for all jobs)

		turnaround time			waiting time		
Job	CPU burst	FCFS	RR	SJF	FCFS	RR	SJF
A	50	50	150	150	0	100	100
B	40	90	140	100	50	100	60
C	30	120	120	60	90	90	30
D	20	140	90	30	120	70	10
E	10	150	50	10	140	40	0
Avg.		110	110	70	80	80	40

Scheduling Algorithms: An Overview

- First-Come-First-Serve (FCFS)
- Round Robin (RR)
- Shortest-Job-First (SJF)
- **Priority Scheduling**
- Multilevel Queue (MLQ)
- Multilevel Feedback-Queue (MLFQ)

Priority Scheduling: Idea

- More general case of SJF, where each job is assigned a **priority** and the job with the highest priority gets scheduled first

Priority Scheduling: Idea

- More general case of SJF, where each job is assigned a **priority** and the job with the highest priority gets scheduled first
- SJF is a priority scheduling where priority is the predicted next CPU burst time

Priority Scheduling: Idea

- More general case of SJF, where each job is assigned a **priority** and the job with the highest priority gets scheduled first
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- In practice, priorities are implemented using integers within a fixed range
  - No convention on whether "high" priorities use large or small numbers
  - Usually, low numbers for high priorities (0 = the highest priority)

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**
- **Internal** priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, etc.

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**
- **Internal** priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, etc.
- **External** priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.

Priority Scheduling: Characteristics

- Priorities can be assigned either **internally** or **externally**
- **Internal** priorities are assigned by the OS using criteria such as average burst time, ratio of CPU to I/O activity, system resource use, etc.
- **External** priorities are assigned by users, based on the importance of the job, fees paid, politics, etc.
- Priority scheduling can be either **preemptive** or **non-preemptive**

Priority Scheduling: Issues

- **Indefinite blocking** (or **starvation**): a low-priority task can wait forever because some other jobs have always higher priority

Priority Scheduling: Issues

- **Indefinite blocking** (or **starvation**): a low-priority task can wait forever because some other jobs have always higher priority
- Stuck jobs may eventually run when the system load is lighter or after a shutdown/crash and a reboot

Priority Scheduling: Issues

- **Indefinite blocking** (or **starvation**): a low-priority task can wait forever because some other jobs have always higher priority
- Stuck jobs may eventually run when the system load is lighter or after a shutdown/crash and a reboot
- **Aging →** solves starvation by increasing the priority of jobs proportionally to the time they wait, until they are eventually scheduled