

CSE 120: Principles of Operating Systems

Lecture 4: Scheduling

Prof. Joseph Pasquale
University of California, San Diego
January 21, 2019

The CPU Scheduling Problem

- Scheduling: which process gets CPU and when
- Given multiple processes, but only one CPU
- How much CPU time does each process get?
- Possibilities
 - Keep CPU till done
 - Each process uses CPU a bit and passes it on
 - Each process gets proportional to what they pay
- Which is the best policy?

There is No Single Best Policy

- Depends on the goals of the system
- Different for
 - your personal computer
 - large time-shared computer
 - computer controlling a nuclear power plant
- Might even have multiple (conflicting) goals

Scheduling: An Example

Process	Arrival Time	Service Time
A	0	5
B	0	3
C	0	1

- Arrival time: time that process is created
- Service time: CPU time needed to complete
- Turnaround time: from arrival to departure
 - Process arrives, waits for CPU, then uses (in bursts)
 - Departs after CPU usage equals service time
- What order minimizes average turnaround time?

Longest First vs. Shortest First

Longest First											
	0	1	2	3	4	5	6	7	8	9	TT
A											5
B											8
C											9
Average Turnaround Time											22/3

Shortest First											
	0	1	2	3	4	5	6	7	8	9	TT
A											9
B											4
C											1
Average Turnaround Time											14/3

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) +$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) +$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) +$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) + ((n-1) \times S_2) +$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) + ((n-1) \times S_2) +$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + \textcolor{red}{S}_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) + ((n-1) \times S_2) +$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) + ((n-1) \times S_2) + ((n-2) \times S_3) + \dots]$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) + ((n-1) \times S_2) + ((n-2) \times S_3) + \dots + S_n] / n$

Shortest First Is Provably Optimal

- Given n processes with service times S_1, \dots, S_n
 - Note: processes are numbered $1, 2, 3, \dots, n$
- Average turnaround time computed as follows
 - $[S_1 + (S_1 + S_2) + (S_1 + S_2 + S_3) + \dots + (S_1 + \dots + S_n)] / n$
 - $[(n \times S_1) + ((n-1) \times S_2) + ((n-2) \times S_3) + \dots + S_n] / n$
- In general: order by shortest to longest
 - S_1 has maximum weight (n), minimize it
 - S_2 has next-highest ($n-1$), minimize it after S_1 ; ...

Consider Different Arrival Times

Process	Arrival Time	Service Time
A	0	5
B	1	3
C	2	1



FCFS: First Come First Served

	0	1	2	3	4	5	6	7	8	9	TT
A											5
B											7
C											7
Average Turnaround Time											19/3

- Allocate CPU to processes in order of arrival
- A B]A B]A CB]A CB]A CB]A C]B C]B C]B C
 - A At 0, A arrives and runs at 0
 - B]A At 1-, B arrives, enters queue (A still running)
 - B]A At 1, B is in the queue, A is running

FCFS: First Come First Served

	0	1	2	3	4	5	6	7	8	9	TT
A											5
B											7
C											7
Average Turnaround Time											19/3

- Allocate CPU to processes in order of arrival
- A B]A B]A CB]A CB]A CB]A C]B C]B C]B C
- Average turnaround time = $(5 + 7 + 7)/3 = 6.3$
- Non-preemptive, simple, no starvation
- Poor for short processes

RR: Round Robin

	0	1	2	3	4	5	6	7	8	TT
A										9
B										6
C										2
Average Turnaround Time										17/3

- Time-slice: each process gets quantum in turn
- A B]A A]B CA]B BC]A AB]C A]B B]A A]B A A
- Average turnaround time = $(9 + 6 + 2)/3 = 5.7$
- Preemptive, simple, no starvation
- Process waits at most $(n - 1) \times \text{quantum}$

SPN: Shortest Process Next

	0	1	2	3	4	5	6	7	8	9	TT
A											5
B											8
C											4
Average Turnaround Time											17/3

- Select process with shortest service time
- A B]A B]A BC]A BC]A BC]A B]C B B B
- Average turnaround time = $(5 + 8 + 4)/3 = 5.7$
- Optimal for non-preemptive, allows starvation
- Assumes service times are known

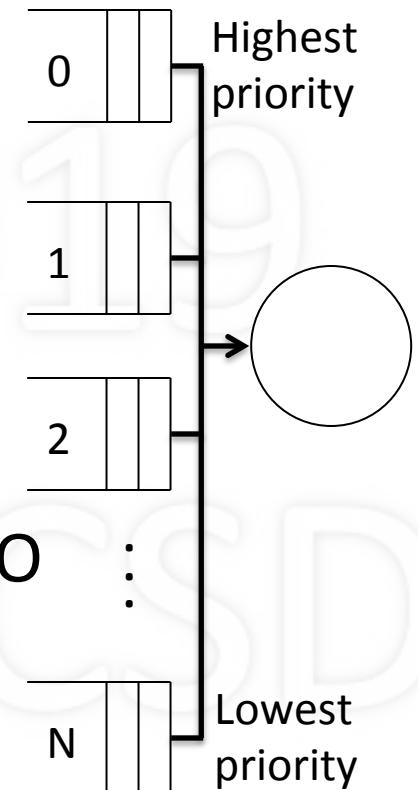
SRT: Shortest Remaining Time

	0	1	2	3	4	5	6	7	8	9	TT
A											9
B											4
C											1
Average Turnaround Time											14/3

- Select process with shortest remaining time
- A B]A A]B AC]B AB]C A]B A]B A A A A
- Average turnaround time = $(9 + 4 + 1)/3 = 4.7$
- Assumes service times are known
- Optimal for preemptive, but allows starvation

Multi-Level Feedback Queues

- Priority queues: 0 (high), ..., N (low)
- New processes enter queue 0
- Select from highest priority queue
- Run for $T = 2^k$ quantums
 - Used T : move to next lower queue, FIFO
 - Used $< T$: back to same queue, RR
 - Due to yield or higher priority arrival
- Periodically boost (e.g., all to highest queue)

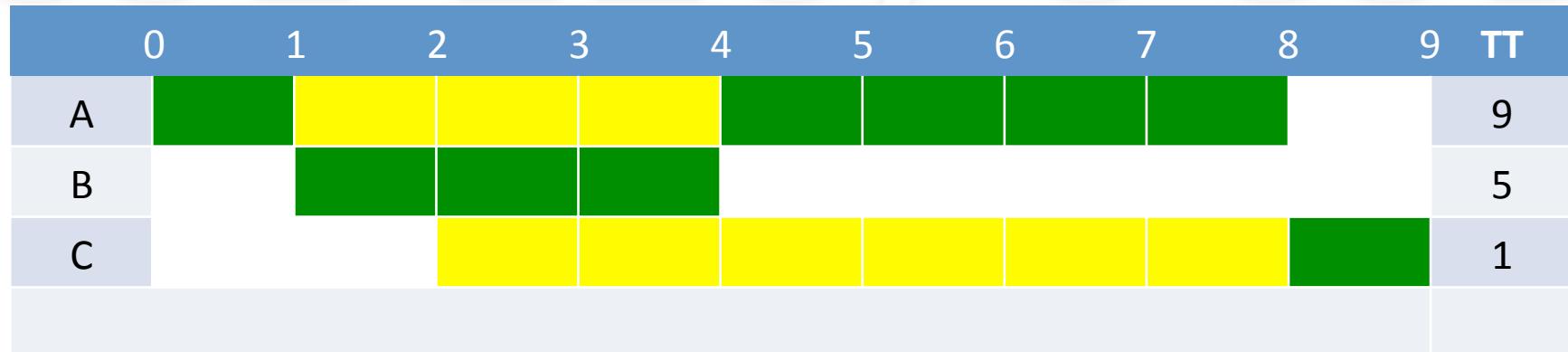


Multi-Level Feedback Queues

	0	1	2	3	4	5	6	7	8	9	TT
A											9
B											6
C											1
Average Turnaround Time											16/3

- Select from highest queue k , run 2^k quantums
- A B]₀A A]₁B A]₁C]₀B BA]₁C B]₁A(2) A]₂B(2) A
- Average Turnaround Time = $(9 + 6 + 1)/3 = 5.3$
- Complex, adaptive, highly responsive
- Favors shorter over longer, possible starvation

Priority Scheduling



- Select process with highest priority
 - Example: $P_A = \text{medium}$, $P_B = \text{high}$, $P_C = \text{low}$
- $A_M \rightarrow B_H \rightarrow A_M \rightarrow A_M \rightarrow B_H \rightarrow C_L \rightarrow A_M \rightarrow B_H \rightarrow C_L \rightarrow A_M \rightarrow (2) \rightarrow C_L \rightarrow A_M \rightarrow (4) \rightarrow C$
- Allows scheduling based on external criteria
 - E.g., priority = $1/\text{CPU_time_used}$, or = $f(\text{pay})$

Fair Share (Proportional Share)

	1	2	3	4	5	6	7	8	9	10
A	100%	50%	33%	50%	40%	50%	43%	50%	44%	50%
B	0%	50%	33%	25%	20%	17%	14%	13%	11%	10%
C	0%	0%	33%	25%	40%	33%	43%	38%	44%	40%

- Each process requests some CPU utilization
 - Utilization: what percentage of time resource is used
 - Example of requests: A: 50%; B: 10%; C: 40%
- Goal: utilization over long run, actual \approx request
- How do we determine who runs each quantum?
- Select process with minimum actual/request ratio

FS Example: A 50%, B 10%, C 40%

	1	2	3	4	5	6	7	8	9	10
A										
B										
C										

- At time 0 A: 0/50 B: 0/10 C: 0/40
- All 0, so arbitrary select A

FS Example: A 50%, B 10%, C 40%

	1	2	3	4	5	6	7	8	9	10
A	100%									
B	0%									
C	0%									

- After 1q A: $100/50$ B: $0/10$ C: $0/40$
- A got 100%, but only requested 50% ($100/50$)
 - Got *more* than its fair share ($100/50 = 2 > 1$)!
- Both B, C got 0%, less than their fair shares
- Select process with min actual/request ratio: B

FS Example: A 50%, B 10%, C 40%

	1	2	3	4	5	6	7	8	9	10
A	100%	50%								
B	0%	50%								
C	0%	0%								

- After 2q A: 50/50 B: 50/10 C: 0/40
- A got 50%: used 1 out of 2q; same for B
 - A got *exactly* its fair share ($50/50 = 1$)
 - B got *more* than its fair share ($50/10 = 5 > 1$)
- Select process with min actual/request ratio: C

FS Example: A 50%, B 10%, C 40%

	1	2	3	4	5	6	7	8	9	10
A	100%	50%	33%							
B	0%	50%	33%							
C	0%	0%	33%							

- After 3q A: $33/50$ B: $33/10$ C: $33/40$
- A got 33% as it used 1 out of 3; same for B, C
 - A, C got *less* than their fair share ($33/50, 33/40 < 1$)
 - B got *more* than its fair share ($33/10 > 1$)
- Select process with min actual/request ratio: A

FS Example: A 50%, B 10%, C 40%

	1	2	3	4	5	6	7	8	9	10
A	100%	50%	33%	50%						
B	0%	50%	33%	25%						
C	0%	0%	33%	25%						

- After 4q A: 50/50 B: 25/10 C: 25/40
- A got 50% as it used 2 out of 4
- B and C got 25%, using 1 out of 4
- A got its fair share; B got more; C got less
- Select process with min actual/request ratio: C

FS Example: A 50%, B 10%, C 40%

	1	2	3	4	5	6	7	8	9	10
A	100%	50%	33%	50%	40%	50%	43%	50%	44%	50%
B	0%	50%	33%	25%	20%	17%	14%	13%	11%	10%
C	0%	0%	33%	25%	40%	33%	43%	38%	44%	40%

- After 10q A: 50/50 B: 10/10 C: 40/40
- Recall goal: over long run, actual \approx request
 - Process utilizations are what they requested!
- However, algorithm shown is inefficient
 - Each q, each p: calculate actual/request, select min

Stride Scheduling

- For processes A, B, C ... with requests $R_A, R_B, R_C \dots$
- Calculate ***strides***: $S_A = 1/R_A, S_B = 1/R_B, S_C = 1/R_C \dots$
- For each process x , maintain ***pass*** value P_x (init 0)
- Schedule: repeat every quantum
 - Select process x with minimum pass value P_x , run
 - Increment pass value by stride value: $P_x = P_x + S_x$
- Optimization: use only integers for R_x, S_x and P_x
 - Calculate $S_x = L/R_x$ using very large L , e.g., $L = 100000$

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		0	0	0	A
quantum 1					
quantum 2					
quantum 3					
quantum 4					
quantum 5					
quantum 6					
quantum 7					
quantum 8					
quantum 9					
quantum 10					

- All pass values 0, select one arbitrarily: A

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		0	0	0	A
quantum 1	A	2000	0	0	B
quantum 2					
quantum 3					
quantum 4					
quantum 5					
quantum 6					
quantum 7					
quantum 8					
quantum 9					
quantum 10					

- $P_A = 2000$, P_B and $P_C = 0$; select B (or C)

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		2000	0	0	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3					
quantum 4					
quantum 5					
quantum 6					
quantum 7					
quantum 8					
quantum 9					
quantum 10					

- $P_A = 2000, P_B = 10000, P_C = 0$; select C

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		2000	10000	0	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4					
quantum 5					
quantum 6					
quantum 7					
quantum 8					
quantum 9					
quantum 10					

- $P_A = 2000, P_B = 10000, P_C = 2500$; select A

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		2000	10000	2500	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5					
quantum 6					
quantum 7					
quantum 8					
quantum 9					
quantum 10					

- $P_A = 4000, P_B = 10000, P_C = 2500$; select C

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		4000	10000	2500	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5	C	4000	10000	5000	A
quantum 6					
quantum 7					
quantum 8					
quantum 9					
quantum 10					

- $P_A = 4000, P_B = 10000, P_C = 5000$; select A

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		4000	10000	5000	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5	C	4000	10000	5000	A
quantum 6	A	6000	10000	5000	C
quantum 7					
quantum 8					
quantum 9					
quantum 10					

- $P_A = 6000, P_B = 10000, P_C = 5000$; select C

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		6000	10000	5000	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5	C	4000	10000	5000	A
quantum 6	A	6000	10000	5000	C
quantum 7	C	6000	10000	7500	A
quantum 8					
quantum 9					
quantum 10					

- $P_A = 6000, P_B = 10000, P_C = 7500$; select A

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		6000	10000	7500	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5	C	4000	10000	5000	A
quantum 6	A	6000	10000	5000	C
quantum 7	C	6000	10000	7500	A
quantum 8	A	8000	10000	7500	C
quantum 9					
quantum 10					

- $P_A = 8000, P_B = 10000, P_C = 7500$; select C

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		8000	10000	7500	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5	C	4000	10000	5000	A
quantum 6	A	6000	10000	5000	C
quantum 7	C	6000	10000	7500	A
quantum 8	A	8000	10000	7500	C
quantum 9	C	8000	10000	10000	A
quantum 10					

- $P_A = 8000, P_B = 10000, P_C = 10000$; select A

Stride Scheduling Example

Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		8000	10000	10000	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5	C	4000	10000	5000	A
quantum 6	A	6000	10000	5000	C
quantum 7	C	6000	10000	7500	A
quantum 8	A	8000	10000	7500	C
quantum 9	C	8000	10000	10000	A
quantum 10	A	10000	10000	10000	... will repeat

- $P_A = 10000, P_B = 10000, P_C = 10000$

Stride Scheduling Example

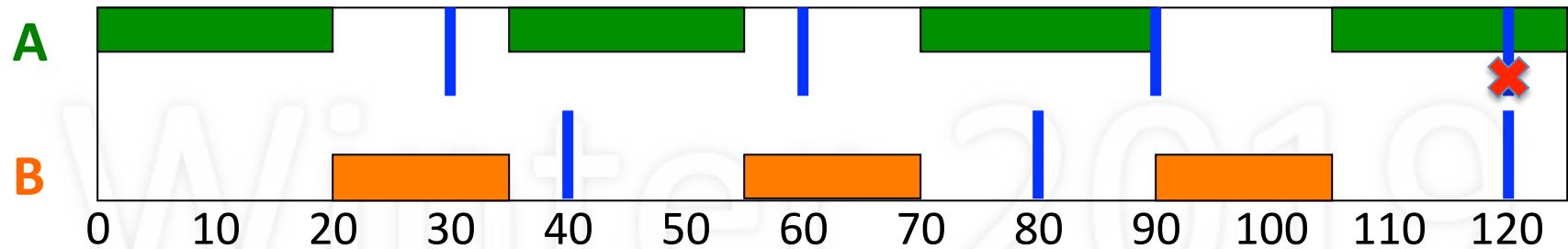
Process x	Runs Now	A	B	C	To Run Next
Requested Utilization: R_x (in %)		50	10	40	
Stride: $S_x = L/R_x$ ($L = 100000$)		2000	10000	2500	
Pass: $P_x = P_x + S_x$; init = 0		10000	10000	10000	A
quantum 1	A	2000	0	0	B
quantum 2	B	2000	10000	0	C
quantum 3	C	2000	10000	2500	A
quantum 4	A	4000	10000	2500	C
quantum 5	C	4000	10000	5000	A
quantum 6	A	6000	10000	5000	C
quantum 7	C	6000	10000	7500	A
quantum 8	A	8000	10000	7500	C
quantum 9	C	8000	10000	10000	A
quantum 10	A	10000	10000	10000	... will repeat

- After 10 quantums, A ran 5; B ran 1; C ran 4

Real Time Scheduling

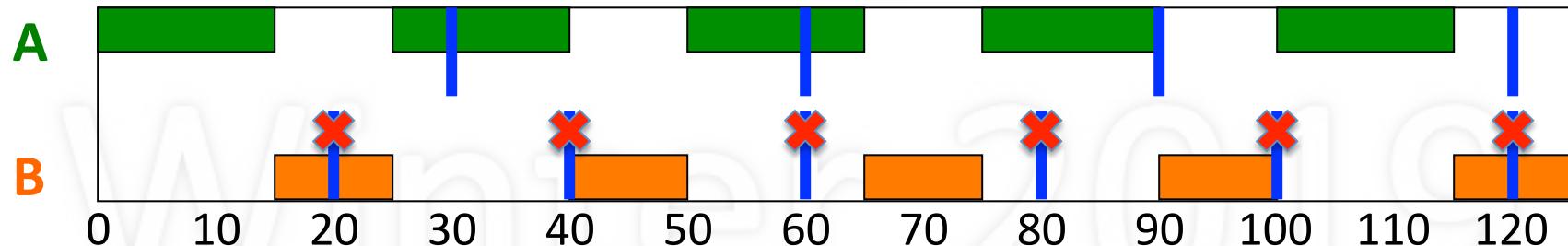
- Correctness of real-time systems depend on
 - logical result of computations
 - *and* the timing of these results
- Type of real-time systems
 - Hard vs. soft real-time
 - Periodic vs. aperiodic
- Scheduling
 - Earliest Deadline First (EDF)
 - Rate Monotonic Scheduling (RMS)

Periodic Processes (or Tasks)



- Periodic processes: computation is cyclic
- For each process, given
 C = CPU burst, T = period, $U = C/T$ = utilization
- Can processes be ordered so deadlines are met?
- Consider orders: ABABAB..., vs. BABABA...

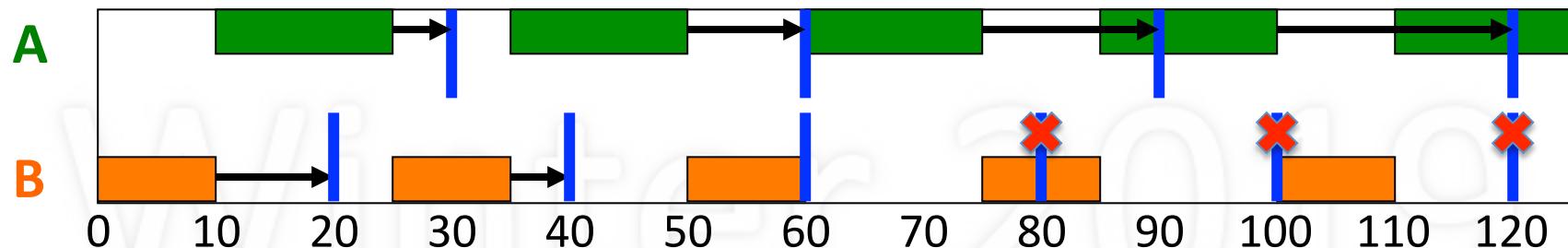
Periodic Processes



	C	T	U
A	15	30	50%
B	10	20	50%

- Sum of utilizations does not exceed 100%
- For order ABABAB..., B misses all deadlines!

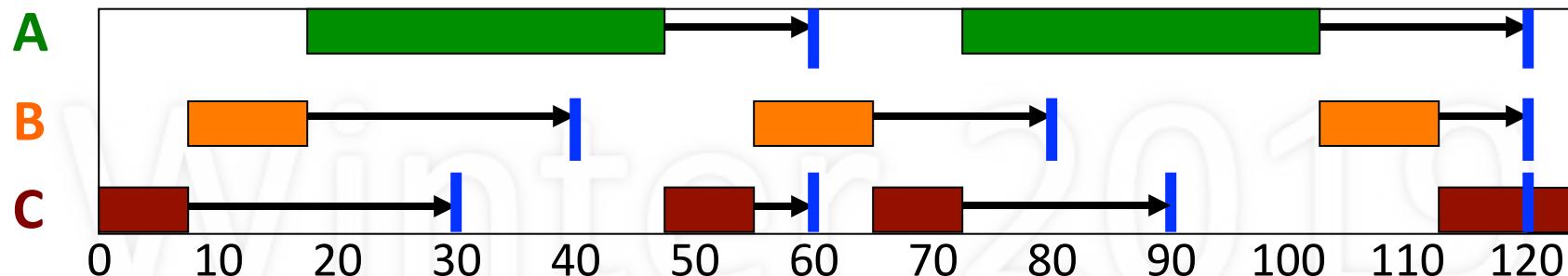
Periodic Processes



	<i>C</i>	<i>T</i>	<i>U</i>
A	15	30	50%
B	10	20	50%

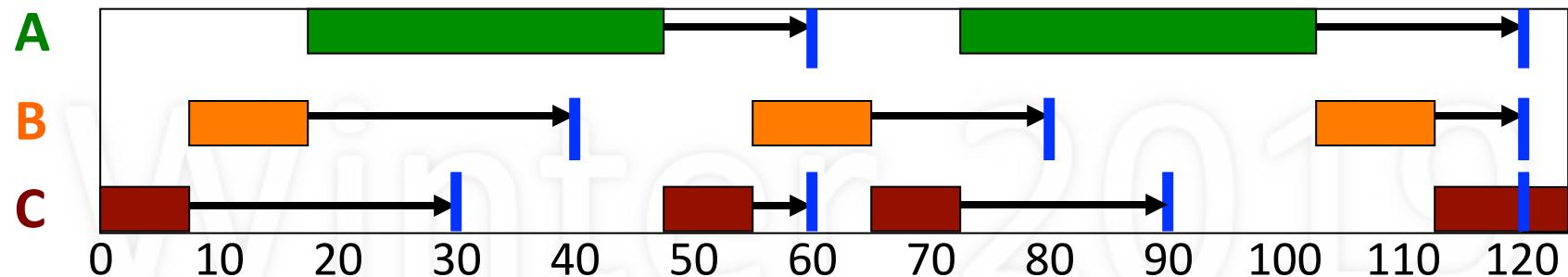
- Sum of utilizations does not exceed 100%
- For order BABABA..., B misses some deadlines

EDF: Earliest Deadline First



- Schedule process with earliest deadline
- If earlier deadline process appears, preempt
- Works for periodic and aperiodic processes
- Achieves 100% utilization (ignoring overhead!)
- Expensive: requires ordering by deadlines

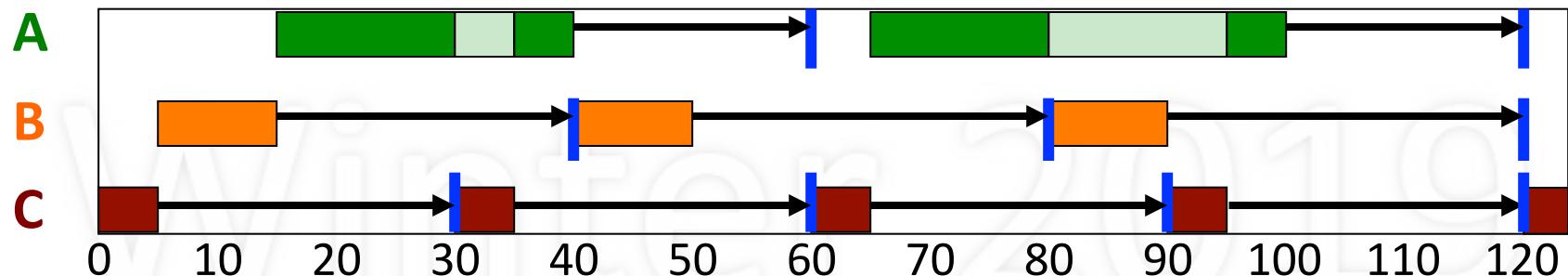
EDF: Earliest Deadline First



	<i>C</i>	<i>T</i>	<i>U</i>
A	30	60	50%
B	10	40	25%
C	7.5	30	25%

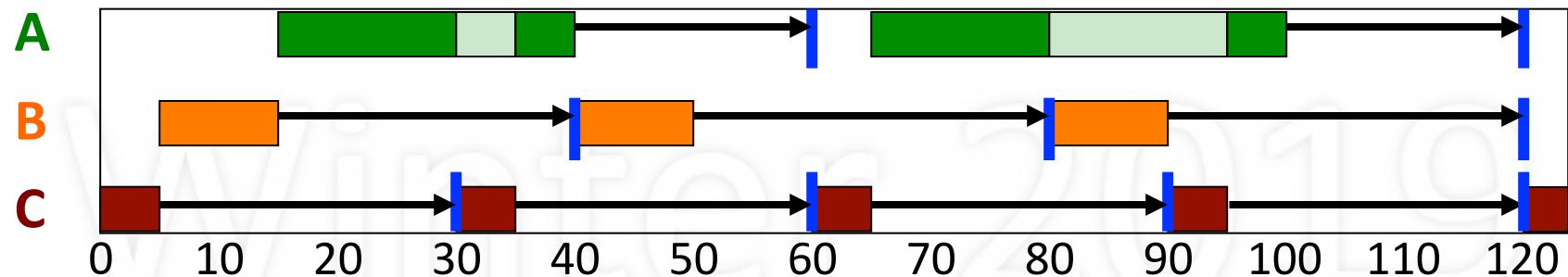
- Meets all deadlines, sum of utilizations = 100%

RMS: Rate Monotonic Scheduling



- If periodic processes, prioritize based on rates
- At start of period, select highest priority
- Preempt if necessary
- When burst done, wait till next period
- If $U_1 + \dots + U_n \leq n (2^{1/n} - 1)$, all deadlines met

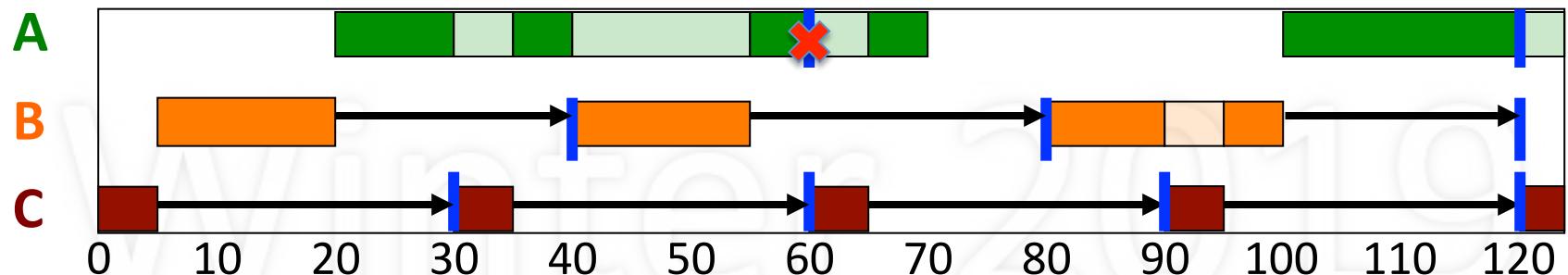
RMS Test Passes, All Deadlines Met



	<i>C</i>	<i>T</i>	<i>U</i>	Rate	Prio
A	20	60	33%	$1/60 = 0.017$	Low
B	10	40	25%	$1/40 = 0.025$	Med
C	5	30	17%	$1/30 = 0.033$	High

- Passes: $U_A + U_B + U_C = 75\% \leq 3(2^{1/3} - 1) = 78\%$

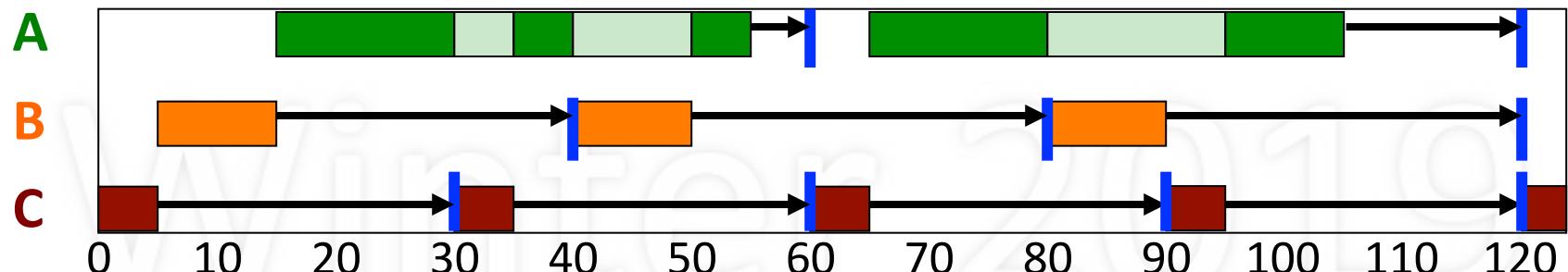
RMS Test Fails, Deadline Missed



	<i>C</i>	<i>T</i>	<i>U</i>	Rate	Prio
A	25	60	42%	$1/60 = 0.017$	Low
B	15	40	38%	$1/40 = 0.025$	Med
C	5	30	17%	$1/30 = 0.033$	High

- Fails: $U_A + U_B + U_C = 97\% > 3(2^{1/3} - 1) = 78\%$

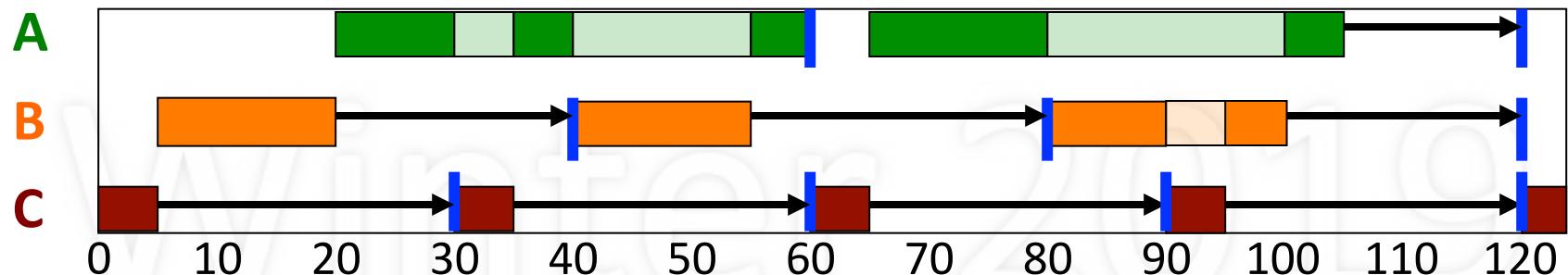
RMS Test Fails, But Deadlines Met



	<i>C</i>	<i>T</i>	<i>U</i>	Rate	Prio
A	25	60	42%	$1/60 = 0.017$	Low
B	10	40	25%	$1/40 = 0.025$	Med
C	5	30	17%	$1/30 = 0.033$	High

- Fails: $U_A + U_B + U_C = 84\% > 3(2^{1/3} - 1) = 78\%$

RMS Test Fails, But Deadlines Met



	<i>C</i>	<i>T</i>	<i>U</i>	Rate	Prio
A	20	60	33%	$1/60 = 0.017$	Low
B	15	40	38%	$1/40 = 0.025$	Med
C	5	30	17%	$1/30 = 0.033$	High

- Fails: $U_A + U_B + U_C = 88\% > 3(2^{1/3} - 1) = 78\%$

RMS Optimal But Limited

- RMS is simple and efficient
 - Static priority scheduling based on rates
- RMS is optimal for static priority algorithms
 - If RMS can't schedule, no other static priority can
- RMS is limited in what it guarantees
 - Utilization bounded by $n (2^{1/n} - 1) > \ln 2 \approx 69\%$
 - Deadline guarantee applies only if test passes
- RMS is limited to periodic processes

Summary

- CPU scheduling is policy: depends on goals
 - First come first served very simple, non-preemptive
 - Round robin simple, preemptive
 - Shortest process next theoretical, non-preemptive
 - Shortest remaining time theoretical, preemptive
 - Multi-level feedback adaptive, responsive, complex
 - Priority external criteria
 - Fair share proportional allocation
 - Earliest deadline first 100% utilization, high overhead
 - Rate monotonic sched < 100% util, low overhead

Textbook

- Chapter 6 (CPU Scheduling)
 - Lecture-related: 5.1-5.3, 5.6, 5.9
 - Recommended: 5.7-5.8

Review & Research

- What is the CPU scheduling problem?
- What are various possibilities for how much CPU a process should get when it runs?
- Is there a best scheduling policy?
- What are various goals that might affect the scheduling policy, and for each goal, what would the effect be?*
- What are two goals that conflict, and in what way would they conflict?*

R&R

- What is meant by “arrival time”?
- What is meant by “service time”?
- What is meant by “turnaround time”?
- What is a CPU “burst”?*
- Can the service time ever be less than the turnaround time, and if so, why?*
- Can the turnaround time ever be less than the service time, and if so, why?*

R&R

- How is average turnaround time computed?
- If all processes arrive at the same time but have different service times, what order minimizes average turnaround time?
- Is there an order that *maximizes* average turnaround time, and if so, what is it?*
- Which is better: minimizing or maximizing the average turnaround time?

R&R

- Given 3 processes A, B, and C, that arrive at the same time, and have service times x , y , and z , respectively, can you provide an intuitive proof as to why they should run in the order A, B, and C, if $x < y < z$ in order to minimize average turnaround time?*
- If processes arrive at *different* times, how does this affect which order is best for minimizing turnaround time?**

R&R

- What is meant by a “non-preemptive” scheduling policy?*
- What is meant by a “preemptive” scheduling policy?*
- What are the pros and cons of each?**
- What mechanism does a preemptive scheduler depend on?**
- What is meant by “starvation”?*

R&R

- What is FCFS?
- Is it preemptive or non-preemptive and why?*
- What are the pros and cons of FCFS?
- Are there certain processes that especially suffer under FCFS, and why?*
- Are there certain processes that especially benefit under FCFS, and why?**

R&R

- What is RR?
- Is it preemptive or non-preemptive and why?*
- What hardware mechanism is needed for RR?
- What are the pros and cons of RR?
- Are there certain processes that especially suffer under RR, and why?*
- Are there certain processes that especially benefit under RR, and why?**

R&R

- What is SPN?
- Is it preemptive or non-preemptive and why?*
- What are the pros and cons of SPN?
- Are there certain processes that especially suffer under SPN, and why?*
- Are there certain processes that especially benefit under SPN, and why?**
- How might starvation occur with SPN?**

R&R

- What is SRT?
- Is it preemptive or non-preemptive and why?*
- What are the pros and cons of SRT?
- Are there certain processes that especially suffer under SRT, and why?*
- Are there certain processes that especially benefit under SRT, and why?**
- How might starvation occur with SRT?**

R&R

- What is MLFQ?
- Is it preemptive or non-preemptive and why?*
- What are the pros and cons of MLFQ?
- Are there certain processes that especially suffer under MLFQ, and why?*
- Are there certain processes that especially benefit under MLFQ, and why?**
- How might starvation occur with MLFQ?**

R&R

- Under MLFQ, say that there are 3 queues, and each is occupied by a single process: what would be their order of execution?**
- Under the same scenario, say a new process arrives at time 2- (i.e., a moment before 2); how does this affect the schedule, and what would be the average turnaround time?**

R&R

- Under the same scenario, say a new process arrives at time 5-; how does this affect the schedule, and what would be the average turnaround time?**
- Under the same scenario, say a new process arrives at time 2- and another arrives at 5-; how does this affect the schedule, and what would be the average turnaround time?**

R&R

- Under MLFQ, what is the rationale for periodically boosting all process priorities?**
- Under MLFQ, what is the rationale for increasing the number of queues?***
- Is having more queues always better, why or why not?***

R&R

- What is Priority Scheduling (PS)?
- Is it preemptive or non-preemptive and why?*
- What are the pros and cons of PS?
- Are there certain processes that especially suffer under PS, and why?*
- Are there certain processes that especially benefit under PS, and why?**
- How might starvation occur with PS?**

R&R

- Say that Priority Scheduling is implemented such that priority is set to $1/\text{CPU_time_used}$: what goal does this scheduling policy try to achieve?***
- Say that Priority Scheduling is implemented such that priority is set to $\text{time_waiting_for_CPU}$: what goal does this scheduling policy try to achieve?***

R&R

- What is Fair Share Scheduling (FSS)?
- Is it preemptive or non-preemptive and why?*
- What are the pros and cons of FSS?
- Are there certain processes that especially suffer under FSS, and why?*
- Are there certain processes that especially benefit under FSS, and why?**
- How might starvation occur with FSS?**

R&R

- What is meant by “utilization”?
- What is the goal of FSS in terms of utilization?
- When are scheduling decisions made in FSS?*
- How is a process selected for FSS?**
- On slide 27, can you explain how each and every number is determined in the chart?**
- On slide 33, how do we know that FSS achieved its goal?**

R&R

- What is the Stride Scheduling (SS) algorithm used for?**
- What are the inputs to SS?
- What is meant by a “stride” value?*
- What is meant by a “pass” value?*
- How often is a scheduling decision made in SS?**
- What calculation is the decision based on?*

R&R

- What is the purpose of the value “L”?**
- Why is it important to make L large?***
- Conversely, what might happen if L were too small?***
- In slide 46, can you explain each and every step, and how the numbers are calculated?**
- In the same slide, did SS achieve its goal, and why?**

R&R

- What is Real-Time Scheduling?**
- How does real-time scheduling differ from the previous scheduling algorithms?**
- What are the various types of real-time scheduling algorithms, and can you give an example for each one?**
- What are the pros and cons of each type of real-time scheduling algorithm?***

R&R

- What is a “periodic process”?*
- What is meant by a “CPU burst”?
- What is meant by a “period”?
- How is the utilization calculated, and why is it calculated this way (why use those parameters, and why in a ratio)?**

R&R

- In slide 49, are all the deadlines met, and if not, why?**
- In slide 50, are all the deadlines met, and if not, why?**
- What is the point of having shown the schedules in slide 49 and slide 50?***

R&R

- What is EDF?
- How does EDF work?**
- What are the pros and cons of EDF?*
- In slide 51, can you explain the schedule shown, and for each scheduling decision, why the particular choice of process was made?**

R&R

- How does RMS work?**
- What are the pros and cons of RMS?*
- In slide 53, can you explain the schedule shown, and for each scheduling decision, why the particular choice of process was made?**
- How does RMS compare to EDF; in what ways is it better, and in what ways is it worse?**

R&R

- How is the inequality at the bottom of slide 53 used?**
- In slide 54, can you explain the schedule shown, and for each scheduling decision, why the particular choice of process was made?**
- What is the point of slide 54?**

R&R

- In slide 55, can you explain the schedule shown, and for each scheduling decision, why the particular choice of process was made?**
- What is the point of slide 55?**
- In slide 56, can you explain the schedule shown, and for each scheduling decision, why the particular choice of process was made?**
- What is the point of slide 56?**

R&R

- In slide 57, can you explain the schedule shown, and for each scheduling decision, why the particular choice of process was made?**
- What is the point of slide 57?**

R&R

- What is meant by “static priority scheduling”?

- Is RMS a static priority scheduling algorithm, and if so why, or why not?***
- Is EDF a static priority scheduling algorithm, and if so why, or why not?***
- Is RMS an optimal static priority scheduling algorithm, and if so why, or why not?***

R&R

- On slide 58, what does it mean that in RMS, utilization is bounded by $\ln 2 \approx 69\%?$ **
- Under RMS, if the RMS “test” passes (or RMS constraint is met), does this guarantee all deadlines will be met?**
- If the RMS “test” fails (or RMS constraint is not met), does this guarantee all deadlines will not be met?**