

Last Class: Processes

- A process is the unit of execution.
- Processes are represented as Process Control Blocks in the OS
 - PCBs contain process state, scheduling and memory management information, etc
- A process is either New, Ready, Waiting, Running, or Terminated.
- On a uniprocessor, there is at most one running process at a time.
- The program currently executing on the CPU is changed by performing a *context switch*
- Processes communicate either with message passing or shared memory



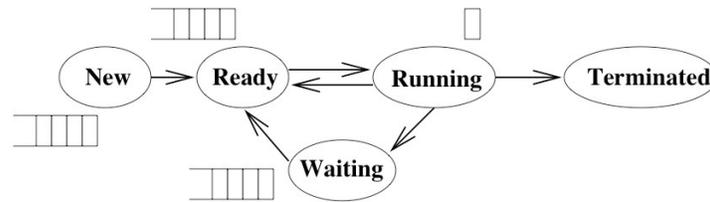
Today: Scheduling Algorithms

- Goals for scheduling
- FCFS & Round Robin
- SJF
- Multilevel Feedback Queues
- Lottery Scheduling



Scheduling Processes

- **Multiprogramming:** running more than one process at a time enables the OS to increase system utilization and throughput by overlapping I/O and CPU activities.
- Process Execution State



- All of the processes that the OS is currently managing reside in one and only one of these state queues.



Scheduling Processes

- **Long Term Scheduling:** How does the OS determine the degree of multiprogramming, i.e., the number of jobs executing at once in the primary memory?
- Short Term Scheduling: How does (or should) the OS select a process from the ready queue to execute?
 - Policy Goals
 - Policy Options
 - Implementation considerations



Short Term Scheduling

- The kernel runs the scheduler at least when
 1. a process switches from running to waiting,
 2. an interrupt occurs, or
 3. a process is created or terminated.
- **Non-preemptive system:** the scheduler must wait for one of these events
- **Preemptive system:** the scheduler can interrupt a running process



Criteria for Comparing Scheduling Algorithms

- **CPU Utilization** The percentage of time that the CPU is busy.
- **Throughput** The number of processes completing in a unit of time.
- **Turnaround time** The length of time it takes to run a process from initialization to termination, including all the waiting time.
- **Waiting time** The total amount of time that a process is in the ready queue.
- **Response time** The time between when a process is ready to run and its next I/O request.



Scheduling Policies

Ideally, choose a CPU scheduler that optimizes all criteria simultaneously (utilization, throughput,..), but this is not generally possible

Instead, choose a scheduling algorithm based on its ability to satisfy a policy

- Minimize average response time - provide output to the user as quickly as possible and process their input as soon as it is received.
- Minimize variance of response time - in interactive systems, predictability may be more important than a low average with a high variance.
- Maximize throughput - two components
 - minimize overhead (OS overhead, context switching)
 - efficient use of system resources (CPU, I/O devices)
- Minimize waiting time - give each process the same amount of time on the processor. This might actually increase average response time.



Scheduling Policies

Simplifying Assumptions

- One process per user
- One thread per process
- Processes are independent
- Single processor, single core

Researchers developed these algorithms in the 70's when these assumptions were more realistic, and it is still an open problem how to relax these assumptions.



Scheduling Algorithms: A Snapshot

FCFS: First Come, First Served

Round Robin: Use a time slice and preemption to alternate jobs.

SJF: Shortest Job First

Multilevel Feedback Queues: Round robin on each priority queue.

Lottery Scheduling: Jobs get tickets and scheduler randomly picks winning ticket.



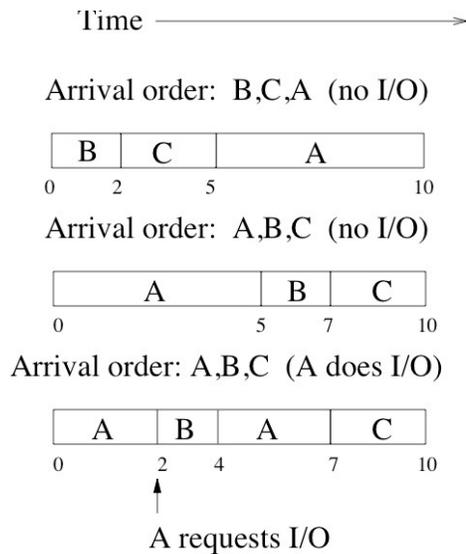
Scheduling Policies

FCFS: First-Come-First-Served (or FIFO: First-In-First-Out)

- The scheduler executes jobs to completion in arrival order.
- In early FCFS schedulers, the job did not relinquish the CPU even when it was doing I/O.
- We will assume a FCFS scheduler that runs when processes are blocked on I/O, but that is non-preemptive, i.e., the job keeps the CPU until it blocks (say on an I/O device).



FCFS Scheduling Policy: Example



- If processes arrive 1 time unit apart, what is the average wait time in these three cases?



FCFS: Advantages and Disadvantages

Advantage: simple

Disadvantages:

- average wait time is highly variable as short jobs may wait behind long jobs.
- may lead to poor overlap of I/O and CPU since CPU-bound processes will force I/O bound processes to wait for the CPU, leaving the I/O devices idle



Round Robin Scheduling

- Variants of round robin are used in most time sharing systems
 - Add a timer and use a preemptive policy.
 - After each time slice, move the running thread to the back of the queue.
 - Selecting a time slice:
 - Too large - waiting time suffers, degenerates to FCFS if processes are never preempted.
 - Too small - throughput suffers because too much time is spent context switching.
- => Balance these tradeoffs by selecting a time slice where context switching is roughly 1% of the time slice.
- Today: typical time slice= 10-100 ms, context switch time= 0.1-1ms
 - **Advantage:** It's fair; each job gets an equal shot at the CPU.
 - **Disadvantage:** Average waiting time can be bad.



Round Robin Scheduling: Example 1

- 5 jobs, 100 seconds each, time slice 1 second, context switch time of 0

Job	Length	Completion Time		Wait Time	
		FCFS	Round Robin	FCFS	Round Robin
1	100				
2	100				
3	100				
4	100				
5	100				
Average					



Round Robin Scheduling: Example 1

- 5 jobs, 100 seconds each, time slice 1 second, context switch time of 0

Job	Length	Completion Time		Wait Time	
		FCFS	Round Robin	FCFS	Round Robin
1	100	100	496	0	396
2	100	200	497	100	397
3	100	300	498	200	398
4	100	400	499	300	399
5	100	500	500	400	400
Average		300	498	200	398



Round Robin Scheduling: Example 2

- 5 jobs, of length 50, 40, 30, 20, and 10 seconds each, time slice 1 second, context switch time of 0 seconds

Job	Length	Completion Time		Wait Time	
		FCFS	Round Robin	FCFS	Round Robin
1	50				
2	40				
3	30				
4	20				
5	10				
Average					



Round Robin Scheduling: Example 2

- 5 jobs, of length 50, 40, 30, 20, and 10 seconds each, time slice 1 second, context switch time of 0 seconds

Job	Length	Completion Time		Wait Time	
		FCFS	Round Robin	FCFS	Round Robin
1	50	50	150	0	100
2	40	90	140	50	100
3	30	120	120	90	90
4	20	140	90	120	70
5	10	150	50	140	40
Average		110	110	80	80



SJF/SRTF: Shortest Job First

- Schedule the job that has the least (expected) amount of work (CPU time) to do until its next I/O request or termination.
- **Advantages:**
 - Provably optimal with respect to minimizing the average waiting time
 - Works for preemptive and non-preemptive schedulers
 - Preemptive SJF is called SRTF - shortest remaining time first
- ⇒ I/O bound jobs get priority over CPU bound jobs
- **Disadvantages:**
 - Impossible to predict the amount of CPU time a job has left
 - Long running CPU bound jobs can starve



SJF: Example

- 5 jobs, of length 50, 40, 30, 20, and 10 seconds each, time slice 1 second, context switch time of 0 seconds

Job	Length h	Completion Time			Wait Time		
		FCFS	RR	SJF	FCFS	RR	SJF
1	50						
2	40						
3	30						
4	20						
5	10						
Average							



SJF: Example

- 5 jobs, of length 50, 40, 30, 20, and 10 seconds each, time slice 1 second, context switch time of 0 seconds

Job	Length h	Completion Time			Wait Time		
		FCFS	RR	SJF	FCFS	RR	SJF
1	50	50	150	150	0	100	100
2	40	90	140	100	50	100	60
3	30	120	120	60	90	90	30
4	20	140	90	30	120	70	10
5	10	150	50	10	140	40	0
Average		110	110	70	80	80	40



Multilevel Feedback Queues (MLFQ)

- Multilevel feedback queues use past behavior to predict the future and assign job priorities
=> overcome the prediction problem in SJF
- If a process is I/O bound in the past, it is also likely to be I/O bound in the future (programs turn out not to be random.)
- To exploit this behavior, the scheduler can favor jobs that have used the least amount of CPU time, thus approximating SJF.
- This policy is **adaptive** because it relies on past behavior and changes in behavior result in changes to scheduling decisions.



Approximating SJF: Multilevel Feedback Queues

- Multiple queues with different priorities.
- Use Round Robin scheduling at each priority level, running the jobs in highest priority queue first.
- Once those finish, run jobs at the next highest priority queue, etc. (Can lead to starvation.)
- Round robin time slice increases exponentially at lower priorities.

	Priority	Time Slice			
<table border="1"><tr><td>G</td><td>F</td><td>A</td></tr></table>	G	F	A	1	1
G	F	A			
<table border="1"><tr><td>E</td></tr></table>	E	2	2		
E					
<table border="1"><tr><td>D</td><td>B</td></tr></table>	D	B	3	4	
D	B				
<table border="1"><tr><td>C</td></tr></table>	C	4	8		
C					



Adjusting Priorities in MLFQ

- Job starts in highest priority queue.
 - If job's time slices expires, drop its priority one level.
 - If job's time slices does not expire (the context switch comes from an I/O request instead), then increase its priority one level, up to the top priority level.
- ⇒ CPU bound jobs drop like a rock in priority and I/O bound jobs stay at a high priority.



Multilevel Feedback Queues: Example 1

- 3 jobs, of length 30, 20, and 10 seconds each, initial time slice 1 second, context switch time of 0 seconds, all CPU bound (no I/O), 3 queues

Queue	Time Slice	Job
1	1	
2	2	
3	4	

Job	Length	Completion Time		Wait Time	
		RR	MLFQ	RR	MLFQ
1	30				
2	20				
3	10				
Average					



Multilevel Feedback Queues: Example 1

- 5 jobs, of length 30, 20, and 10 seconds each, initial time slice 1 second, context switch time of 0 seconds, all CPU bound (no I/O), 3 queues

Job	Length	Completion Time		Wait Time	
		RR	MLFQ	RR	MLFQ
1	30	60	60	30	30
2	20	50	53	30	33
3	10	30	32	20	22
Average		46 2/3	48 1/3	26	28 1/3

Queue	Time Slice	Job
1	1	$1_1^1, 2_2^1, 3_3^1$
2	2	$1_5^3, 2_7^3, 3_9^3$
3	4	$1_{13}^7, 2_{17}^7, 3_{21}^7$ $1_{25}^{11}, 2_{29}^{11}, 3_{32}^{10} \dots$



Multilevel Feedback Queues: Example 2

- 3 jobs, of length 30, 20, and 10 seconds, the 10 sec job has 1 sec of I/O every other sec, initial time slice 2 sec, context switch time of 0 sec, 2 queues.

Job	Length	Completion Time		Wait Time	
		RR	MLFQ	RR	MLFQ
1	30				
2	20				
3	10				
Average					

Queue	Time Slice	Job
1	1	
2	2	



Multilevel Feedback Queues: Example 2

• 3 jobs, of length 30, 20, and 10 seconds, the 10 sec job has 1 sec of I/O every other sec, initial time slice 1 sec, context switch time of 0 sec, 2 queues.

Job	Length	Completion Time		Wait Time	
		RR	MLFQ	RR	MLFQ
1	30	60	60	30	30
2	20	50	50	30	30
3	10	30	18	20	8
Average		46 2/3	45	26 2/3	25 1/3

Queue	Time Slice	Job
1	1	1 ₁ , 2 ₁ , 3 ₁ 3 ₃ , 3 ₅ , 3 ₇ , 3 ₉ , 3 ₁₁ , 3 ₁₃ , 3 ₁₅ , 3 ₁₇ , 3 ₁₉ , 3 ₂₁ , 3 ₂₃ , 3 ₂₅ , 3 ₂₇ , 3 ₂₉
2	2	1 ₃ , 2 ₃ , 1 ₅ , 2 ₅ , 1 ₇ , 2 ₇ , 1 ₉ , 2 ₉ , 1 ₁₁ , 2 ₁₁ , 1 ₁₃ , 2 ₁₃ , 1 ₁₅ , 2 ₁₅ , 1 ₁₇ , 2 ₁₇ , 1 ₁₉ , 2 ₁₉ , 1 ₂₁ , 2 ₂₁ , 1 ₂₃ , 2 ₂₃ , 1 ₂₅ , 2 ₂₅ , 1 ₂₇ , 2 ₂₇ , 1 ₂₉ , 2 ₂₉ , 1 ₃₁ , 2 ₃₁ , 1 ₃₃ , 2 ₃₃ , 1 ₃₅ , 2 ₃₅ , 1 ₃₇ , 2 ₃₇ , 1 ₃₉ , 2 ₃₉



Improving Fairness

Since SJF is optimal, but unfair, any increase in fairness by giving long jobs a fraction of the CPU when shorter jobs are available will degrade average waiting time.

Possible solutions:

- Give each queue a fraction of the CPU time. This solution is only fair if there is an even distribution of jobs among queues.
- Adjust the priority of jobs as they do not get serviced (Unix originally did this.)
 - This ad hoc solution avoids starvation but average waiting time suffers when the system is overloaded because all the jobs end up with a high priority,.



Lottery Scheduling

- Give every job some number of lottery tickets.
- On each time slice, randomly pick a winning ticket.
- On average, CPU time is proportional to the number of tickets given to each job.
- Assign tickets by giving the most to short running jobs, and fewer to long running jobs (approximating SJF). To avoid starvation, every job gets at least one ticket.
- Degrades gracefully as load changes. Adding or deleting a job affects all jobs proportionately, independent of the number of tickets a job has.



Lottery Scheduling: Example

- Short jobs get 10 tickets, long jobs get 1 ticket each.

# short jobs/ # long jobs	% of CPU each short job gets	% of CPU each long job gets
1/1	91%	9%
0/2		
2/0		
10/1		
1/10		



Lottery Scheduling Example

- Short jobs get 10 tickets, long jobs get 1 ticket each.

# short jobs/ # long jobs	% of CPU each short job gets	% of CPU each long job gets
1/1	91% (10/11)	9% (1/11)
0/2		50% (1/2)
2/0	50% (10/20)	
10/1	10% (10/101)	< 1% (1/101)
1/10	50% (10/20)	5% (1/20)



Summary of Scheduling Algorithms:

- **FCFS:** Not fair, and average waiting time is poor.
 - **Round Robin:** Fair, but average waiting time is poor.
 - **SJF:** Not fair, but average waiting time is minimized assuming we can accurately predict the length of the next CPU burst. Starvation is possible.
 - **Multilevel Queuing:** An implementation (approximation) of SJF.
 - **Lottery Scheduling:** Fairer with a low average waiting time, but less predictable.
- ⇒ Our modeling assumed that context switches took no time, which is unrealistic.

