Last Class: Threads and Scheduling

- **Thread:** sequential execution stream within a process
- Kernel threads versus user-level threads
- Goals for Scheduling:
  - Minimize average response time
  - Maximize throughput
  - Share CPU equally
  - Other goals?
- **Scheduling Algorithms:**
  - Selecting a scheduling algorithm is a policy decision
  - FCFS: simple, but typically fails to meet above goals

Today: More on Scheduling Algorithms

- Round Robin
- SJF
- Multilevel Feedback Queues
- Lottery Scheduling

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

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

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

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

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
G F A	1	1
E	2	2
D B	3	4
C	4	8

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

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

Queue	Time Slice	Job
1	1	
2	2	
3	4	

Multilevel Feedback Queues: Example 2

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

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

Queue	Time Slice	Job
1	2	
2	4	

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		

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.

$\Rightarrow$ Our modeling assumed that context switches took no time, which is unrealistic.

Round Robin Scheduling: Example 1

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

Job	length	Completion Time FCFS	Completion Time Round Robin	Wait Time FCFS	Wait Time 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

- **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	Completion Time FCFS	Completion Time Round Robin	Wait Time FCFS	Wait Time 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: Example

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

		Completion Time			Wait Time		
Job	length	FCFS	Round Robin	SJF	FCFS	Round Robin	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: 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

		Completion Time		Wait Time	
Job	length	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 2/3	28 1/3

Queue	Time Slice	Job
1	1	$1^{1}_{1}, 2^{1}_{2}, 3^{1}_{3}$
2	2	$1^{3}_{5}, 2^{3}_{7}, 3^{3}_{9}$
3	4	$1^{7}_{13}, 2^{7}_{17}, 3^{7}_{21}$
		$1^{11}_{25}, 2^{11}_{29}, 3^{10}_{32} \ldots$

Multilevel Feedback Queues: Example 2

- 3 jobs, of length 30, 20, and 10 seconds, the 10 sec job has with 1 sec of I/0 every other sec, initial time slice 1 sec, context switch time of 0 sec, 2 queues
- This is one possible solution. Other solutions are possible depending on when the 10 sec job starts its I/O.

		Completion Time		**Wait Time**	
Job	length	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	$1^1_1, 2^1_2, 3^1_3$ $3^3_6, 3^5_9, 3^7_{12}, 3^9_{15}, 3^{10}_{18}$
2	2	$1^3_5, 2^3_8, 1^5_{11}, 2^5_{14}, 1^7_{17}, 2^7_{12},$ $1^9_{20}, 2^9_{22}, 1^{11}_{24}, 2^{11}_{26}, 1^{13}_{28}, 2^{13}_{30},$ $1^{15}_{32}, 2^{15}_{34}, 1^{17}_{36}, 2^{17}_{38}, 1^{19}_{40}, 2^{19}_{42},$

Lotttery 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)