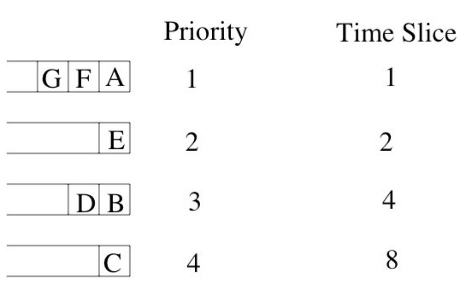
Multilevel Feedback Queues (MLFQ)

- 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.



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	

		Completion Time		**Wait Time**	
Job	Length	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}
2	2	1_{5}
3	4	1_{13} 1_{25}

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.

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

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

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.