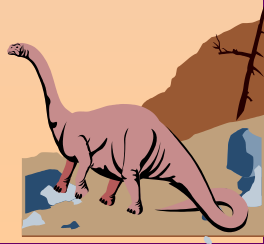
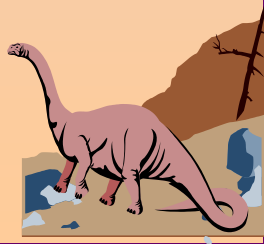
Operating Systems

Lecture 6
Scheduling



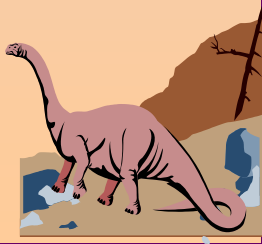
Schedulers

- Short term scheduler: Select a process in the ready queue for CPU allocation.
- Medium term scheduler (swapper): determine which process to swap in/out of disk to/from memory.
- Long term scheduler: Determine which processes are admitted into the system.
- Schedulers determine which processes will wait and which will progress.
- Schedulers affect the performance of the system.
- Scheduling is a fundamental O.S. function.
- We will discuss short-term schedulers (CPU allocation).

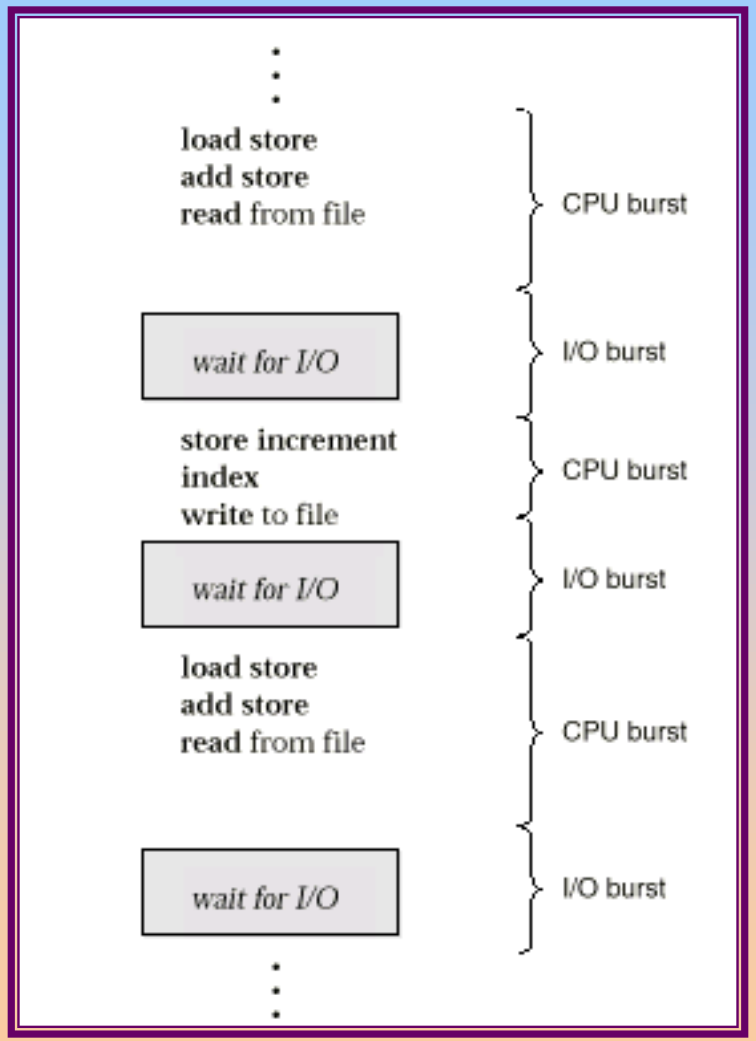


CPU-I/O Burst cycle

- Process execution consists of a cycles of CPU execution and I/O waiting.
- A process begins with a CPU burst.
- When a process waits for I/O, it is called an I/O burst.
- A process alternates between CPU bursts and I/O bursts.
- Eventually a CPU burst will end with process termination.
- The length of CPU bursts vary by process and computer.
- Typically there are many short bursts and a few long bursts.
- Question: What length of CPU bursts would an I/O bound process have?
- Question: What length of CPU bursts would a CPU bound process have?

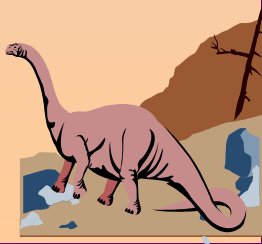


Alternating Sequence of CPU And I/O Bursts



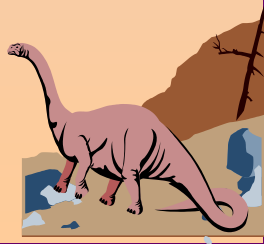
Preemptive vs. Non-preemptive scheduling

- Non-preemptive scheduling:
  - Each process completes its full CPU burst cycle before the next process is scheduled.
  - No time slicing or CPU stealing occurs.
  - Once a process has control of the CPU, it keeps control until it gives it up (e.g. to wait for I/O or to terminate).
  - Works OK for batch processing systems, but not suitable for time sharing systems.
- Preemptive scheduling:
  - A process may be interrupted during a CPU burst and another process scheduled. (E.g. if the time slice of the first process expires).
  - More expensive implementation due to process switching.
  - Used in all time sharing and real time systems.



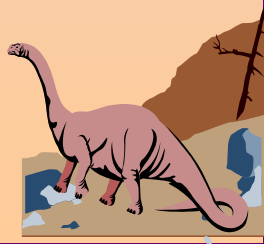
Dispatcher

- Process scheduling determines the order in which processes execute.
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running.



Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)



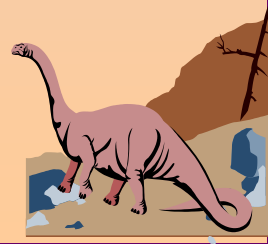
First-Come, First-Served (FCFS) Scheduling

Process that requests the CPU first is allocated the CPU first.
Easily managed with a FIFO queue.
Often the average waiting time is long.

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for P_1 = ; P_2 = ; P_3 =
- Average waiting time:

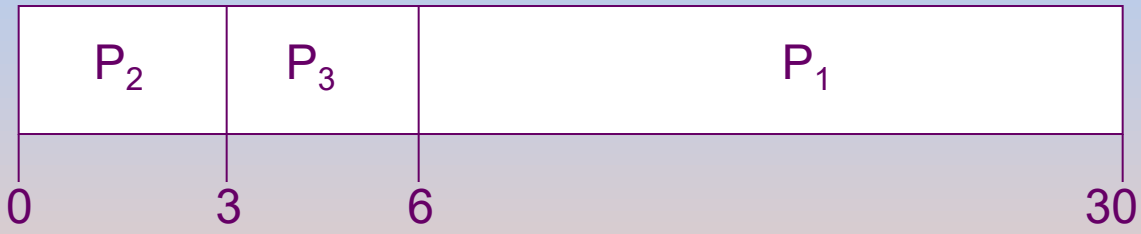


FCFS Scheduling (Cont.)

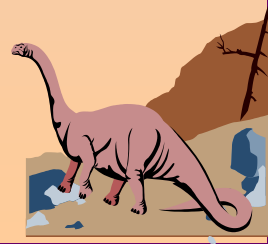
Suppose that the processes arrive in the order

$$P_2 , P_3 , P_1 .$$

- The Gantt chart for the schedule is:

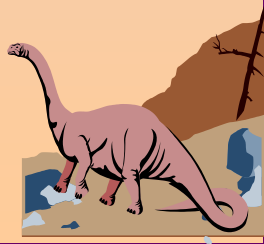


- Waiting time for P_1 = ; P_2 = ; P_3 =
- Average waiting time:
- Much better than previous case.
- *Convoy effect:* short processes line up behind long process.
- FCFS is not good for time-sharing systems. (Non-preemptive).



Round Robin (RR)

- FCFS Scheme: Potentially bad for short jobs!
  - Depends on the submitted order
- Round Robin Scheme
  - Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds
  - After quantum expires, the process is preempted and added to the end of the ready queue.
  - *n* processes in the ready queue and the time quantum is *q* ⇒
    - Each process gets 1/*n* of the CPU time
    - In chunks of at most *q* time units
    - No process waits more than (*n*-1)*q* time units
- Performance
  - *q* large ⇒ FCFS
  - *q* small ⇒ Interleaved (really small ⇒ hyperthreading?)
  - *q* must be large with respect to context switch, otherwise overhead is too high (all overhead)



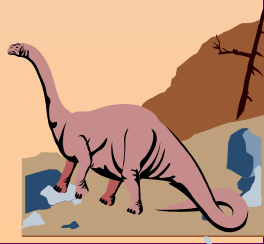
Example of RR with Time Quantum = 20

- Example:

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

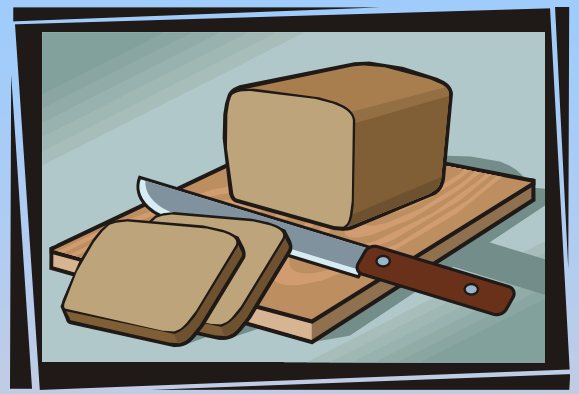
  - The Gantt chart is:

  - Waiting time for P_1=(68-20)+(112-88)=72
    P_2=(20-0)=20
    P_3=(28-0)+(88-48)+(125-108)=85
    P_4=(48-0)+(108-68)=88
  - Average waiting time = (72+20+85+88)/4=66¼
  - Average completion time = (125+28+153+112)/4 = 104½
- Thus, Round-Robin Pros and Cons:
  - Better for short jobs, Fair (+)
  - Context-switching time adds up for long jobs (-)

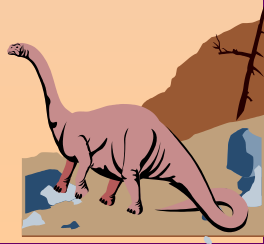


Round-Robin Discussion

- How do you choose time slice?
  - What if too big?
    - Response time suffers
  - What if infinite (∞)?
    - Get back FIFO
  - What if time slice too small?
    - Throughput suffers!



- Actual choices of timeslice:
  - In practice, need to balance short-job performance and long-job throughput:
    - Typical time slice today is between 10ms – 100ms
    - Typical context-switching overhead is 0.1ms – 1ms
    - Roughly 1% overhead due to context-switching

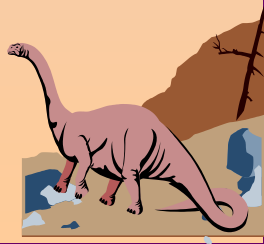


Comparisons between FCFS and Round Robin

- Assuming zero-cost context-switching time, is RR always better than FCFS?
- Simple example: 10 jobs, each takes 100s of CPU time RR scheduler quantum of 1s. All jobs start at the same time
- Completion Times:

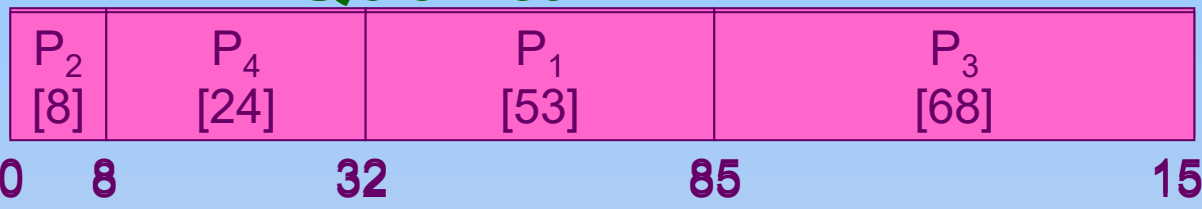
Job #	FIFO	RR
1	100	991
2	200	992
...	...	...
9	900	999
10	1000	1000

  - Both RR and FCFS finish at the same time
  - Average response time is much worse under RR!
    - Bad when all jobs same length
  - Total time for RR longer even for zero-cost switch!



Earlier Example with Different Time Quantum

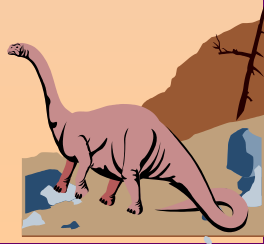
Best FCFS:



	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	$31\frac{1}{4}$
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	$61\frac{1}{4}$
	Q = 8	80	8	85	56	$57\frac{1}{4}$
	Q = 10	82	10	85	68	$61\frac{1}{4}$
	Q = 20	72	20	85	88	$66\frac{1}{4}$
	Worst FCFS	68	145	0	121	$83\frac{1}{2}$
Completion Time	Best FCFS	85	8	153	32	$69\frac{1}{2}$
	Q = 1	137	30	153	81	$100\frac{1}{2}$
	Q = 5	135	28	153	82	$99\frac{1}{2}$
	Q = 8	133	16	153	80	$95\frac{1}{2}$
	Q = 10	135	18	153	92	$99\frac{1}{2}$
	Q = 20	125	28	153	112	$104\frac{1}{2}$
	Worst FCFS	121	153	68	145	$121\frac{3}{4}$

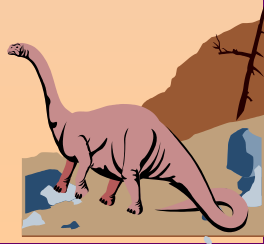
What if we Knew the Future?

- Could we always mirror best FCFS?
- Shortest Job First (SJF):
  - Run whatever job has the least amount of computation to do
  - Sometimes called “Shortest Time to Completion First” (STCF)
- Shortest Remaining Time First (SRTF):
  - Preemptive version of SJF: if job arrives and has a shorter time to completion than the remaining time on the current job, immediately preempt CPU
  - Sometimes called “Shortest Remaining Time to Completion First” (SRTCF)
- These can be applied either to a whole program or the current CPU burst of each program
  - Idea is to get short jobs out of the system
  - Big effect on short jobs, only small effect on long ones
  - Result is better average response time



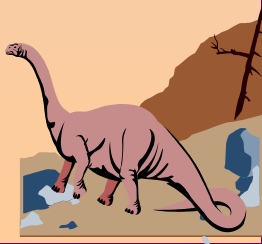
Discussion

- SJF/SRTF are the best you can do at minimizing average response time
  - Provably optimal (SJF among non-preemptive, SRTF among preemptive)
  - Since SRTF is always at least as good as SJF, focus on SRTF
- Comparison of SRTF with FCFS and RR
  - What if all jobs the same length?
    - SRTF becomes the same as FCFS (i.e. FCFS is best can do if all jobs the same length)
  - What if jobs have varying length?
    - SRTF (and RR): short jobs not stuck behind long ones



Shortest-Job-First (SJF) Scheduling

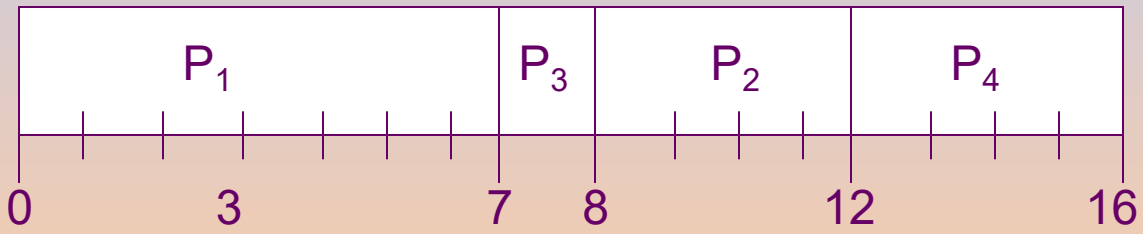
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.



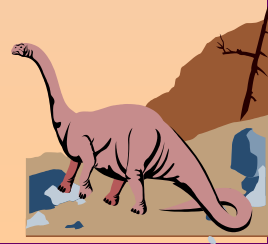
Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



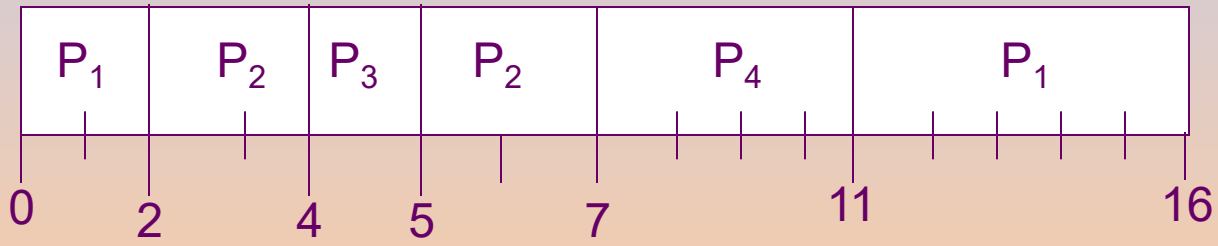
- Average waiting time =



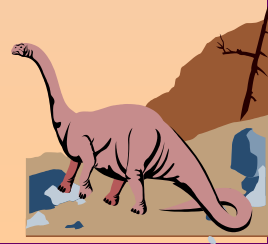
Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time =



Thanks