

Scheduling

Scheduling

- scheduling: share CPU among processes
- scheduling should:
 - be fair
 - all processes must be similarly affected
 - no indefinite postponement
 - “aging” as a possible solution
 - » adjust priorities based on waiting time for resource
 - max. possible no of processes per unit time
 - reduce response time for interactive users

Scheduling

- priorities should be used
- not fail even under very heavy load
 - e.g. accept no new processes to system
 - e.g. lower quantum

Scheduling Criteria

- I/O bound
- CPU bound
- interactive / batch
- importance of quick response
- priority
- real execution time
- time to completion

Scheduling

- preemptive x non-preemptive scheduling
- preemptive
 - high cost of context switching
 - to be effective, there must be a sufficient amount of processes ready to run in memory

Priorities

- static x dynamic priorities
- static priorities
 - fixed during execution
 - easy to implement
 - not efficient
- dynamic priorities
 - change based on environment changes
 - harder to implement + more CPU time
 - enhances response times

Scheduling Example

Process	Time of Arrival	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

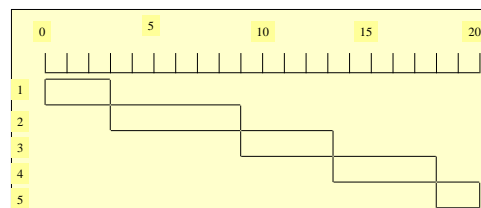
Scheduling Techniques

- Deadline scheduling
 - order processes based on their ending times
 - useless if process is not completed on time
 - process must declare all resource requests beforehand
 - may not be possible
 - plan resource allocation based on ending times
 - new resources may become available

Scheduling Techniques

- FIFO scheduling
 - simplest technique
 - order based on arrival times
 - non-preemptive
 - processes with short service times wait unnecessarily because of processes requiring long service times
 - ineffective for interactive processes
 - response times may be too long
 - ineffective for I/O bound processes
 - I/O ports may be available while the process waits for a CPU bound process to complete
- ⇒ FIFO usually used together with other techniques

Example: FIFO Scheduling



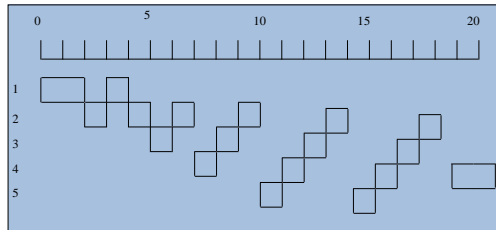
Scheduling Techniques

- Round-Robin scheduling
 - FIFO-like
 - assign CPU to processes for fixed time units in turn
 - preemptive
 - quantum = time slice
 - if not completed within quantum: move to end of queue
 - effective for interactive processes
 - has context switching

Scheduling Techniques

- selection of quantum is critical
 - has effect on performance of system
 - short x long
 - fixed x variable
 - same x different for each user
 - if too long quantum ⇒ becomes FIFO
 - if too short quantum ⇒ too much time for context switches
 - correct quantum sizes different for different types of systems

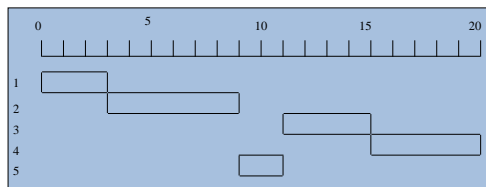
Example: Round-Robin Scheduling



Scheduling Techniques

- shortest-job-first scheduling
 - non-preemptive
 - order based on shortest time to completion
 - decreased average waiting times compared to FIFO
 - better service for short jobs
 - not suitable for interactive processes
 - total running time must be known beforehand
 - user provides estimate
 - if requires more than estimate, stop process and run later
 - if jobs repeat, may know running time

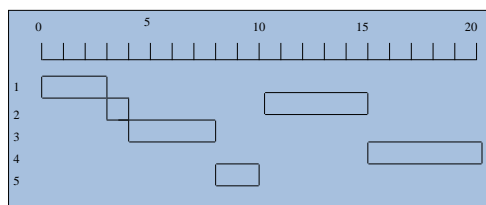
Example: Shortest-Job-First Scheduling



Scheduling Techniques

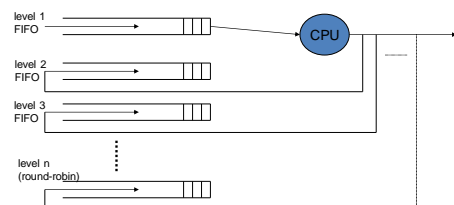
- shortest time remaining
 - preemptive version of previous technique
 - good performance for time-sharing systems
 - run process with least time remaining to completion
 - consider new arrivals too
 - a running process may be preempted by a new, short process
 - total running time must be known beforehand
 - more time wasted
 - used / remaining time calculations
 - context switching

Example: shortest time remaining



Scheduling Techniques

- Multilevel queues



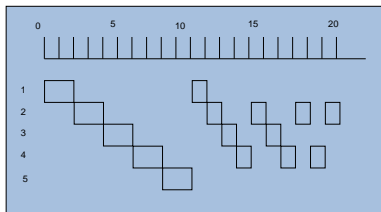
Scheduling Techniques

- new process to end of level 1
- FIFO within levels
- if not completed within quantum, go to end of lower level
- limited no of levels
- in last level, round-robin instead of FIFO
- short, new jobs completed in a short time
- in some systems, longer quantum at lower levels

Scheduling Techniques

- processes at higher level queues finished before those in lower levels can be run
- a running process may be preempted by a process arriving to a higher level
- in some systems stay in same queue for a few rounds
 - e.g. at lower level queues

Example: Multilevel Queues



Assumption: Max 3 levels in system.

Example 2: Multilevel Queues

Previous example assumes equal quantum values at all levels. How will the scheduling diagram be if at each level, the quantum (Q) assigned to each process at each level (l) was calculated based on the level.

$$Q = 2^{(l-1)}$$

at level 1: 2^0 (1 units)

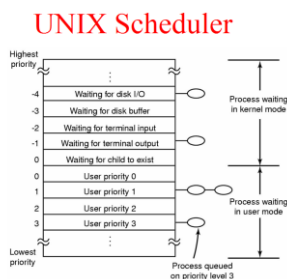
at level 2: 2^1 (2 units)

...

at level 6: 2^5 (32 units)

...

Scheduling in UNIX Systems



Scheduling in UNIX Systems

$$\text{Priority} = \text{CPU_usage} + \text{nice} + \text{base}$$

$$\text{CPU_usage} = \Delta T / 2$$

Example:

- Assume only 3 processes
- base=60
- no nice value
- clock interrupts system 60 times per quantum
- start with the order Process A, B and C

Time	Process A		Process B		Process C	
	Priority	Cpu Count	Priority	Cpu Count	Priority	Cpu Count
0	60	0	60	0	60	0
		1				
		2				
		60				
1	75	30	60	0	60	0
				1		
				2		
				60		
2	67	15	75	30	60	0
						1
						2
						60
3	63	7	67	15	75	30
		8				
		9				
		67				
4	76	33	63	7	67	15

Priority = (CPUusage/2) + 60