Uniprocessor Scheduling
Types of scheduling in OS

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long-term scheduling</td>
<td>The decision to add to the pool of processes to be executed.</td>
</tr>
<tr>
<td>Medium-term scheduling</td>
<td>The decision to add to the number of processes that are partially or fully in main memory.</td>
</tr>
<tr>
<td>Short-term scheduling</td>
<td>The decision as to which available process will be executed by the processor.</td>
</tr>
<tr>
<td>I/O scheduling</td>
<td>The decision as to which process's pending I/O request shall be handled by an available I/O device.</td>
</tr>
</tbody>
</table>
Long-Term Scheduling

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
- More processes, smaller percentage of time each process is executed

Not very common (crond, “planned activity”)
Medium-Term Scheduling

- process suspension
- based on the need to manage the degree of multiprogramming
- the swapper
Short-Term (cpu) Scheduling

- a cpu scheduling policy decides for each cpu...
  - which process should be executed among the ready ones
  - how long it will be executed
- the scheduler executes very frequently
  - Invoked when an event occurs
    - Timer interrupts (time quantum expired)
    - system calls (blocking I/O operations)
      - if policy is preemptive: on I/O interrupts
  - e.g. 200 times per second
- after the scheduler the dispatcher runs
  - usually used as synonymous
process states
scheduling and process state transitions
short term cpu scheduling
decision mode

• non-preemptive
  – Once a process is in the running state, it will continue until it terminates or blocks itself for I/O

• preemptive
  – currently running process may be interrupted
    • better service since a process does not monopolize the cpu
  – can a process be preempted while running in kernel mode?
    • kernel preemptability: good kernels (especially real time ones) are mostly preemptable
    • in Linux kernel preemptability for version >2.6, large parts are still not preemptable
queue

\[ \lambda \leq \frac{1}{S} \]

(statistically, always verified in an operating systems)

for cpu scheduling the server is the cpu
User Oriented, Performance Related

Turnaround time  This is the interval of time between the submission of a process and its completion. Includes actual execution time plus time spent waiting for resources, including the processor. This is an appropriate measure for a batch job.

Response time  For an interactive process, this is the time from the submission of a request until the response begins to be received. Often a process can begin producing some output to the user while continuing to process the request. Thus, this is a better measure than turnaround time from the user's point of view. The scheduling discipline should attempt to achieve low response time and to maximize the number of interactive users receiving acceptable response time.

Deadlines  When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.

User Oriented, Other

Predictability  A given job should run in about the same amount of time and at about the same cost regardless of the load on the system. A wide variation in response time or turnaround time is distracting to users. It may signal a wide swing in system workloads or the need for system tuning to cure instabilities.
System Oriented, Performance Related

**Throughput**  The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed. This clearly depends on the average length of a process but is also influenced by the scheduling policy, which may affect utilization.

**Processor utilization**  This is the percentage of time that the processor is busy. For an expensive shared system, this is a significant criterion. In single-user systems and in some other systems, such as real-time systems, this criterion is less important than some of the others.

System Oriented, Other

**Fairness**  In the absence of guidance from the user or other system-supplied guidance, processes should be treated the same, and no process should suffer starvation.

**Enforcing priorities**  When processes are assigned priorities, the scheduling policy should favor higher-priority processes.

**Balancing resources**  The scheduling policy should keep the resources of the system busy. Processes that will underutilize stressed resources should be favored. This criterion also involves medium-term and long-term scheduling.
optimality criteria

- **cpu bound processes**
  - tend to monopolize the CPU making other processes to starve
  - unfair with respect to other processes concerning CPU usage

- **I/O bound processes**
  - do not need very much CPU
  - quickly provide something to do for other devices
  - when other devices run parallelism (and hence, throughput and response time) is improved

- **CPU scheduler** should prefer I/O bound vs. CPU bound processes
scheduling queuing diagram
Priorities

- a priority is assigned to each process
- a ready process queue for each priority
- Scheduler will always choose a process of higher priority over one of lower priority
- Lower-priority may suffer starvation
  - Allow a process to change its priority based on its age or execution history

- preemption may be based on priority
priority queuing and preemption
Process Scheduling Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

- **arrival time**: when the process enter the ready queue
- **service time**: the process virtual time elapsed till the next blocking operation
First-Come-First-Served (FCFS)

- A short process may have to wait a very long time before it can execute
- Favors CPU-bound processes
  - I/O processes have to wait until CPU-bound process completes
First-Come-First-Served (FCFS)

- When the current process ceases to execute, the oldest process in the Ready queue is selected (non preemptive)
Round-Robin

- Clock interrupt is generated at periodic intervals
- When an interrupt occurs, the currently running process is placed in the read queue (preemption based on timer)
  - Next ready job is selected
- a.k.a. time slicing
Round-Robin (RR), $q=1$

- **preemption** based on a timer
- **time quantum** $q$: each process is allowed to use the processor for the time quantum and then preempted
RR and queues

not on the book

blocked

interactive event

ready

round robin

time quantum expired

cpu

response time
effect of quantum on response time
effect of quantum on response time

(b) Time quantum less than typical interaction
unfairness of RR

- I/O-bound processes usually release CPU before expiration of their quantum
- CPU-bound processes run for the whole quantum
- RR prefers CPU-bound processes
- We would like to prefer I/O-bound processes!
Shortest Process Next (SPN, SJF)

- process with shortest expected processing time is selected next (expected processing time to the next blocking i/o operation)
- need to know future! approximated.
- non-preemptive policy optimal w.r.t. minimum total waiting time
Shortest Process Next

- Predictability of longer processes is reduced
- Possibility of starvation for longer processes
- Estimation of time length of the next cpu-burst may be done by exponential averaging

\[ S_{n+1} = \alpha T_n + (1 - \alpha) S_n \]

\[ \alpha \in (0, 1] \]
Shortest Remaining Time (SRT)

- Preemptive version of shortest process next policy
- Must estimate processing time
feedback

- does not need service time estimation
- preemptive as in RR
- demotes processes at each expired time quantum into lower priority queues

Figure 9.10 Feedback Scheduling
Feedback

$\text{Feedback } q = 1$

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Service Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

© 2004 – 2011 william stallings, maurizio pizzonia - sit
feedback: varianti

- un processo scala di priorità...
  - sempre quando scade il suo quanto di tempo
  - oppure
  - quando scade il quanto e c'è almeno una altro processo nel sistema (Stallings)
feedback: varianti

- un processo aumenta di priorità quando va in blocco...
  - aumento fisso ogni volta che va in blocco
  oppure
  - aumento dipende dal tempo speso in blocco
Linux scheduling policies in kernel 2.6

- **Conventional processes**
  - FB with preemption (also in kernel mode)
  - "estimation" of the cpu-burst
  - Dynamically set higher priority for processes with shorter cpu-burst
  - User defined parameter (command: nice)

- **Real time processes**
  - FCFS
  - RR (user defined time quantum)
  - Priorities and preemption

*Not on the book*