Chapter 9: Uniprocessor Scheduling

**Aim:** Assign processes to the processor over time in a way that meets objectives such as:

- response time
- turnaround time
- throughput
- processor efficiency
- fairness
- lack of starvation
- low overhead
- predictability
- ...
- **Long-term scheduling**
  decision to admit new process (new job on batch system, or allowing login)

- **Medium-term scheduling**
  decision to have process partially/fully in main memory, i.e. swapping/suspending

- **Short-term scheduling**
  by *dispatcher*: which process gets the CPU

- **I/O scheduling**
  decision which process’s I/O request shall be handled

Chapter 9 concerns short-term scheduling
Figure 9.1  Scheduling and Process State Transitions
Basic assumptions behind scheduling

- several independent processes competing for CPU

- dispatcher has to share CPU among processes "fairly", "efficiently", . . .

Will these assumptions always remain true? Eg. multi-processor machines, single applications as multiple processes?

Processes compete for other resources than CPU, and may not be independent (eg. may have to synchronize)
Scheduling Criteria

- turnaround time batch
- response time interactive
- deadlines
- predictability
- throughput
- scheduling efficiency low overhead
- CPU utilization
- fairness to short/long/IO-bound/CPU-bound
- enforcing priorities
- balancing resources not just CPU
- relative or normalized ...
Typical Process Behaviour

CPU/IO bursts

- CPU-bound process:
  little IO, long CPU bursts

- IO-bound process:
  lot of IO, short CPU bursts
Scheduling algorithm can be classified by

- decision mode, i.e. when do we take a scheduling decision

- selection function, which process gets dispatched

Main categories of decision mode

- nonpreemptive:  
  never interrupt running process

- preemptive:  
  possibly interrupt running process
**When** does scheduling decision take place?

All (incl. nonpreemptive) algorithms:

- when running process becomes blocked because of IO request, waiting for child, lock acquisition, ... 
- when running process terminates

For preemptive algorithms maybe also

- at clock interrupt
- when blocked process becomes ready e.g. IO completion
- a new process starts
First-Come-First-Served (FCFS)

Run process on head of the queue until it terminates or performs IO (blocks)

Pros

- simple, low overhead
- no starvation

Cons

- bad response time
- throughput not emphasized
- unfair to short processes
- unfair to IO bound processes
Round Robin (RR)

FCFS + preemption at time quantum:
Run process on head of the queue until it terminates, blocks, or its quantum runs out

Key issue: length of quantum

Pros

- good response time
- no starvation
- relatively simple

Cons

- throughput can be low if quantum too small
- unfair to IO-bound processes (CPU-bound processes jump queue)
Figure 9.6  Effect of Size of Preemption Time Quantum
Variations of Round Robin

- **Virtual Round Robin**
  
  RR with separate queue for processes that have blocked

  fairer for IO-bound processes

- **Multilevel Queue Scheduling**
  
  RR with several queues, with different priorities (MINIX)
Figure 9.7 Queuing Diagram for Virtual Round-Robin Schedulor
Shortest Process Next (SPN)

Nonpreemptive (like FCFS), but always pick shortest process next

How do we know which process is shortest?

- for batch system: user estimate, statistics
- for interactive processes:
  estimate next CPU-burst by
  - taking average
  - taking exponential average

of previous bursts
Shortest Process Next (SPN)

Pros

• good (normalized) turnaround time

Cons

• overhead to estimate process time
• unfair to long processes
• risk of starvation
Shortest Remaining Time (SRT)

Shortest Process Next (SPN) + preemption when new process becomes ready

Pros

- even better turnaround time than SPN

Cons

- more overhead than SPN preemption and keeping track of elapsed process time
- still unfair to long processes
- still risk of starvation
Highest Response Ratio Next (HRRN)

Process with highest response ratio first

\[
\text{response ratio} = \frac{\text{waiting} + \text{service}}{\text{service}}
\]

Pros

- fairer than SPN, SRT
- no starvation

Short jobs still favoured, but long ones will get a turn

Cons

- overhead
(Multilevel) Feedback

What if we don’t know expected process time?

Penalize jobs that have been running long

Feedback: Processes with priorities (typically in several queues) that change over time.

(Lots of variations possible! Tuning to meet requirements)

Pro

• may favour IO-bound processes

Cons

• overhead
• unfair to long processes
• possible starvation of long processes (solution: aging)
Figure 9.10 Feedback Scheduling
Performance Analysis

How "good" is a scheduling algorithm?

Hard to answer. Lots of factors involved.

Ways to answer this question:

- Queuing analysis
- Simulation
- Trying in real-life, "tuning"
Priorities & Aging

Many scheduling algorithms (SPN, SRT, HRRN, Feedback, ...) use some form of priorities.

Potential problem with using priorities

- unfair to processes with low priorities
- possible starvation of process with low priorities

A solution

- Aging: increase priority with age
  (HRRN has aging built-in)
Priority Inversion

A process can block processes that have lower priority (of course).

Priority Inversion is the (unwanted!) situation where a process blocks processes that have higher priority.

Example

- high-priority process $P_0$
- medium-priority processes $P_1 \ldots P_{99}$
- low-priority process $P_{100}$
- $P_0$ has to wait for $P_{100}$, eg because $P_{100}$ holds a lock that $P_0$ needs

Now processes $P_1 \ldots P_{99}$ block $P_{100}$ but then also $P_0$!

This caused serious problems in NASA Pathfinder spacecraft that landed on Mars!

Possible Solution: priority inheritance
Convoy Effect

One process occupies a resource for a long time, blocking many processes that need it only for a short time.

Example:

- one CPU-bound process holding up many IO-bound processes
- one blocked process with a lock (eg exclusive access to some IO device) holding up many processes that need lock

Inefficient use of the other resources of the system.