Operating system concepts

Process Scheduling
Slides Set #5

By Prof K R Chowdhary

JNV University

2023

Why (context-) switch between processes?

- In case OS leaves a process and goes into *kernel mode*, it cannot return back to the same process it has left, because:
  - that process might have exited or terminated,
  - process has made a *blocking* system call (e.g., doing IO).
- Sometimes, the OS does not want to return back to the same process, because:
  - The process has run for too long, or
  - Must timeshare CPU with other processes.
- In above casse the OS performs a *context switch* to switch from one process to other.

Scheduling Criteria (deciding the order of execution)

A variety of metrics may be used:

1. *CPU utilization*: the fraction of the time the CPU is being used (and not for idle process!)
2. *Throughput*: Number of processes that complete their execution per time unit.
3. *Turnaround time*: amount of time to execute a particular process.
4. *Waiting time*: amount of time a process has been waiting in the ready queue.
5. *Response time*: amount of time it takes from when a request was submitted until the first response is produced (in time-sharing systems)
6. Sensible scheduling strategies might be:
   - Maximize throughput or CPU utilization, and
   - Minimize average turnaround time, waiting time or response time. Also need to worry about fairness and liveness.

The OS scheduler (i.e. process scheduler)

OS scheduler has two parts:

- *Policy* to pick which process to run next, and
- *Mechanism* to switch to that process.

Non-preemptive (cooperative) schedulers are polite:

- Switch only if process blocked or terminated.

Preemptive (non-cooperative) schedulers can switch even when process is ready to continue:

1. CPU generates periodic timer interrupt,
2. After servicing interrupt OS checks if the current process has run for too long.

What resources are we trying to optimize?

- Maximize *utilization* ($=$ fraction of time CPU is used)
- Minimize average *turnaround time* ($=$ time duration of process arrival to completion)
- Minimize *average response time* ($=$ time from process arrival to first scheduling)
- *Fairness*: all processes must be treated equally
- Minimize *overhead*: run process long enough to *amortize** cost of context switch ($\approx$ 1 microsecond)

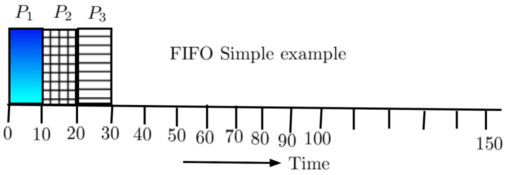
  *=gradually write of the initial cost.

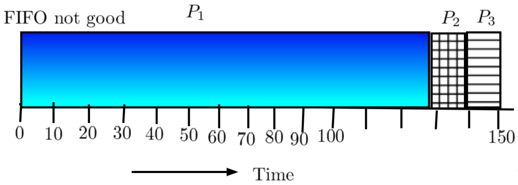
Types of Process scheduling

- First-In-First-out (FIFO), also called FCFS (First-come-first-served)
- Shortest job first (SJF) Scheduling
- Shortest Running/remaining Time First (SRTF) scheduling
- Round Robin Scheduling
- Static Priority Scheduling
- Dynamic Priority Scheduling
- Schedulers in real systems (e.g., Linux, Multi Level Feedback Queue) MLFQ

First-In-First-Out (FIFO)

- Also called first-come-first-served
- Let three processes arrive at time t=0, in the order P_1, P_2, P_3
- Problem: *Convoy effect* (Convoy Effect is phenomenon associated with the FCFS algorithm, in which the whole Operating System slows down due to few slow processes.)
- Turn around time tend to be high

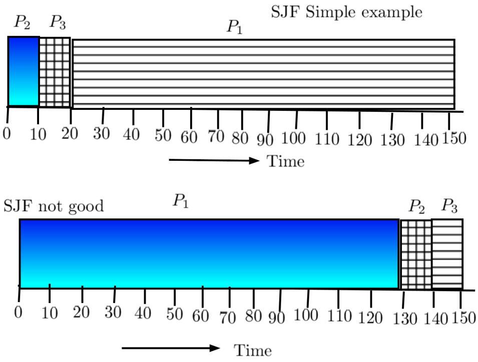




First-In-First-Out (FIFO)...

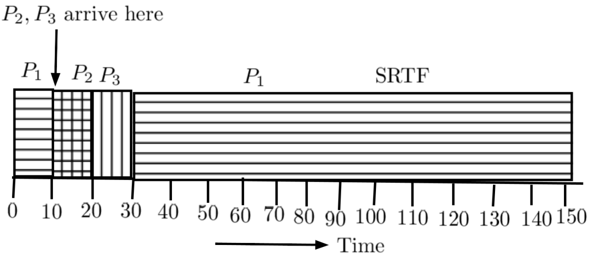
- FCFS depends on order processes arrive, e.g. P_1, P_2, P_3 have bust time of $25, 4, 7$.
- So, waiting time for $P_1 = 0$, for $P_2 = 25$, for $_P3 = 29$. so, average waiting time $= (0 + 25 + 29)/3 = 18$
- If these arrive in the order P_3, P_2, P_1, then wating time for $P_1 = 11$, for $P_2 = 7$, for $P_3 = 0$, so average waiting time is $11 + 7 + 0)/3 = 6$.
- First case is poor due to convoy *effect*

Shortest job First (SJF)



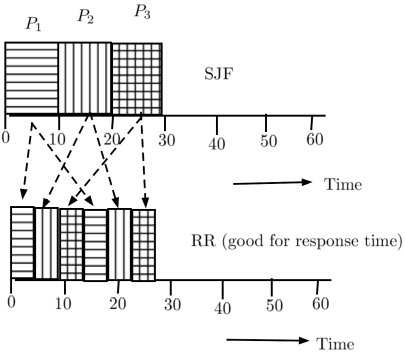
- Optimal when all processes arrive together.
- SJF is non-preemptive, so short jobs can still get stuck behind long ones.
- Average time in 1st: $(P_1, P_2, P_3), = (20+0+10)=10.$

Shortest Remaining Time First(SRTF)



- A Preemptive (?) scheduler
- Preempts running task if time left is more than that of new arrival

Round Robin (RR)



- Every process is executed for a fixed quantum of time
- Slice is big enough to reduce or pay off for the cost of context switch
- Premeptive
- Good for response time and fairness
- Bad for turn around time

Round Robin (RR)....

A small fixed unit of time called a *quantum* (or time-slice) is defined (10-100 millisecc.).

- Process at head of the ready queue is allocated the CPU for one quantum.
- When the time has elapsed, the process is preempted and added to the tail of the ready queue.

Following are good properties of RR:

- *Fair*: if there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$th of the CPU.
- *Live*: no process waits more than $(n-1)q$ time units before receiving a CPU allocation.
- Typically get higher average *turnaround time* than SRTF, and better average *response time*.
- By trickily choosing correct size quantum (q):
  - q too large $\Rightarrow$ FIFO
  - q too small $\Rightarrow$ context switch overhead too high.

Static Priority Scheduling

- Associates an integer with each process type, e.g.
  - Priority 0: for internal processes,
  - Priority 1: interactive processes,
  - Priority 2: students interactive processes,
  - Priority 3: batch processes
- Allocate CPU to the highest priority process (lowest integer)
- How to solve ties?
  - Round robin with time-slicing
  - This has Problem: Biased towards CPU intensive jobs
  - The less priority processes will go starvation

Dynamic Priority Scheduling

- Use same scheduling algorithm, but allow priorities to change over time.
- Simple aging:
  - processes have a (static) base priority and a dynamic effective priority.
  - If a process starves for k-seconds, increment effective priority.
  - Once a process runs, reset the effective priority.

Schedulers in Real (actual) Systems

- Real schedulers are more complex
- For example, Linux uses a Multi-Level Feedback Queue (MLFQ)
  - Many queues, in order of priority
  - Process from highest priority queue scheduled first
  - Within same priority, any algorithm like RR
  - Priority of process decays with its age

Process control block

OS maintains information about every process in a data structure called a process control block (PCB):

- Unique process identifier
- Process state (Running, Ready, etc.)
- CPU scheduling & accounting information
- Program counter & CPU registers
- Memory management information