Section 2 : Scheduling

When does scheduling happen?

The OS makes a scheduling decision when one of these things happen :

- Running → Waiting (ie process makes an I/O request)
- Running → Ready (timer interrupt in Round Robin)
- Waiting → Ready (I/O finishes)
- Process terminates
- New process created

Scheduling Goals (What are we trying to optimize?)

- Keep the CPU busy 100% of the time
- Throughput - # of processes completed per time unit
- Turnaround time - time from submission to completion
- Waiting time - Time from submission to first CPU execution
- Response time - Time from submission to first CPU execution
- Fairness - No processor starves, all get a turn, everyone eats

Resource types

Memory

Physical memory (RAM)

- Limited, only so much space for programs to live in
- OS allocates memory when a program starts and frees it when it exits

Virtual memory

- Tricks programs into thinking that they have more mem than is physically available
- Allows multiple processes to co exist by using disk as backup
- Enable memory protection and process isolation

I/O - disk drives, printers network cards

- I/O is slow so it must be scheduled carefully, can request I/O and block until its done

CPU

- Time shared, only one process can run per core at a time
- OS uses scheduling algorithms to decide who gets the CPU next

Multilevel Queue (MLQ)

How it works: Multiple separate queues for different process types

- i.e. System processes, interactive, batch
- Each queue can use its own scheduling algorithm
- Queue scheduling order is fixed (ie system → interactive → batch)

Pros: Great for managing process types with different needs

Cons: Rigid - processes are stuck in one queue

- Starvation possible if low priority queues are never served

Def : Starvation - When a process wait indefinitely to get the CPU because other processes keep getting chosen instead

Multilevel Feedback Queue (MFLQ)

How it works: Like MLQ, but processes can move between queues

- Processes that use too much CPU time get "demoted"
- New/interactive processes start in higher-priority queues
- More flexible and adaptive

Pros: Adjust priority based on behavior, Very responsive for short jobs and interactive processes

Cons: Hard to configure, Can be gamed if not implemented carefully

- Starvation is possible, but is mitigated through policy

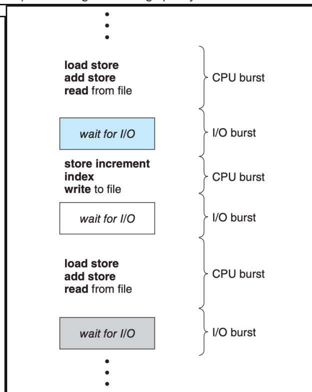

Figure 5.1 Alternating sequence of CPU and I/O bursts.

CPU Scheduling Algorithms

First come first serve (FCFS)

How it works: Processes run in the order they arrive

- No preemption - each process runs to completion

Pros: Simple to implement, Fair in order of arrival

Cons: **Convoy effect** - One long job delays everything, Poor responsiveness for short processes.

- No starvation - every process will eventually get the CPU

Shortest Job First (SJF)

How it works : Pick the process with the shortest estimated CPU burst

Can be either Non preemptive (Classic), Preemptive (Shortest Remain Time First SRTF)

Pros: Optimal average waiting time

Cons: Need to predict burst times, hard, Starvation of long job if short jobs keep arriving

Round Robin (RR)

How it works :

- Fixed time quantam for each, If it doesn't finish in time, sent to back of the queue.

Pros: Simple and fair - every process gets a turn, Good for interactive systems

Cons: Lots of context switching if quantum is too small. Too large → behaves like FCFS

Priority Scheduling

How it works: Each process is assigned a priority

- Can be
  - Preemptive : higher priority interrupts
  - Non-preemptive : lower priority finishes first

Pros: Good for real time tasks or critical jobs

Cons: Starvation if low priority jobs never run, Aging is needed to prevent starvation

Def : Aging - Gradually increases the priority of a process the longer it waits.

Def - Preemption - OS's ability to interrupt/pause running process so that another can run

CPU - I/O Bursts : Pattern in which a process alternates between CPU bursts (doing computation) and I/O bursts (waiting for I/O operations like reading from disk or network)

- Most processes follow the cycle : short CPU bursts, long I/O bursts

Turnaround Time - The total time from when a process is submitted to when it completes.

Turnaround Time = Completion Time - Arrival Time

Response Time - The time from when a process is submitted to when it first starts execution

Response Time = First Run Time - Arrival Time

Waiting Time - The total time a process spends in the ready queue, waiting to get CPU time

Waiting Time = Turnaround Time - CPU Burst Time

Throughput - The number of processes completed per unit of time

Queue Type	Purpose	Common Algorithm	Priority Level
System	OS-level processes	Preemptive Priority	Very High
Interactive	Real-time apps (GUIs, shells)	Round Robin, SRTF	High
Interactive Edit	Editors, IDEs	Round Robin	Medium-High
Batch	Background tasks	FCFS, SJF	Medium-Low
Student / Low	Low-priority jobs	FCFS	Low

Exponential Averaging Equation

In SJF, the scheduling must pick the process with the shortest next CPU burst. It predicts this time using the processes previous behavior.

$\tau(n+1) = \alpha \cdot t(n) + (1-\alpha) \cdot \tau(n)$

- **τ(n+1)**: Predicted burst time for the next CPU burst
- **t(n)**: Actual burst time of the current burst
- **τ(n)**: Predicted burst time for the current burst
- **α (alpha)**: Smoothing factor between 0 and 1

This is a weighted average. α controls how much weight is given to the recent burst time vs. the historical prediction

a = 1 → Use only the most recent burst → Reacts quickly but may be unstable

a = 0 → Use only the past predictions → Very stable, but doesn't learn

0 < α < 1 → Weighted mix of past + present → Balance between adaptability and stability

Preemption in Real Time

Real time schedulers are often preemptive so that critical tasks can interrupt less important ones to meet deadlines

- Hard real time systems always use preemptive scheduling
- Soft real time systems may use non preemptive algorithms to simplify design

Challenges in real time scheduling

- Priority inversion: A low priority process holds a resource needed by a high priority one
  - Solving using priority inheritance (temporarily raises the priority of a process to match the priority of another)
- Overhead from using too many preemptions
- Deadline overhead: When too many high priority tasks arrive at once, system many drop or reject low priority jobs

Real Time Scheduling

Two kinds of real time scheduling

- Hard Real Time
  - Deadlines must be met, missing even one can cause system failure.
- Soft real time
  - Deadlines are important, but missing them occasionally is okay

Goals of Real Time Scheduling

- Determinism : Predictable execution timing
- Deadline guarantees: Meet task specific timing contraints
- Minimal jitter: Keeping timing consistent
- High priority for timecritical tasks

Real Time Scheduling Algorithms

1. **Rate Monotonic Scheduling (RMS)**

   The more frequently a task needs to run, the more important it is

   - Priorities are fixed, they don't change during execution
   - Works only if all tasks are
     - Periodic (runs on a fixed schedule)
     - Independent (no shared resources that cause blocking)
   - Best for systems with predicable and repetitive jobs
2. **Earliest Deadline First (EDF)**

   Always run the task that is closest to its deadline

   - Priorities are dynamic - they change depending on the deadline
   - Can handle both
     - Periodic tasks
     - Aperiodic tasks (come in randomly)
   - Best for systems where timing is more flexible, or tasks arrive at random

Short Term Scheduler

- Selects which ready process runs on the CPU next; runs very frequently; must be fast

Long Term Scheduler

- Decided which jobs to enter the system from the job pool; runs less often; controls degree of multiprogramming

Def - Preemptive Scheduling - OS can interrupt a running process and switch to another

- Happens when high priority process arrives or time quantum expires

Def - Non Preemptive Scheduling - Once process starts running it runs to completion or until it voluntarily yields

Def - Priority Inversion - low priority processes blocks high priority, one by holding a needed resource. Fixed with priority inheritance

Def - Priority Inheritance - low priority process, temporarily inherits the higher priority of a blocked process to prevent priority, inversion, then reverts back after releasing the resource

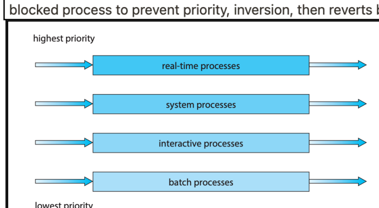

Figure 5.8 Multilevel queue scheduling.