Process Management

1 Process State

As a process executes, it changes state:

- **New** - The process is being created
- **Running** - Instructions are being executed
- **Waiting** - The process is waiting for some event to occur
- **Ready** - The process is ready to be dispatched to the CPU
- **Terminated** - The process has completed its execution, or some other event causing termination

2 Multiprogramming

Independent processes cannot affect or be affected by the execution of other processes
Co-operating processes can affect or be affected by the execution of another process - there are advantages of implementing process co-operation

The advantages of multiprogramming/implementing process co-operation may include::

- Computation speed-up: if there are multiple CPUs and/or cores
- Convenience: single user wants to run many tasks
- Information sharing: for example shared files
- Modularity: programming (e.g. OOP)

Multiprogramming supports the co-operation of processes

The aim of multiprogramming is to **maximise** CPU utilisation

With multiprogramming several processes are kept in memory concurrently

When one process is **waiting**, the operating system executes (to **running**) one of the processes sitting in the **ready** queue

CPU scheduling is the method used to support multiprogramming in process management

3 CPU Scheduling

CPU scheduling can be viewed as processes 'taking turns'

It provides a basis of fair and efficient sharing of system, in this case CPU, resources

As the CPU is a primary computer resource therefore scheduling is fundamental to operating system design

4 Schedulers

An operating system operates three types of scheduler: long-term, medium, and short term

- The **long term scheduler(job scheduler)** selects which process should be brought into the ready queue
- The **medium-term scheduler** removes processes from active contention for the CPU bv swapping processes in and out of the ready queue. This temporarily reduces the number of processes that the short-term scheduler (CPU scheduler must choose between)
- The **short-term scheduler (CPU scheduler)** selects the process to be executed next and allocated to the CPU

  This decision is initiated when processes:

  - Switch from running to waiting state
  - Switch from running to ready state
  - Switch from waiting to ready
  - Terminate

4.1 Considerations

Design considerations for scheduling.

When the CPU switches to another process:

- The system must save the context of the old process
- Load any saved context for the new process

Context-switch time is an **overhead**: the system undertakes no user processing while switching

4.2 Criteria

- CPU utilisation
- Throughput: the number of processes that complete their execution within a specific number of time unit
- Turnaround time: amount of time to execute a particular process
- Waiting time: total (accumulated) amount of time a process has been waiting in the ready queue
- Response time: amount of time it takes from when a request was first submitted to the ready queue until the first response is produced

5 Scheduling Algorithms

5.1 Algorithm Evaluation

Deterministic modelling:
Taking a predetermined workload (case) and analyse the performance of each algorithm

Simulation:
Complex model of the system and the way it is used. Need to beware of Bonini's paradox

Post-implementation:
Examine the running operating system

5.2 Priority Scheduling

In **priority scheduling**, the algorithm associates a priority number (integer for each process)

The CPU is allocated by the dispatcher to the process with the highest priority

Problem: Starvation - The low priority processes may never execute
Solution: Ageing - as time progresses increase the priority of the process

Priority based CPU scheduling can be either pre-emptive or non pre emptive

5.3 Multilevel queue scheduling

Multilevel queue scheduling is where processes are partitioned into different queues, e.g. foreground vs background processes.

The approach is based on the different queues having different (performance) requirements e.g. response time

Problem: provides differentiation but not flexible, since a process remains in the same queue regardless of adapting requirements.
Feedback queues: An adaptation of multilevel queue scheduling, however the process may proceed from one queue to the next. This is the most common approach to scheduling but also difficult to implement.

The **ready** queue is partitioned into separate queues

Each queue has its own scheduling algorithm

Scheduling must be undertaken between the queues

- Fixed priority scheduling
- Time slice: each queue gets a certain amount of CPU time which it can schedule among its processes

5.3.1 Multilevel feedback queue

A process can move between the various queues

A multilevel feedback queue scheduler may use the following parameters:

- The number of queues
- The scheduling algorithms for each queue
- The method used to determine when to upgrade a process
- The method used to determine when to demote a process
- The method used to determine which queue a process will enter

5.4 First Come, First Served (FCFS)

- The first process to request the CPU is allocated the CPU until it is completed

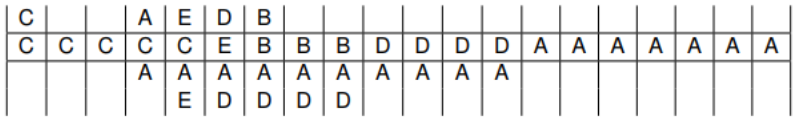
Process	Priority	Arrival Time	Burst Time
A	–	3	7
B	–	6	3
C	–	0	5
D	–	5	4
E	–	4	1

- The average waiting time may or may not be lengthy
- A simple algorithm to implement
- May result in a 'convoy' effect, for example short processes waiting on a long process to finish

5.5 Shortest-Job-First

- Compares each process waiting to determine the length of its next CPU burst
- Use these lengths to schedule the process with the shortest time (if two have the same length then FCFS)
- Non pre emptive - once the CPU is given a process it cannot be pre-emptive until the process has completed its CPU Burst

Process	Priority	Arrival Time	Burst Time
A	–	3	7
B	–	6	3
C	–	0	5
D	–	5	4
E	–	4	1



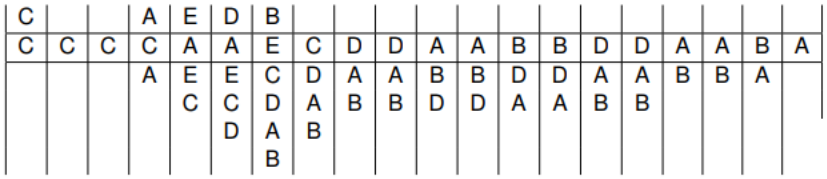
5.6 Round Robin (RR)

Each process gets a small unit of CPU time: a time quantum or time slice usually q=10 to 100 milliseconds

After this time has elapsed, the process is pre-empted and added to the end of the ready queue

If there are n processes in the ready queue and the time quantum is q, them each process gets $\frac{1}{n}$ of the CPU time in chunks of at most q time units at once

Process	Priority	Arrival Time	Burst Time
A	1	3	7
B	3	6	3
C	5	0	5
D	2	5	4
E	4	4	1

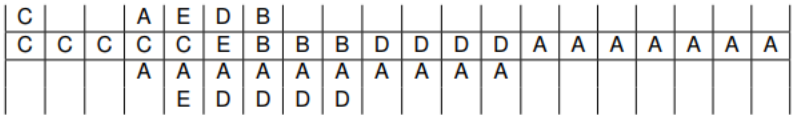


5.7 Shortest Remaining Time First

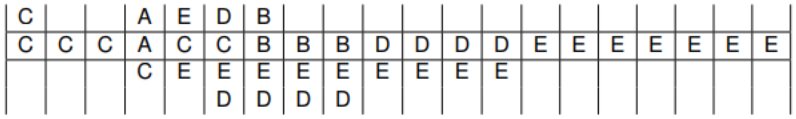
If a new process arrives in the ready queue with a CPU burst length time less than the remaining time of the executing process then pre-empt the running process and run the process with the shorter CPU burst length.

It can be viewed as a pre-emptive version of SJF.

Process	Priority	Arrival Time	Burst Time
A	–	3	7
B	–	6	3
C	–	0	5
D	–	5	4
E	–	4	1



Process	Priority	Arrival Time	Burst Time
A	–	3	1
B	–	6	3
C	–	0	5
D	–	5	4
E	–	4	7



5.8 Round Robin (With Priority)

With priority in round robin, we can choose to implement either:

- a priority queue, or
- a normal queue where the priority is used only when competing for entry onto the queue

  We will choose the latter

This is with time slice 2

Process	Priority	Arrival Time	Burst Time
A	5	0	3
B	4	5	7
C	3	6	1
D	2	2	5
E	1	7	4