Lecture 2: Process Management – Part 1

Barnaby Martin

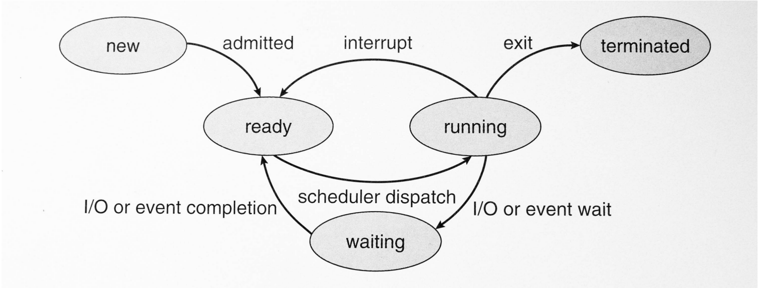
barnaby.d.martin@dur.ac.uk

Process State

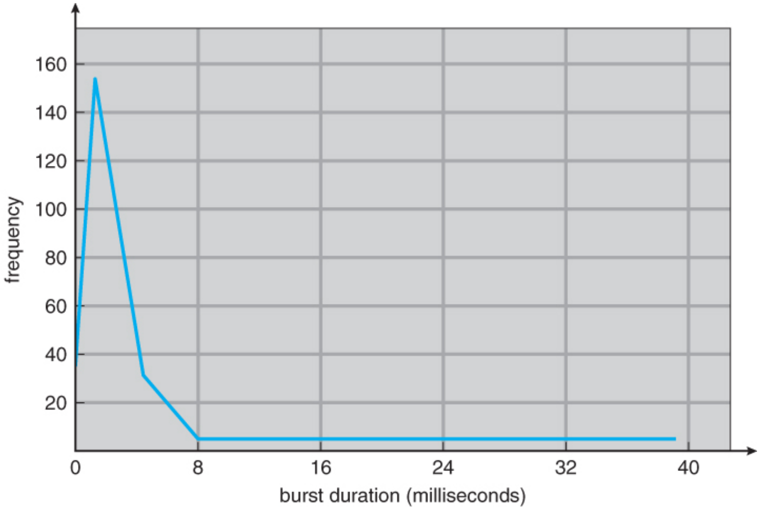
As a process executes, it changes state (thus status):

- **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.

Process State



CPU Bursts



frequency
160
140
120
100
80
60
40
20
0
8
16
24
32
40
burst duration (milliseconds)

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.

Multiprogramming

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.

Multiprogramming

The aim of multiprogramming is to **maximize** CPU utilization.

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.

CPU Scheduling

CPU scheduling can be viewed as processes ‘taking turns’.

It provides a basis of a 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.

Schedulers

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

The **Long-term scheduler (job scheduler)** selects which processes should be brought into the ready queue.

The **Medium-term scheduler** removes processes from active contention for the CPU by 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.

CPU Scheduler

The **Short-term scheduler (CPU scheduler)** selects the process to be executed next and allocated to the CPU.

This decision is initiated when processes:

1. Switch from running to waiting state, or
2. Switch from running to ready state, or
3. Switch from waiting to ready, or
4. Terminate

CPU Scheduling - 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.

Cost vs. Benefit.

Scheduling Criteria

- **CPU utilization**: keep the CPU busy!
- **Throughput**: number of processes that complete their execution within a specific number of time units.
- **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 (access to the CPU). NB: Not the time to complete the process!

Scheduling Algorithms

There are a number of different algorithms including:

- First Come, First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time First (SRTF)
- Priority (with or without pre-empting)
- Priority (with or without ageing)
- Round-Robin (RR)
- Multilevel queue / feedback queue

Algorithm Evaluation

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

Simulation:
Complex model of the system and the way it is used. Beware of Bonini's paradox!

Post-implementation:
Examine the running operating system (a.k.a. 'live' testing).

First Come, First Served (FCFS)

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

First Come, First Served (FCFS)

Example(s)

First Come, First Served (FCFS)

- 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.

Shortest-Job-First (SJF)

- 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 processes have the same length then FCFS.
- Non pre-emptive.
  - Once the CPU is given to the process it cannot be pre-empted until the process has completed its CPU burst.

Shortest-Job-First (SJF)

Example(s)

Shortest-Job-First (SJF)

- The average waiting time?
- A simple algorithm to implement?
- Long processes waiting on a short process to finish. Worse?

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, then each process gets $\frac{1}{n}$ of the CPU time in chunks of at most q time units at once.

Round Robin (RR)

Example(s)

Round Robin (RR)

- The average waiting time?
- A simple algorithm to implement?
- Short vs. long processes?