Anthony Ventresque

anthony.ventresque@ucd.ie

Operating Systems

COMP30640

Process Scheduling

School of Computer Science, UCD

Scoil na Ríomheolaíochta, UCD

Outline

- Understand the goals of Process Scheduling
- Understand the Queuing Process for Scheduling
- Understand Process Behaviour and Scheduling Criteria

Introduction

- **Burst cycles:** process execution typically consists of a cycle of two states
  1. **CPU burst**: CPU execution interval with no I/O usage
  2. **I/O burst:** wait for I/O signal

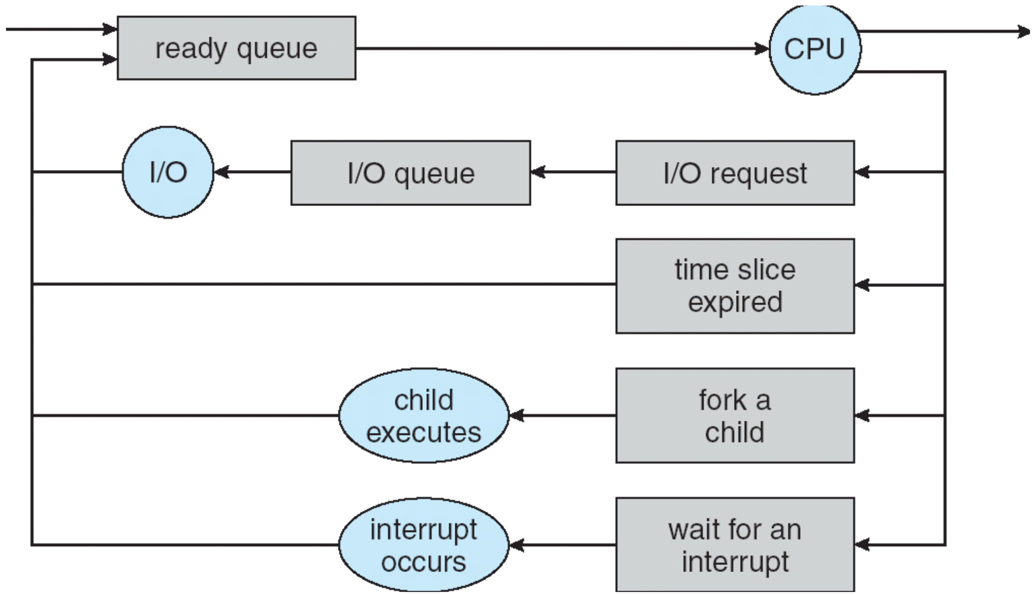
- Example of CPU bursts and I/O bursts in process execution:

fscanf(file1,"%i,%i",&a,&b)	CPU burst
wait for I/O	I/O burst

c=a+b;
d=(a*b)-c;
fprintf(file2,"%i,%i\n",c,d);
	CPU burst

wait for I/O	I/O burst

Recall: Thread/Process Lifecycle



- Earlier, we talked about the life-cycle of a thread
  - Active threads work their way from Ready queue to Running to various waiting queues.
- Question: How is the OS to decide which of several tasks to take off a queue?
  - Obvious queue to worry about is ready queue
  - Others can be scheduled as well, however
- **Scheduling**: deciding which threads are given access to resources from moment to moment

Scheduling

- ***Principle of multiprogramming*:** intertwine CPU bursts and I/O bursts of different processes to increase system utilisation
- Assume that an executing process reaches an I/O wait (or also that it finishes or times out)
  - What process (among those ready) shall the CPU execute next?
  - Not always an obvious choice, since there are virtually always more ready processes than CPUs available
- ***Scheduler*:** part of the OS that makes that decision relying on a ***scheduling algorithm***
  - In an OS that supports kernel threads, these are the smallest units of scheduling
  - Most issues that apply to process scheduling also apply to thread scheduling

Definitions

- Task/Job
  - User request: e.g., mouse click, web request, shell command, …
- Latency/Turnaround time
  - How long does a task take to complete?
- Response Time
  - How long does it take to react to a given input?
- Throughput
  - How many tasks can be done per unit of time?
  - How much time to execute a particular process?
- Overhead
  - How much extra work is done by the scheduler?
- Fairness
  - How equal is the performance received by different users?
- Predictability
  - How consistent is the performance over time?

More Definitions

- Workload
  - Set of tasks for system to perform
- Preemptive scheduler
  - If we can take resources away from a running task
- Work-conserving
  - Resource is used whenever there is a task to run
  - For non-preemptive schedulers, work-conserving is not always better
- Scheduling algorithm
  - Takes a workload as input
  - Decides which tasks to do first
  - Performance metric (throughput, latency) as output
  - Only preemptive, work-conserving schedulers to be considered

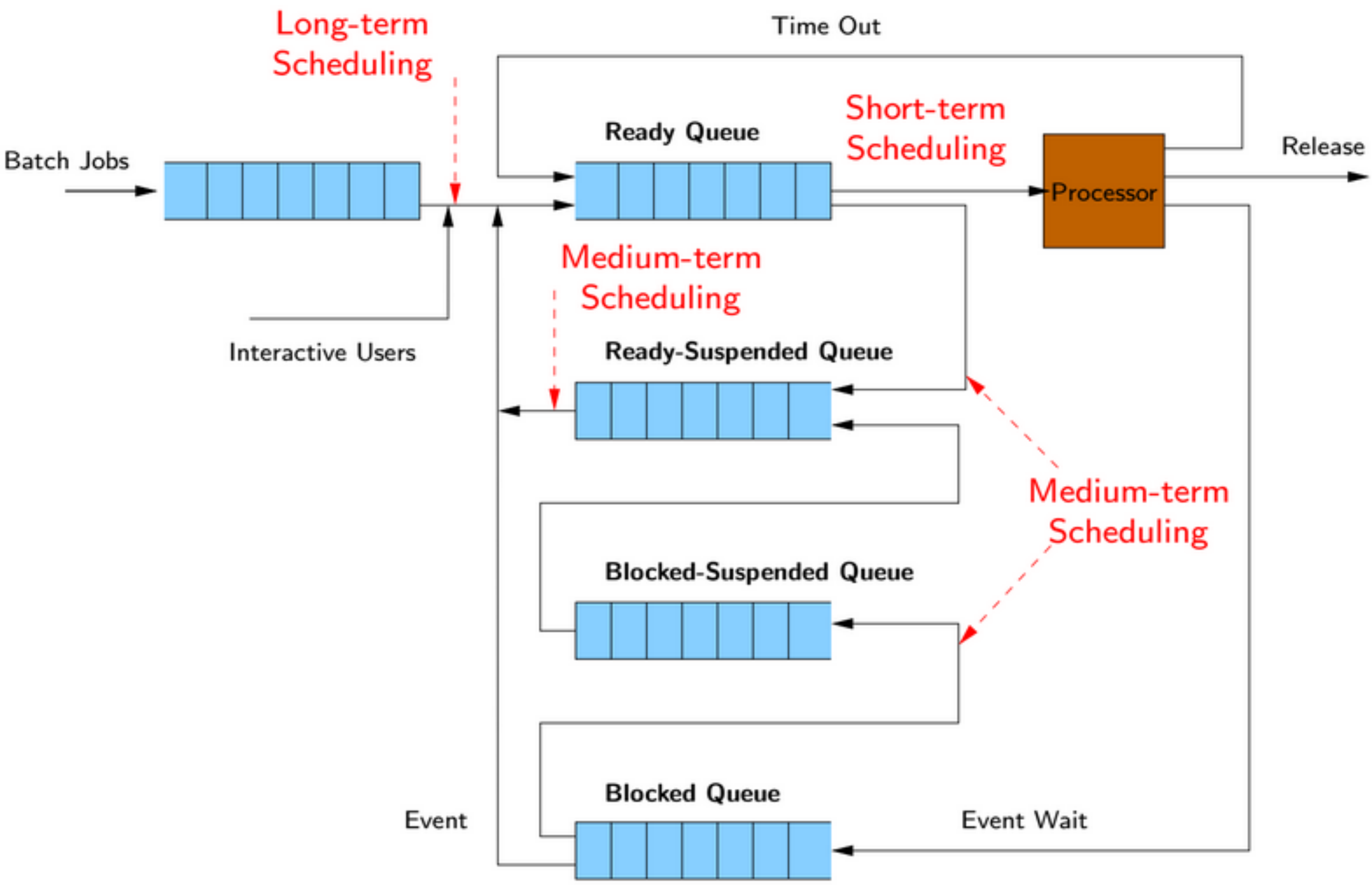
Levels of Scheduling

1. **Long-term (high-level) scheduler:** controls the pool of processes admitted to compete for system resources
   - A program (job) becomes a process once selected by the long-term scheduler, and it is added to the ready (or ready-suspended) queue
   - It controls the degree of multiprogramming
   - The more processes admitted, the smaller the percentage of time that each process can possibly be executed

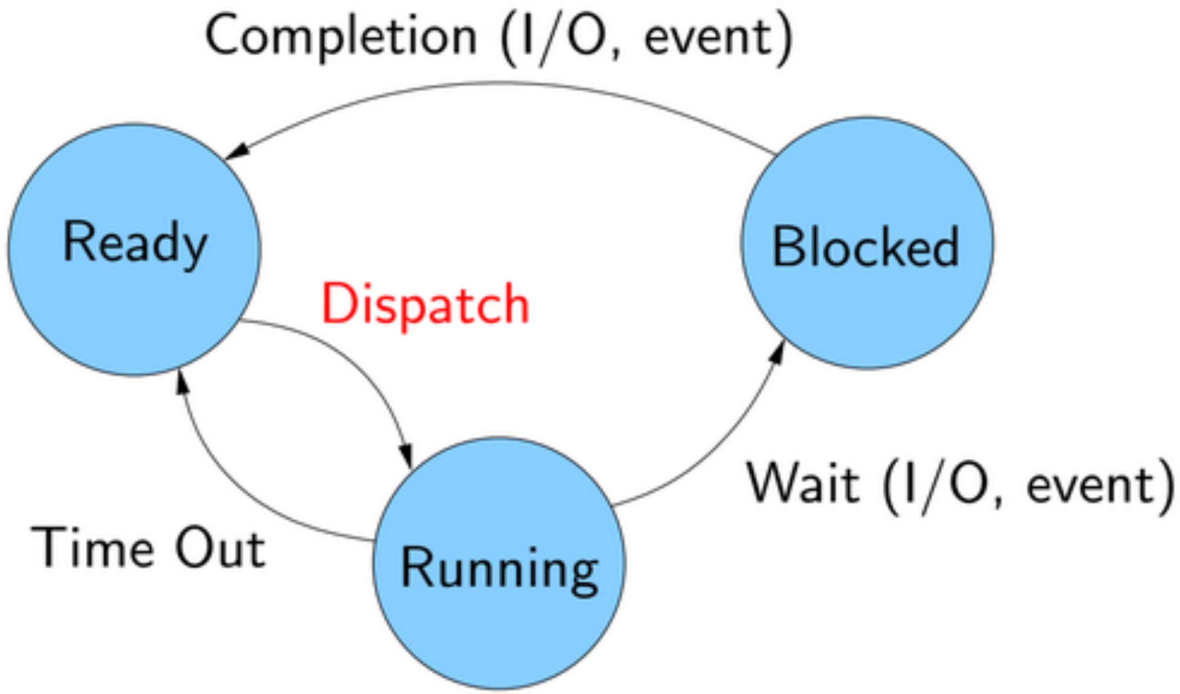
2. **Medium-term (mid-level) scheduler:** selects what processes are kept in memory, actively competing for CPU acquisition
   - The medium-term scheduler acts as a buffer, suspending and resuming processes to adaptively fine tune the system load
   - It determines which processes are in suspended-ready & suspended-blocked states

3. **Short-term (low-level) scheduler:** selects what process in the ready queue is assigned next to the CPU

Queueing Diagram for Scheduling



Process States (Simplified)



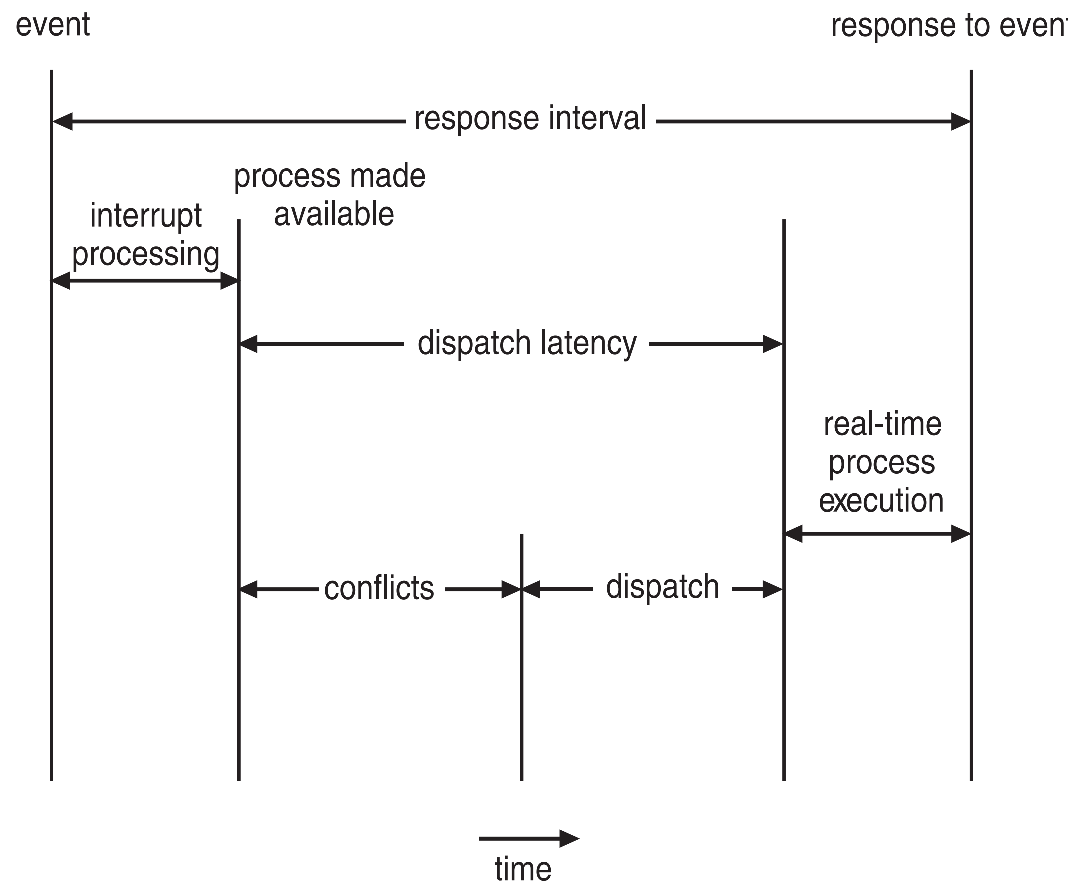
- Short-term scheduler dispatches processes in the ready state (ready queue)
  - As we will see, the discipline of this queue is not necessarily FIFO (first-in, first-out)

Dispatcher

- When the short-term scheduler selects the next process, the dispatcher routine gives it control of the CPU
- Dispatcher functions:
  - Switch context
    - Store relevant data in the PCB of running process
    - Swap in PCB contents of process to be run
  - Jump to the right location in the program to (re)start it, switching to user/kernel mode if required
- It must be as fast as possible (low dispatch latency), since it is invoked during every process switch

Dispatch Latency (Context Switch)

- Before the process can actually be dispatched, it must go through a conflicts phase
- Examples of conflicts:
  - Acquisition of resources needed by new process to execute
  - Pre-emption of running process resources if they should only be held while running



(in diagram we assume scheduler dispatches process immediately after it becomes ready)

Non Pre-emptive Scheduling

- ***Non pre-emptive*** scheduling (collaborative scheduling): some scheduling decisions take place when a process relinquishes the processor voluntarily:
  - It switches from "run" to "wait" (e.g. I/O request)
  - It terminates (runs until completion)

- Features:
  - Errant processes can block the system (e.g. infinite loops)
  - Short processes can experience long delays, if long processes are running non pre-emptively
    - Unimportant processes can make important ones wait
  - Time from process submission to process completion is quite predictable without pre-emption

Pre-emptive Scheduling

- ***Pre-emptive scheduling***: Scheduling decisions may take place where a process is forced to relinquish the processor it is running in, in order to give it to another process

- Features:
  - Malicious or errant processes can be removed from the CPU
  - Improved response times are possible:
    - Important for interactive systems, time-sharing
    - Fundamental for soft real-time systems (but hard real-time systems do not use pre-emption)

- ***Pre-emption cost***: <u>context switching overheads</u>
  - CPU utilisation may be high, but a lot of CPU cycles may be wasted doing switches
  - To minimise overheads, pre-empted processes tend to be kept in memory (i.e. not suspended by medium-term scheduler)

Scheduling Policy Goals/Criteria

- What does the scheduler take into account in its decisions?
- Different schedulers will have different <u>goals and policies</u>

- Examples:
  - Maximise processor utilisation
  - Maximise throughput: processes completed per time unit
  - Minimise response time (latency): waiting time in the ready
  - Queue for the first acquisition of the CPU
  - Minimise waiting time in the ready queue
  - Minimise turnaround (total waiting time + total execution time)
  - Complete processes by given deadlines (real time)
  - Fairness
  - Enforce priorities
  - . . .

- All of these goals <u>cannot be met simultaneously</u>

Scheduler Objectives

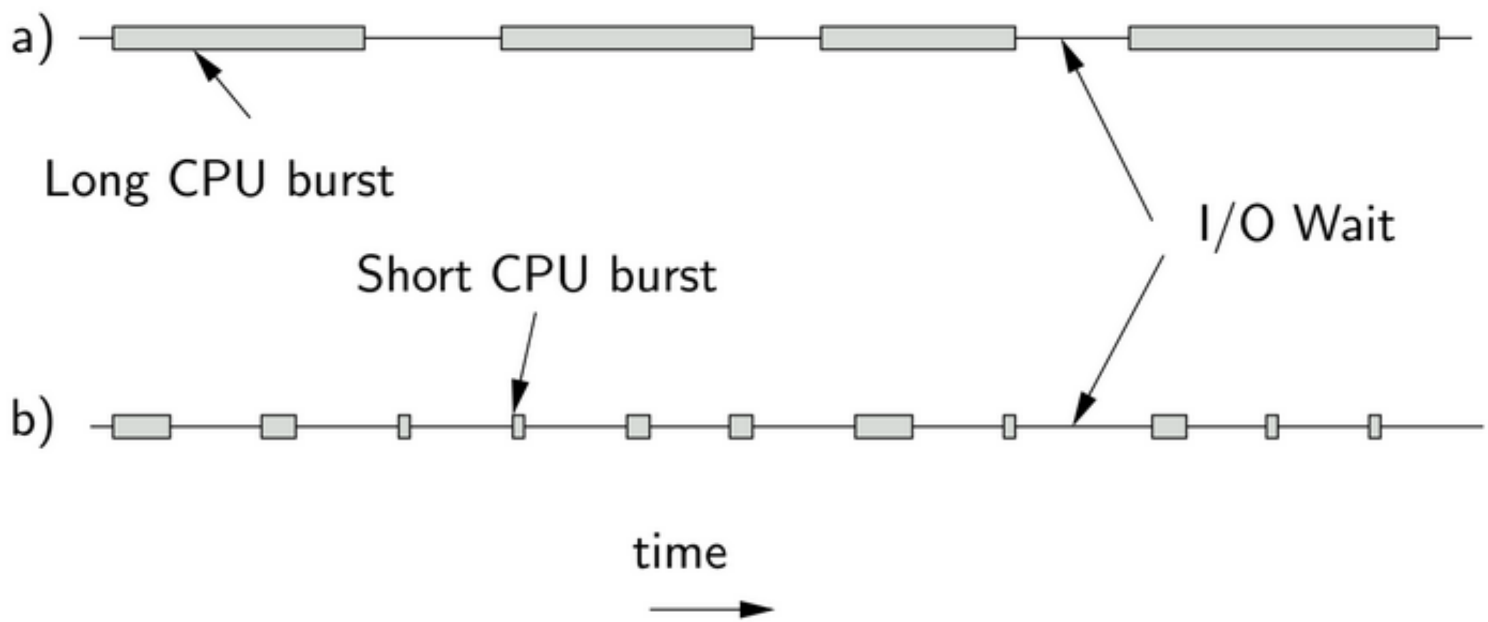
- Scheduling disciplines should achieve:
  - ***Policy enforcement*:** guarantee that the system scheduling policy is actually carried out
  - ***Fairness and balance*:**
    - No process is starved (no indefinite postponement)
    - Similar processes are treated similarly
  - ***Predictability*:** under a similar load, the same process should run in the approximately same amount of time
  - ***Scalability***: graceful performance degradation under heavy loads

Process Behaviour & Scheduling Criteria

To realise its scheduling objectives, the scheduler should also take ***process behaviour*** into account:

- Process classification according to their burst pattern:
  - ***Processor-bound***: it tends to use all available processor time (CPU utilisation)
  - ***I/O-bound***: it tends to generate I/O requests quickly and relinquish the processor

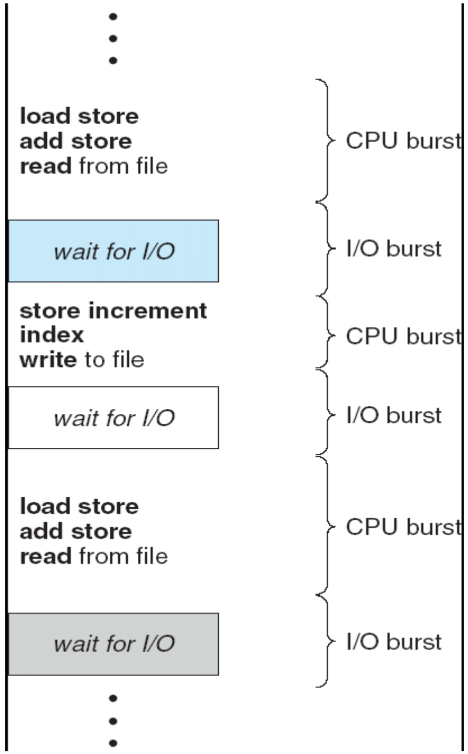
Processor-bound vs I/O-bound Processes

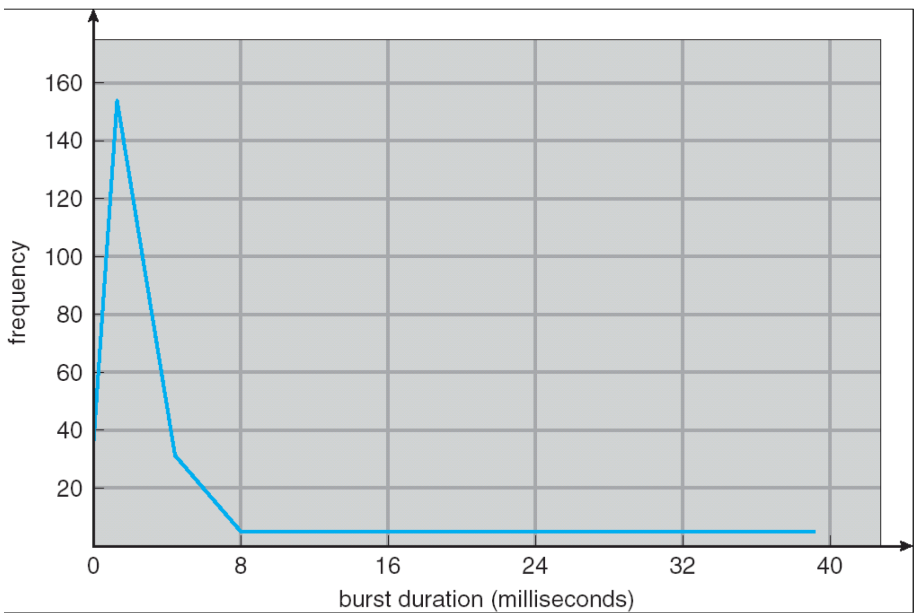


- Examples:
  - CPU-bound: compute the first 10^6 prime numbers
  - I/O-bound: print a series of documents

CPU Bursts and I/O Bursts in Typical Processes

- As processor technology improves faster than disk technology, most processes tend to be I/O-bound nowadays
- Considering all processes, the probability distribution of CPU burst times in any system typically resembles this diagram:





Process Behaviour & Scheduling Criteria

- Process classification according to their interactivity:
  - **Batch:** it performs work with no user interaction
    - Usually CPU-bound (although not necessarily so)
  - **Interactive:** it requires frequent user input
    - Typically I/O-bound

- Typical scheduling objectives depending on process behaviour
  - **Batch systems:** throughput, turnaround time
  - **Interactive systems:** response time, proportionality
    - Pre-emption is usually needed

How to Handle Simultaneous Mix of Different Types of Applications?

- Can we use Burst Time (observed) to decide which application gets CPU time?
- ***Consider mix of interactive and high throughput apps*:**
  - How to best schedule them?
  - How to recognize one from the other?
    - Do you trust app to say that it is "interactive"?
  - Should you schedule the set of apps identically on servers, workstations, pads, and cellphones?
- ***Assumptions encoded into many schedulers*:**
  - Apps that sleep a lot and have short bursts must be interactive apps – they should get high priority
  - Apps that compute a lot should get low(er?) priority, since they won't notice intermittent bursts from interactive apps
- ***Hard to characterize apps*:**
  - What about apps that sleep for a long time, but then compute for a long time?
  - Or, what about apps that must run under all circumstances (say periodically)

Conclusion

- Understand the goals of process scheduling
- Understand the queuing process for scheduling
- Understand process behaviour and scheduling criteria