04. Scheduling

9th ed: Ch. 6
10th ed: Ch. 5
Objectives

• To introduce CPU scheduling, the basis for multi-programmed operating systems, and the CPU I/O burst cycle

• To distinguish pre-emptive and non-preemptive scheduling

• To understand some different metrics used to make scheduling decisions
  • Utilisation, Throughput
  • Turnaround time, Waiting time, Response time
Outline

• Queues
• Scheduling
• Multiple processor scheduling
Outline

• Queues
  • CPU I/O burst cycle
  • CPU scheduler vs job scheduler
  • Idling

• Scheduling

• Multiple processor scheduling
Queues

- **Job Queue**: batch processes awaiting admission
- **Ready Queue**: processes in main memory, ready and waiting to execute
- **Wait Queue(s)**: set of processes waiting for e.g., I/O devices or other processes
Queues

- For example,
  - Two processes (7, 2) in the Ready queue
  - No processes waiting for either magnetic tape unit
  - Three processes (3, 14, 6) waiting for the disk
  - One process (5) waiting for the terminal
- ...etc
CPU I/O Burst Cycle

- Process execution interleaves CPU execution with waiting for I/O
- Maximising CPU utilization means multiprogramming
  - Need something to do while waiting for I/O
- CPU burst distribution helps parameterise scheduling
  - Often (hyper-)exponential
- I/O-bound
  - Many short CPU bursts
- CPU-bound
  - Fewer long CPU bursts

Figure 6.2 Histogram of CPU-burst durations.
An I/O-bound program typically has many short CPU bursts. A CPU-bound program might have a few long CPU bursts. This distribution can be important in the selection of an appropriate CPU-scheduling algorithm.

6.1.2 CPU Scheduler
Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler, or CPU scheduler. The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Note that the ready queue is not necessarily a first-in, first-out (FIFO) queue. As we shall see when we consider the various scheduling algorithms, a ready queue can be implemented as a FIFO queue, a priority queue, a tree, or simply an unordered linked list. Conceptually, however, all the processes in the ready queue are lined up waiting for a chance to run on the CPU. The records in the queues are generally process control blocks (PCBs) of the processes.

6.1.3 Preemptive Scheduling
CPU-scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for example, as the result of an I/O request or an invocation of wait() for the termination of a child process)
Schedulers

• Short-term or **CPU scheduler**
  • Selects which process should be executed next and allocates it to the CPU
  • Sometimes the only scheduler in a system
  • Invoked frequently (milliseconds) so must be fast

• Long-term or **Job scheduler**
  • Controls the degree of multiprogramming
  • Selects which processes should be brought into the ready queue
  • Invoked infrequently (seconds, minutes) so may be slow
  • Strives for good process mix between CPU- and I/O-bound processes
Idling

• Will assume there’s always something to do – but what if there isn’t?
  • An important question on a modern (interactive) machine

• Three options:
  1. Busy wait in the scheduler: short-response times but ugly, inefficient
  2. Halt CPU until interrupted: saves energy but increases latency
  3. Invent an **idle process**:
     • nice uniform structure and could do some housekeeping
     • ...but consumes resources and might slow interrupt response
Outline

• Queues

• Scheduling
  • Dispatcher
  • Pre-emptive vs non-preemptive
  • Criteria

• Multiple processor scheduling
Dispatcher

• After scheduler, the Dispatcher gives control of the CPU to the selected process by
  • Switching context,
  • Switching to user mode,
  • Executing the user process from the selected location

• **Dispatch latency** is the time it takes to complete this stop/start procedure

• Two important questions:
  1. When to make a scheduling decision to select the next process?
  2. How to order the queue – which process to select next?
When to enter the scheduler?

- When can the scheduling decision be made? When
  1. ...a running process blocks (running → waiting)
  2. ...a running process terminates (running → terminated)
  3. ...a timer expires (running → ready)
  4. ...a waiting process unblocks (waiting → ready)

- If the scheduler is only invoked under 1 and 2, it is non-preemptive
  - Running process decides if/when to enter scheduler

- Otherwise, it is pre-emptive
  - OS can force scheduler entry
Pre-emptive vs Non-preemptive

- **Non-preemptive** scheduling
  - Typically uses an explicit `yield` system call or similar so running process can enter the scheduler, alongside implicit yields when, e.g., performing IO
  - Simple to implement: no timers required, process holds CPU as long as desired
  - Open to denial-of-service: malicious or buggy process can refuse to yield

- **Pre-emptive** scheduling
  - Hardware support for regular timer interrupts required to ensure scheduler entered
  - Precludes denial-of-service: the OS simply pre-empts a long-running process
  - More complex to implement: Timer management, concurrency issues

- Almost all modern schedulers are **pre-emptive**
Scheduling Criteria

• Typically there will be more than one process *runnable* – how to decide which one to pick?

• Many different metrics may be used, with different trade-offs and leading to different operating regimes

• Data structures introduce time and space overheads
  • …of measurement and computation for the metric
  • …of selecting the “best” next process
Scheduling Criteria

• **Turnaround time**, minimising the time for any process to complete
  • Aims to minimise total time from process submission to completion across all states

• **Waiting time**, minimising the time a process sits in the Ready queue
  • Scheduler only controls time in the Ready queue – rest is up to the process
  • But may penalise IO heavy processes that spend a long time in the wait queue

• **Response time**, minimising the time to *start* responding
  • In interactive/time-sharing systems, users may prefer to total efficiency
  • But may penalise longer running sessions under heavy load
Scheduling Criteria

• **CPU utilisation**, maximising the time the CPU is actively in use
  - Aims to keep the (expensive) CPU as busy as possible
  - But may penalise I/O heavy processes as they appear to leave the CPU idle

• **Throughput**, maximising the rate at which processes complete execution
  - Aims to get useful work done at the highest possible rate
  - But may penalise long-running processes as short-run processes will be preferred

• Typically want to maximise utilisation and throughput, and minimise turnaround, waiting and response times
  - ...but what exactly – optimise the average? Minimise the maximum?
  - What about the distribution, e.g., variance, confidence intervals?
Outline

• Queues
• Scheduling
• Multiple processor scheduling
  • NUMA
  • Load balancing, multicore, virtualisation
Multiple processor scheduling

• Everything becomes more complex when multiple CPUs are available
  • Assume homogeneous processors within a multiprocessor
• Asymmetric multiprocessing
  • Only one processor accesses the system data structures
  • Alleviates the need for data sharing
• Symmetric multiprocessing (SMP) – currently the most common
  • Each processor is self-scheduling
  • All processes can be in a single ready queue, or each processor has its own private ready queue
• Processor affinity when a process has affinity for which processor it runs
  • Soft affinity indicates preference
  • Hard affinity indicates constraint
  • Variations including processor sets
Non-Uniform Memory Access (NUMA)

- Affects CPU scheduling as it means different CPUs have faster or slower access to parts of memory
  - E.g., because have combined CPU and memory boards
- Memory placement then affects affinity
- Costs of switching to a different CPU could be very much higher than without NUMA
Load balancing, multicore, virtualisation

• SMP means OS needs to keep all CPUs loaded for efficiency

• **Load balancing** attempts to keep workload evenly distributed
  
  • **Push migration** has a periodic task check load on each CPU and push tasks off overloaded CPUs onto other CPUs
  
  • **Pull migration** has idle CPUs pull waiting tasks off busy CPUs

• Recent trends include
  
  • **Multicore**, placing multiple CPU cores on same physical chip, increasing speed and efficiency
  
  • **Hyperthreading**, increasing the number of threads per core so that one thread can make progress while another is stalled on memory read
  
  • **Virtualisation** challenges OS scheduler as hypervisor and guests are all scheduling against each other
Summary

• Queues
  • CPU I/O burst cycle
  • CPU scheduler vs job scheduler
  • Idling

• Scheduling
  • Dispatcher
  • Pre-emptive vs non-preemptive
  • Criteria

• Multiple processor scheduling
  • NUMA
  • Load balancing, multicore, virtualisation