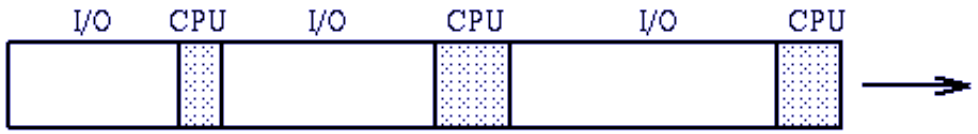
CPU Scheduling, Threads, & Processes

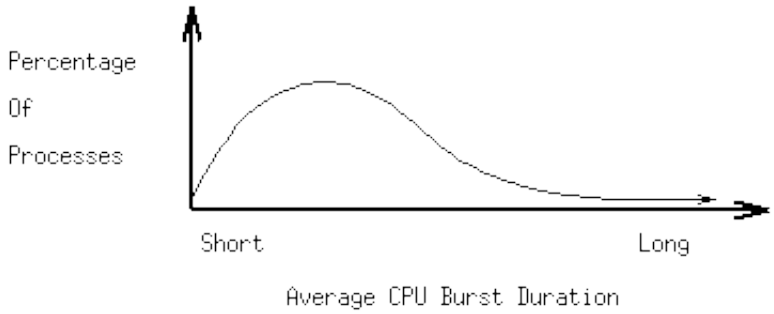
- What is CPU Scheduling?
- Context Switches
- Threads
  - Thread Implementation
  - Threads on multi-core systems
  - CPU Scheduling for threads
- Processes
  - Process Scheduling Algorithms
  - Multilevel Queues

What is CPU Scheduling?

- CPU scheduling (or *multiprogramming*) is when the OS switches the CPU between programs in memory. Doing so increases:
  - CPU Utilization
  - Throughput (amount of useful work CPU does / unit time)
- Processes typically execute in cycles of CPU "bursts" and I/O "bursts"



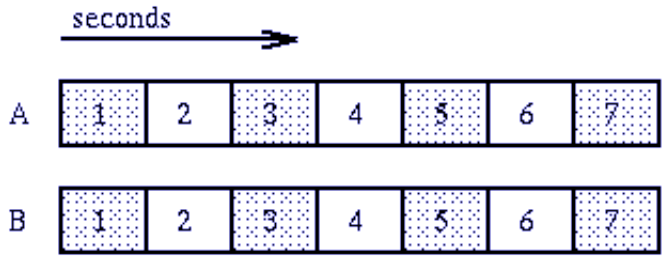
- Studies have shown that a processes CPU bursts are usually of small duration as compared to its I/O bursts I/O devices are very slow.



Multiprogramming Example

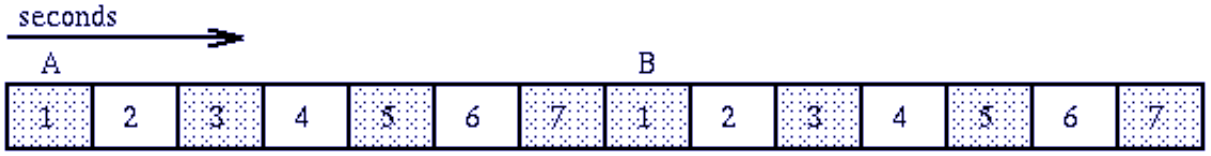
- Consider interleaving 2 processes with identical burst lengths

Grey area = useful work



White area = time waiting for I/O device

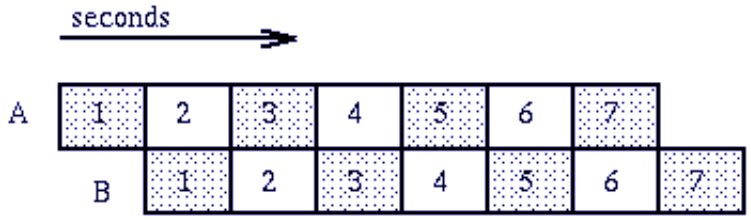
- If we run A, and then B, the total execution time is 14 secs



- Throughput = 2 jobs/14 s

CPU Utilization = 8 / 14 = 57%

Ex: interleaved processes



not realistic because we're assuming there's no time inbetween switching processes

- Throughput = 2 jobs/8s

CPU Utilization = 8/8 = 100%

Context Switches

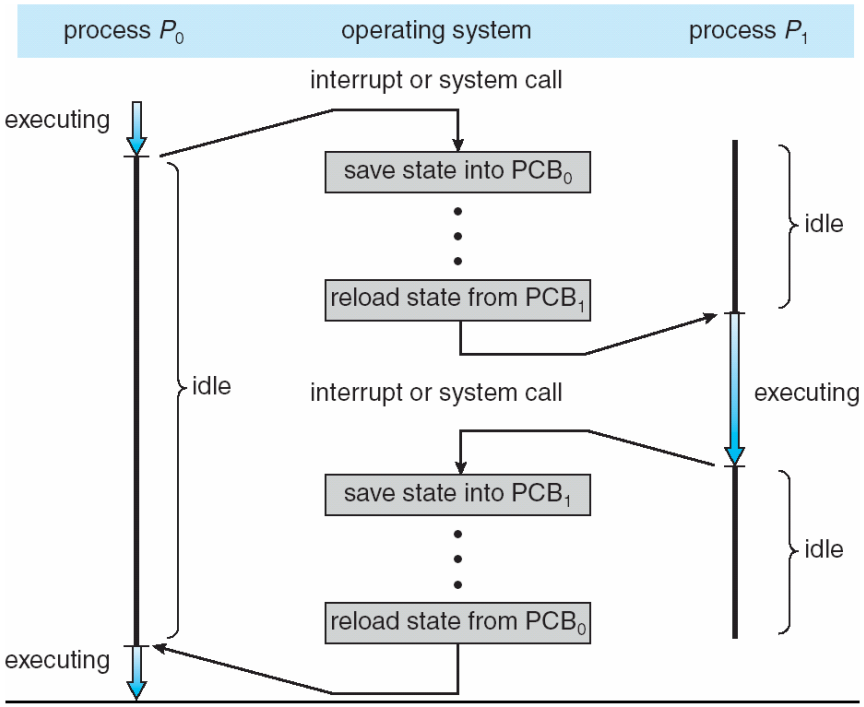
- A *context switch* is when the CPU is switched from one process to another
- For the process being taken out of the CPU, the OS must save the following information in the *process control block* to resume execution later:
  - General purpose registers (data and address), stack pointer register, program counter register
- The following information must either not be altered while the process is suspended, or must be saved:
  - Main memory occupied by the process (code and data)
  - Process stack (local variables, return addresses, subroutine parameters, etc.)

must protect process memory in some way

Context Switching Example

- To reactivate a process, all saved values are reloaded, with the program counter loaded last of all

each process has its own PCB



kernel saves everything into PCB

last thing make sure program counter is up to date

- Time taken during context switch is PURE OVERHEAD - no processes are running

no progress made on any process

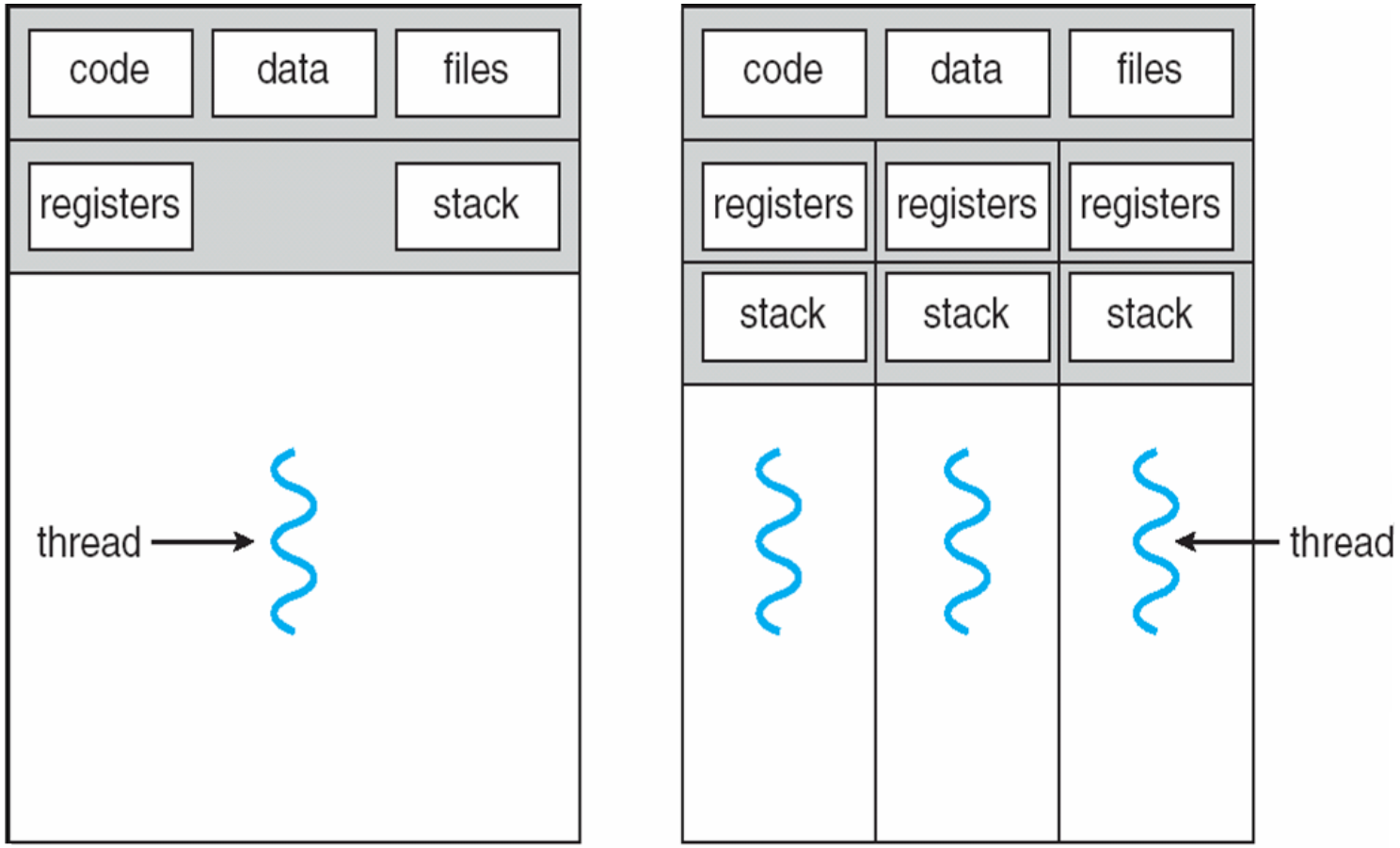
Chapter 4: Threads & Concurrency

- Overview
- Thread Implementation
- Threads on multi-core systems
- CPU Scheduling for threads

Threads

- Consider taking a single application and decomposing it into multiple processes to speed up execution
  - Problem: context switches take up time/resources
  - Solution: make context switches faster!
- One way to do this is to reduce the things we save at context switch time
  - only save PC and other registers
  - do not swap or save the memory of the process because all the threads share the memory
- This means that all of the *lightweight processes* or *threads* would share the same memory address space!

Single and Multithreaded Processes



single-threaded process

multithreaded process

Benefits of Threads

- *Responsiveness*: one thread blocking does not stop the process from doing other useful work
  - E.g. a multithreaded web browser can do user interaction in one thread and load an image in another
- *Economy*:
  - Creating new threads is easy compared to creating new processes
    - E.g we do not have to allocate much memory for a new thread
  - Context switches are faster

Benefits of Threads

- *Resource Sharing*: IPC is simplified since all threads share memory (data and code)
- *Scalability*: in a multiprocessor environment threads can run on different processors
  - A single-threaded process can only run on one processor
  - Having multiple threads increases parallelism

Multithreaded Server Architecture