CS 310 Operating Systems

Lecture 4: Process – Multiprogramming, Multiprocessing, Boot Sequence

Ravi Mittal
IIT Goa

Acknowledgements !

- Contents of this class presentation has been taken from various sources. Thanks are due to the original content creators:
  - Class presentation: University of California, Berkeley: David Culler, Anthony D. Joseph, John Kubiatowicz, AJ Shankar, George Necula, Alex Aiken, Eric Brewer, Ras Bodik, Ion Stoica, Doug Tygar, and David Wagner.
  - Operating Systems: Principles and Practice (2nd Edition) Anderson and Dahlin
    - Volume 1, Kernel and Processes
      - Chapter 2

Use the following for your learning..

- Operating Systems: Principles and Practice (2nd Edition) Anderson and Dahlin
  - Volume 1, Kernel and Processes
    - Chapter 2
- https://inst.eecs.berkeley.edu/~cs162/su21/
  - CS 162, UCB
- https://youtu.be/itfEcA3TXq4

We will study..

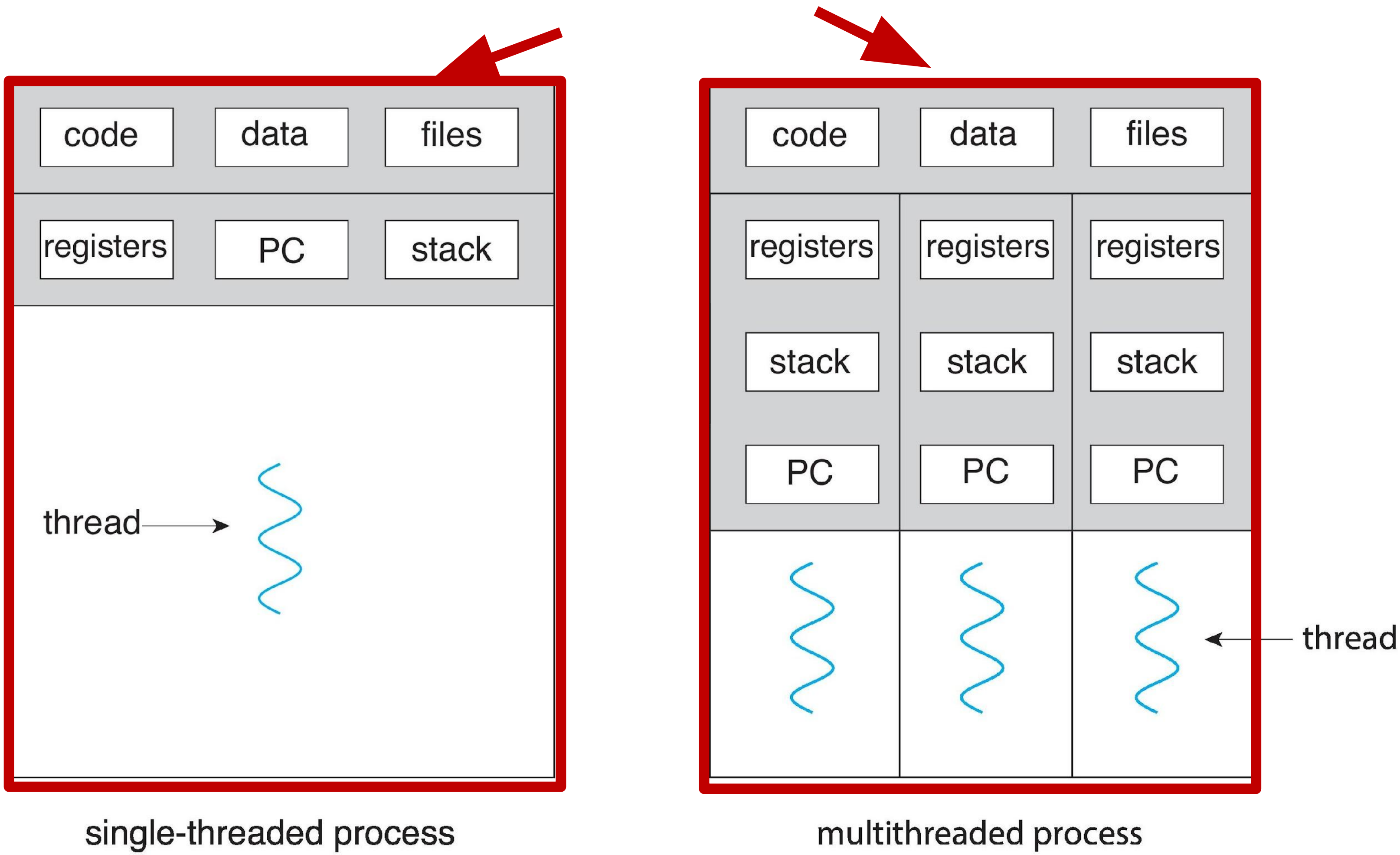
- Multiprogramming, context switching, Multiprocessing
- Process – Creation

Concepts we learnt so far

Four Fundamental OS Concepts

- **Process: an instance of a running program**
  - Protected Address Space + One or more Threads
- **Address space**
  - Set of memory addresses accessible to program (for read or write)
- **Thread: Execution Context**
  - Fully describes program state
- **Dual mode operation / Protection**
  - User / Kernel mode

Single and Multithreaded Processes

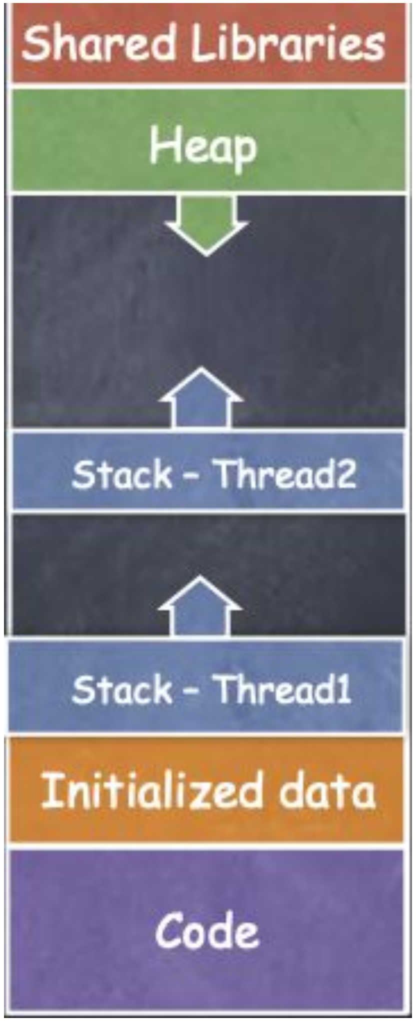


- Threads encapsulate concurrency: “Active” component
- Address spaces encapsulate protection
- Threads share memory; In processes – sharing memory is complex

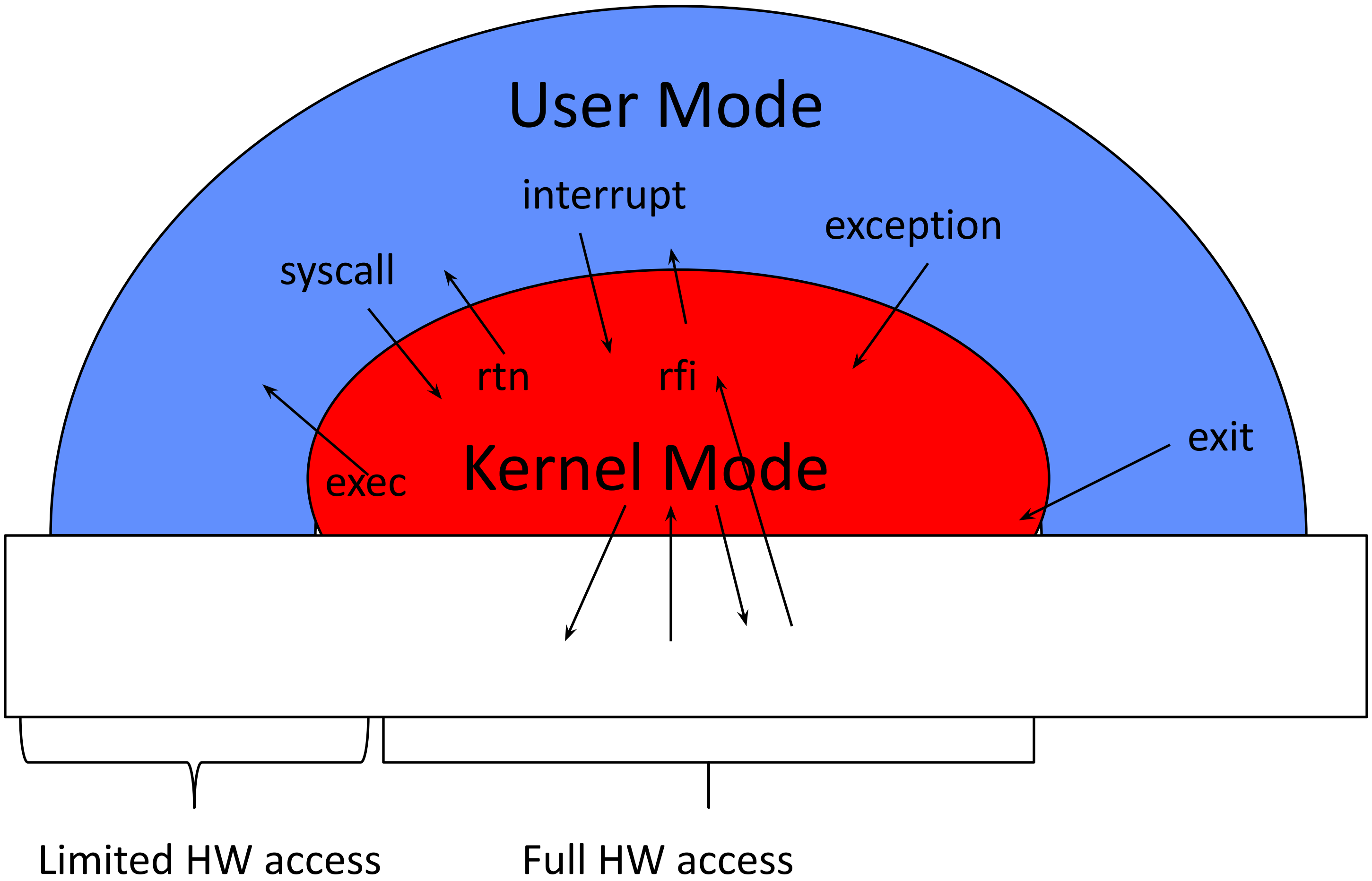
Thread

- All threads within a process share
  - Heap
  - Global/static data
  - Libraries
- Each thread has a separate
  - Program Counter
  - Stack
  - Registers
- Note two threads of the same process

Process Address Space



User/Kernel (Privileged) Mode

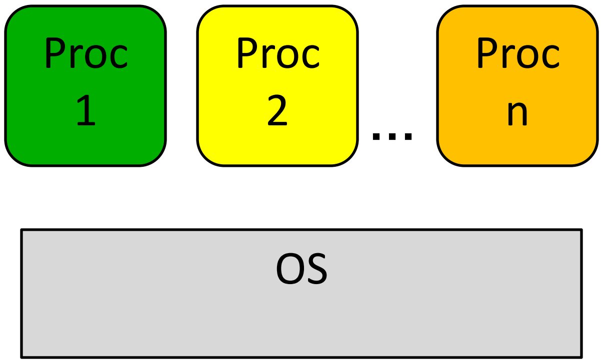


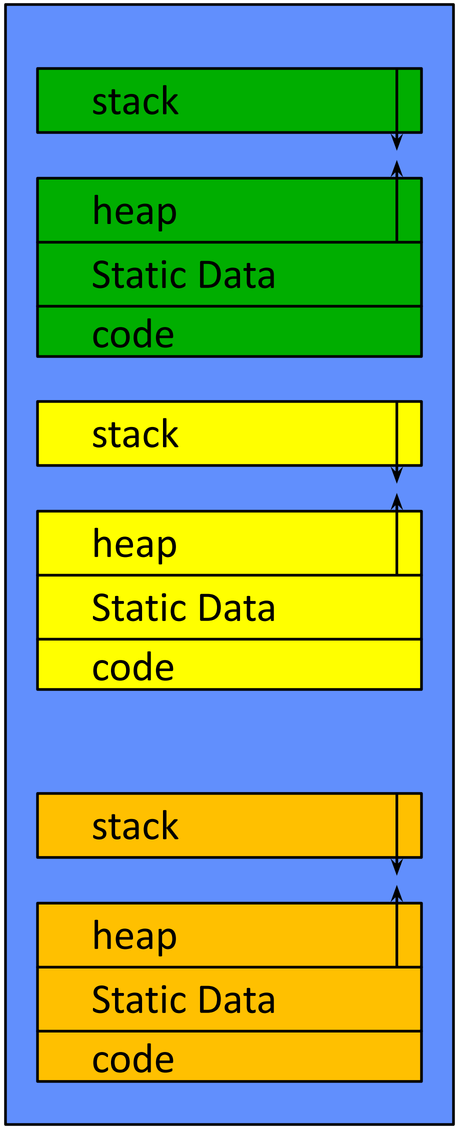
Multiprogramming, Context Switching, and Multiprocessing

Multiprogramming

- On a modern computer, there are hundreds of different processes running simultaneously -- but only a few CPUs
- In the period of microseconds, the OS rapidly switches between all processes to allow each process to run on a CPU
  - In the multi-programming system, one or multiple programs can be loaded into its main memory for execution
- When the OS swaps out one process from one CPU and allows a new process to run, this is called a context switching

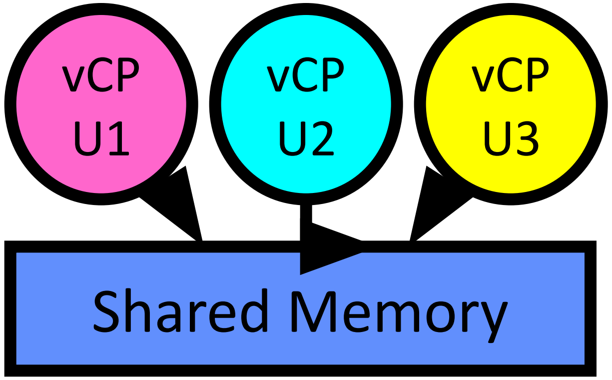
Multiprogramming

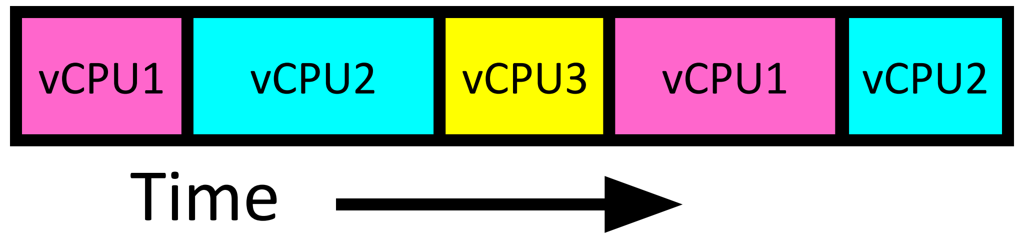




OS timeshares CPU across multiple Processes

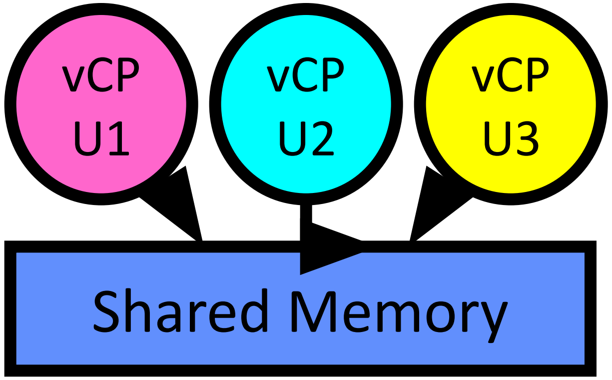
Illusion of Multiple Processors

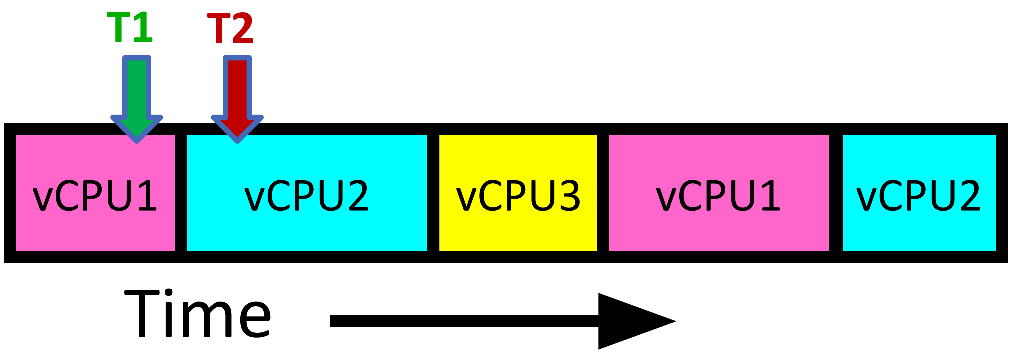




- Assume a single processor (core). How do we provide the illusion of multiple processors?
  - Multiplex in time!
- Threads/processes are *virtual cores*
- Contents of virtual core (thread/process):
  - Entire address space: Process
  - Program counter, stack pointer, Registers : Thread
- Where is “it” (the process / thread)?
  - On the real (physical) core, or
  - Saved in chunk of memory – PCB or TCB

Illusion of Multiple Processors (Continued)





- Consider:
  - At T1: vCPU1 on real core, vCPU2 in memory
  - At T2: vCPU2 on real core, vCPU1 in memory
- What happened?
  - OS ran, triggering context switch
  - Saved state in control block (for Process PCB ; For Thread: TCB)
  - Loaded PC, SP, … from vCPU2's processor block, jumped to PC
- What triggered this switch?
  - Timer, voluntary yield, I/O, other things
- OS has a CPU scheduler that picks up on of the process/thread
  - Policy: Which Process
  - Mechanism: How to context switch

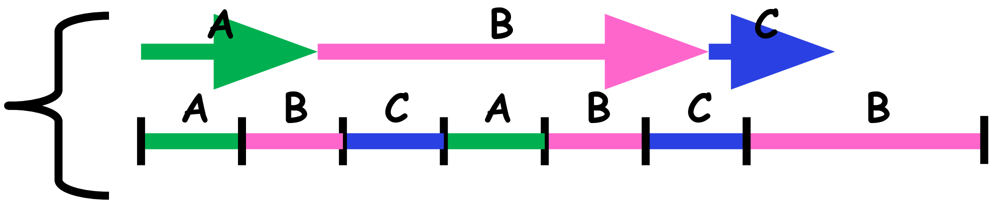
What is required during context switching?

- CPU
  - Save all CPU registers
- Caches
  - Save all CPU caches specific to the process
- Page Table
  - Page tables are unique to each process
  - Switch to the new page table (for the process to be executed)
- Overall Cost
  - Expensive
  - OS must minimize time for context switching
    - A lot of progress is made in the few decades
      - Still it is a few microseconds

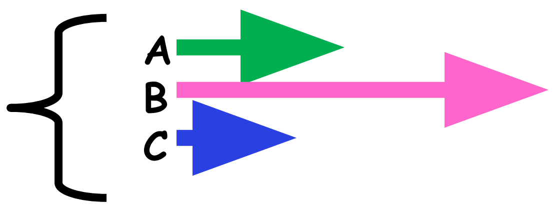
Multiprocessing vs. Multiprogramming

- Multiprocessing: Multiple CPUs(cores)
- Multiprogramming: Multiple jobs or multiple processes
- Multithreading: Multiple threads per process

Multiprogramming



Multiprocessing

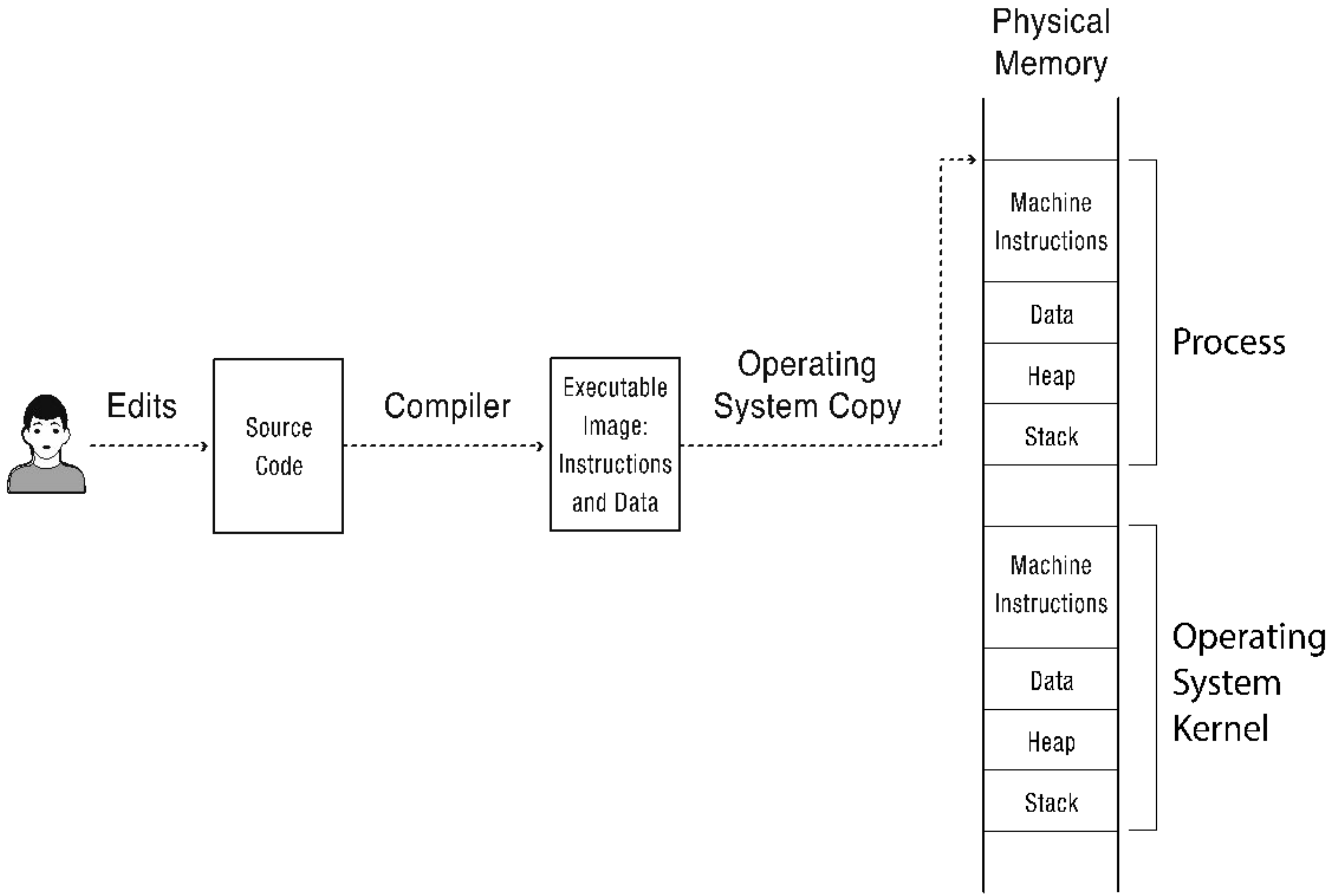


Concurrency is not Parallelism

- Concurrency is about handling multiple things at once
- Parallelism is about doing multiple things *simultaneously*
- Example: Two threads on a single-core system...
  - … execute concurrently …
  - … but *not* in parallel
- Parallel => concurrent, but not the other way round!

Process - Basics

The Process abstraction (Repeat)

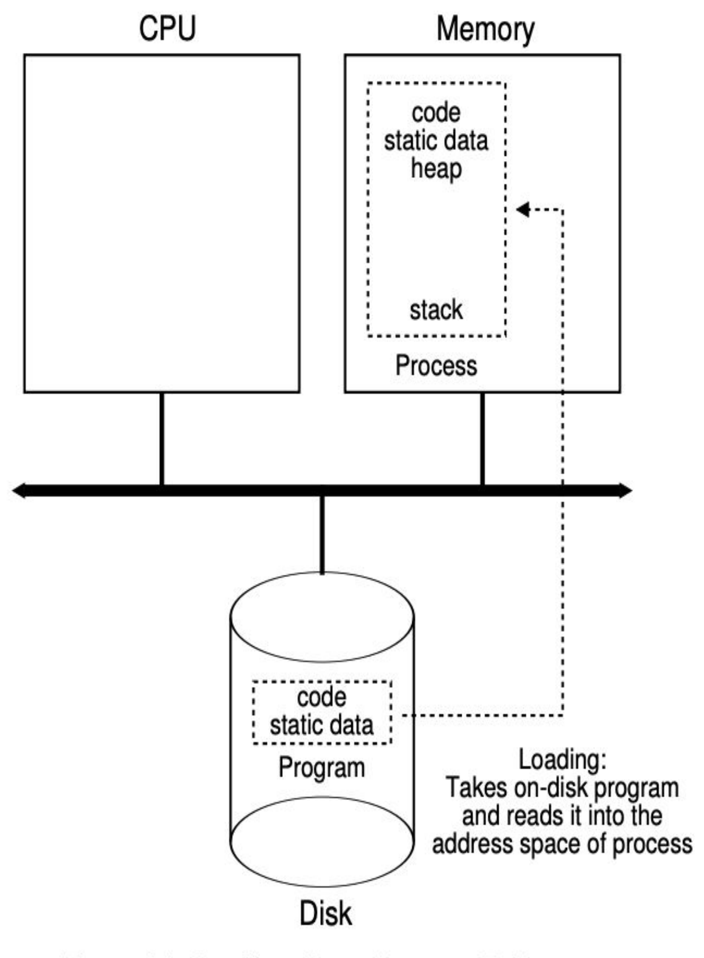


The Process Abstraction

- Compiler
  - converts the code into machine instructions and stores them into a file – executable image
  - Also defines any static data that the program needs along with its initial values – include them in the executable image
- To Run the program
  - OS copies the instructions and data from the executable image into physical memory
  - OS also sets aside a memory region for execution stack – that hold local variables during execution and a memory region called heap - to hold any dynamically allocated data structures
- Note that the OS must already be in the memory so that it can copy executable file into the memory
  - With its own stack and heap

Process Creation

- OS allocates memory and creates memory image
  - Loads code, static data from disk executable (eg a.out)
  - Creates and initialized runtime stack
  - Create heap
- Opens basic files
  - Standard Input, Output, Error
    - Standard Input Output let programs read input from terminal and print output to screen
- Initializes CPU registers
  - PC points to first instruction



Kernel code/data in process Virtual Address Space?

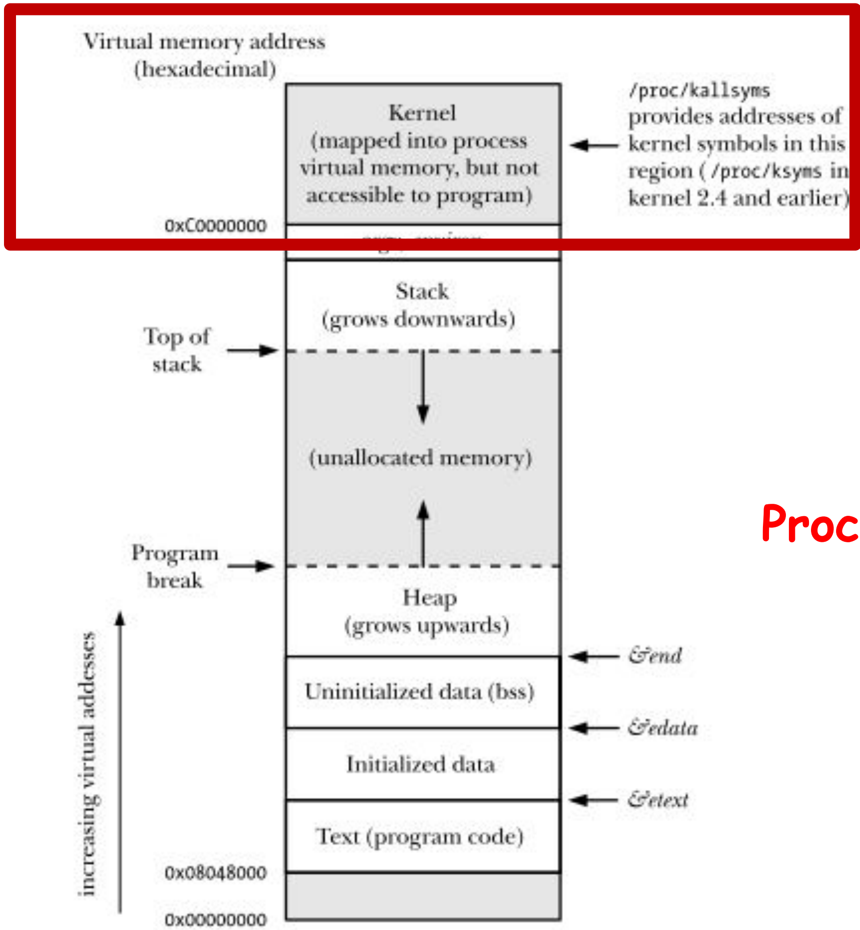


Figure 6-1: Typical memory layout of a process on Linux/x86-32

Process image

- Unix: Kernel space is mapped in high - but inaccessible to user processes

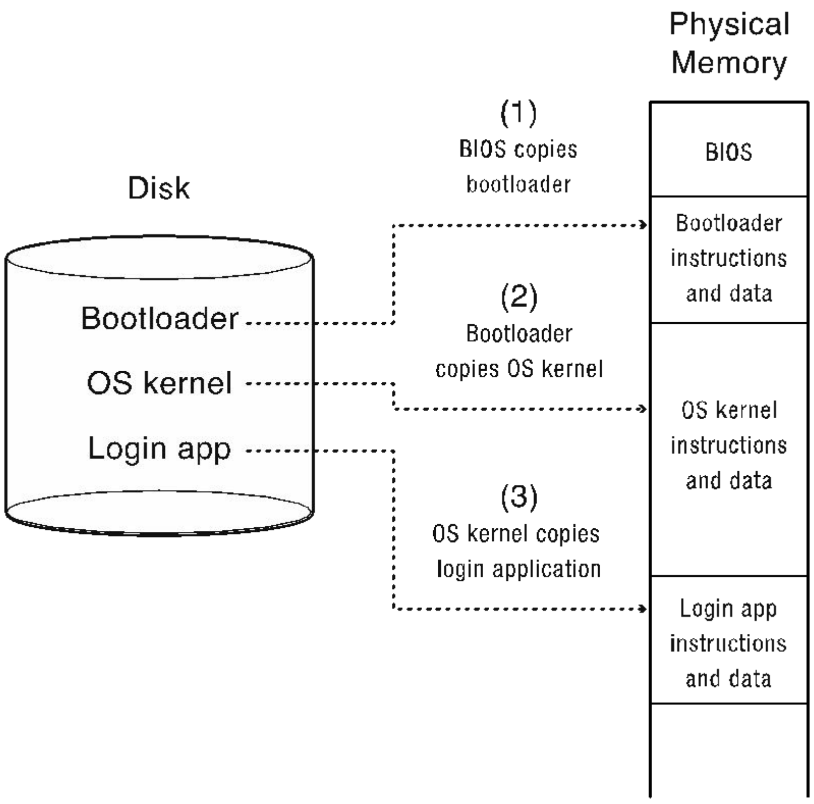
When you switch on computer – fresh start

- When you press power button
- RAM is empty
  - It does not contain OS
    - OS is in disk in the beginning
- Goals of a computer at boot
  - Ensure a minimal set of hardware is functional
    - CPU, Screen, Hard Disk, Keyboard
  - Hand control over to the OS
    - Let OS handle remaining operations

Boot Sequence (1)

- Set PC to start execution at ta pre-determined position in memory
- System uses a special read-only hardware memory (Boot ROM)
  - Boot ROM stores boot instructions
  - On x86, the boot program is called BIOS: Basic Input/Output System
    - ROM instructions are fixed at the time of manufacturing (computer) ; they are never changed
      - Hence, it is not good to keep entire OS in the ROM
      - OS needs frequent updates for bugs and security fixes
    - ROM storage is slow and expensive
    - So, small amount of code is kept in BIOS in ROM

Booting Sequence (2)



Boot Sequence (3)

- The BIOS reads a fixed-size block from a fixed position on disk (or SSD) into memory
  - This block is called bootloader
- Once the bootloader is copied to memory
  - It jumps to the first instruction of the bootloader
- Bootloader in turn loads the kernel into memory
    - (Note that Kernel's executable image is usually stored in the file system (on disk))
  - The bootloader jumps to the Kernel
- Note that
  - BIOS just reads a block of raw data from disk
  - Bootloader, in turn, needs to know how to read kernel image (as a file) in the file system

Boot Sequence (4)

- Kernel initializes data structures
  - Setting up Interrupt vector table
    - To point to various interrupts, processor exceptions, sys call handler
- Kernel starts the first process – typically the user login page
  - OS reads login program code from it's disk location and jumps to the first location of the program

Lecture Summary

- In the multi-programming system, one or multiple programs can be loaded into its main memory for execution
- Multiprocessing – multiple jobs parallelly running
- Multithreading – Multiple threads per process

- Starting computer system involves
  - Pointing PC to fixed location in Boot ROM
  - Boot ROM loads Bootloader image from disk into memory
  - Bootloader, in turn, loads OS Kernel image from hard disk to memory
  - OS kernel starts running
- A process has following states: New, Waiting, Ready, Running, and Terminated – will study in the next class