COMP30023 – Computer Systems

Operating systems: Process Management

Olya Ohrimenko

Outline

- Last lecture: Operating Systems Overview
- Introduction to Processes and Process Management
- Prelab

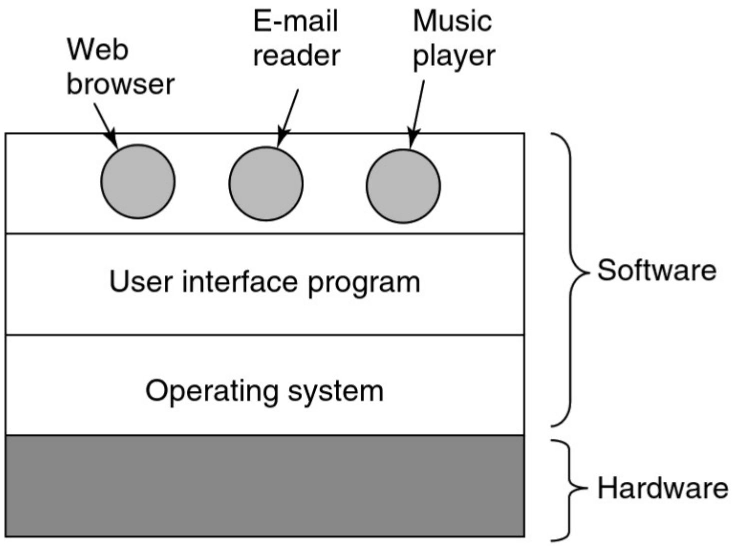
Prelab: Common issues

- Not putting "International" if you're outside of Australia
- The authenticity of host '...' can't be established.

  RSA key fingerprint is ....
  Are you sure you want to continue connecting (yes/no)? yes

- Not specifying the correct (lowercase) username and ip
- Permission denied errors - not specifying the private key (public key instead), corrupt private key
- Connect to university network or configure VPN

Operating system (OS)



TB 1-1

One or more processors
Main memory
Disks
Printers
Keyboard
Mouse
Display
Network interfaces
I/O devices

- Provides an abstraction of hardware resources
- Manages these resources

Processor/CPU

- Central processing unit (CPU)
- Fetches instructions from registers (memory); executes them
- Registers keep some process state information
- Two modes of execution:
  - Kernel (execute any CPU instruction and reference any memory address) e.g., initiate I/O
  - User (only a subset) e.g., add, sub

Processes

- A **program** is static; a **process** is dynamic.
- At any given time, the number of processes running a given program can be 0, 1, 2, 3, ...
- The state of a process consists of the code or “text” of the program, the values of all the variables, both in memory and in registers, the address of the current instruction, and probably some other data as well (e.g. current directory).
- As an analogy, consider cooking:
  - A recipe is a static entity; it is analogous to a program.
  - The act of cooking is analogous to a process.

Processes vs. processors

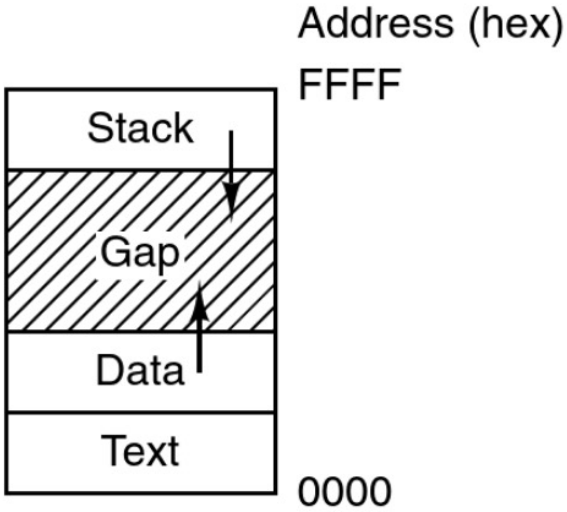
How many processes do you think your machine is running?

How many CPUs does your computer have?

Try ps -ef, lscpu, activity monitor etc. depending on your machine.

Processes

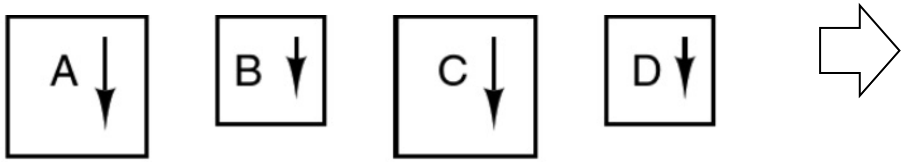
- Program in execution
- Each process has its own address space
- Process memory has three segments:
  - text (program code, usually “read only”)
  - data (constant data strings, global vars)
  - stack (local vars)

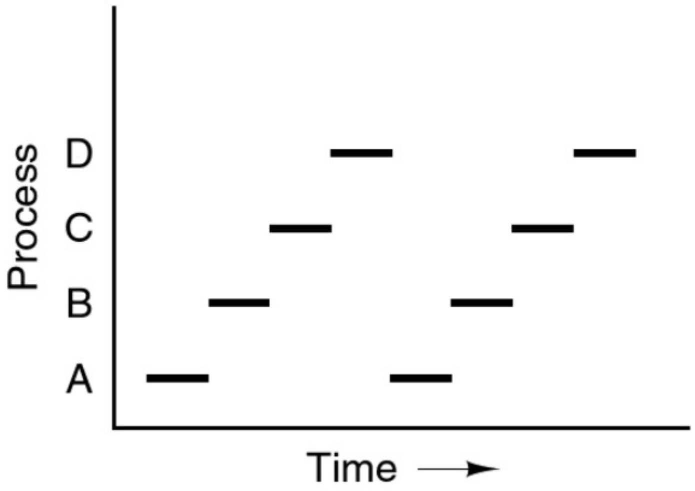


TB 1-20

Multiple processes

4 processes running in “parallel”:



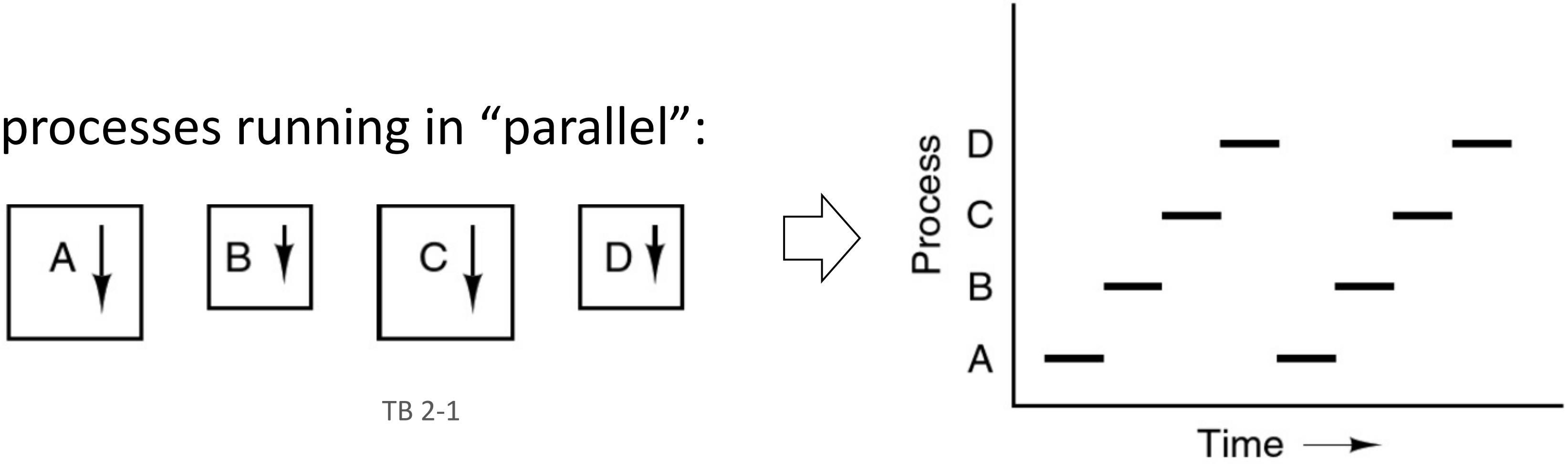


TB 2-1

- Conceptually each process has its own **virtual** CPU.
- In reality, multiple processes will share a CPU, each running for a small period of time in turn. This is called **multiprogramming**.
- Introduced in 1960s: use CPU while waiting for I/O

Multiprogramming

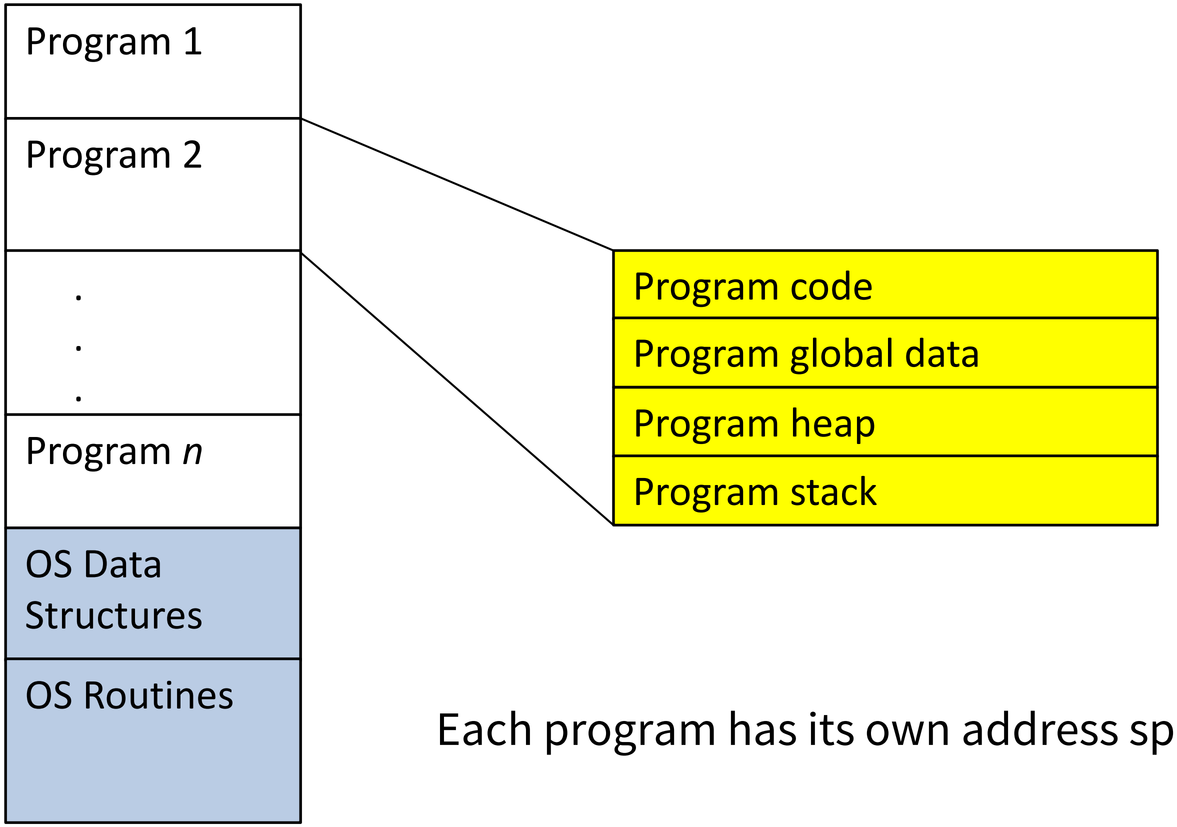
4 processes running in “parallel”:



TB 2-1

- Multiprogramming increases system **efficiency**. When one process needs to wait for e.g. data from disk or keyboard input, another process can make use of the CPU.
- Multiprogramming is useful even when the machine has two or more CPUs, since (for the foreseeable future) the number of processes will sometimes exceed the number of CPUs.

Multiple processes – a simple illustration


Program 1

Program 2

.
.
.

Program *n*

OS Data Structures

OS Routines

Program code

Program global data

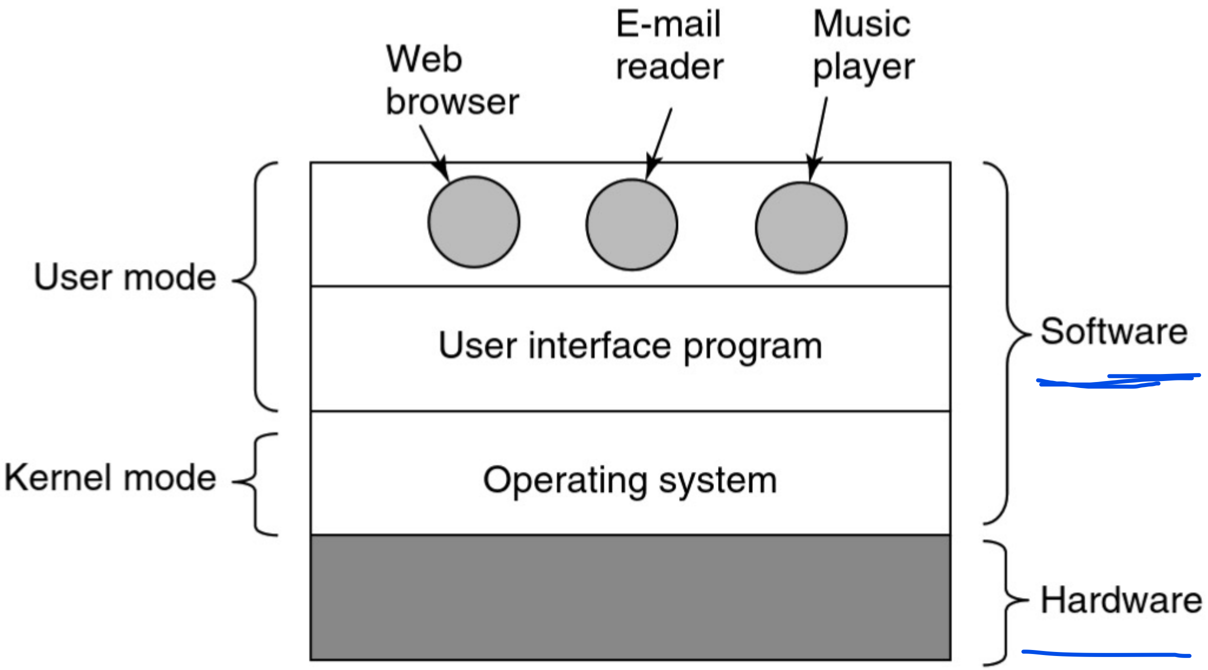
Program heap

Program stack

Each program has its own address space

Operating system as a process

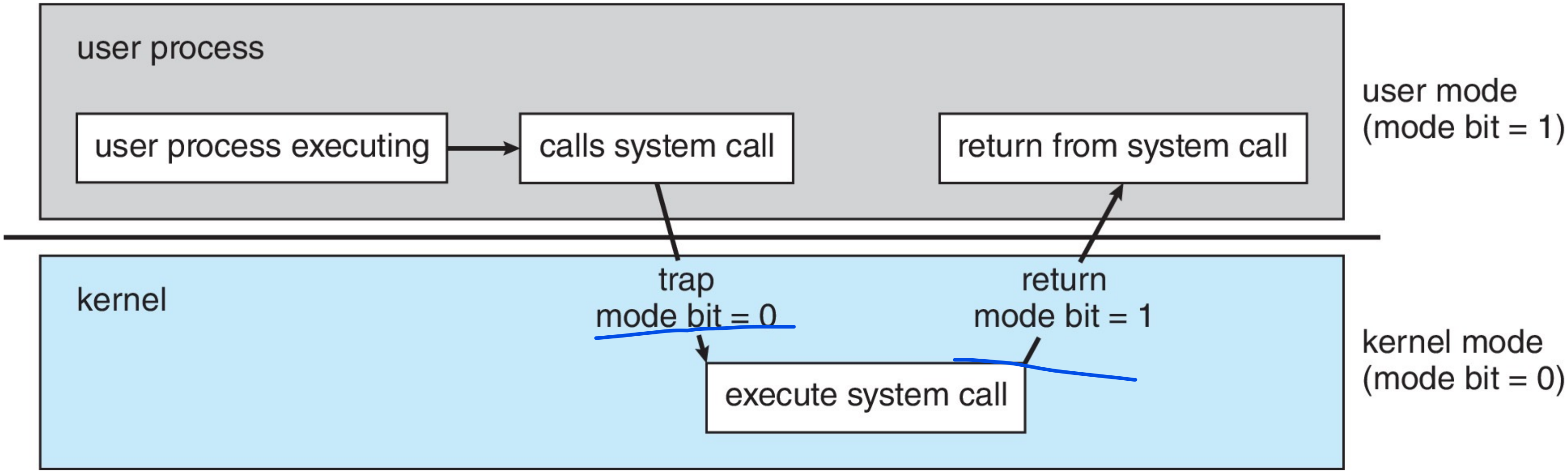
- If several processes are to be active at the “same” time, something has to ensure that they do not get in each other’s way.
- It provides services such as “read N bytes from this file”. These services are used by application programs, utilities, and by the non-privileged parts of the operating system.


Web browser
E-mail reader
Music player
User mode
User interface program
Software
Kernel mode
Operating system
Hardware

User-kernel mode distinction

- Most CPUs have two modes (some have more). The program status word (PSW) register gives the current mode.
  - Code running in user mode cannot issue privileged instructions, and can access only the parts of memory allowed it by kernel mode code.
  - Code running in kernel mode can issue all instructions, and can access all memory.
- Instructions and memory locations whose use could interfere with other processes (e.g. by accessing I/O devices) are privileged.
- The user mode / kernel mode distinction is the foundation needed by the OS for the building of its security mechanisms.

Transition from User to Kernel mode



Mode bit (in program status word register) provided by hardware. It provides the ability to distinguish when the system is running in user mode or kernel mode.

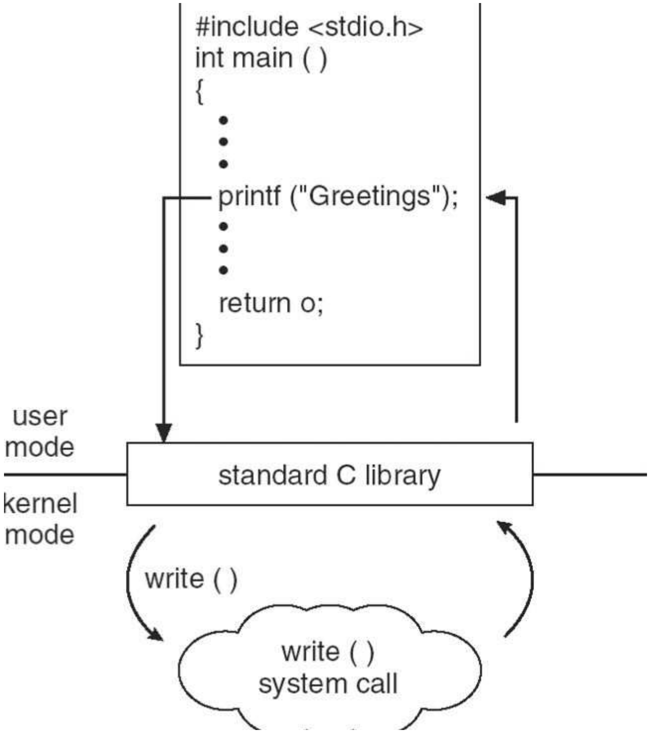
System calls

- System calls exist to allow user programs to ask the OS to execute privileged instructions and access privileged memory locations on their behalf. Since the OS checks those requests before executing them, this scheme preserves system integrity and security.
- To make system calls more convenient to use, a system call will typically do several privileged things in carrying out one logical operation, e.g. reading N bytes from a file on disk.
- To the application programmer, a system call is a call to a privileged function.
- Example system calls on Unix:
  - open, read, write, close
  - fork, exec, exit, wait

Typically system calls (OS independent list)

- Process control, e.g.. load/execute; create and terminate a process; get/set process attributes
- File management e.g.. create/delete/open/close/read from a file; get/set file attributes
- Device management e.g.. request/release a device; read/write from a device
- Information maintenance e.g.. get/set time or date
- Communication e.g.. create/delete communication connections.

System call example



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return o;
}
```

- Standard C library handling of write(). The library provides a portion of the system-call interface for many versions of Unix and Linux.

Summary

Processor vs. process

What is a process

Multiprogramming concept

User vs. kernel mode

Next:

- This class part 2: Source control (Git)
- Week 2: process lifetime, scheduling and interprocess communication

Acknowledgement

- The slides were prepared by Olya Ohrimenko based on material developed previously by: Lachlan Andrew, Chris Culnane, Michael Kirley, Zoltan Somogyi, Rao Kotagiri, James Bailey and Chris Leckie.
- Some of the images included in the notes were supplied as part of the teaching resources accompanying the text books listed in lecture 1.
  - (And also) Figures 1.10, 2.6, 2.11 of Modern Operating Concepts

- References: TB 1.3.1, 1.5.1, 1.6 (before 1.6.1), 2.1, 2.2.1

Operating system concepts

- Processes (running program)
- Address spaces (memory)
- Files and file system
- Input and Output
- Protection (manage security)