CS-202 Week 1

Notes

Lecture 1

Admin & Goals

Com Sys is a new course replacing the following:

- Introduction to Computer Systems, CS-323
- Computer Networks, COM-208
- Programmation Orienté Systems, CS-207
- Projet de programmation orienté Systems, CS-212

Course Aims:

- Understanding timeless principles of os and networking
- Understand the difference between application-oriented and system-oriented thinking
- Separate the ends, understand middle-boxes and routers
- Build hands on experience

Requirements:

- Linux os or VM

Structure:

- Main Lectures:
  - 4 hours of non recorded / streamed lectures
  - Slides available before-hand
  - Summary videos available
  - Weekly graded quiz to check understanding
- C Bootcamp (first 3 weeks):
  - Flipped teaching
  - 3 weeks theory -> practice
  - 2 person graded programming project starting week 4

The role of the operating system

Operating System (OS) is a special type of program, it should never stop and never fail. It has a trust value.

Hardware Foundations of Computer Systems

The von Neumann Architecture:

CPU:

- Arithmetic/Logic Unit: Does operations

- Control Unit: Controls what happens next
- has at least one register, the instruction pointer (IP):
  - interprets instructions at address IP
  Programs are stored in memory

Protection and memory management units

Key enhancements to the von neumann:

CPU privileges:

- user mode
- kernel mode (special access)

Virtual memory with a Memory Management Unit (MMU):

- MMU converts virtual adresses to physical addresses

CPU operates with virtual addresses only:

- Instruction pointer is a virtual address
- All pointers are virtual addresses

What is an OS

An OS is a software that **interfaces** between the physical hardware ressources and the one or many apps running on the machine. Implements **abstraction** thats is easier to program than the raw hardware.

OS wears three main hats:

- Referee: manages protection, isolation and sharing of resources
- Illusionist: Provide abstraction of physical ressources
- Glue: Provide a set of common ressources

The OS as a referee

Falut isolation:

- Isolate programs from each other and from OS
  Ressource sharing:
- Choosing which program to execute
- Splitting physical ressources for programs
  Communication:
- Primitives that allows communication among programs

The os as a glue

Makes sharing easier
Maximise reuse:

- avoid re-implementation
- decouple hardware and software developement

- [x] ~~re-read class ✅ 2024-03-12~~

Lecture 2

2.1 - The process abstraction

Focus of lecture
the need of a process abstraction
what is a process
how os creates a process

Abstractions provide an **interface** to application programmers that separates **policy** - what the interface commits to accomplishing - from **mechanism** - how the interface is implemented

Executable file

Contains:

- Executable code: CPU instructions
- Data: Information manipulated by these instructions

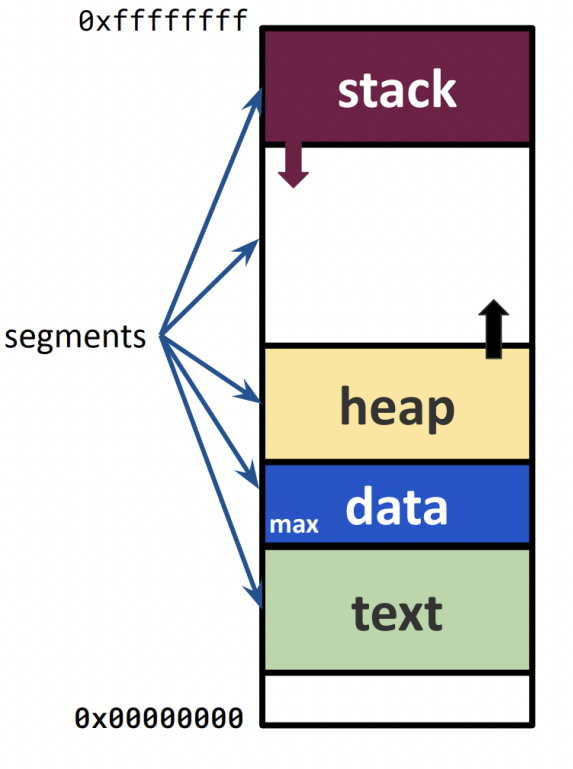
What constitutes a process ?

- A unique identifier: Process ID (PID)
- Memory image (contains 4 *segments):
  - Code and data (static)
  - Stack and heap (dynamic)
- CPU context: registers
  - Program counter, current operands, stack pointer
- File descriptors: pointers to open files and descriptors

Running a program -> creating a process
a process is an instance of an executable

The register file of a process can be in memory or in CPU but not in both at the same time.

What is in the process memory



Stack

- Temporary data like function parameters, local vars and return adresses
- Grows from higher to lower adresses
- How does Stack Pointer (SP) know when it has reached heap ?
  - OS protects SP from pointing to heap
  - Stack overflow: when stack grows beyond allocated memory due to to many function calls

Heap

- Used for dynamic memory allocation during program runtime

Data

- Statically allocate global variables and program data structures

Text

- Read-only
- Contains code and constants
- These are program executable machine instructions

Stack is necessary because of recursion

Heap is where we allocate objects that can live outside a function
Null pointer:

How to OS creates a process

- Loading: OS loads the static code and data into memory
- Memory Allocation: allocate process memory regions (heap and stack)
- Initialization: Initialize tasks related to IO (setting up STDIN, STDOUT, STDERR)

- Ready: OS sets up the stage for running the process by transferring CPU control to beginning of program entry point

Sharing of resources: Two approaches

Time sharing

Running one task at a time and quickly switching between ressources

Space sharing

Each task gets a portion of available space

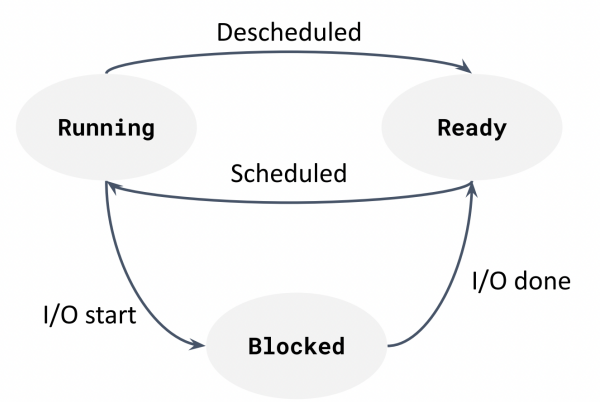
OS time shares CPU among multiple processes

2.2 - Process state execution diagram

State transition

3 states:

- Running: process is running on CPU
- Ready: ready to run but OS has not chosen to run it yet
- Blocked: Not ready to run until some event occurs



OS mechanism for process management

Process control block

Information for each process is stored in a process control block
(slide 27)

Process API

pid 1: init

fork()

- Process calling fork is parent process
- creates a child process which is duplicate of parent process
- returns child pid
- (contine)

exec()

- Often used not long after fork
- Loads a new program in the context of an already running process (child), replacing the previous executable program
- replaces the memory (address space) - code data heap and stack are replaced
- No new PID
- STDIN, STDOUT and STDERR are kept to allow parent to redirect/rewire child's output

wait()

- blocking call
- wait for child to finish executing

exit()

- exits process