Systems programming

4 – Processes introduction

MEEC LEEC MEAer LEAer MEIC-A

João Nuno Silva

TÉCNICO LISBOA

Bibliography

- Time-Sharing Computer Systems John McCarthy, MIT - 1964
- McCarthy, John, et al. "THOR: a display based time sharing system." 1967
  - Proceedings of the April 18-20, 1967, spring joint computer conference.
- Bauer, Walter F. "Computer design from the programmer's viewpoint."
  - 1958, eastern joint computer conference: Modern computers: objectives, designs, applications
- Beej's Guide to Interprocess Communication, Brian Hall
  - Chapters 2
- The Linux Programming Interface, Michael Kerrisk
  - Chapter 24

First computers

- Batch processing computers
  - First computers
  - Slow, one task at the time
  - Transistors, magnetic core memory
- 1945 – ENIAC
  - 18,000 vacuum tubes, 7,200 crystal diodes, 1,500 relays, 70,000 resistors, 10,000 capacitors,
  - 150 kW, 27 t
  - 100 word memory, 5000 SUMs per second
- 1957 - IBM 709
  - 100 kW, 1 Ton
  - 32K words memory, 42000 SUMs per second

Batch-processing

- One work at a time
  - Load code, Load data
  - Process
  - Save results
- Repeat
- Other users
  - Wait
- No interactivity
  - Results only at end

1958 Time sharing computing

- The central idea here is that each large metropolitan area would have one or more of these super computers.
- The computers would handle a number of problems concurrently.
  - Input-output equipment installed on their own premises
  - would buy time on the computer
    - the same way we buy power and water from utility companies.
- Tasks are scheduled by a supervisor
  - Each work has a priority

1964 John McCarthy view

- Private computing
  - It is done by time sharing a large computer.
  - Each user has a console that is connected to the computer by a wired channel such as a telephone line.
- The consoles are two kinds,
  - cheap console
    - electric typewriter used for input and output.
  - expensive console
    - includes a cathode-ray-tube unit for the computer to display pictures and text

Computing consoles

- https://www.youtube.com/watch?v=S81GyMKH7zw
- https://www.youtube.com/watch?v=jxkygWI-Wfs

1964 - Time sharing operation

- Operation of such a system as it appears to the user of the typewriter console.
  - When the user wants service
    - he simply starts typing in a message requesting the service.
  - The computer is always ready to pay attention to any key that he strikes
    - just as the telephone system is always ready for youu to lift the receiver of the hook.
  - As soon as the key is depressed on the typewriter
    - a signal is sent to the computer.
  - The effect of the signal is
    - make the computer interrupt the current program after the current instruction
    - jump temporarily to a program that determines what the typewriter wants.

1964 - Time sharing operation

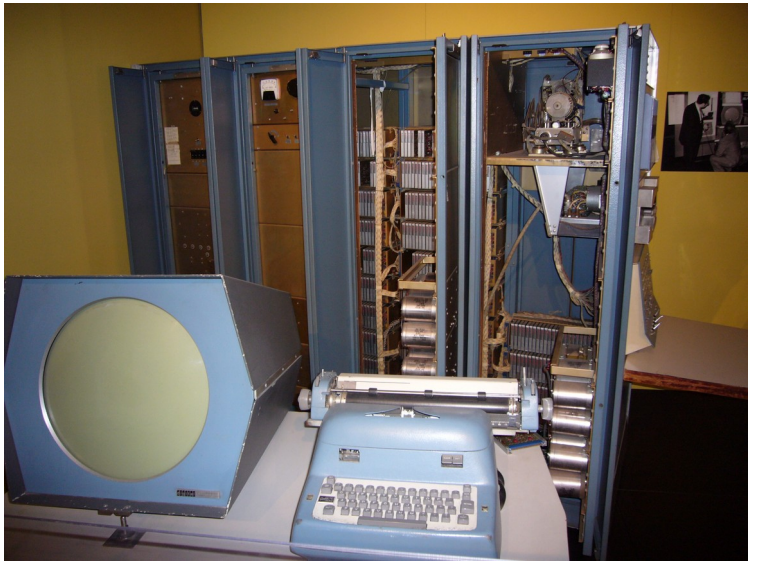
- At any moment some of the typewriters
  - will be inactive
  - some will be in the middle of entering messages into the computer,
  - some will be in the process of typing out characters
  - some will be in a status of wanting programs run.
- This is one example of a time-sharing system.
  - There are a large number of users
  - Each user has his own console,
  - Each user gets service from the computer whenever he desires it
  - Each user has the computer maintain his files for him.

Time sharing requirements

- Large primary memory
- Interruption system to handle errors as well as input and output
- Completely nonstop operation
- Erroneous programs must be prevented from damaging other programs
  - it must interrupt if the program attempts any memory references outside its allotted region,
  - programs that get into endless loops must be prevented from wasting computer time.
- Advances memory management (relocation/virtual)
- Secondary storage enough to maintain users' files.

PDP-1

- Main memory
  - magnetic-core memory
  - 18-bit word size (three six-bit chars)
  - 4096 words as standard main memory
    - <=> 9Kbytes (8bit) 12000 chars
  - upgradable to 65,536 words
- 93458 operations per second
  - (10.7ms/instruction)
- 2,700 transistors and 3,000 diodes
- 730Kg

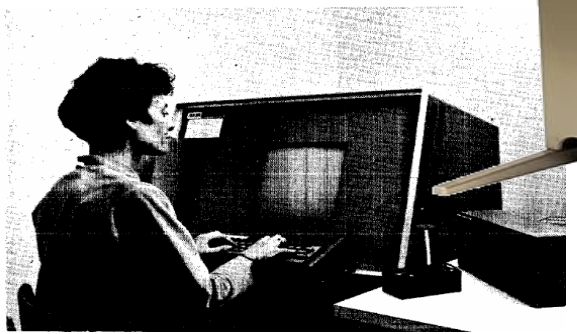


Time-sharing application

- PDP-1
  - 8K Words
  - 22x4Kw drum memory
- BBN time-sharing system
  - increase the effectiveness of the PDP-1 computer
  - for applications involving man-machine interaction
- Allowing the five users
  - each at his own typewriter to interact with the computer
  - just as if he had a computer all to himself.
  - 4kwords per users
  - Response time < 1s

Time-sharing application

- THOR 1966
  - Control computer based teaching laboratories
  - Text editing
  - General purpose programming
  - Better interaction CRT vs teletype
  - Understand trade-offs
- 20 user programs
- 28 consoles



Time sharing Computers

- Cheaper
- Transistors
- More compact
- Multiple users
- Single CPU

- 1959 - Thor on PDP-1
  - 4K Words
  - 1.3 M USD – 2024 equiv
- 1964 – DEC PDP-7
  - 4K words
  - 700 kUSD – 2024 equiv
- 1957 – IBM 709 (vacuum tubes)
  - 32 kwords
- 1960 IBM 7090
  - 6x faster 3x cheaper than 709
- 1964 - IBM S/360 Model 67
  - 512 Kbyte

After time-sharing...

- Multitasking/Interactive Operating Systems
  - multiple-programs
  - Single CPU
  - Word and excel running at the same time
- Parallel computers
  - Multiple processors in the same computer
- Multi-threaded Operating Systems
  - Multiple tasks inside the same program
  - Excel, loading a file while updating a table

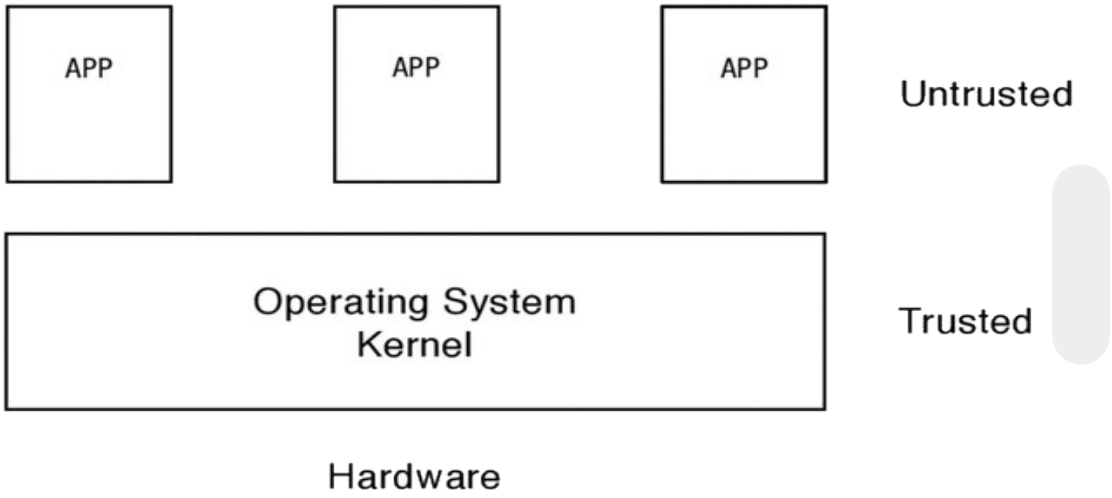
Processes

Protection

- Operating Systems central role
  - isolation of misbehaving applications
- Fundamental to other OS goals
  - Reliability
  - Security
  - Privacy
  - fairness
- OS kernel
  - lowest level SW running on the system
  - implements protection

Protection

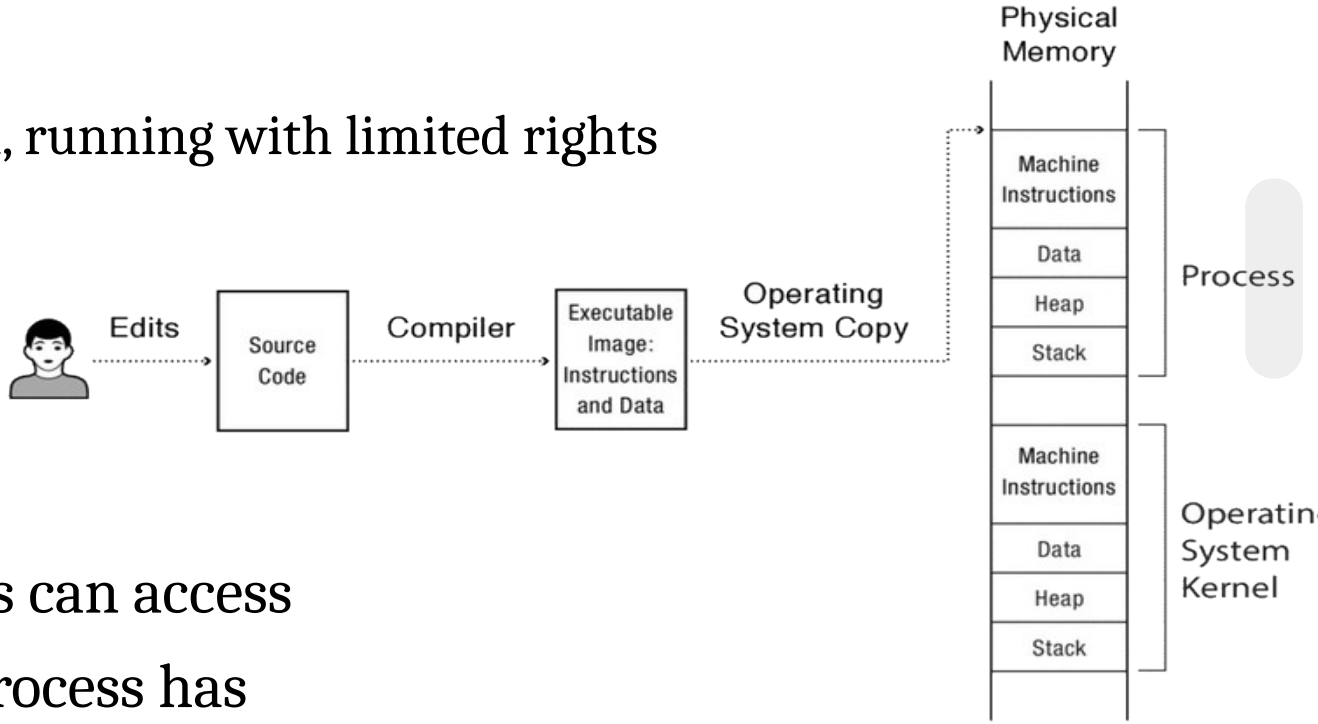
- Applications
  - untrusted



- Process
  - execution of application program with restricted rights
  - Needs permission
    - from OS kernel
    - to access resources (memory from other processes, I/O. ...)

Process Abstraction

- Process:
  - an instance of a program, running with limited rights
- Address space:
  - set of rights of a process
- Memory that the process can access
- Other permissions the process has
  - e.g., which system calls it can make, what files it can access)



UNIX Process Management

- Basic API to manage processes
  - Creation
  - Termination
  - Reception of return code
    - Remember C exit function

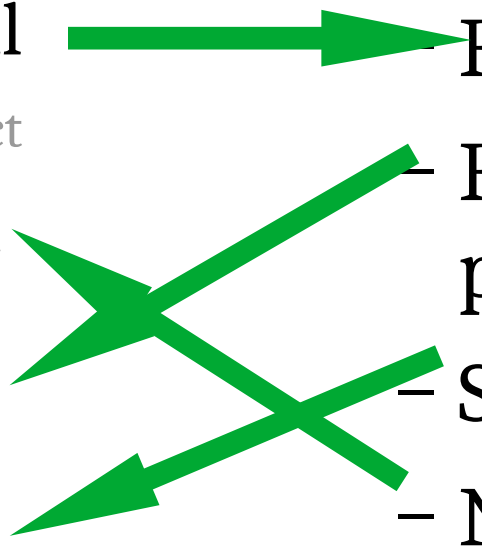
UNIX Process Management

- UNIX fork
  - system call to create a copy of the current process, and start it running
  - No arguments!
- UNIX exit
  - system call to terminate a process
- UNIX wait
  - system call to wait for a process to finish
- UNIX signal
  - system call to send a notification to another process

Unix process creation

- **Existing process (father)**
  - Calls fork system call
  - Waits for start of new project
  - Continues execution
- **New process (son)**
  - Is created
  - Continues execution

- **Kernel**
  - Receives request
  - Replicates calling process
  - Starts new process
  - Notifies existing process



Unix process creation

- To start a new process call:
  - `#include <unistd.h>`
  - `pid_t fork();`
- A new process is created

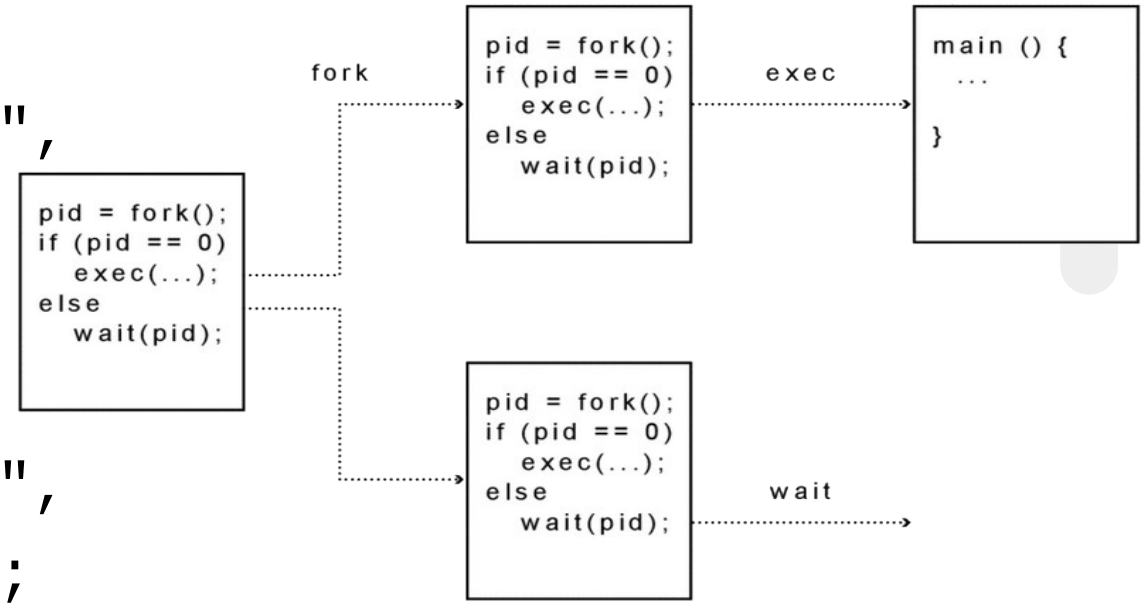
- **`Man fork`**

Unix `fork`

- The new instruction to be executed is the one following the fork
  - On the father and son !!!!
- All variables are duplicated with the same value
  - After the fork all changes in values are local to each process.
- Fork is a system call that return two different values:
  - In the son return 0
  - In the father return the son PID

Unix `fork`

```
int new_pid = fork();
if (new_pid == 0) {
   printf("child PID #%d\n",
                  getpid());
   exit(0);
} else {
   printf("Parent PID %d\n",
                  child_pid);
   exit(0);
}
```



Unix Process creation policies

- Execution mode
  - Father and son execute concurrently
- Memory management
  - Son gets a NON SHARED copy of father memory
- Resource sharing (files, ...)
  - Father and son share all resources
- Child inherits
  - Copy of most information
  - Copy of Memory space
    - Variables
    - Allocated memory
  - Code
  - Opened files
    - pipes, fifos, sockets

Process termination

- When a process executes the exit(int) function:
  - The calling process is terminated "immediately".
  - Any open file belonging to the process are closed;
  - Process's parent is sent a SIGCHLD signal.

- **Man 2 exit / man 3 exit**
  - What happens to the return code?

Reception of return code

- If parent needs the child return code:
  - It must wait for its dead:
  - Call **`wait` /`waitpid`** function

pid_t waitpid(pid_t pid, int *status, int options);

  - Process waits for a specifi child termination.
  - 1st argument – ID of child (-1 any process)
  - 2nd argument – child status
  - 3rd argument
    - WNOHANG: return immediately if no child has exited
    - WUNTRACED: also return if a child has stopped

Reception of return code

- **WIFEXITED(status)**
  - returns true if the child terminated normally, exit(3) _exit(2) or by returning from main().
- **WEXITSTATUS(status)**
  - returns the exit status of the child.
- **WIFSIGNALED(status)**
  - returns true if the child process was terminated by a signal.
- **WTERMSIG(status)**
  - returns the number of the signal that caused the child process to terminate.
- **WIFSTOPPED(status)**
  - returns true if the child process was stopped by delivery of a signal
- **WSTOPSIG(status)**
  - returns the number of the signal which caused the child to stop.

Next on PSIS

- Pipes