Faculty of Computer Science & Engineering

Operating Systems

Contact:
Trần Ngọc Anh Tú
51304672@hcmut.edu.vn

Ho Chi Minh City
University of Technology

Lab 3.1 - Process

- What we will learn?
  - System Programming Techniques
  - Concurrency
  - Synchronization
  - Communication
  - Scheduling
  - Memory Management
- Environment: *nix systems (CentOS, Ubuntu, Mac OS?)

Objective

- Understand to concept of process
- Know how the operating system manages the execution of processes
- Understand how Linux create a new process

What is a process?

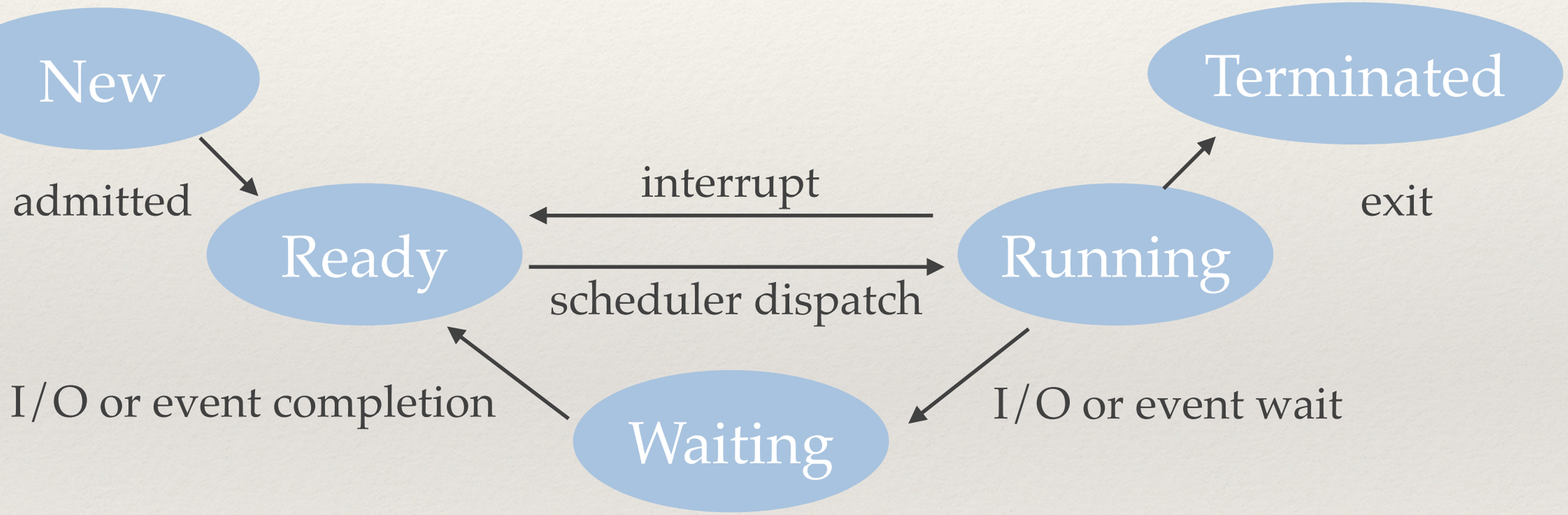
- A process is an instance of a computer program that is being executed. It contains the program code and its current activity.
- Typically, a process consists of:
  - Instructions
  - Memory: text, data, stack, heap
  - Descriptor or resources (file descriptor in *nix)
  - Processor state (context): register, physical memory addressing
- Process holds the information of its resources in Process Control Blocks (PCB)

Isolation

- Process is one of the most important concepts in Operating System.
- Programs are isolated and protected from each others
- Programmers and compilers do not care about the state of the system they are running on, they just have to focus on their job

Process State

- Exercise: describe the states of the process running hello world program




```
#include <stdio.h>
int main() {
  printf("Hello, world!");
  return 0;
}
```

Create a new process

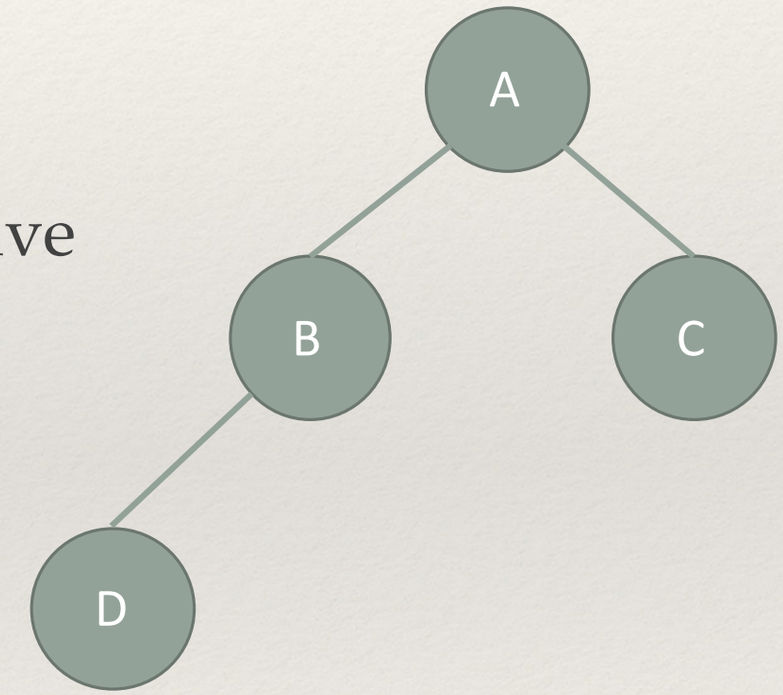
- Linux allows a process to create a new one by calling fork system call.
- When process **A** invokes *fork,* if the OS could create a new process then following events have happened:
  - The operating system created a new process (we call **B**).
  - **A** became **B**'s parent.
  - The content of process **A** was copied to process **B**. (**B** is a clone of **A**).
- The fork system call returns two different values to **A** and **B**.
  - **A** receives a positive value which is the PID of its new child.
  - **B** receives a 0.
- A and B concurrently continue executing the next instruction after the *fork* system call.

Process in Linux

- A process is identified by its unique Process ID (PID).
- PIDs are assigned (generally) sequentially to processes, starting from 0.
- Special process:
  - 0 -> Scheduler: responsible for paging
  - 1 -> Init startup and shutdown the system
  - 2 -> Support memory management
  - …
- Process's state could be found in /proc directory.

Process in Linux

- When a process A creates process B:
  - A is B's parent
  - B is A's child
  - A process has only one parent but could have many children
- The relationships between processes could be represented by a tree
  - A has two children: B and C.
  - B has only one child D and only one parent A.

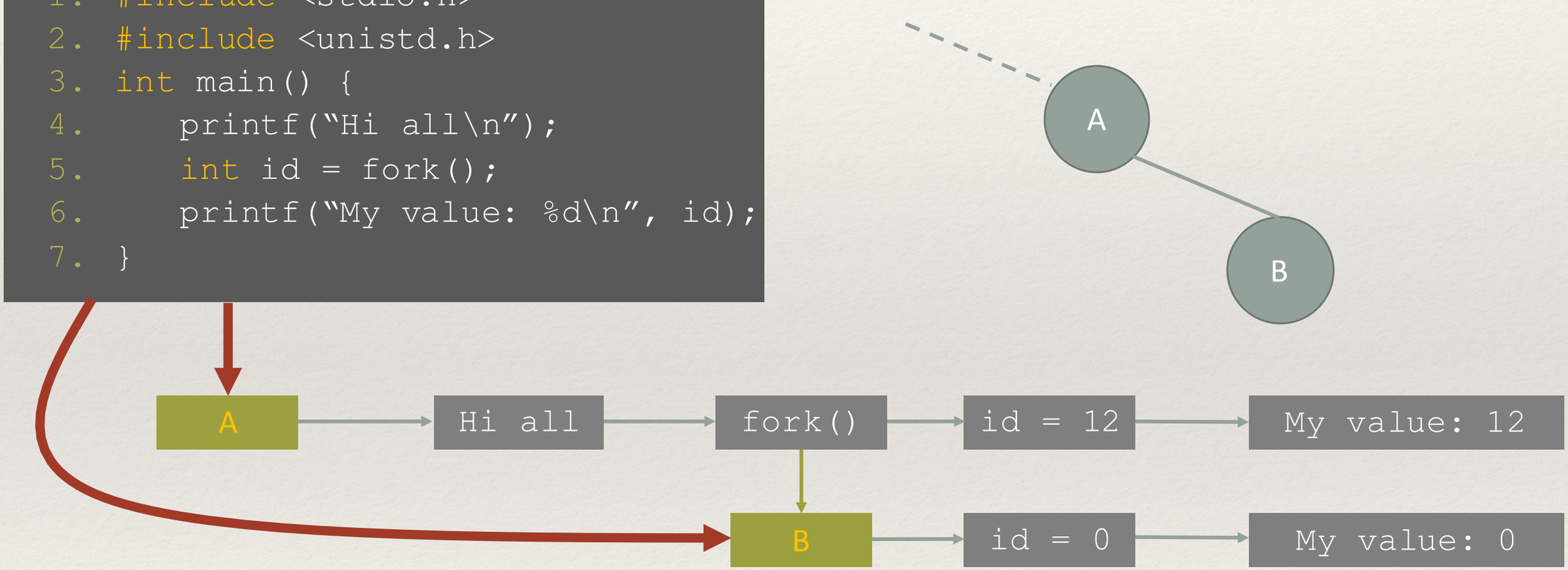


Process in Linux

- If parent process terminates before its child then the child becomes an orphan process and process *init* will become its parent.
- In Linux (and probably other Operating Systems), each process are isolated from others. Which means its does not knows the existence of other processes and what they are doing.
- A process, however, could know a little information about its children and parent through system calls.
  - *getppid()* -> Get the PID of parent process
  - *wait()* -> Wait for one of children process terminating

Example

```
1.  #include <stdio.h>
2.  #include <unistd.h>
3.  int main() {
4.      printf("Hi all\n");
5.      int id = fork();
6.      printf("My value: %d\n", id);
7.  }
```



Exercises

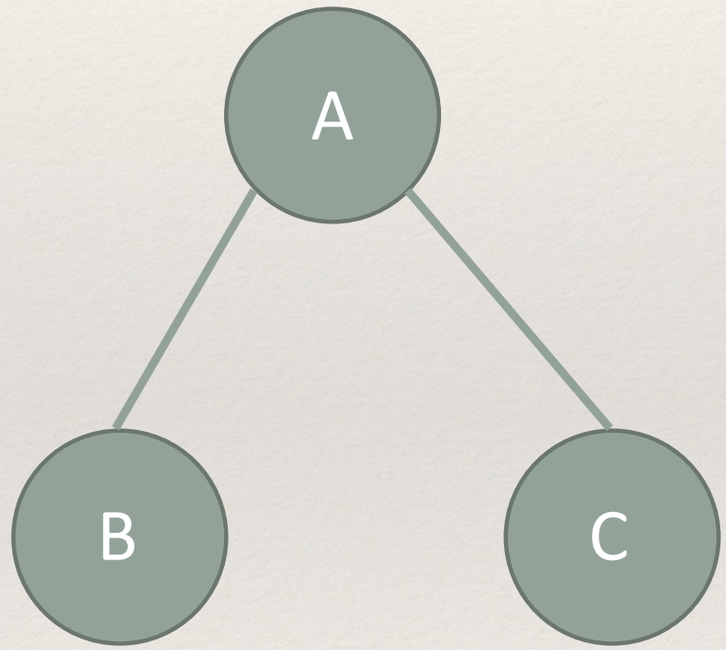
- What will we see on the screen?

```
#include <stdio.h>
#include <unistd.h>
int main() {
   int a = 10;
   if (fork() == 0) {
   a++;
   }else{
   wait();
   printf("%d\n", a);
   }
}
```

- Write a program that when executing, it create a new process and then both of them print their PID to stdout.

Exercises

- Write a program that when executing, it has the OS to create two duplicate copies of itself.



Exercises

- Draw the tree of processes created by running the following program

```
#include <stdio.h>
#include <unistd.h>
int main() {
 fork();
 fork();
}
```

Appendix: Useful commands

- Use command ps -aux to show brief information about running processes.
- Use command ptree to display a tree of processes
- To terminate a foreground process, press Ctrl+C
- In other to stop a running process, use kill –INT <Process's PID>.

Appendix: Suspend a process

- If a foreground process taking so much time while we want to take the control of the shell back to do another task then we could suspend it by pressing Ctrl+Z.
- To display suspended processes, type jobs.
- To resume a suspended process, use
  - fg <n> : suspended process will be waken up and take the control of shell again.
  - bg <n> : suspended process will be waken up and run in foreground.
  - Note: n is the index of the suspended process which placed between a pair of square bracket before the name of process in the string that appears just after we press Ctrl+Z.

End

Thanks!