KOÇ UNIVERSITY

COMP304 PS-1
Spring 2021

Koç University, Istanbul, Turkey

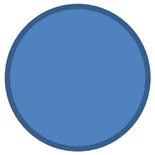
Agenda

- A1
  - Problem1
  - Problem3
- Linux
- Options to Install a Distro
- Ubuntu Shell Introduction
- Basic Shell Commands
  - Quick break
- Shell Scripting
- C Programming and Execution

Consecutive process forks

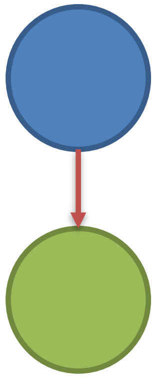
Part (a) Write a C program, which calls **fork()** 4 times consecutively. Then each of the forked processes prints its ID, its parents ID and its level in the process tree (The level of the main process is 0). Sample output for 3 forks is shown in Figure 1.

- 3 forks:



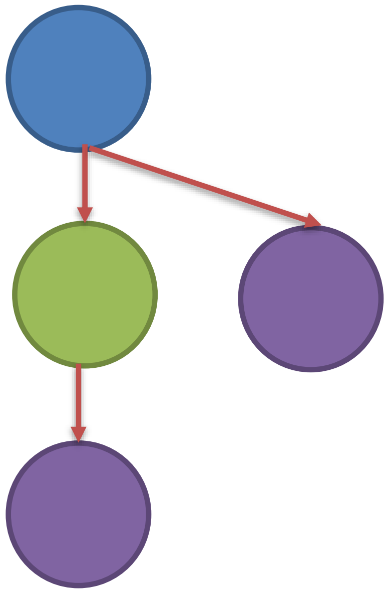
Consecutive process forks

- 3 forks:
  - Fork1



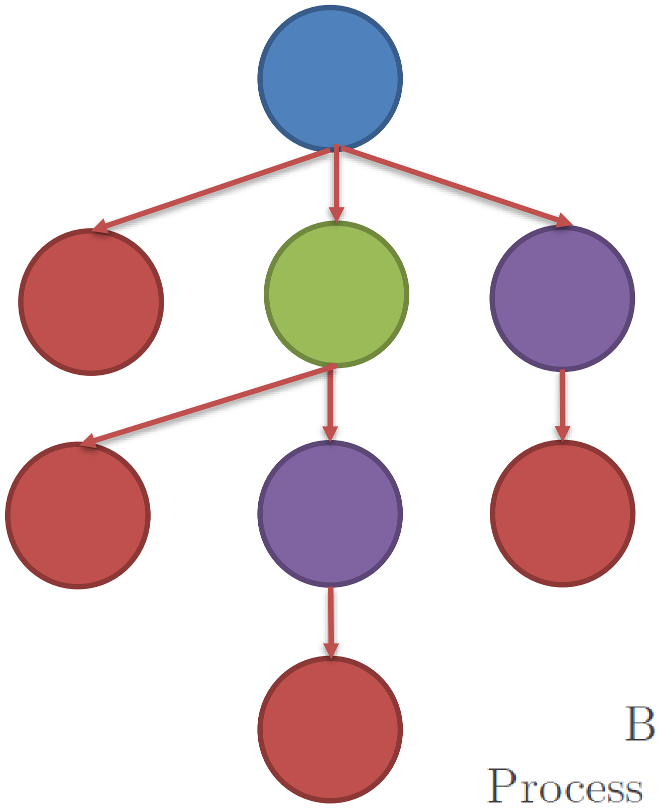
Consecutive process forks

- 3 forks:
  - Fork2



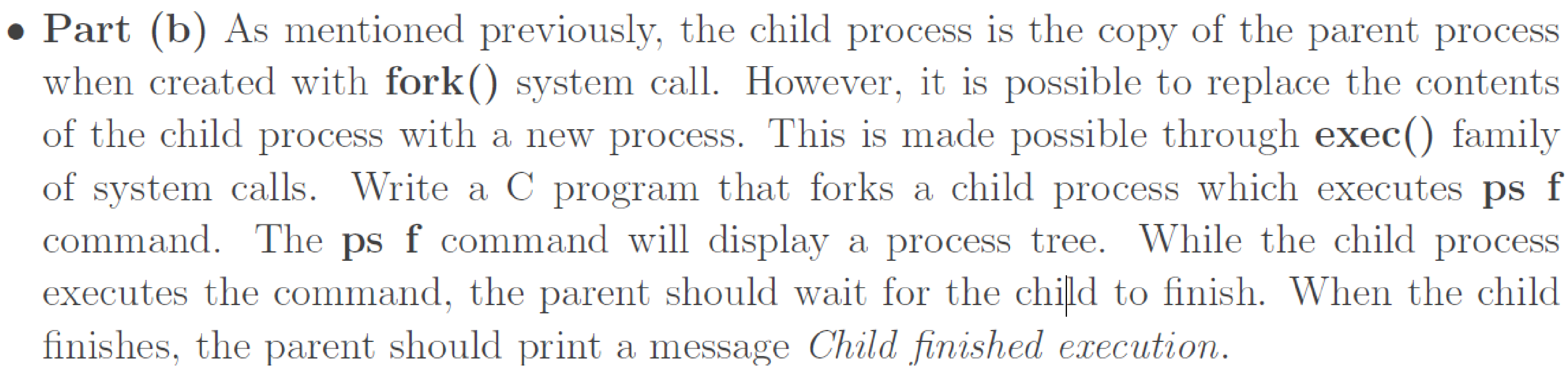
Consecutive process forks

- 3 forks:
  - Fork3



```
Base Process ID: 30511, level: 0
Process ID: 30512, Parent ID: 30511, level: 1
Process ID: 30513, Parent ID: 30511, level: 1
Process ID: 30514, Parent ID: 30511, level: 1
Process ID: 30515, Parent ID: 30512, level: 2
Process ID: 30517, Parent ID: 30512, level: 2
Process ID: 30516, Parent ID: 30513, level: 2
Process ID: 30518, Parent ID: 30515, level: 3
```

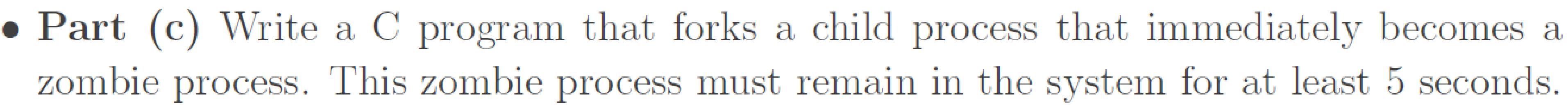
Exec family system Calls



- **Part (b)** As mentioned previously, the child process is the copy of the parent process when created with **fork()** system call. However, it is possible to replace the contents of the child process with a new process. This is made possible through **exec()** family of system calls. Write a C program that forks a child process which executes **ps f** command. The **ps f** command will display a process tree. While the child process executes the command, the parent should wait for the child to finish. When the child finishes, the parent should print a message *Child finished execution*.

- Fork.
- Child
  - exec() family system call // ps f
- Parent
  - Wait

Zombie Process



- **Part (c)** Write a C program that forks a child process that immediately becomes a zombie process. This zombie process must remain in the system for at least 5 seconds.

- A zombie process is created when a process terminates but its parent does not invoke wait() immediately. By not invoking wait(), the child maintains its PID and an entry in the process entry table.
  - –sleep()
  - –Run in the background (&)
  - –Execute the command (ps –l)

Task and Task List

- The Kernel should keep track of the processes
  - Task List
    - Linux: Circular doubly linked list
- Each element in the task list is a process descriptor of the type struct ***task_struct***
  - Contains all the information about a specific process
    - The state of the process, scheduling and memory-management information, list of open files, and pointers to the process's parent and a list of its children and siblings
  - task_struct is linux representation of PCB (Process Control Block)

Kernel Modules

- loadable kernel module is an object file that contains code to extend the running kernel, or so-called base kernel, of an operating system
- With kernel modules it is a relatively easy to interact with the kernel, by writing programs that directly invoke kernel functions.
- Advantages of having loadable Kernel Modules:
  - Performance
    - Core kernel is lighter, additional modules are loaded only on demand
  - Extendibility & Maintainability:
    - Can we know all the anticipated functionalities?
    - What if we need to add/modify some module?

Problem3

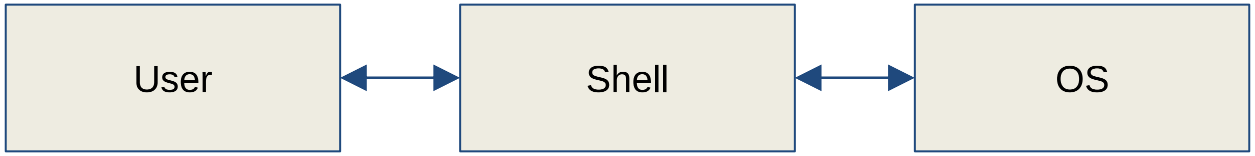
- Part a:
  - On module load:
    - Create a linked list
    - Create and add to the list
      - five struct birthday elements
  - On exit:
    - delete the elements from the linked list
  - All the required API calls are provided in the book
- Part b:
  - https://www.informit.com/articles/article.aspx?p=368650
    - The Process Family Tree(Every process on the system has exactly one parent. Likewise, every process has zero or more children.)

Options to Install a GNU/Linux Distro

- You may install Ubuntu 18.04 Desktop for this course.
- You can install a distro using:
  1. Virtualization using Oracle VirtualBox, WMware, etc.
     - https://www.virtualbox.org/
  2. Dual-boot (multi-boot) or installing it as main OS

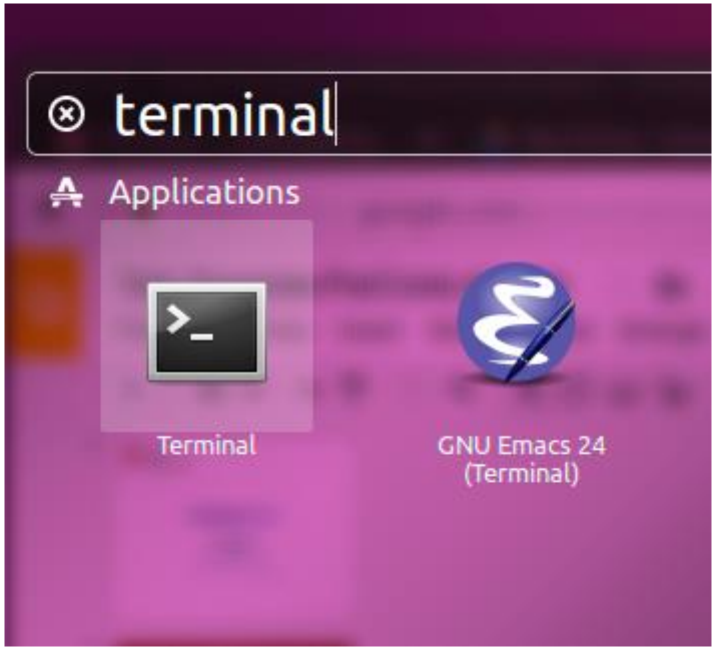
What is Shell?

- Shell is the interface between the user and the OS
  - Takes command from user or script and gives them to OS
- Shell is also often called Terminal
- There are many applications for terminal:
  - Bash, Sh, Zsh, Csh, etc.



Terminal

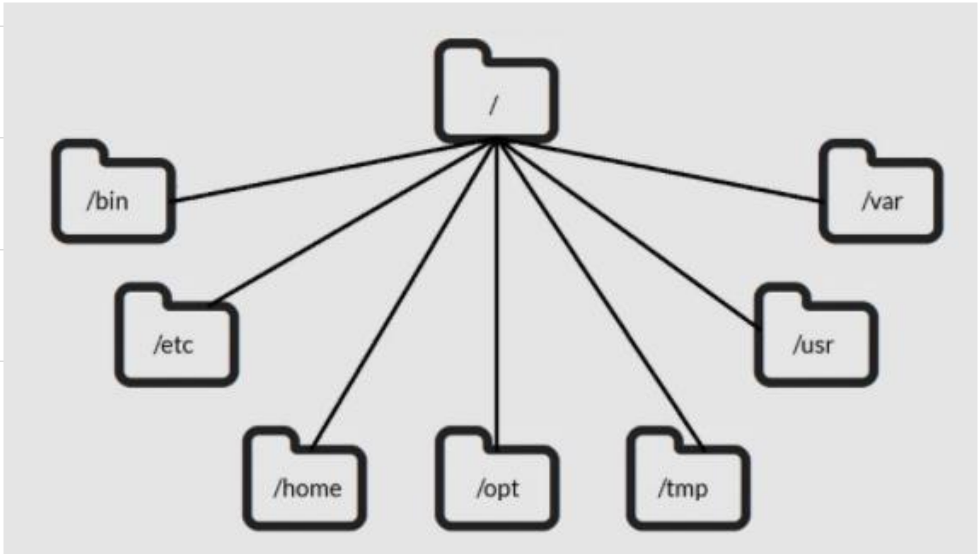
- Accessing terminal
  - Search terminal using Ubuntu search button



  - Or press CTRL+ALT+T

Directory Hierarchy

Dir	Description
`/`	The directory called “root.” It is the starting point for the file system hierarchy. Note that this is not related to the root, or superuser, account.
`/bin`	Binaries and other executable programs.
`/etc`	System configuration files.
`/home`	Home directories.
`/opt`	Optional or third party software.
`/tmp`	Temporary space, typically cleared on reboot.
`/usr`	User related programs.
`/var`	Variable data, most notably log files.



Basic Navigation Commands

- Current directory is called working directory
  - `pwd` command: Print working directory
  - `echo` command: Print the input passed to it
  - `ls` command: List files in working directory
    - http://www.gnu.org/software/coreutils/ls
    - https://www.gnu.org/software/coreutils/manual/html_node/Common-options.html
  - `cd` command: Change working directory
    - Dot operator (.): Current directory
    - Double dot operator (..): Working directory’s parent
    - ~ operator: Change to home directory

Creating Directories/Files

- Creating directory
  - `mkdir` *directoryname*
- Output a file
  - `cat` *filename.ext*
- View text file/long output
  - `less` *filename.ext*
- Clear terminal output
  - reset
  - clear

Deleting Directories/Files

- Deleting a file
  - `rm` *filename*
  - rm -r directory (delete recursively)
  - http://www.gnu.org/software/coreutils/rm
- Deleting a directory
  - `rmdir` *directory*
  - *There should be no files inside*

Copying/Renaming Files

- Copy file
  - `cp` *filename1 filename2*
  - *http://www.gnu.org/software/coreutils/cp*
- Rename/move file
  - `mv` *oldfilename newfilename*
  - *http://www.gnu.org/software/coreutils/mv*

Manipulating Files

- Searching file contents
  - Using `grep` command
    - Grep searches one or more input files for lines containing a match to a specified pattern. By default, Grep outputs the matching lines.
    - https://www.gnu.org/software/grep/
- Word count
  - `Wc`
    - counts the number of bytes, characters, whitespace-separated words, and newlines in each given file, or standard input
    - http://www.gnu.org/software/coreutils/wc

I/O Redirection

- Redirecting output
  - > operator (write output in file)

    `ls` > *list.txt*
- Append output
  - >> operator (append output in file)

    `ls` >> *list.txt*
- Redirecting input
  - < operator (read input from file)

    `sort` < *list.txt*

File/Directory Permissions

- Checking permissions
  - `ls -l`
- Changing permissions
  - `chmod` command
    - *changes the access permissions of the named files*
    - *http://www.gnu.org/software/coreutils/chmod*

Processes

- Stop a process (and terminate)
  - Ctrl+C
- Running process in background
  - & operator
- Process list
  - `ps` command
- Killing a process
  - `kill` command
    - `kill pid`

Editing Files

- You can edit files inside terminal or with a GUI editor (if you're using a desktop distro)
  - Open a file on Desktop for editing using GEdit:
    ```
    gedit ~/Desktop/file.txt
    ```
  - Open the same file inside terminal for editing:
    ```
    nano ~/Desktop/file.txt
    ```
  - vim and emacs are alternatives

Useful Commands for Learning

- whatis: tells what a command does briefly
    - E.g., whatis make
- command --help/command -h: shows the command specific help and arguments
    - E.g., ls --help
- man: manual pages of Linux
    - Can be used to read about programs and commands
        - E.g., man ls
    - Can be used to read about C functions and system calls
        - E.g., man fread

Intro to Bash Scripting

- Script is a list of commands to be executed by the shell
- The most popular shell is *bash*
- Creating a bash script file
  - Add the following as the first line in the file

    ```
    #!/bin/bash
    ```
  - Mark the file as executable (chmod +x)
  - Usually bash scripts are named *something.sh*

Introduction to Bash Scripting

- Variables

  VAR=value

  Output variable using `echo` bash command:

  echo $VAR

- Functions

```
function funcName {   }
```

- Passing arguments to Bash script

```
./bashfile.sh Arg1 Arg2 Arg3
```

- Accessing arguments inside script

```
echo $1 $2 $3
```

If/else statements

- Conditionally execute commands

```
if [ <some test> ]
then
        <some commands>
else
        <some other commands>
fi
```

  - Example

```
        if [ $1 -eq 4 ]
then
        echo Option 4 is selected
else
        echo Option 4 not selected
fi
```

if/else Example

- Conditional command execution based on input argument

```
if [ $1 -eq 1 ]
then
     echo “Printing working directory”
     pwd
else
     echo “Unidentified command”
fi
```

Loops

- While loop

```
    while [ <some test> ]
do
    <some commands>
 done
```

  - Example

```
var1 = 10
while [ $var1 -le 0 ]
do
    echo Current value of ‘$var1’ is
    ((var1--))
done
```

Loops

- For loop

```
for value in <list>
do
    <commands>
done
```

  - Example

```
fruits=”Apple Banana Strawberry”
for name in $fruits
do
    echo $name
done
```

Environment Variables

- An environment variable is a dynamic-named value that can affect the way running processes will behave on a computer. They are part of the environment in which a process runs.
- Some standard bash variables
  - $PATH: Colon separated directory list for command search
  - $HOME: Currents user's home directory
  - $LOGNAME: Current user's name
  - $SHELL: User's preferred shell
  - $EDITOR: User's preferred editor

$PATH Environment Variable

- PATH is an environmental variable that tells the shell which directories to search for executable files in response to commands issued by a user.
- consists of a series of colon-separated absolute paths.
- It increases both the
  - Convenience
  - Safety

Installing Compiler

- Install the package *build-essential* to have compiler and build tools:

  ```
  sudo apt-get install build-essential
  ```
- Gcc and G++ and make included
- Some other useful tools to install:

```
sudo apt-get install htop vim emacs git
```

Executing C programs from shell

- Write your program using any text editor
- Save the file with extension .c or .cpp
- Compile c/c++ files with gcc/g++ respectively.

```
g++ hello-world.cpp -o Hello-World
```

-o defines the output executable filename

- Execute the output file

```
./Hello-World
```

- ./ is needed because of $PATH
- File names are case-sensitive!

Questions?