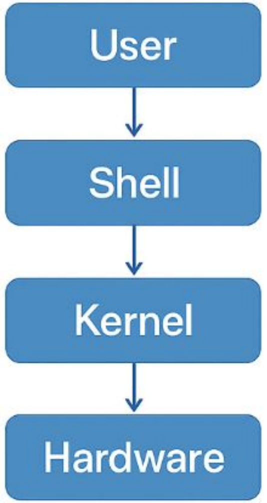
HY345
Assignment 1 Tutorial

30/09/2025

What is a shell

- A shell is a command-line interface that allows users to interact with the operating system.
- It interprets user commands and passes them to the OS kernel.
- Provides access to programs, file management, and system utilities.
- Acts as a bridge between the user and the computer
- Commonly used shells: Bash, Zsh, Fish, PowerShell.



What is a shell

- You’re asked to implement a shell that can reads and executes commands provided by the user

- The command prompt should be **<user>@<AM>-hy345sh:<dir>$** , where:
  - <user> is the currently logged in user name (use the getlogin() function)
  - <dir> is the current working directory path (use the getcwd() function)
  - <AM> is your student ID number (e.g. csd1234)

Command Execution

- Some simple commands like:
  - ls
  - ls -l
  - exit
  - cd
  - cat
  - mkdir
- Moreover you have to implement the execution of sequences of commands like:
  - ls ; pwd ; whoami

Global Variables

Commands like:

- x=10
- my_var="Hello world"
- echo $x #prints: 10
- echo $my_var #prints: Hello world

Redirection

The input of a command is given from a file instead of stdin.

The output of a command is printed to a file instead of stdout.

- ls > y #creates or overwrites a file named “y” with the contents of ls
- cat < y #feeds as input the content of “y” to the “cat” command
- ls -al >> y #appends the output of “ls -al” to the “y” file

Pipes

The output of the previous command passes to the the next command as input.

- cat < y | sort | uniq | wc > y1 #count of unique lines of y stored in y1

- ls | sort | uniq | wc #count of unique file names in a directory

Program control flow and loops

If

- x=5
  if [$x -lt 10] #be careful, there’s a space after the opening square bracket, and before the closing one
  then
  #do commands
  fi

Program control flow and loops

for

- for i in <element list> #the element list can be strings or numbers

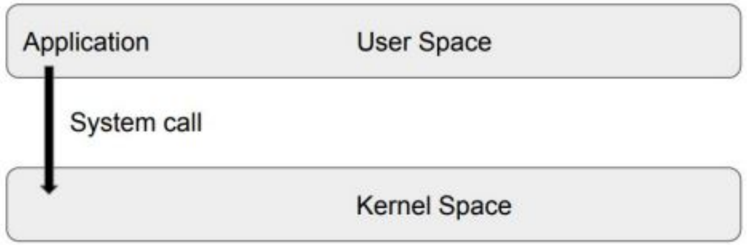
  do

      #do commands

  done

System Calls

- When the user needs to call a system service a trap instruction needs to be called.
- The trap instruction is a procedure call that transfers control to the os.



fork()

- fork() creates a new process (child process)
  - It creates a process duplicate to the original one (parent process), including all file descriptors, registers etc.
  - The child process inherits all variables from the parent at their current state.If in a later step the child process change the value of a variable this change takes place locally (the parent will still have the old value).
- There are three cases which you need to consider regarding the return value of fork():
  - pid < 0 : That means that fork was unsuccessful
  - pid == 0 : This is the pid of the child process (child execution space)
  - pid > 0 : The pid of the child process passed to the parent (parent execution space)

exec()

The exec family contains multiple calls:

- int execl(char *path, char *arg, ...);
- int execlp(const char *file, const char *arg, ...);
- int execle(const char *path, const char *arg, ..., char * const envp[]);
- int execv(const char *path, char *const argv[]);
- **int execvp(const char *file, char *const argv[]);**
- int execvpe(const char *file, char *const argv[], char *const envp[]);

Executes a command. **Replaces the current process with the specified program**

wait()

- The wait() syscall **forces the parent to suspend its execution and wait for the children process** to finish its execution (or to be terminated e.g. by a signal)

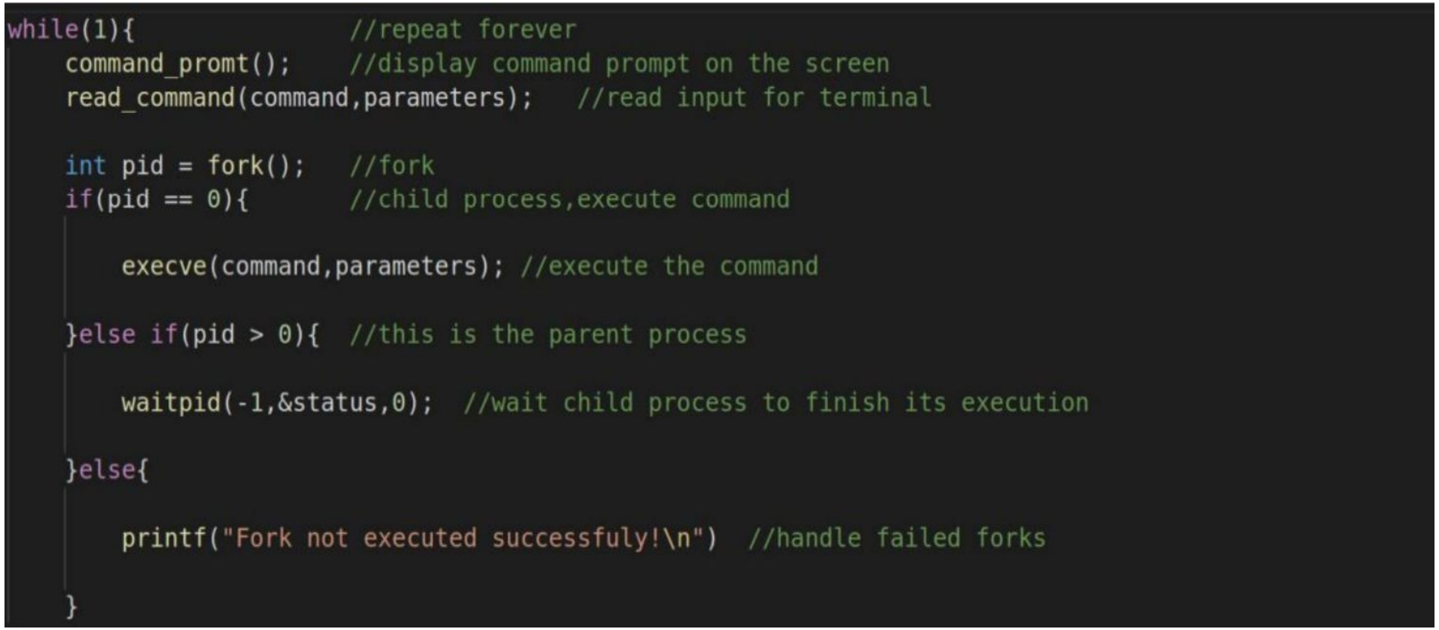
- When a child process terminates, it returns an exit status to the OS, which is returned to the parent process waiting. Upon this value is received by the parent, it continues the execution

exit()

- This call **gracefully terminates the process of execution**, meaning that it cleans up and releases resources taken by this process.

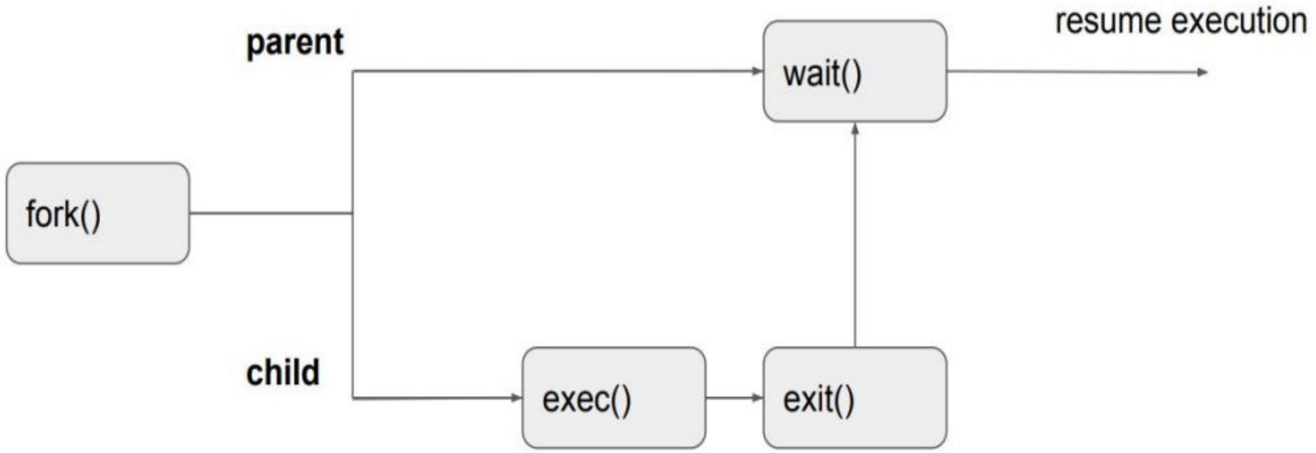
- The **exit status is captured by the parent via the wait() system call**.If the parent waited and receives such signal then the child terminates (dies).If the parent wasn't waiting then the child process enters a **zombie state**.

fork()



```
while(1){                   //repeat forever
    command_promt();        //display command prompt on the screen
    read_command(command,parameters);   //read input for terminal

    int pid = fork();       //fork
    if(pid == 0){           //child process,execute command

        execve(command,parameters); //execute the command

    }else if(pid > 0){  //this is the parent process

        waitpid(-1,&status,0);  //wait child process to finish its execution

    }else{

        printf("Fork not executed successfuly!\n")  //handle failed forks

    }
```

Assignment Execution Diagram



QEMU: How to use it

- Qemu is already installed in the department’s computers.
- You have to copy the disk image from the course’s directory to one of your own local directories.
- The disk image can be found here : **~hy345/qemu-linux/hy345-development.img**
  - To copy the image execute:
    - cp ~hy345/qemu-linux/hy345-development.img <your_dir_path>
    - <your_dir_path> is the path to the directory you want to place the disk image

QEMU: How to use it

Now that you have the disk image locally you can boot it up:

- To launch qemu run:
  - qemu-system-i386 -hda hy345-development.img -display curses
  - The **-display curses** parameter dictates that the Virtual machine will run without graphics
  - If you are working remotely using this argument in the boot up is recommended (if not necessary).
- To login as a user the credentials are:
  - username : hy345
  - password : hy345

QEMU: How to use it

- To transfer a file from your local directory to a directory in qemu:
  - From within qemu run : **scp username@10.0.2.2:<path>/test1.c <qemu-dir>**
  - Where:
    - **username** is your username (e.g. csd1234)
    - **<path>** is the path to your file, in your local machine
    - **<qemu_dir>** is the directory you wish to copy the file inside of qemu
- To transfer a file within qemu to your local machine:
  - **From within qemu run** : **scp test1.c username@10.0.2.2:~/<path>** (parameters same as the above in meaning).
- ❖ To exit qemu simply use the command **turnoff**

QEMU:Why use it

- An emulator mimics the properties of a system to run in another platform efficiently
  - It might bring some additional overhead but:
    - It is inexpensive
    - Easy to access
    - Helps us run programs that might be obsolete to the available system
- You are asked to use the QEMU emulator in order to utilize the department’s computers **safely** amongst all students (e.g. prevent crashes)
- You should **always compile and run** your shell **in** the virtual environment of **qemu**. You can implement the code in the machine you prefer but the testing of your shell should not take place in that machine.

GNU Screen: What is it

- If you're working remotely on **qemu** and the connection drops, qemu will still be running but it will be inaccessible, losing all your work.
    - Also, if you run qemu again, and again, the system will crash.

- That's why you should use **screen**

- **screen** is a terminal multiplexer that lets you manage multiple shell sessions inside one terminal.

- Sessions persist even if you disconnect (useful for remote servers).

GNU Screen: How to use it

- To start a screen session run:
    - **screen -S hy345** #instead of hy345 you could use any name
        - A fresh terminal will open, in where you can run **qemu**
- To **end** a screen session press **CTRL+D** or run **exit**
    - when a session is ended you'll return to the previous terminal used
- To **detach** a screen session press **CTRL+A** and then press **D**
    - when a session is detached you'll return to the previous terminal used
- To continue a detached session run:
    - **screen -ls** #to see all sessions
    - **screen -r hy345** #again you can use any name instead of hy345
    - A detached session **can be the one you lost connection to!**

Turnin of the assignment

- This year the exams are turned in via **E-Learn**
- You have to include a **makefile**, with the commands **make all** to compile the executable and **make clean** to remove all the compiled files.
- A **README** file is also required, in which you'll explain your work
- All questions about the assignment must be sent to the **mailing list and not the E-Learn forum.**

The End

Any questions?