CS302
Operating System
Lab 2

Process, Pipe and Signals

CS302 TA group

Thanks Chuang Yang for linux environment

Process

- Today, we will focus these five main keywords:
  - pid: process identifier
  - fork: a function that duplicate a process
  - exec*: a function family that replace a process
  - signal: handle unexpected situations
  - pipe: Inter-process Communication

Process Identification

- Command of showing Process in Linux:
  - –top, ps, (htop)…
  - –just man them to see details if you are not familiar of them

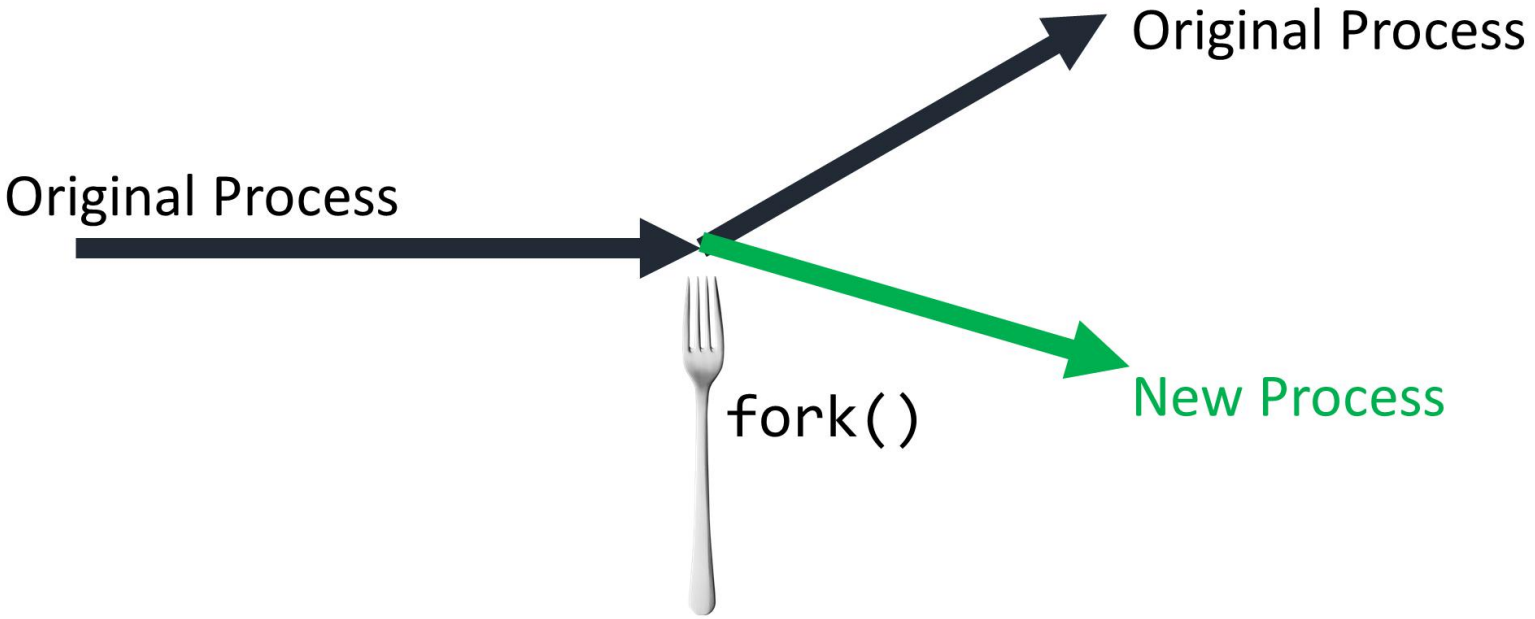
- We can identify one process uniquely by its **Process ID (PID)**
  - – we can use getpid() to get the pid of current process

In class practice

- Write a C program that use getpid() to display the pid of current process.
  - Try man getpid if you do not know how to use it
  - Don't be shy to ask questions

Process Creation

- Use system call **fork** to create a process
- after fork, the original process splits into two
  - –fork copies almost all things about the original process
  - –OS allocates the pid, and pass the value to fork()



In class practice

- Write a program that call fork() to generate a new process, and print the pid of the processes(both parent and child process)
  - Try man fork if you are not familiar with the function
  - Don't be shy to ask questions

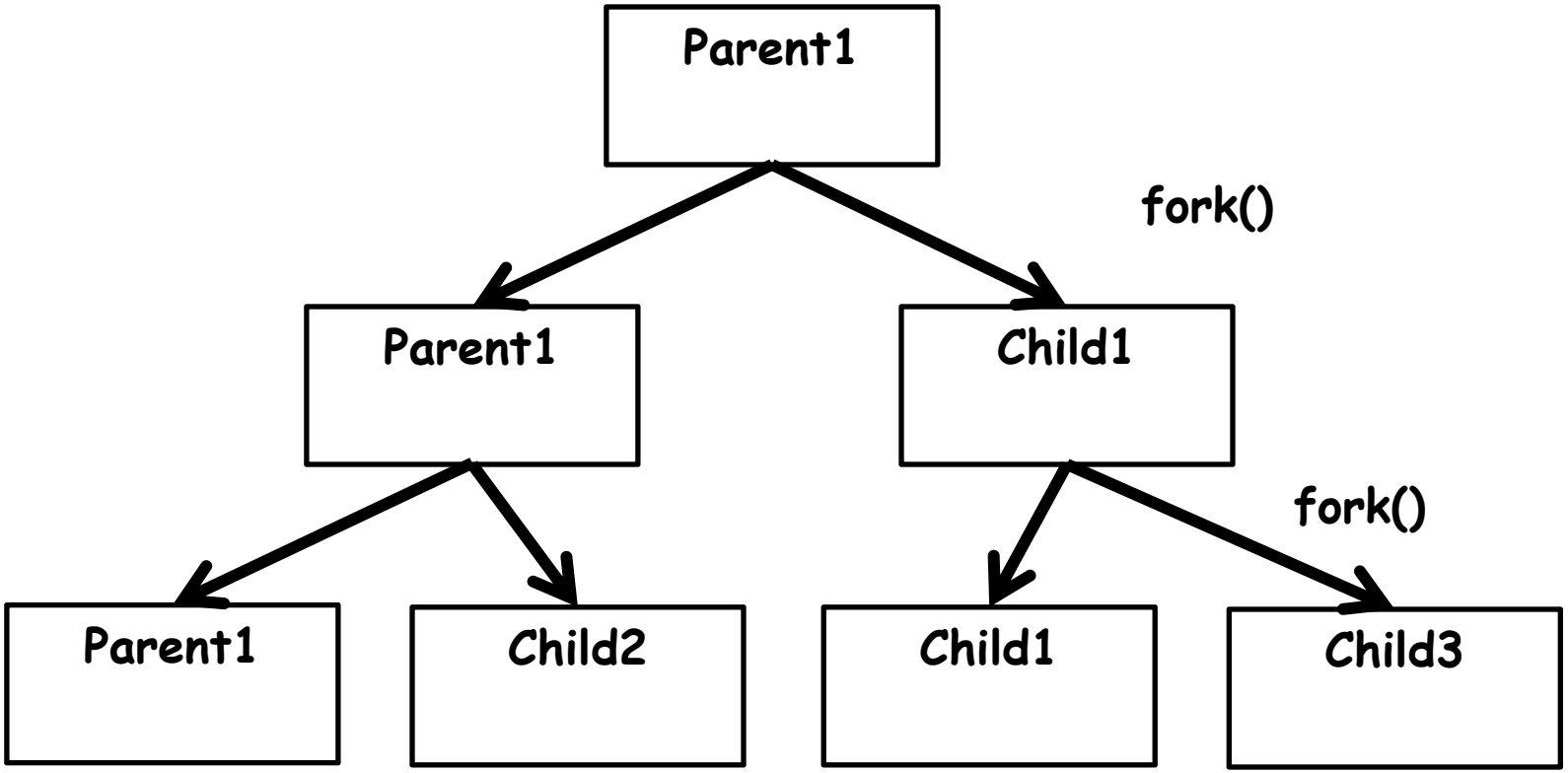
Process Creation

- Exercises : How many A, B and C will be printed?
  a) A: 1 time, B: 2 times, C: 2 times
  b) A: 1 time, B: 2 times, C: 4 times
  c) A: 1 time, B: 1 times, C: 2 times
  d) A: 1 time, B: 2 times, C: 3 times

```c
#include <stdio.h>
#include <unistd.h>
int main(int argc,char *argv[]){
    printf("A\n");
    fork();
    printf("B\n");
    fork();
    printf("C\n");
    return 0;
}
```

Process Creation

- The sequence of process running after fork:



Who runs first ?

Process Creation

- Exercises : The result of running the following program is ?
- (A) A\nB\nB\nC\nC\nC\nC\n
- (B) A\nB\nC\nB\nC\nC\nC\n
- (C) Not sure

```c
#include <stdio.h>
#include <unistd.h>
int main(int argc,char *argv[]){
    printf("A\n");
    fork();
    printf("B\n");
    fork();
    printf("C\n");
    return 0;
}
```

In class practice

- Write a program that verify the return value of fork() function.
  - Try man fork if you are not familiar with the function
  - Don't be shy to ask questions

Process Execution

- exec*() function family
  - Contains: execl, execlp, execle, execv, execvp, execve.
  - Only one system call: execve()
  - "l" means list, "v" means vector
  - "p" means path, "e" means environment

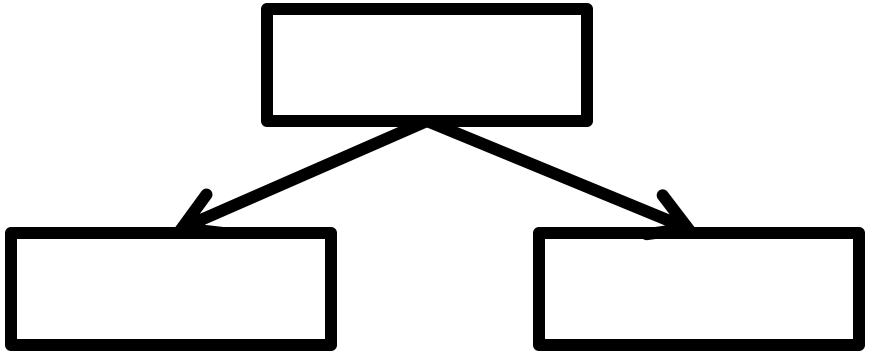
- Try man exec to see details

In class practice

- Write a simple program that print anything you want. And then call exec*() function to execute it.
  - Try man exec if you are not familiar with these functions
  - Don't be shy to ask questions

fork() + exec*()

- Now, we have the method to duplicate a process

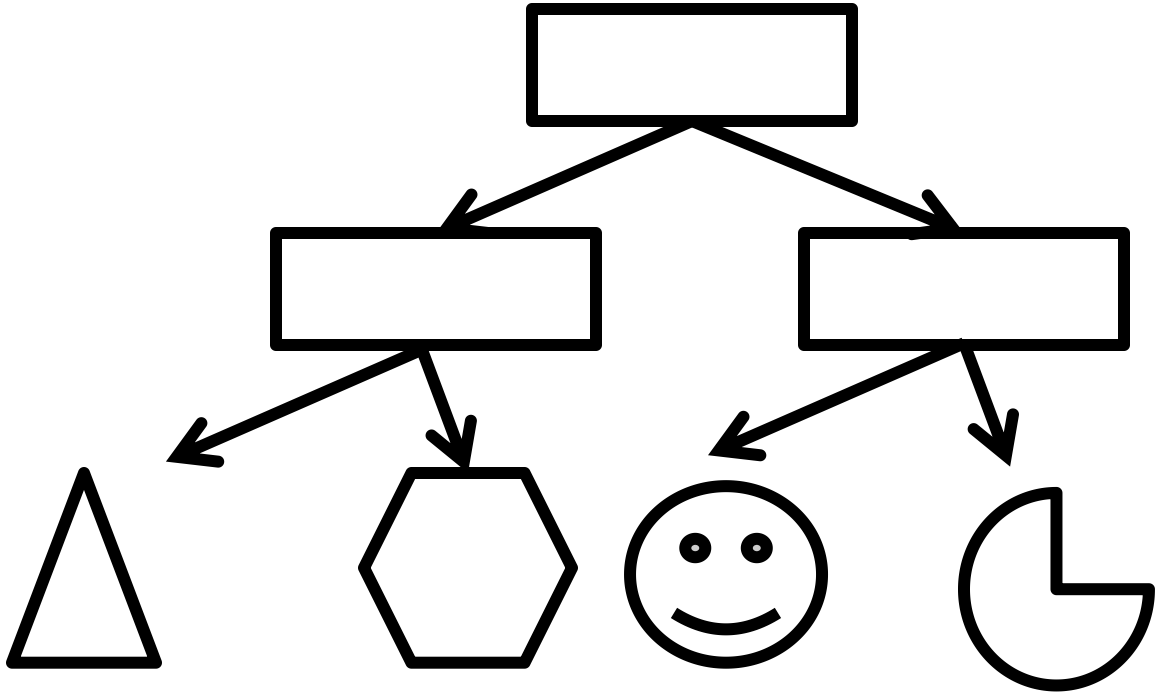


- And, we have the method to replace a process

- Then we can get ...

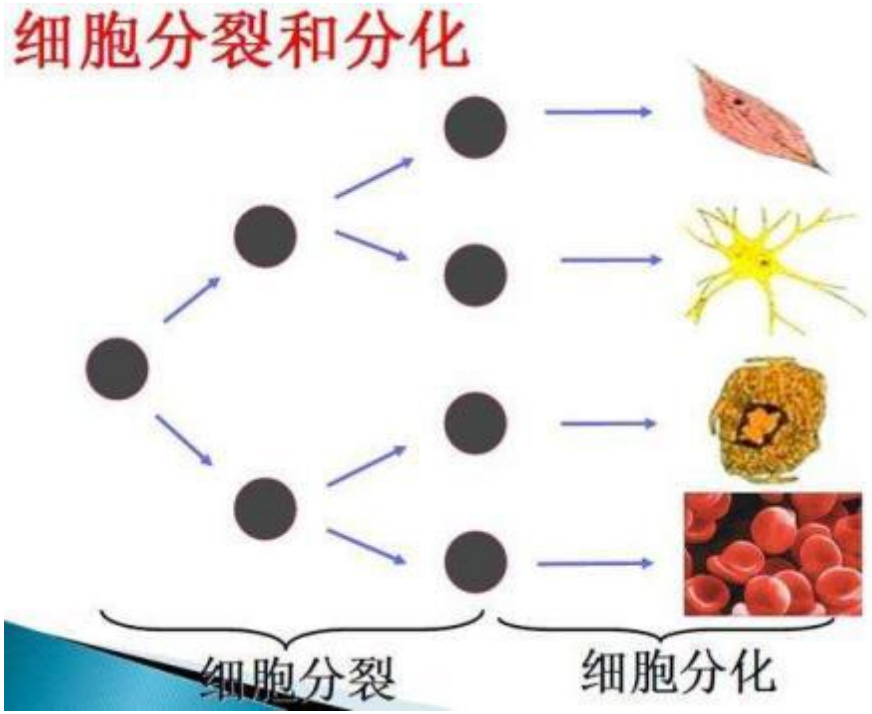
fork() + exec*()

- We can get diversity functions we need



Interesting idea of mine

- fork() like cell division
- exec*() like cell differentiation
- Init process like Fertilized egg
- What we can get is actually a human.



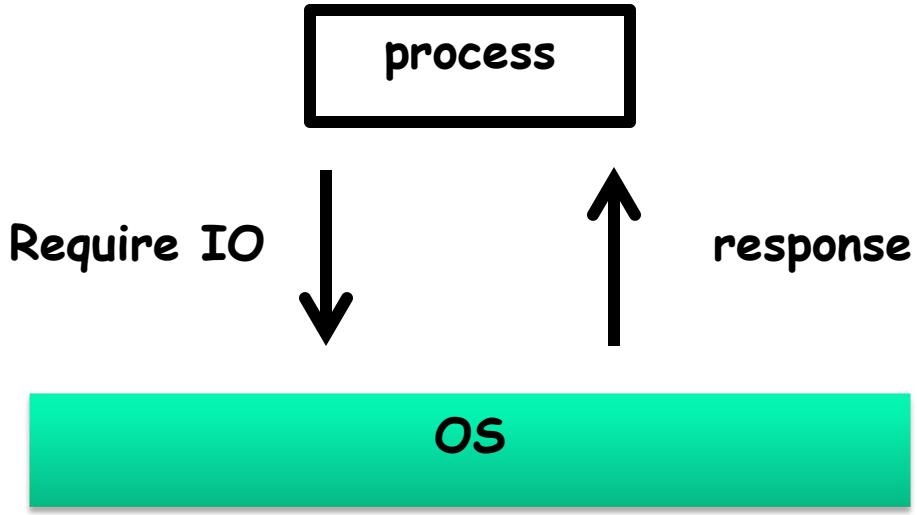
fork() + exec*()

- Let's return to the topic we discussed before:
  - The sequence of running a program.
  - What does shell exactly do.

- But if we want the process running in the way we want ?
  - wait*(), signal(), kill()
  - Try man wait to know details

fork() + exec*() + wait()

- After require IO, our process can call wait*() to suspend until be waked up.
  - –wait() for any one of the children, and only detect the termination
  - –waitpid() depends on the parameters



In class practice

- Write a program that after fork(), the child process runs first. And then parent runs.

- Recall shell's command

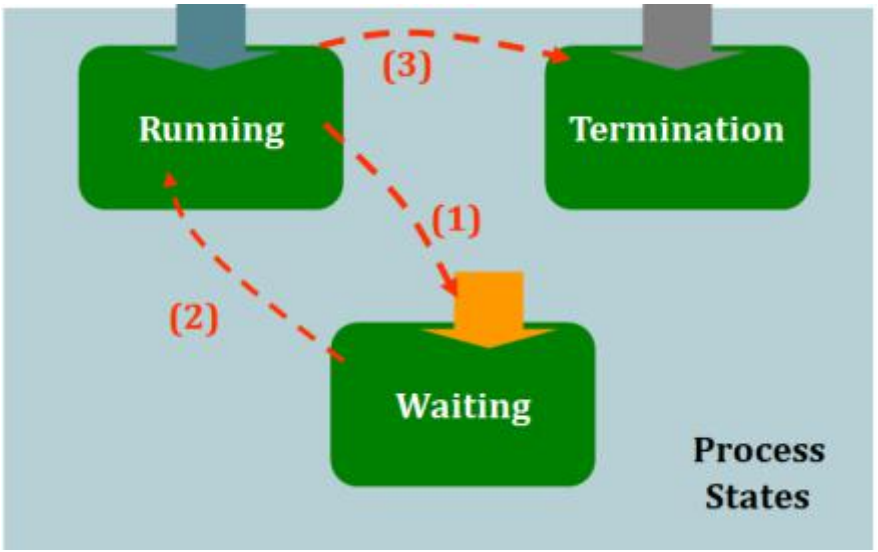
Signal

- Signal is a kind of interrupt to the running process.
  - SIGINT, SIGCHLD, SIGHUP, SIGALARM

- Signal handling
  - Ignore
    - SIGKILL, SIGSTOP never be ignored
  - Catch
    - By user function
  - Default
    - By kernel default handler

Signal

- Signals
  - Signals play a really important role in switching process states.



Signal

- Signal sending(kill, sigqueue)
  - kill() send a signal to process
  - Try kill -l to see all signals

- Signal registration(signal, sigaction)
  - signal() functions is used to set the handler when the process received a given signal.
  - E.g. signal(SIGINT, my_handler) will let the process run function my_handler when reveived signal SIGINT.

In class practice

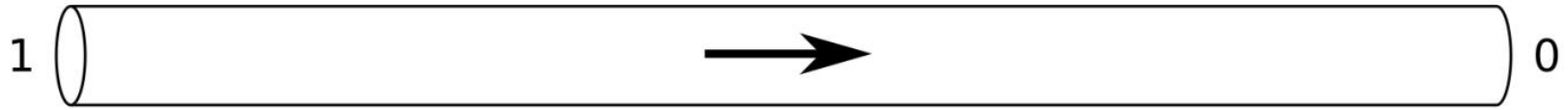
- Write a program that waiting for signal SIGINT. After the first time from receiving SIGINT, the process must ignore SIGINT. And find a way to test your program.
  - Hint: You may need getpid(), signal() functions and kill command
  - If you are not familiar with these functions/commands, use man
  - Be free to ask questions, but I hope you can solve your problems by yourself.

Pipe

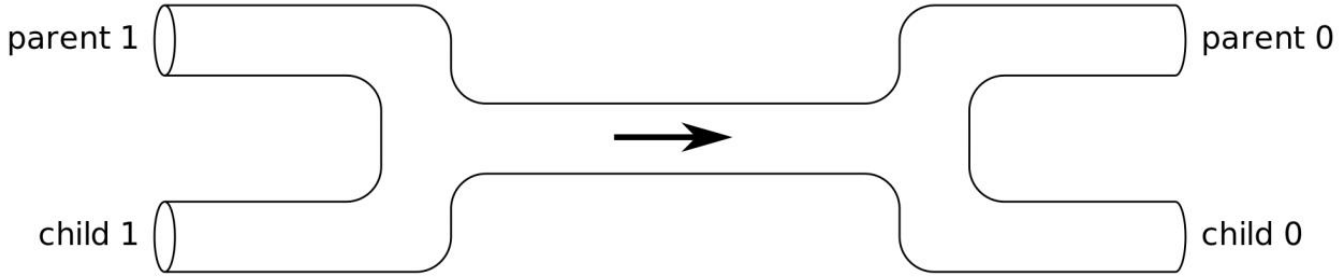
- pipe overview
  - Pipe is a unidirectional data channel for inter-process communication.
- System call pipe ()
  - pipe() will create a pipe.
  - Try man pipe to know details.
  - File descriptor: handle used to access a file or other input/output resource.

Learn to use pipe

- A pipe



- A pipe with fork
  - Why they share the same pipe?



In class practice

- As you know, if we want to do inter-process communication, we need at least two processes. Write a program that call fork() to generate a child process, and use pipe() to create a pipe between them. And then, you should input message in one of the process and display the message in the other process.
  - If you are not familiar with these functions/commands, use man
  - Be free to ask questions, but I hope you can solve your problems by yourself.
  - An example:

```
mark@marklinux:~/os/lab2$ ./pipe_new
parent process begins, pid = 8897
child process begins, pid = 8898

My pid is 8898
Please input a string:
hello
Write finished

My pid is 8897
read begins
Read finished
Message is: hello

My pid is 8898
Please input a string:
```

Read code together

```c
int pid;
int pipe_fd[2];
char buff[1024], input[1024];

void write_data(){
    sleep(1);
    printf("\nMy pid is %d\n", getpid());
    printf("Please input a string:\n");
    scanf("%s", input);
    write(pipe_fd[1], input, strlen(input));
    printf("Write finished\n");
    kill(getppid(), SIGALRM);
}

void finish_write(){
    close(pipe_fd[1]);
    printf("%d finish write\n", getpid());
    exit(0);
}

void read_data(){
    sleep(1);
    printf("\nMy pid is %d\n", getpid());
    printf("read begins\n");
    memset(buff, 0, sizeof(buff));
    read(pipe_fd[0], buff, 1024);
    printf("Read finished\n");
    printf("Message is: %s\n", buff);
    kill(pid, SIGALRM);
}
```

```c
void finish_read(){
    close(pipe_fd[0]);
    printf("%d finish read\n", getpid());
    exit(0);
}

int main(){

    if (pipe(pipe_fd) < 0){
        printf("pipe create failed\n");
    }

    if ((pid = fork()) < 0){
        printf("fork failed\n");
    }

    if (!pid){
        printf("child process begins, pid = %d\n", getpid());
        signal(SIGALRM, write_data);
        signal(SIGINT, finish_write);
        kill(getpid(), SIGALRM);
        while (1){
        }
    }
    else{
        printf("parent process begins, pid = %d\n", getpid());
        signal(SIGALRM, read_data);
        signal(SIGINT, finish_read);
        while (1){
        }
    }
}
```

Assignment

- Finish the Lab report.