Project 2a: Shell (Linux)

Outline

- Shell Overview
- Strings in C

Outline

- Shell Overview
- Strings in C

Shell

- Shell is a program that takes input as "commands" line by line, parses them, then
  - For most of the commands, the shell will create a new process and load it with an executable file to run. E.g. gcc, ls
  - For some commands, the shell will react to them directly without creating a new process. These are called "built-in" commands. E.g. alias
    - Temp: alias wr=”cd /var/www/html”
    - Permanent:
      - Bash – ~/.bashrc
      - ZSH – ~/.zshrc
      - Fish – ~/.config/fish/config.fish

If you are curious about which catalog a command falls into, try which cmd_name

- The output of the following commands may vary depending on what shell you are running
- $ which gcc
- $ which alias
- $ which which

Fork: process creation

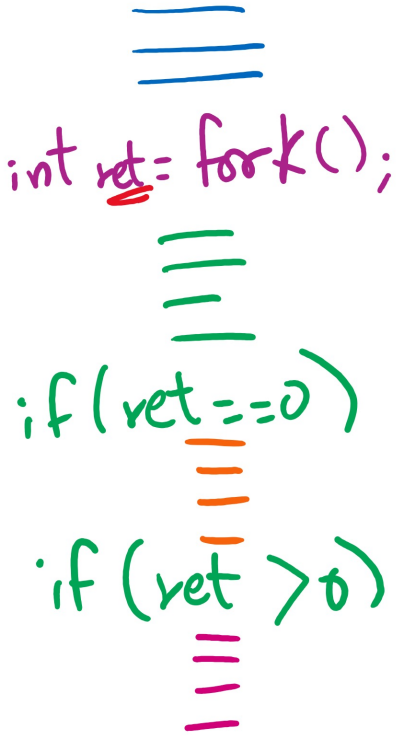
- The syscall fork() creates a copy of the current process. We call the original process "parent" and the newly created process "child".

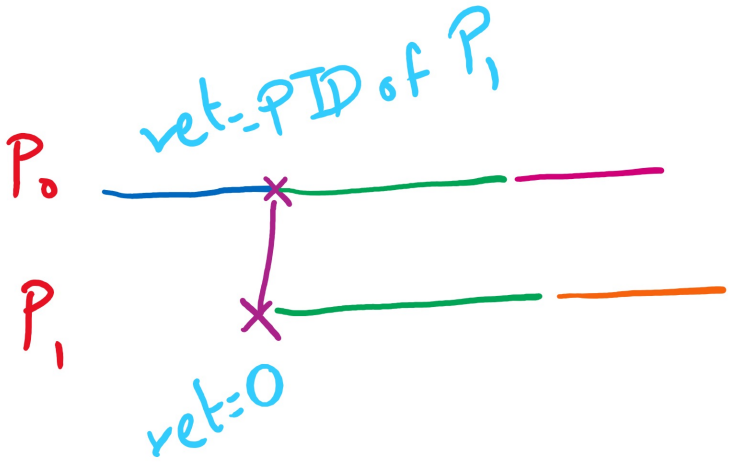
What will the output look like?

```
pid_t pid = fork();
if (pid == 0) { // the child process will execute this
    printf("I am child with pid %d. I got return value from fork: %d\n", getpid(), pid);
    exit(0); // we exit here so the child process will not keep running
} else { // the parent process will execute this
    printf("I am parent with pid %d. I got return value from fork: %d\n", getpid(), pid);
}
```

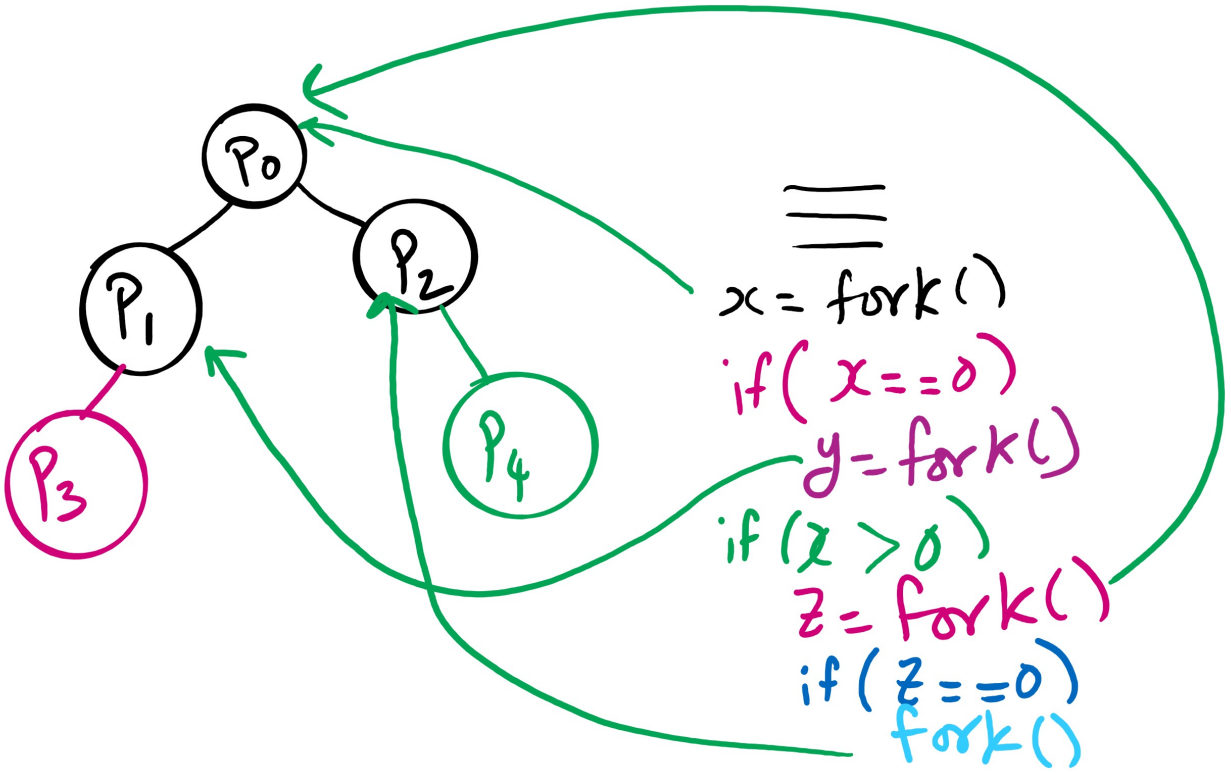
- I am parent with pid 46565. I got return value from fork: 46566
- I am child with pid 46566. I got return value from fork: 0

fork() tracing





fork()'s of fork()'s



exec

- fork itself is not sufficient to run an operating system. It can only create a copy of the previous program, but we don't want the exact same program all the time.
- That's when exec shines. exec is actually a family of functions, including execve, execl, execle, execlp, execv, execvp, execvP...
- For this project, execv is probably what you need. It is slightly less powerful than execve, but is enough for this project.

exec

- What exec does is: it accepts a path/filename and finds this file. If it is an executable file, it then destroys the current address space (including code, stack, heap, etc), loads in the new code, and starts executing the new code.

```
int execv(const char *path, char *const argv[])
```

- it takes a first argument path to specify where the executable file locates, and then provides the command-line arguments argv, which will eventually become what the target program receives in their main function int main(int argc, char* argv[]).

ls -l

```
pid_t pid = fork();
if (pid == 0) {
    // the child process will execute this
    char *const argv[3] = {
        "/bin/ls", // string literial is of type "const char*"
        "-l",
        NULL // it must have a NULL in the end of argv
    };
    int ret = execv(argv[0], argv);
    // if succeeds, execve should never return
    // if it returns, it must fail (e.g. cannot find the executable file)
    printf("Fails to execute %s\n", argv[0]);
    exit(1);
}
// do parent's work
```

waitpid: Wait for child to finish

- $ sleep 10 # this will create a new process executing /usr/bin/sleep
- pid_t waitpid(pid_t pid, int *status, int options);

Outline

- Shell Overview
- Strings in C

StrTok

- The C library function char *strtok(char *str, const char *delim) breaks string str into a series of tokens using the delimiter delim.

```
/* get the first token */
   token = strtok(str, delim);

   /* walk through other tokens */
   while( token != NULL ) {
      printf( " %s\n", token );

      token = strtok(NULL, delim);
   }
```

StrDup

- copy_str = strdup(source_str);

- The function strdup() is used to duplicate a string. It returns a pointer to null-terminated byte string.

\0

- memory management is taken care of by programmar in C
- pointer has access to memory beyond \0 but anything after '\0' is ignored by string manipulation functions

What you will implement

- Built-in Functions
  - cd, path, exit
- Redirect
  - It could, like most of the unix shells, redirect the output of program to a local files.
- Multi-Commands
  - It could receive multiple commands, seperated by &, and execute them one by one. It could also work with multiple redirected commands.
- Program Error
  - All of the error will be printed to stderr.