System Call Implementation in FreeBSD

(Haniya Gebeyehu – FreeBSD Operating system – BDU1601700)

Overview

This program illustrates the creation and management of process using fork(), setpgid(), and other POSIX system calls. It isdesigned to shoe how parent and child processes relate in terms of Processes ID and group ID on FreeBSD system.

Requirements

- A terminal or shell environment
- A C compiler

Code steps at a glance

- First, the parent process creats three child processes using fork()
- next, the first and the third child processes set their own process group ID using setpgid(0,0)
- then, each process prints its own Process ID, Parent ID and group ID.
- Finally, the parent waits for all children to compile using waitpid()

Key Functionalities used

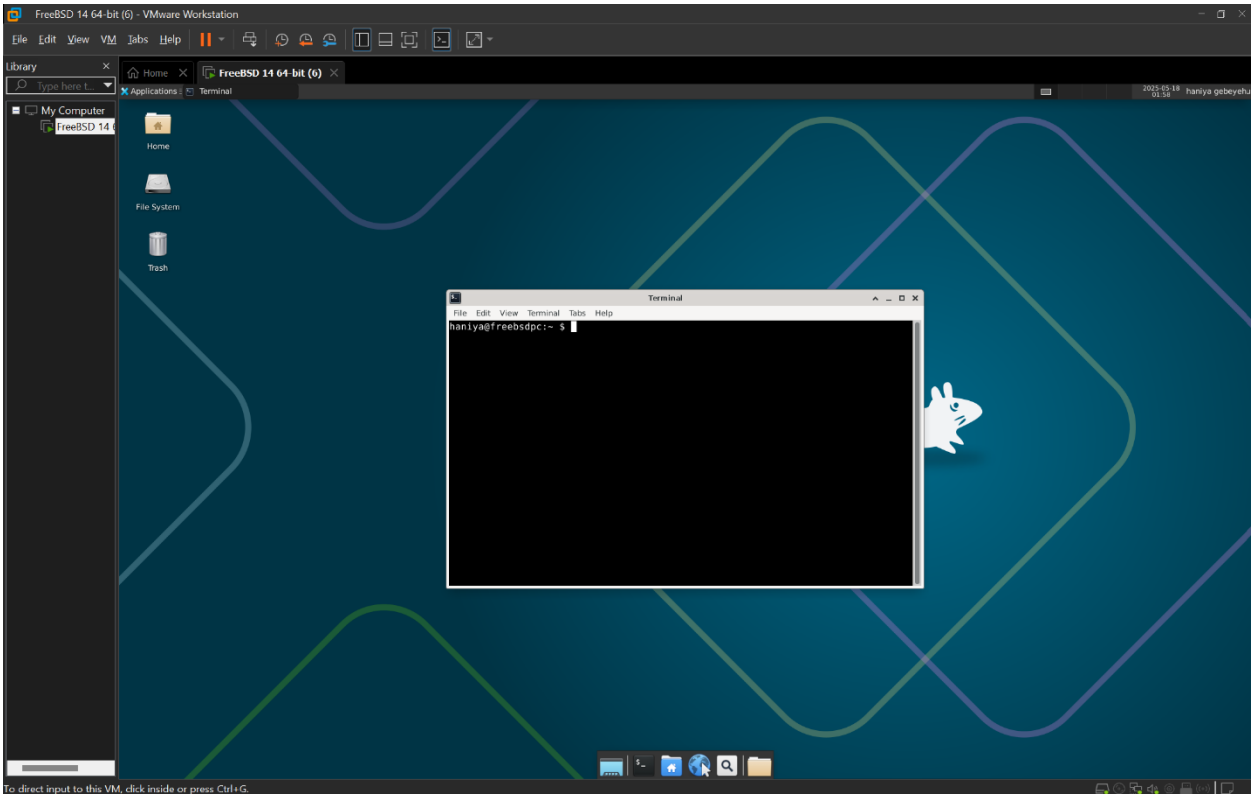
Function	**Purpose**
fork()	Create a new child process
setpgid()	Change the process group ID
getpid()	Get current process ID

Function	Purpose
getppid()	Get parent process ID
getpgrp()	Get current process group ID
waitpid()	Wait for child process to finish
sleep()	Delay execution for readability

How to write and compile the code in the terminal

We can write this code either in the terminal or GNU. Because of simplicity let's use the terminal emulator.

Step 1: open the terminal emulator in FreeBSD



Step 2: write this code in the terminal

```
ee process_groups.c
```

- ee stands for Easy Editor, beginner-friendly text editor that runs in the terminal
- therefore, we are going to open a file process_groups.c using the Easy Editor(ee)

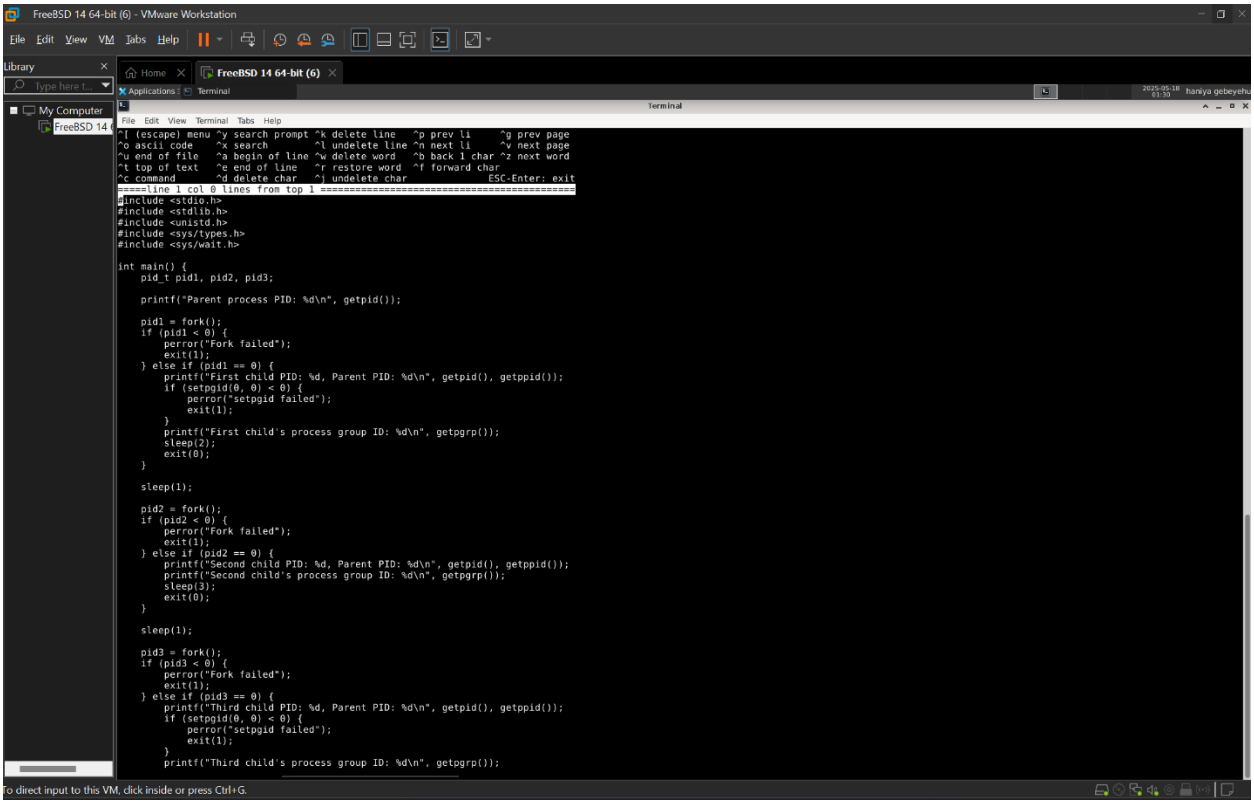
step 3: write the source code (you can find the source code in my GitHub repo: https://github.com/Haniya9/FreeBSD_OS.git)

Code Blame 67 lines (56 loc) · 1.53 KB Code 55% faster with GitHub Raw

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    pid_t pid1, pid2, pid3;

    printf("Parent process PID: %d\n", getpid());

    pid1 = fork();
    if (pid1 < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid1 == 0) {
        printf("First child PID: %d, Parent PID: %d\n", getpid(), getppid());
        if (setpgid(0, 0) < 0) {
            perror("setpgid failed");
            exit(1);
        }
        printf("First child's process group ID: %d\n", getpgrp());
        sleep(2);
        exit(0);
    }

    sleep(1);

    pid2 = fork();
    if (pid2 < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid2 == 0) {
        printf("Second child PID: %d, Parent PID: %d\n", getpid(), getppid());
        printf("Second child's process group ID: %d\n", getpgrp());
        sleep(3);
        exit(0);
    }

```

```
    sleep(1);

    pid3 = fork();
    if (pid3 < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid3 == 0) {
        printf("Third child PID: %d, Parent PID: %d\n", getpid(), getppid());
        if (setpgid(0, 0) < 0) {
            perror("setpgid failed");
            exit(1);
        }
        printf("Third child's process group ID: %d\n", getpgrp());
        sleep(4);
        exit(0);
    }

    sleep(1);

    printf("\nParent process group ID: %d\n", getpgrp());

    waitpid(pid1, NULL, 0);
    waitpid(pid2, NULL, 0);
    waitpid(pid3, NULL, 0);

    printf("All child processes have completed.\n");
    return 0;
}
```

Step 4: write the code in the terminal



Step 5: save and exit

Once we write the code

- **Press Esc** key (usually top-left on your keyboard).
- A menu appears at the bottom.
- Press **a** to “save and exit”.

Now you're back in the terminal.

Step 6: Compile the program

```
cc -o process_groups process_groups.c
```

Breakdown:

- cc: the C compiler

- -o process_groups: this tells the compiler to name the output executable process_groups.
- process_groups.c: the C source file we are compiling

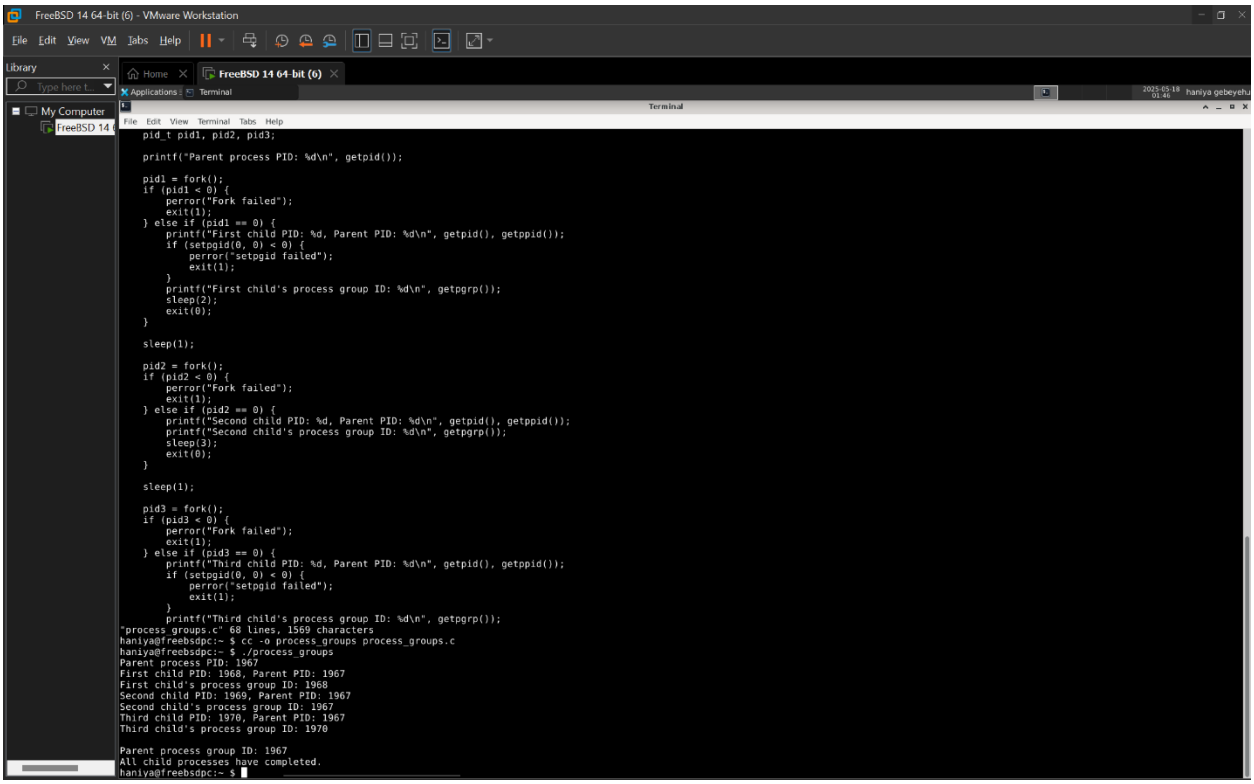
Step 7: Write the last code if the earlier runs without error

```
./process_groups
```

Breakdown:

- . /: refers to the current directory. It tells the shell to look the process_group here instade if searching system paths
- process_group: the name of the compiled executable file created by cc -o process_group process_group.c

Step 8: checking the result



Breakdown:

- As we see, the code compiled without errors.