

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 is designed to show 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 creates 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 complete using `waitpid()`

Key Functionalities used

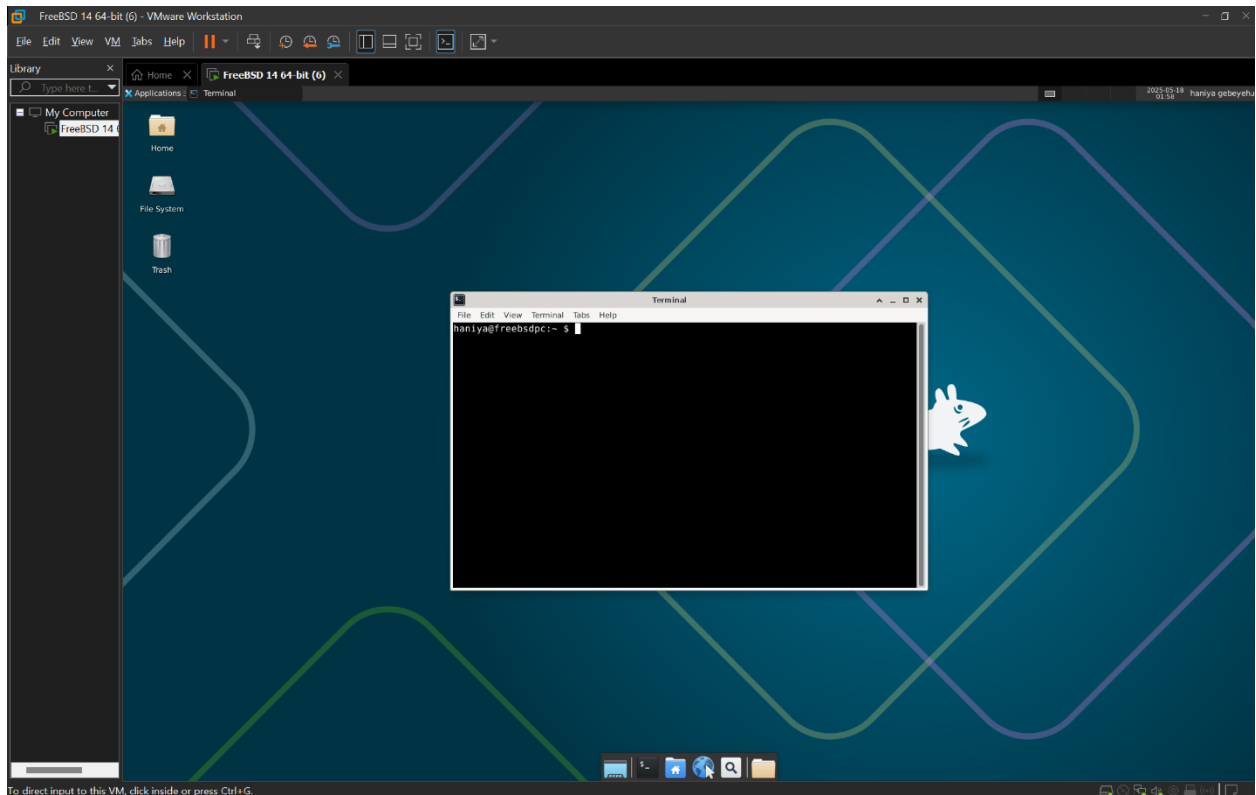
<u>Function</u>	<u>Purpose</u>
<code>fork()</code>	Create a new child process
<code>setpgid()</code>	Change the process group ID
<code>getpid()</code>	Get current process ID

<u>Function</u>	<u>Purpose</u>
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](#)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  int main() {
8      pid_t pid1, pid2, pid3;
9
10     printf("Parent process PID: %d\n", getpid());
11
12     pid1 = fork();
13     if (pid1 < 0) {
14         perror("Fork failed");
15         exit(1);
16     } else if (pid1 == 0) {
17         printf("First child PID: %d, Parent PID: %d\n", getpid(), getppid());
18         if (setpgid(0, 0) < 0) {
19             perror("setpgid failed");
20             exit(1);
21         }
22         printf("First child's process group ID: %d\n", getpgrp());
23         sleep(2);
24         exit(0);
25     }
26
27     sleep(1);
28
29     pid2 = fork();
30     if (pid2 < 0) {
31         perror("Fork failed");
32         exit(1);
33     } else if (pid2 == 0) {
34         printf("Second child PID: %d, Parent PID: %d\n", getpid(), getppid());
35         printf("Second child's process group ID: %d\n", getpgrp());
36         sleep(3);
37         exit(0);
38     }
39 }
```

```
40     sleep(1);
41
42     pid3 = fork();
43     if (pid3 < 0) {
44         perror("Fork failed");
45         exit(1);
46     } else if (pid3 == 0) {
47         printf("Third child PID: %d, Parent PID: %d\n", getpid(), getppid());
48         if (setpgid(0, 0) < 0) {
49             perror("setpgid failed");
50             exit(1);
51         }
52         printf("Third child's process group ID: %d\n", getpgrp());
53         sleep(4);
54         exit(0);
55     }
56
57     sleep(1);
58
59     printf("\nParent process group ID: %d\n", getpgrp());
60
61     waitpid(pid1, NULL, 0);
62     waitpid(pid2, NULL, 0);
63     waitpid(pid3, NULL, 0);
64
65     printf("All child processes have completed.\n");
66     return 0;
67 }
```

Step 4: write the code in the terminal

```
FreeBSD 14 64-bit (6) - VMware Workstation
File Edit View VM Tabs Help
Library
My Computer
FreeBSD 14
File Edit View Terminal Tabs Help
[ (escape) menu ^y search prompt ^k delete line ^o prev ll ^g prev page
^o ascii code ^s search ^l undelete line ^n next ll ^v next page
^u end of file ^a begin of line ^w delete word ^b back 1 char ^z next word
^t top of text ^e end of line ^r restore word ^f forward char
^c command ^d delete char ^l undelete char ESC-Enter: exit
=====line 1 col 0 lines from top 1 =====
#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());
    }
}
```

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

```

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", getpgid());
    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", getpgid());
    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", getpgid());
}

"process_groups.c" 68 lines, 1569 characters
haniya@freebsdpc:~$ cc -o process_groups process_groups.c
haniya@freebsdpc:~$ ./process_groups
Parent process PID: 1967
First child PID: 1968, Parent PID: 1967
First child's process group ID: 1968
Second child PID: 1969, Parent PID: 1967
Second child's process group ID: 1967
Third child PID: 1970, Parent PID: 1967
Third child's process group ID: 1970

Parent process group ID: 1967
All child processes have completed.
haniya@freebsdpc:~$

```

Breakdown:

- As we see, the code compiled without errors.