LAB 5: PROCESSES AND SIGNALS

Lab Exercises:

1. Write a C program to block a parent process until the child completes using a wait() system call.

Open | wait_example.c ~/Documents/ASS LAB/lab 5 | Save

wait_example.c | exec_example.c | process_id.c | *zombie.c

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

int main() {
    pid_t pid;

    printf("Parent process started\n");

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) {
        // Child process
        printf("Child process started\n");
        sleep(3); // Simulate some work
        printf("Child process completed\n");
    } else {
        // Parent process
        wait(NULL); // Wait for the child to complete
        printf("Parent process completed\n");
    }

    return 0;
}
```

rohithsaidatta@rohithsaidatta-VirtualBox: ~/Documents/ASS LAB/lab 5

```
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ gcc -o wait_example wait_example.c
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ ./wait_example
Parent process started
Child process started
Child process completed
Parent process completed
```

2. Write a C program to load the binary executable of the previous program in a child process using the exec system call.

Open | *exec_example.c ~/Documents/ASS LAB/lab 5 | Save

wait_example.c | *exec_example.c | process_id.c | *zombie.c

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

int main() {
    pid_t pid;

    printf("Parent process started\n");

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (pid == 0) {
        // Child process
        printf("Child process started\n");
        execl("./wait_example", "wait_example", NULL);
        perror("Exec failed");
        exit(1);
    } else {
        // Parent process
        wait(NULL);
        printf("Parent process completed\n");
    }

    return 0;
}
```

```
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ gcc -o exec_example exec_example.c
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ ./exec_example
Parent process started
Child process started
Parent process started
Child process started
Child process completed
Parent process completed
Parent process completed
```

3. Write a program to create a child process. Display the process IDs of the process, parent and child (if any) in both the parent and child processes.

Open *process_id.c ~/Documents/ASS LAB/lab 5 Save

wait_example.c | *exec_example.c | *process_id.c | *zombie.c

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

int main() {
    printf("Parent PID: %d\n", getpid());

    pid_t child_pid = fork();

    if (child_pid == 0) {
        // This is the child process
        printf("Child PID: %d, Parent PID: %d\n", getpid(), getppid());
    }

    return 0;
}
```

```
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ gcc -o process_id process_id.c
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ ./process_id
Parent PID: 3231
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ Child PID: 3232, Parent PID: 1218
```

4. Create a zombie (defunct) child process (a child with exit() call, but no corresponding wait() in the sleeping parent) and allow the init process to adopt it (after parent terminates). Run the process as a background process and run the “ps” command.

Open *zombie.c ~/Documents/ASS LAB/lab 5 Save

wait_example.c | *exec_example.c | *process_id.c | *zombie.c

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

int main() {
    pid_t child_pid = fork();

    if (child_pid < 0) {
        perror("Fork failed");
        exit(1);
    } else if (child_pid == 0) {
        // This is the child process
        printf("Child PID: %d\n", getpid());
        exit(0); // Child exits without waiting
    } else {
        // This is the parent process
        printf("Parent PID: %d, Child PID: %d\n", getpid(), child_pid);
        // Parent sleeps briefly, allowing the child to become a zombie
        sleep(10);
    }

    return 0;
}
```

```
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ gcc -o zombie zombie.c
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab 5$ ./zombie & ps aux | grep zombie
[1] 3321
Parent PID: 3321, Child PID: 3324
Child PID: 3324
rohiths+    3321  0.0  0.0   2772  1408 pts/2    S    12:54   0:00 ./zombie
rohiths+    3323  0.0  0.1  17864  2560 pts/2    S+   12:54   0:00 grep --color=auto zombie
rohiths+    3324  0.0  0.0      0     0 pts/2    Z    12:54   0:00 [zombie] <defunct>
```