

LAB 5: PROCESSES AND SIGNALS

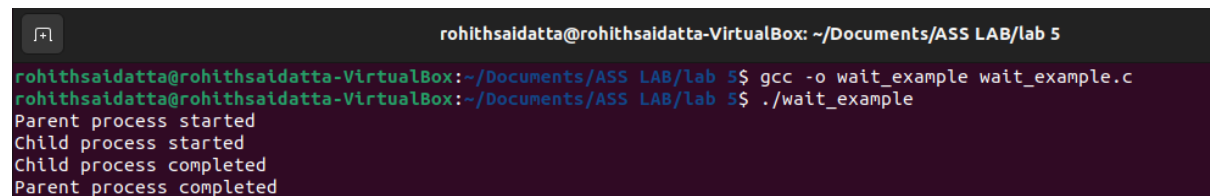
Lab Exercises:

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



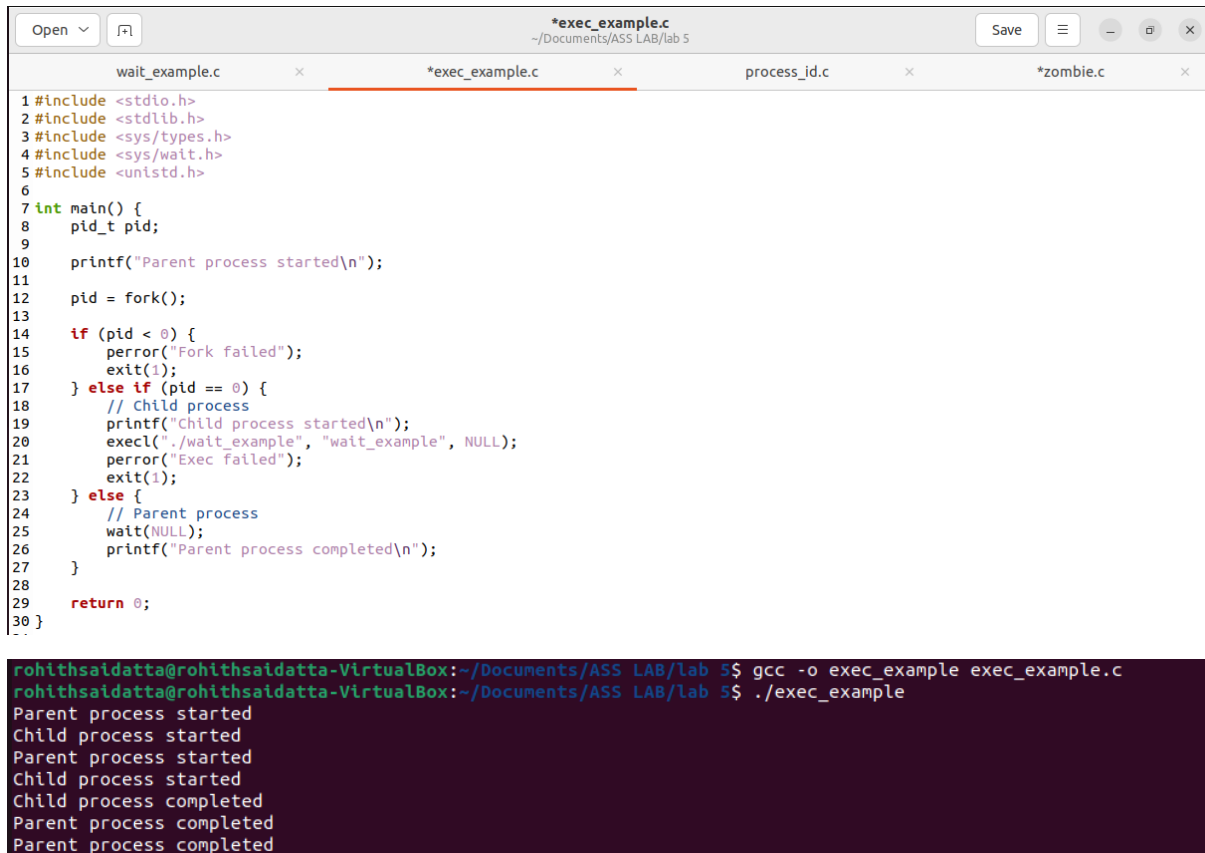
```
wait_example.c
~/Documents/ASS LAB/lab 5
Save

wait_example.c x exec_example.c x process_id.c x *zombie.c x
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6
7 int main() {
8     pid_t pid;
9
10    printf("Parent process started\n");
11
12    pid = fork();
13
14    if (pid < 0) {
15        perror("Fork failed");
16        exit(1);
17    } else if (pid == 0) {
18        // Child process
19        printf("Child process started\n");
20        sleep(3); // Simulate some work
21        printf("Child process completed\n");
22    } else {
23        // Parent process
24        wait(NULL); // Wait for the child to complete
25        printf("Parent process completed\n");
26    }
27
28    return 0;
29 }
```



```
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.




The image shows a code editor window with four tabs: `wait_example.c`, `*exec_example.c` (active), `process_id.c`, and `*zombie.c`. The active tab contains the following C code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <unistd.h>
6
7 int main() {
8     pid_t pid;
9
10    printf("Parent process started\n");
11
12    pid = fork();
13
14    if (pid < 0) {
15        perror("Fork failed");
16        exit(1);
17    } else if (pid == 0) {
18        // Child process
19        printf("Child process started\n");
20        execl("./wait_example", "wait_example", NULL);
21        perror("Exec failed");
22        exit(1);
23    } else {
24        // Parent process
25        wait(NULL);
26        printf("Parent process completed\n");
27    }
28
29    return 0;
30 }
```

Below the code editor is a terminal window showing the execution of the program. The terminal output is as follows:

```
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab $ gcc -o exec_example exec_example.c
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab $ ./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.



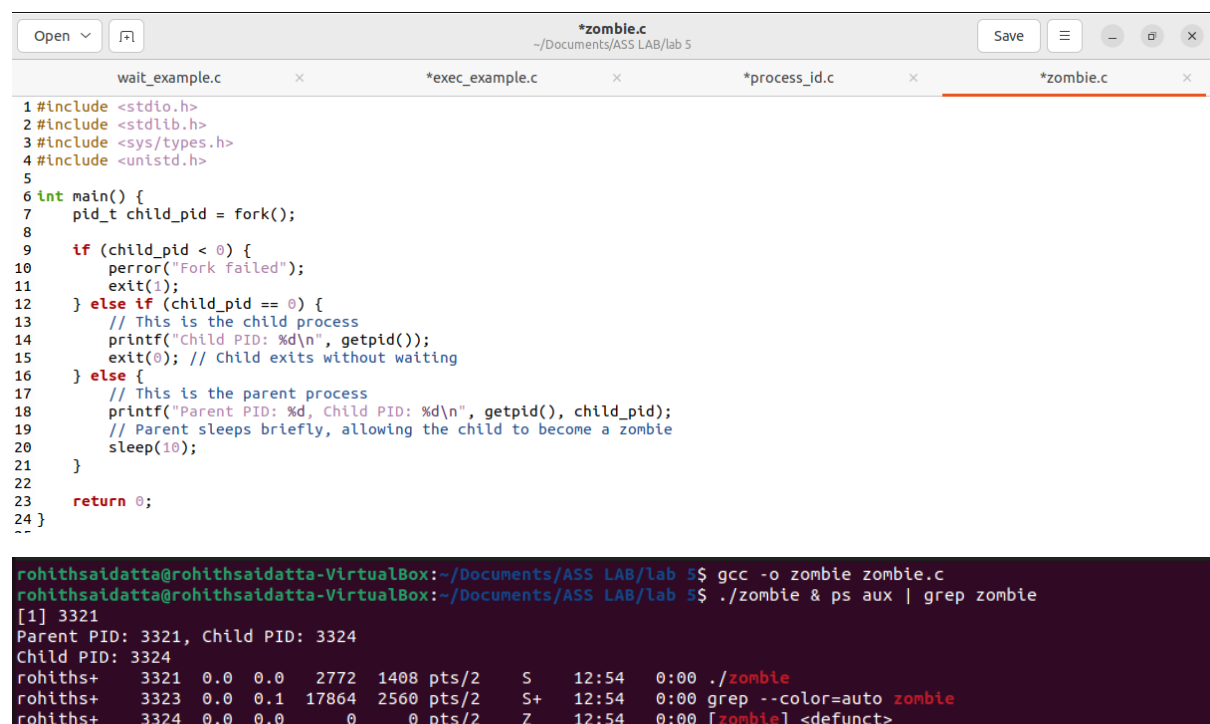
The screenshot shows a code editor with four tabs: wait_example.c, *exec_example.c, *process_id.c (active), and *zombie.c. The *process_id.c file contains the following C code:

```
1#include <stdio.h>
2#include <sys/types.h>
3#include <unistd.h>
4
5int main() {
6    printf("Parent PID: %d\n", getpid());
7
8    pid_t child_pid = fork();
9
10   if (child_pid == 0) {
11       // This is the child process
12       printf("Child PID: %d, Parent PID: %d\n", getpid(), getppid());
13   }
14
15   return 0;
16}
```

Below the code, the terminal output is shown:

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



The screenshot shows a code editor with four tabs: wait_example.c, *exec_example.c, *process_id.c, and *zombie.c (active). The *zombie.c file contains the following C code:

```
1#include <stdio.h>
2#include <stdlib.h>
3#include <sys/types.h>
4#include <unistd.h>
5
6int main() {
7    pid_t child_pid = fork();
8
9    if (child_pid < 0) {
10        perror("Fork failed");
11        exit(1);
12    } else if (child_pid == 0) {
13        // This is the child process
14        printf("Child PID: %d\n", getpid());
15        exit(0); // Child exits without waiting
16    } else {
17        // This is the parent process
18        printf("Parent PID: %d, Child PID: %d\n", getpid(), child_pid);
19        // Parent sleeps briefly, allowing the child to become a zombie
20        sleep(10);
21    }
22
23    return 0;
24}
```

Below the code, the terminal output is shown:

```
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab $ gcc -o zombie zombie.c
rohithsaidatta@rohithsaidatta-VirtualBox:~/Documents/ASS LAB/lab $ ./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>
```