



# **Discipline**

**Activity**

**CS5003**

Activity 2

## ❖ Problem Statement:

In this activity, you have to gain Process API familiarity with process management.

You are asked to refer to the classroom GitHub repository (Classroom codes) to get started with this activity.

Problems to address and Inquiries to be made.

1. Write a piece of code that calls `fork ()`. Before calling `fork ()`, have the main process access a variable (e.g., `test`) and set its value to some number (eg.,500). Execute, inquire and validate the outcome -
  - a. What value is the variable in the child process?
  - b. What happens to the variable mentioned when both parent and child change the value?
2. Write a program that uses a `wait ()` system call for the child process to finish in the parent what does `wait` return? What will happen if you use `wait ()` system call for child?

## ❖ Solution:

1)

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the `fork ()` call (parent process). After a new child process is created, both processes will execute the next instruction following the `fork ()` system call. A child process uses the same pc (program counter), same CPU registers, same open files which use in the parent process.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<sys/wait.h>
5
6 int main(int argc, char *argv[]){
7     printf("hi cs5003 students I am your Instructor (pid=%d)",(int) getpid());
8     int tadeeb = 400;
9     int cr=fork();
10    if (cr<0){
11        fprintf(stderr,"fork failed \n");
12        exit(1);
13    }else if(cr==0){
14        wait (NULL);
15        tadeeb += 1;
16        printf("hello i am student (pid=%d)\n", (int) getpid());
17        printf("Variable Value (variable=%d)\n", (int) tadeeb);
18    }else {
19        wait (NULL);
20        tadeeb += 1;
21        printf("hello i am instructor of %d (pid=%d)\n",cr,(int) getpid());
22        printf("Variable Value (variable=%d)\n", (int) tadeeb);
23    }
24    return 0;
25 }
```

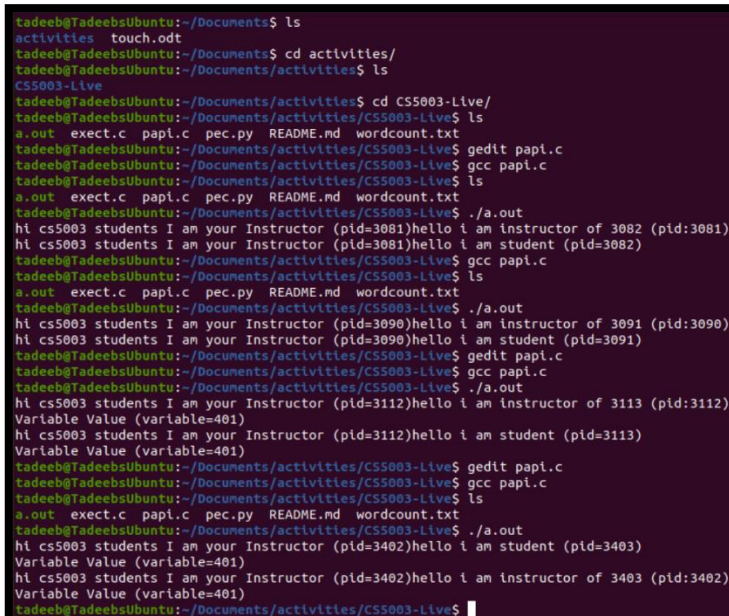
It takes no parameters and returns an integer value. Above are different values returned by fork ().

- a) The variable value will be incremented by 1 i.e., it will be 401 after execution.
- b) The variable will also change the value and its value will remain same as that of parent because fork () creates a clone of its parent and give it to its child.

2)

A call to wait () blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction. Child process may terminate due to any of these:

- It calls exit ();



```
tadeeb@TadeebUbuntu:~/Documents$ ls
activities touch.odt
tadeeb@TadeebUbuntu:~/Documents$ cd activities/
tadeeb@TadeebUbuntu:~/Documents/activities$ ls
CS5003-Live
tadeeb@TadeebUbuntu:~/Documents/activities$ cd CS5003-Live/
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ls
a.out exect.c papi.c pec.py README.md wordcount.txt
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ gedit papi.c
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ gcc papi.c
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ls
a.out exect.c papi.c pec.py README.md wordcount.txt
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ./a.out
hi cs5003 students I am your Instructor (pid=3081)hello i am instructor of 3082 (pid:3081)
hi cs5003 students I am your Instructor (pid=3081)hello i am student (pid=3082)
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ gcc papi.c
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ls
a.out exect.c papi.c pec.py README.md wordcount.txt
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ./a.out
hi cs5003 students I am your Instructor (pid=3090)hello i am instructor of 3091 (pid:3090)
hi cs5003 students I am your Instructor (pid=3090)hello i am student (pid=3091)
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ gedit papi.c
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ gcc papi.c
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ./a.out
hi cs5003 students I am your Instructor (pid=3112)hello i am instructor of 3113 (pid:3112)
Variable Value (variable=401)
hi cs5003 students I am your Instructor (pid=3112)hello i am student (pid=3113)
Variable Value (variable=401)
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ gedit papi.c
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ gcc papi.c
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ls
a.out exect.c papi.c pec.py README.md wordcount.txt
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$ ./a.out
hi cs5003 students I am your Instructor (pid=3402)hello i am student (pid=3403)
Variable Value (variable=401)
hi cs5003 students I am your Instructor (pid=3402)hello i am instructor of 3403 (pid:3402)
Variable Value (variable=401)
tadeeb@TadeebUbuntu:~/Documents/activities/CS5003-Live$
```

From The above figure we can see that if any process has more than one child processes, then after calling wait (), parent process has to be in wait state if no child terminates. If only one child process is terminated, then return a wait () returns process ID of the terminated child process. If more than one child processes are terminated than wait () reap any arbitrarily child and return a process ID of that child process.