**Terna Engineering College**

**Computer Engineering Department**

Program: Sem IV

**Course: Operating System Lab**

**Faculty:**

LAB Manual

 PART A

(PART A : TO BE REFFERED BY STUDENTS)

**Experiment No-04**

PART A

(PART A: TO BE REFFERED BY STUDENTS)

**A.1 Aim:**

Create a child process in Linux using **fork** system call From the child process obtain the process ID of both child and parent by using getpid and getppid system call. Explore wait and waitpid before termination of process.

**A-2 Prerequisite**

Knowledge about Linux operating system environment and Process concepts.

.

1. **OutCome**

After successful completion students will able to explore various system calls.

1. **Theory:**

* **Process Creation:**

To create a new process we use the fork() system call. The fork system call actually clones the calling process, with very few differences. The clone has a different process id (PID) and parent process id (PPID).

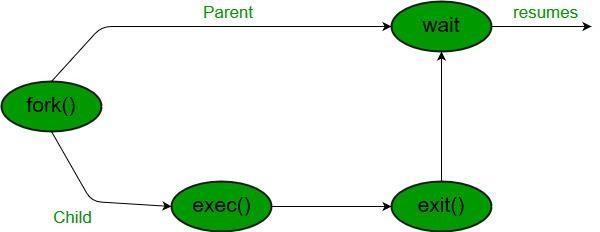
* The new process created by fork is called the **child process**. This function is called once but returns twice. The only difference in the returns is that the return value in the child is **0**, whereas the return value in the parent is the process ID of the new child.
  + fork returns child's process ID in parent: a process can have more than one child, and there is no function that allows a process to obtain the process IDs of its child

fork returns 0 in child: a process can have only a single parent, and the child can always call **getppid** to obtain the process ID of its parent

* Both the child and the parent continue executing with the instruction that follows the call to fork. The child is a copy of the parent. For example, the child gets a copy of the parent’s data space, heap, and stack. Note that this is a copy for the child the parent and the child do not share these portions of memory. The parent and the child do share the text segment.

**Useful Funtions**

* pid\_tgetpid(void): get process id (PID) of calling process
* pid\_tgetppid(void): get process id (PID) of the parent of calling process, a.k.a parent process id (PPID)
* wait() and waitpid() :

There are certain situations where when a child process terminates or changes state then the parent process should come to know about the change of the state or termination status of the child process. In that case functions like **wait()** are used by the parent process where the parent can query the change in state of the child process using these functions.

For the cases where a parent process has more than one child processes, there is a

function **waitpid()** that can be used by the parent process to query the change state of a particular child.

By default, waitpid() waits only for terminated children, but this behavior is modifiable via the

options argument, as described below.

The value of pid can be:

* < -1 : Wait for any child process whose process group ID is equal to the absolute value of pid.
* -1 : Wait for any child process.
* 0 : Wait for any child process whose process group ID is equal to that of the calling process.
* >0 : Wait for the child whose process ID is equal to the value of pid.

The value of options is an OR of zero or more of the following constants:

* WNOHANG : Return immediately if no child has exited.
* WUNTRACED : Also return if a child has stopped. Status for traced children which have stopped is provided even if this option is not specified.
  + WCONTINUED : Also return if a stopped child has been resumed by delivery of

SIGCONT

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the ERP or emailed to the concerned lab conducting faculties at the end of the practical in case the there is no ERP access available)***

|  |  |
| --- | --- |
| Roll. No. B75 | Name: Nilesh Gawli |
| Class: comp SE(B) | Batch: B4 |
| Date of Experiment: | Date of Submission: |
| Grade: | |

1. **Software Code written by student:**

***(Paste your Search material completed during the 2 hours of practical in the lab here)***

Program code in C.

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

int main(void) {

  pid\_t pid = fork();

  if(pid==0) {

printf("Child => PPID: %d PID: %d\n",getppid(),getpid());

exit(EXIT\_SUCCESS);

  }

  else if(pid > 0) {

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

printf("Waiting for child process to finish.\n");

wait(NULL);

printf("Child process finished.\n");

  }

  else {

printf("Unable to create child process.\n");

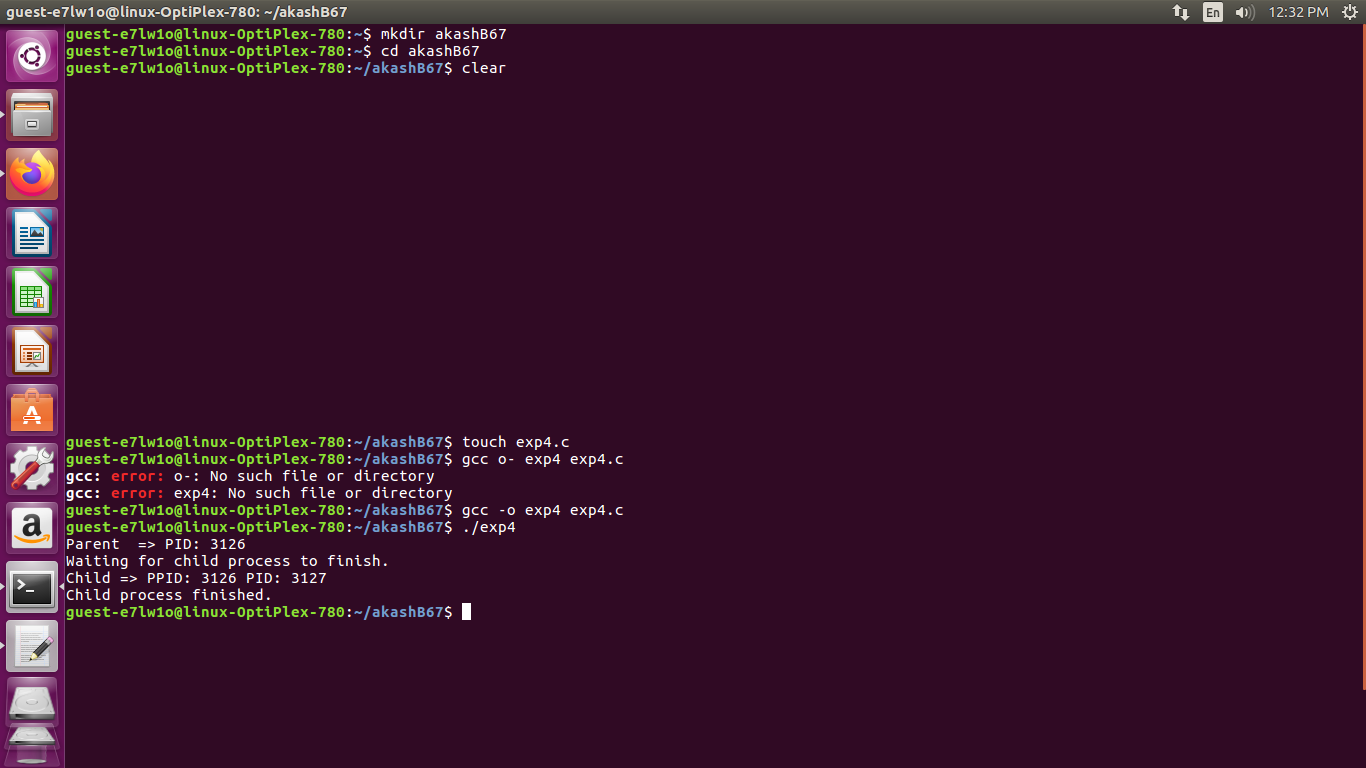
  }

  return EXIT\_SUCCESS;

}

1. **Input and Output:**

***(input and output of the program)***

******

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

Hence, we have successfully performed this practical and learnt to create child process using fork() system call.