**Learning Objectives:** Students will be able to learn UNIX process management system calls and library functions.

# Step 1:

Consider the following C program. #include <stdio.h>

#include <unistd.h>

for(i = 0; i < 3; i++)

{

if (pid = fork() < 0)

// error else if (pid == 0)

{

function\_A(); return 0;

}

printf(“process ID: %d \n”, pid); // Line A

}

for(i = 0; i < 10; i++) //Line B wait();

1. How many new processes are created in the program? Why?

**fork() is to create a new process by duplicating the existing process**

**Answer: 7 child processes will be created**

1. Which process, the parent or the child, executes function\_A()? Why?

**Child Process: because pid == 0 means its executing under the new process of child process.**

1. Whose PID, the parent or the child, is printed in Line A? Why?

**Both Parent and child process id’s will be printed. Whatever created by the parent must be created by the Child.**

1. What is the purpose of the loop (with its wait()) in Line B?

**Telling the parent process to wait until all the child processes to be executed.**

# Step 2:

Consider the following C program.

// Assume variables *i* and *pid*, and constant *N* have been properly defined, and/or initialized and there is no syntax error.

int main ()

{

for(*i* =0; *i* < *N*; *i*++) {

*pid*=fork ();

}

}

1. For *N*=5, How many processes are created when the program is executed?

32 total processes are created

1. Modify the program so that only the parent process creates 3 child processes(meaning the initial execution of the parent process being 3 child processes) , and each new created process calls a function CPU(). In addition, make the parent process wait for each child’s termination.

CPU Function will be called 7 times

**#include <stdio.h>**

**#include <unistd.h>**

**#define N 3**

**int main()**

**{**

**int i;**

**int pid;**

**for(i = 0;i < N; i++)**

**{**

**if(pid = fork() == 0)**

**{**

**CPU();**

**}**

**else**

**{**

**wait();**

**}**

**}**

**printf("Process Created %d, %d\n",getpid(),pid);**

**}**

**void CPU()**

**{**

**printf("Info Stack: %d, %d\n",getppid(),getpid());**

**}**

**"l3q2mod.c" 29L, 306C**

# Step 3:

Consider the following program. What will be the output in Line A?

int value = 60; int main()

{

pid\_t pid; pid = fork();

if (pid == 0) {

value = value + 20;

}

else if (pid > 0) { value = value -20;

printf("PARENT: value= %d \n", value); **//Line A**

wait (NULL);

}

}

# Step 4:

Consider the following program. #include <sys/types.h>

#include <stdio.h>

#include <unistd.h>4

int value = 100; int main()

{

pid\_t pid; pid = fork();

if (pid == 0) {

value = value + 15;

}

else if (pid > 0) {

value = value -15;

printf("PARENT: value= %d \n", value); wait (NULL);

}

}

1. What will be the output in Line A? Justify your answer.
2. Do you think there is synchronization problem in updating the variable value? Justify your answer.

# Step 5:

Consider the following programs **A** and **B**.

* 1. How many times will the fork () function be called in **Program A**? (*i.e*., how many processes are created?) Justify your answer.

**16 total processes will be created**

* 1. What is the output of **Line A** in **Program B**? Justify your answer.

**In Program B, since the fork creates two processes in which one is parent and other is child, parent process returns value 15 and child process returns value 45**

In program A

# // Program A

int main()

{

pid\_t pid; int i;

}

# // Program B

int value = 30;

for (i=0; i<4; i++)

pid = fork();

int main()

{

pid\_t pid; pid = fork();

if (pid == 0)

value = value + 15; else if (pid > 0) {

value = value -15; wait (NULL);

}

printf("Value= %d \n", value); //Line A

}