



西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

Lab **2** Report

Report Subject: OS Experiment - Lab **2**

Student ID: 2018380039

Student Name: Dikshya Kafle

Submission Date: **2021/11/18**

Computer Operating System Experiment

Computer Operating System

Experiment Laboratory 2

Process

1) **Objective:**

Learn to work with Linux system calls related to process creation and control.

Including: ☐

- ❖ Process create and data sharing between parent and child ☐
- ❖ The execution order of parent and child process ☐
- ❖ Create the specified num of child processes ☐
- ❖ Process termination ☐
- ❖ Zombie process ☐
- ❖ Process create a child process and load a new program

2) **Equipment:**

- ❖ VirtualBox with Ubuntu Linux 20.04

3) **Experiments:**

Experiment 1: Process Creation

1. If you change the values of variable x ,y and i in parent process, do the variable in the child process will affected? Please give the reason.

Answer:

- ❖ If we change the values of variable x, y and i in the parent process the variables in the child **will not be affected**. This is because both the child and the parent process **have different copies of the data** and this data is not shared between them.

2. Please modify the fork-ex.c, and create a Makefile that builds all the programs you created. Test your expectation.

Answer:

- ❖ We modify the fork-ex.c to change the variables x,y and j on the parent process and test our last answer. The results are as expected and changing the variables in the parent process does not affect the variables in the child process.

```
Open ▾ [F1] Makefile  
~/Desktop/Lab 2 OS  
1 fork-ex: fork-ex.c  
2      gcc fork-ex.c -o fork-ex  
_
```

```
Open ▾ [F1] fork-ex.c  
~/Desktop  
6 double x = 3.14159;  
7 int pid;  
8  
9 int main(int argc, char *argv[])  
10 {  
11  
12     int j = 2;  
13     double y = 0.12345;  
14  
15     pid = fork();  
16     if (pid < 0)  
17     {  
18         fprintf(stderr, "Fork failed");  
19         return 1;  
20     }  
21     if (pid > 0)  
22     {  
23         // parent code  
24         printf("parent process -- pid= %d\n", getpid());  
25         fflush(stdout);  
26  
27         i = 20;  
28         y = 0.78912;  
29         x = 1.428571;  
30  
31         printf("parent sees: i= %d, x= %lf\n", i, x);  
32         fflush(stdout);  
33         printf("parent sees: j= %d, y= %lf\n", j, y);  
34         fflush(stdout);  
35     }  
36     else  
37     {
```

```
dikshya@dikshya-VirtualBox: ~/Desktop
dikshya@dikshya-VirtualBox:~/Desktop$ gcc fork-ex.c -o fork-ex
dikshya@dikshya-VirtualBox:~/Desktop$ ./fork-ex
parent process -- pid= 20169
parent sees: i= 20, x= 1.428571
parent sees: j= 2, y= 0.789120
child process -- pid= 20170
child sees: i= 10, x= 3.141590
child sees: j= 2, y= 0.123450
dikshya@dikshya-VirtualBox:~/Desktop$
```

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ make
gcc fork-ex.c -o fork-ex
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ./fork-ex
parent process -- pid= 25498
parent sees: i= 20, x= 1.428571
parent sees: j= 2, y= 0.789120
child process -- pid= 25499
child sees: i= 10, x= 3.141590
child sees: j= 2, y= 0.123450
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

I have changed the variables in the parent process and run.

As we can see that changes of the variables in the parent process didn't affect child process.

Experiment 2: the execution order of parent and child process

1. The global variable num is declared before the call to fork() as shown in this program. After the call to fork(), when a new process is spawned, does there exist only one instance of num

in the memory space of the parent process shared by the two processes or do there exist two instances: one in the memory space of the parent and one in the memory space of the child?

Answer:

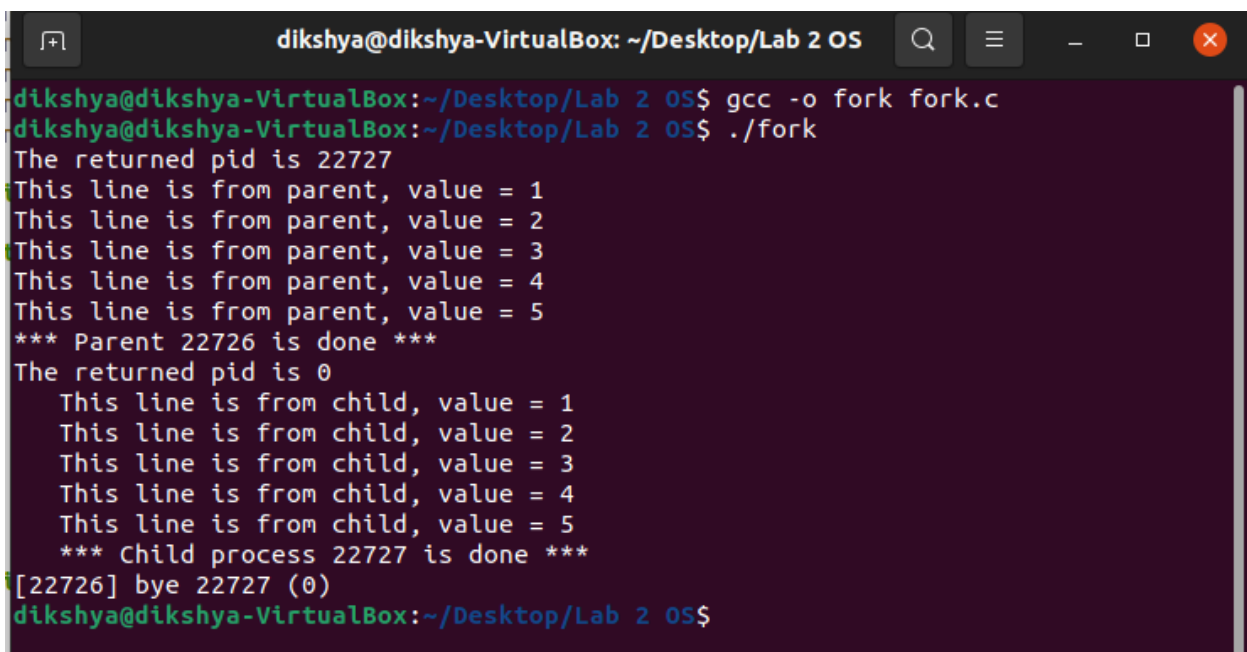
- ❖ There exist two instances of num, one in the memory space of the parent and one in the memory space of the child. This is because the fork will create a new copy of the parent process with a new pid that has a separate address space. So each process will have its own copy of the data instead of this data being shared between them.

2. Can you infer the order of execution of these lines? Please try to decrease the num, If the value of num is so small that a process can finish in one time quantum, you will see two groups of lines, each of which contains all lines printed by the same process.

Answer:

- ❖ In this code we can't infer the order of execution of the lines. It is true that the parent is using wait () but this wait is not called right after the fork and the parent will still execute the for loop concurrently with the child process. However, the parent process will not return until the child process has returned. To illustrate this I increased the value of num from 5 to 40 (5 and 10 was too small that each process could finish in one time quantum) and tested the results. As expected that at one point the parent and the child process were executing concurrently.

When num=5



```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 2 OS
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ gcc -o fork fork.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ./fork
The returned pid is 22727
This line is from parent, value = 1
This line is from parent, value = 2
This line is from parent, value = 3
This line is from parent, value = 4
This line is from parent, value = 5
*** Parent 22726 is done ***
The returned pid is 0
  This line is from child, value = 1
  This line is from child, value = 2
  This line is from child, value = 3
  This line is from child, value = 4
  This line is from child, value = 5
  *** Child process 22727 is done ***
[22726] bye 22727 (0)
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

When num=10

```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 2 OS
This line is from parent, value = 2
This line is from parent, value = 3
This line is from parent, value = 4
This line is from parent, value = 5
This line is from parent, value = 6
This line is from parent, value = 7
This line is from parent, value = 8
This line is from parent, value = 9
This line is from parent, value = 10
*** Parent 22682 is done ***
The returned pid is 0
  This line is from child, value = 1
  This line is from child, value = 2
  This line is from child, value = 3
  This line is from child, value = 4
  This line is from child, value = 5
  This line is from child, value = 6
  This line is from child, value = 7
  This line is from child, value = 8
  This line is from child, value = 9
  This line is from child, value = 10
  *** Child process 22683 is done ***
[22682] bye 22683 (0)
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

When num=40

```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 2 OS
This line is from parent, value = 24
  This line is from child, value = 17
This line is from parent, value = 25
  This line is from child, value = 18
This line is from parent, value = 26
  This line is from child, value = 19
This line is from parent, value = 27
  This line is from child, value = 20
This line is from parent, value = 28
  This line is from child, value = 21
This line is from parent, value = 29
  This line is from child, value = 22
This line is from parent, value = 30
  This line is from child, value = 23
This line is from parent, value = 31
  This line is from child, value = 24
This line is from parent, value = 32
  This line is from child, value = 25
This line is from parent, value = 33
This line is from parent, value = 34
  This line is from child, value = 26
This line is from parent, value = 35
  This line is from child, value = 27
This line is from parent, value = 36
  This line is from child, value = 28
This line is from parent, value = 37
  This line is from child, value = 29
This line is from parent, value = 38
  This line is from child, value = 30
This line is from parent, value = 39
  This line is from child, value = 31
This line is from parent, value = 40
  This line is from child, value = 32
*** Parent 22855 is done ***
  This line is from child, value = 33
  This line is from child, value = 34
  This line is from child, value = 35
  This line is from child, value = 36
  This line is from child, value = 37
  This line is from child, value = 38
  This line is from child, value = 39
  This line is from child, value = 40
  *** Child process 22856 is done ***
[22855] bye 22856 (0)
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

Experiment 3: Create the specified num of child processes

1. Please observe the pid and can you tell the policy about pid allocation? Can you determine the parent process and child process from the pid?

Answer:

- ❖ We can observe that the policy about pid allocation Under Unix implies that process IDs are allocated on a sequential basis, beginning at 0. However, we cannot determine the parent and child process from the pid. For example if we run `pgrep -P 22713` (where 22713 is the bash pid) we get: 22695 (not 22712)(see last screen-shot)

```
File  Machine  View  Input  Devices  Help
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <sys/wait.h>
6
7 int forkChildren(const char *whoami,int i)
8 {
9     printf("I'm a %s %d PID is:%d  my parent id : is %d\n",
10           whoami, i,getpid(), getppid() );
11     return 1;
12 }
13 int i=0;
14
15 int main(void)
16 {
17     int n= 5;
18
19     int status=0;
20
21
22     printf("Creating %d children\n", n);|
23     //forkChildren("parent",i);
24     for(i=0;i<n;i++)
25     {
26         pid_t pid=fork();
27
28         if (pid==0) /* only execute this if child */
29         {
30             forkChildren("child",i);
31
32             exit(0);
33
34         }
35         wait(&status);  /* only the parent waits */
36     }
37 }
```

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ touch forkN.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ gcc -o forkN forkN.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ./forkN
Creating 5 children
I'm a child 0 PID is:23478 my parent id : is 23477
I'm a child 1 PID is:23479 my parent id : is 23477
I'm a child 2 PID is:23480 my parent id : is 23477
I'm a child 3 PID is:23481 my parent id : is 23477
I'm a child 4 PID is:23482 my parent id : is 23477
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

- ❖ Can you determine the parent process and child process from the pid? --> No - example:

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ touch forkN.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ gcc -o forkN forkN.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ./forkN
Creating 5 children
I'm a child 0 PID is:23478 my parent id : is 23477
I'm a child 1 PID is:23479 my parent id : is 23477
I'm a child 2 PID is:23480 my parent id : is 23477
I'm a child 3 PID is:23481 my parent id : is 23477
I'm a child 4 PID is:23482 my parent id : is 23477
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ps
  PID TTY          TIME CMD
 22713 pts/0        00:00:00 bash
 23582 pts/0        00:00:00 ps
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ps -o ppid=22713
 22713
 22695
 22713
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

Experiment 4: Process termination

Copy fork_ex.c to file fork-wait.c and modify it so that you can guarantee that the parent process will always terminate after the child process has terminated. Your solution cannot rely on the termination condition of the for loops or on the use of sleep. The right way to handle this is using a syscall such as wait or waitpid – read their man pages before jumping into this task. One more thing: Modify the child process so that it makes calls to getpid(2) and getppid(2) and prints out the values returned by these calls


```
1 #include <unistd.h> // need this for fork
2 #include <sys/types.h>
3 #include <sys/wait.h>
4 #include <stdio.h> // need this for printf and fflush
5 #include <stdlib.h>
6
7 int i = 10;
8 double x = 3.14159;
9 int pid;
10
11 int main(int argc, char *argv[])
12 {
13     int j = 2;
14     double y = 0.12345;
15
16     pid = fork();
17     if (pid < 0)
18     {
19         fprintf(stderr, "Fork failed");
20         return 1;
21     }
22     if (pid > 0)
23     {
24         int status;
25
26         // parent code
27         printf("parent process -- pid= %d\n", getpid());
28         fflush(stdout);
29     }
```

```

29     fflush(stdout);
30
31     i = 20;
32     y = 0.78912;
33     x = 1.428571;
34
35     printf("parent sees: i= %d, x= %lf\n", i, x);
36     fflush(stdout);
37     printf("parent sees: j= %d, y= %lf\n", j, y);
38     fflush(stdout);
39
40     wait(&status);
41 }
42 else
43 {
44     // child code
45     printf("child process -- pid= %d, ppid= %d\n", getpid(), getppid());
46     fflush(stdout);
47     printf("child sees: i= %d, x= %lf\n", i, x);
48     fflush(stdout);
49     printf("child sees: j= %d, y= %lf\n", j, y);
50     fflush(stdout);
51 }
52
53 return (0);
54 }

```

```

dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 05$ touch fork-wait.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 05$ gcc -o fork-wait fork-wait.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 05$ ./fork-wait
parent process -- pid= 23798
parent sees: i= 20, x= 1.428571
parent sees: j= 2, y= 0.789120
child process -- pid= 23799, ppid= 23798
child sees: i= 10, x= 3.141590
child sees: j= 2, y= 0.123450
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 05$

```

Experiment 5: Zombie process

- ❖ To make a zombie process, I made the parent process sleep before executing the wait() and made the child process exit. I execute the ./zombie command on a shell window and executed “ps aux” in the another window while the parent process is sleeping and I found that child process has a “zombie” state.

```
Open  [icon] zombie.c ~/Desktop/Lab 2 OS
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 int main(int argc, char **argv)
5 {
6     int i = 0;
7     pid_t pid = fork();
8     if (pid == -1)
9         return 0;
10    else if (pid == 0)
11    { // child
12        printf("child pid is %d\n", getpid());
13        printf("child terminated\n");
14    }
15    else if (pid > 0)
16    { // Parent
17        printf("parent pid is %d\n", getpid());
18        while (1)
19            sleep(100);
20    }
21    return 0;
22 }
```

The zombie process created through this code will run for 100 seconds. We can increase the time duration by specifying a time (in seconds) in the sleep() function.

Now I run the zombie program:

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ gcc -o zombie zombie.c
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ./zombie
parent pid is 24035
child pid is 24036
child terminated

dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$ ps -al
F S  UID      PID     PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000      982      976  0  80   0 - 57621 do_pol  tty2        00:00:00 gnome-session-b
0 S  1000    24035    24014  0  80   0 -   688 hrtime  pts/1        00:00:00 zombie
1 Z  1000    24036    24035  0  80   0 -     0 -      pts/1        00:00:00 zombie <defunct>
0 R  1000    24133    24041  0  80   0 - 5376 -      pts/0        00:00:00 ps
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS$
```

The **ps** commands will now also show this defunct process, we open a new terminal and use the command to check the defunct process(**zombie process**):

1. Can you use kill command to kill the zombie process? If not, how can you reap the zombie process?

```
[icon] dikshya@dikshya-VirtualBox: ~
dikshya@dikshya-VirtualBox:~$ gcc -o zombie zombie.c
cc1: fatal error: zombie.c: No such file or directory
compilation terminated.
dikshya@dikshya-VirtualBox:~$
```

A Zombie process can't be killed because it is already dead. One thing we can try to reap it is to notify its parent process explicitly so that it can retry to read the child (dead) process's status and eventually clean it from the process table. This can be done by sending a SIGCHLD signal to the parent process.

\$ kill -s SIGCHLD <Parent PID>

We have seen that to get the parent pid we can simply run: *\$ ps -o ppid= <Child PID>*

However, this might not always work. In our case the parent process is sleeping, so the only way to reap the child zombie process is by killing its parent. After the parent dies, the zombie will be inherited by pid 1, which will wait on it and clear its entry in the process table (This solution always works):

\$ kill <Parent PID>

The init process regularly performs the necessary cleanup of zombies, so to kill them, you just have to kill the process that created them

Experiment 6: create a child process and load a new program

I first code a Scan() function, which performs a lexical analysis to the user input. The contents of the command entered by the user are loaded into the args array as specified. Then the parent process forks a new child process that run the execvp() function. If the user added "&" the parent process will wait for the child process to exit, otherwise they will run concurrently.

Modified Code:

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <errno.h>
```

```
#include <stdlib.h>
```

```
#include <sys/types.h>
```

```
#include <string.h>
```

```
#define MAX_LINE      80 /* 80 chars per line, per command, should be enough. */
```

```
#define MAX_COMMANDS5 /* size of history */
```

```
char history[MAX_COMMANDS][MAX_LINE];
```

```
char display_history [MAX_COMMANDS][MAX_LINE];
```

```
int command_count = 0;
```

```
/**
```

```
 * Add the most recent command to the history.
```

```
 */
```

```
void addtohistory(char inputBuffer[]) {
```

```
    int i = 0;
```

```
    // add the command to history
```

```
    strcpy(history[command_count % MAX_COMMANDS], inputBuffer);
```

```
    // add the display-style command to history
```

```
    while (inputBuffer[i] != '\n' && inputBuffer[i] != '\0') {
```

```
        display_history[command_count % MAX_COMMANDS][i] = inputBuffer[i];
```

```
        i++;
```

```
    }
```

```
    display_history[command_count % MAX_COMMANDS][i] = '\0';
```

```
    ++command_count;
```

```
    return;
```

```
}
```

```
int setup(char inputBuffer[], char *args[],int *background)
```

```
{
```

```
    int length,          /* # of characters in the command line */
```

```
    i,                   /* loop index for accessing inputBuffer array */
```

```
    start,               /* index where beginning of next command parameter is */
```

```

    ct,                /* index of where to place the next parameter into args[] */
    command_number;    /* index of requested command number */

ct = 0;

/* read what the user enters on the command line */

do {
    printf("os>");
    fflush(stdout);
    length = read(STDIN_FILENO,inputBuffer,MAX_LINE);
}
while (inputBuffer[0] == '\n'); /* swallow newline characters */


start = -1;

if (length == 0)
    exit(0);          /* ^d was entered, end of user command stream */


if ( (length < 0) && (errno != EINTR) ) {
    perror("error reading the command");
    exit(-1);         /* terminate with error code of -1 */
}


/**
 * Check if they are using history
 */

if (inputBuffer[0] == '!') {

```

```

    if (command_count == 0) {
        printf("No history\n");
        return 1;
    }

    else if (inputBuffer[1] == '!') {
        // restore the previous command

        strcpy(inputBuffer,history[(command_count - 1) % MAX_COMMANDS]);

        length = strlen(inputBuffer) + 1;
    }

    else if (isdigit(inputBuffer[1])) { /* retrieve the nth command */

        command_number = atoi(&inputBuffer[1]);

        strcpy(inputBuffer,history[command_number]);

        length = strlen(inputBuffer) + 1;
    }
}

/**
 * Add the command to the history
 */

addtohistory(inputBuffer);

/**
 * Parse the contents of inputBuffer
 */

for (i=0;i<length;i++) {

    /* examine every character in the inputBuffer */

```

```

switch (inputBuffer[i]){

    case ' ':

    case '\t':          /* argument separators */

        if(start != -1){

args[ct] = &inputBuffer[start];  /* set up pointer */

            ct++;

        }

        inputBuffer[i] = '\0'; /* add a null char; make a C string */

        start = -1;

        break;


    case '\n':          /* should be the final char examined */

        if (start != -1){

            args[ct] = &inputBuffer[start];

            ct++;

        }

        inputBuffer[i] = '\0';

        args[ct] = NULL; /* no more arguments to this command */

        break;


    default :          /* some other character */

        if (start == -1)

            start = i;

        if (inputBuffer[i] == '&') {

            *background = 1;

            inputBuffer[i-1] = '\0';

        }

    } /* end of switch */

} /* end of for */

```



```
/**
```

```
 * If we get &, don't enter it in the args array
```

```
 */
```

```
if (*background)
```

```
    args[--ct] = NULL;
```

```
args[ct] = NULL; /* just in case the input line was > 80 */
```

```
return 1;
```

```
} /* end of setup routine */
```

```
int main(void)
```

```
{
```

```
    char inputBuffer[MAX_LINE];    /* buffer to hold the command entered */
```

```
    int background;                /* equals 1 if a command is followed by '&' */
```

```
    char *args[MAX_LINE/2 + 1];    /* command line (of 80) has max of 40 arguments */
```

```
    pid_t child;                   /* process id of the child process */
```

```
    int status;                    /* result from execvp system call */
```

```
    int shouldrun = 1;
```

```
    int i, upper;
```

```
    while (shouldrun){              /* Program terminates normally inside setup */
```

```
        background = 0;
```

```
        shouldrun = setup(inputBuffer,args,&background);    /* get next command */
```

```

if (strcmp(inputBuffer, "exit", 4) == 0)

    return 0;

else if (strcmp(inputBuffer,"history", 7) == 0) {

    if (command_count < MAX_COMMANDS)

        upper = command_count;

    else

        upper = MAX_COMMANDS;

    for (i = 0; i < upper; i++) {

        printf("%d \t %s\n", i, display_history[i]);

    }

    continue;

}

```

```

if (shouldrun) {

    child = fork();      /* creates a duplicate process! */

    switch (child) {

        case -1:

            perror("could not fork the process");

            break;

        case 0: /* this is the child process */

            status = execvp(args[0],args);

            if (status != 0){

                perror("error in execvp");

                exit(-2); /* terminate this process with error code -2 */

            }

            break;

    }

}

```

```

        default : /* this is the parent */

        if (background == 0) /* handle parent,wait for child */

            while (child != wait(NULL))

                ;

    }

}

}

return 0;

}

```

Procedure & Results:

```

dikshya@dikshya-VirtualBox: ~/Desktop/Lab 2 OS/Q6
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS/Q6$ cat prog.c
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>

#define MAX_LINE      80 /* 80 chars per line, per command, should be enough. */
#define MAX_COMMANDS  5 /* size of history */

char history[MAX_COMMANDS][MAX_LINE];
char display_history [MAX_COMMANDS][MAX_LINE];

int command_count = 0;

/**
 * Add the most recent command to the history.
 */
void addtohistory(char inputBuffer[]) {
    int i = 0;

    // add the command to history
    strcpy(history[command_count % MAX_COMMANDS], inputBuffer);

    // add the display-style command to history
    while (inputBuffer[i] != '\n' && inputBuffer[i] != '\0') {
        display_history[command_count % MAX_COMMANDS][i] = inputBuffer[i];
        i++;
    }
    display_history[command_count % MAX_COMMANDS][i] = '\0';
}

```

```

dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS/Q6$ cat prog.c &
[1] 43146
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS/Q6$ #include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <string.h>

#define MAX_LINE          80 /* 80 chars per line, per command, should be enough. */
#define MAX_COMMANDS      5 /* size of history */

char history[MAX_COMMANDS][MAX_LINE];
char display_history [MAX_COMMANDS][MAX_LINE];

int command_count = 0;

/**
 * Add the most recent command to the history.
 */

void addtohistory(char inputBuffer[]) {
    int i = 0;

    // add the command to history
    strcpy(history[command_count % MAX_COMMANDS], inputBuffer);

    // add the display-style command to history
    while (inputBuffer[i] != '\n' && inputBuffer[i] != '\0') {
        display_history[command_count % MAX_COMMANDS][i] = inputBuffer[i];
    }
}

```

```

dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS/Q6$ ./prog
os>ps -ael

```

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S	0	1	0	0	80	0	-	41455	-	?	00:00:14	systemd
1	S	0	2	0	0	80	0	-	0	-	?	00:00:00	kthreadd
1	I	0	3	2	0	60	-20	-	0	-	?	00:00:00	rcu_gp
1	I	0	4	2	0	60	-20	-	0	-	?	00:00:00	rcu_par_gp
1	I	0	6	2	0	60	-20	-	0	-	?	00:00:00	kworker/0:0H-events_highpri
1	I	0	9	2	0	60	-20	-	0	-	?	00:00:00	mm_percpu_wq
1	S	0	10	2	0	80	0	-	0	-	?	00:00:00	rcu_tasks_rude
1	S	0	11	2	0	80	0	-	0	-	?	00:00:00	rcu_tasks_trace
1	S	0	12	2	0	80	0	-	0	-	?	00:00:00	ksoftirqd/0
1	I	0	13	2	0	80	0	-	0	-	?	00:00:35	rcu_sched
1	S	0	14	2	0	-40	-	-	0	-	?	00:00:02	migration/0
1	S	0	15	2	0	9	-	-	0	-	?	00:00:00	idle_inject/0
1	S	0	16	2	0	80	0	-	0	-	?	00:00:00	cpuhp/0
1	S	0	17	2	0	80	0	-	0	-	?	00:00:00	cpuhp/1
1	S	0	18	2	0	9	-	-	0	-	?	00:00:00	idle_inject/1
1	S	0	19	2	0	-40	-	-	0	-	?	00:00:02	migration/1
1	S	0	20	2	0	80	0	-	0	-	?	00:00:01	ksoftirqd/1
1	I	0	22	2	0	60	-20	-	0	-	?	00:00:00	kworker/1:0H-events_highpri
5	S	0	23	2	0	80	0	-	0	-	?	00:00:00	kdevtmpfs
1	I	0	24	2	0	60	-20	-	0	-	?	00:00:00	netns
1	I	0	25	2	0	60	-20	-	0	-	?	00:00:00	inet_frag_wq
1	S	0	26	2	0	80	0	-	0	-	?	00:00:00	kauditd
1	S	0	27	2	0	80	0	-	0	-	?	00:00:00	khungtaskd
1	S	0	28	2	0	80	0	-	0	-	?	00:00:00	oom_reaper
1	I	0	29	2	0	60	-20	-	0	-	?	00:00:00	writeback
1	S	0	30	2	0	80	0	-	0	-	?	00:00:12	kcompactd0
1	S	0	31	2	0	85	5	-	0	-	?	00:00:00	ksmd

```
dikshya@dikshya-VirtualBox: ~/Desktop/Lab 2 OS/Q6
1 S 0 17011 2 0 80 0 - 0 - ? 00:00:00 jfsCommit
1 S 0 17012 2 0 80 0 - 0 - ? 00:00:00 jfsSync
0 S 1000 19963 984 0 80 0 - 99375 do_pol ? 00:00:00 gvfsd-network
0 S 1000 19975 984 0 80 0 - 81380 do_pol ? 00:00:00 gvfsd-dnssd
0 S 1000 41232 953 0 80 0 - 10844 do_wai ? 00:00:00 gnome-terminal
0 S 1000 41233 41232 0 80 0 - 97483 do_pol ? 00:00:00 gnome-terminal.
0 R 1000 41238 953 0 80 0 - 142627 - ? 00:00:31 gnome-terminal-
0 S 1000 41256 41238 0 80 0 - 5022 do_sel pts/0 00:00:00 bash
0 S 1000 41391 953 0 80 0 - 10844 do_wai ? 00:00:00 gnome-terminal
0 S 1000 41392 41391 0 80 0 - 97472 do_pol ? 00:00:00 gnome-terminal.
0 S 1000 41397 41238 0 80 0 - 5022 do_sel pts/1 00:00:00 bash
0 S 1000 41764 953 0 80 0 - 237922 do_pol ? 00:00:14 nautilus
1 I 0 42271 2 0 80 0 - 0 - ? 00:00:02 kworker/0:2-events
1 I 0 42537 2 0 80 0 - 0 - ? 00:00:04 kworker/1:2-cgroup_destroy
1 S 0 42872 2 0 60 -20 - 0 - ? 00:00:00 loop12
1 I 0 42898 2 0 80 0 - 0 - ? 00:00:00 kworker/u4:2-events_unbound
0 S 1000 42951 1069 0 80 0 - 700494 do_pol ? 00:00:01 gjs
0 S 1000 42970 42951 0 80 0 - 10844 do_wai ? 00:00:00 gnome-terminal
0 S 1000 42971 42970 0 80 0 - 97487 do_pol ? 00:00:00 gnome-terminal.
0 S 1000 42976 41238 0 80 0 - 5022 do_sel pts/2 00:00:00 bash
1 I 0 43031 2 0 80 0 - 0 - ? 00:00:00 kworker/1:3-cgroup_destroy
1 I 0 43115 2 0 80 0 - 0 - ? 00:00:00 kworker/u4:0-events_unbound
1 I 0 43131 2 0 80 0 - 0 - ? 00:00:00 kworker/0:1-events
0 S 1000 43134 41238 0 80 0 - 5022 do_sel pts/4 00:00:00 bash
1 I 0 43143 2 0 80 0 - 0 - ? 00:00:00 kworker/1:0-mm_percpu_wq
0 S 1000 43150 41238 0 80 0 - 5022 do_wai pts/3 00:00:00 bash
0 S 1000 43163 43150 0 80 0 - 688 do_wai pts/3 00:00:00 prog
0 R 1000 43168 43163 0 80 0 - 5376 - pts/3 00:00:00 ps
os>exit
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS/Q6$
```

```
dikshya@dikshya-VirtualBox:~/Desktop/Lab 2 OS/Q6$ ./prog
os>echo "Dikshya"
"Dikshya"
os>echo "Dikshya" &
os>"Dikshya"
```

Results: Notice how “OS>” may be printed by the parent process before the child return if “&” is not added to the command (child runs still runs in the background). However if it is added the parent won’t as for the command until the child process terminates.