

Processes in UNIX

LAB # 03



Fall 2023

CSE-302L Systems Programming Lab

Submitted by: **Ali Asghar**

Registration No.: **21PWCSE2059**

Class Section: **C**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Submitted to:

Engr. Abdullah Hamid

Date:

1st February 2024

Department of Computer Systems Engineering
University of Engineering and Technology, Peshawar

3.1. UNIX Process Creation and fork

Task 1:

Create process chain as shown in figure 3.1(b) and fill the figure 3.1 (b) with actual IDs. The program shall take a single command-line argument that specifies the number of processes to be created. Before exiting, each process shall output its i value (loop variable), its process ID (using getpid()), its parent process ID (getppid()) and the process ID of its child (return value of fork). The parent does not execute wait.

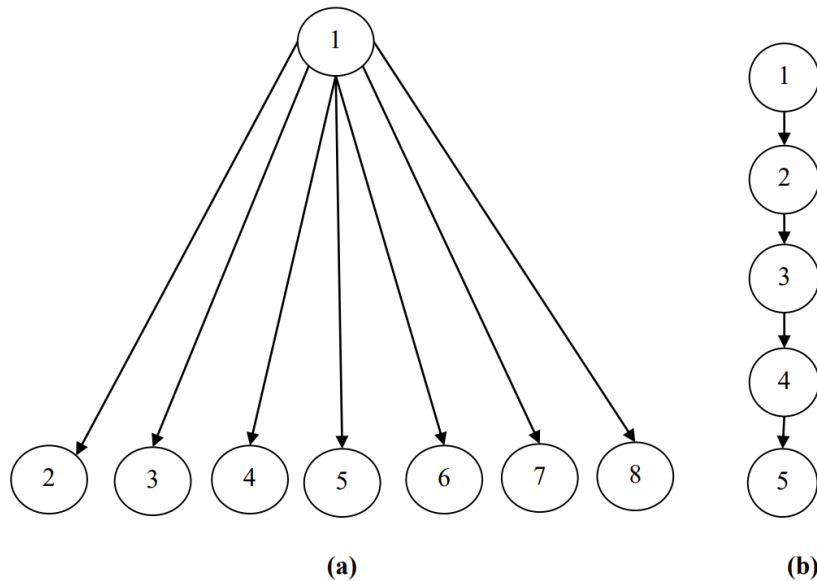


Figure 3.1 Multiple Processes (a) Process Fan (b) Process Chain

Code:

```
task1.c
~/Desktop/SP Lab/Lab 3

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
8     int pid,r,status;
9     int i;
10    for(i = 1; i <= atoi(argv[1]); i++){
11        pid = fork();
12        if(pid > 0){
13            printf("Process %d with PID = %d and PPID = %d and Child ID = %d \n", i , getpid(), getppid(),pid );
14            break;
15        }
16
17        if(pid == 0){
18            //printf("Child Process %d with PID = %d and PPID = %d \n",i,getpid(),getppid());
19        }
20    }
21
22    if(pid > 0){
23        r = wait(&status);
24    }
25
26 }
27
28 return 0;
29
30 }
```

Output:

```
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ gcc task1.c -o task1.o
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ ./task1.o 4
Process 1 with PID = 34867 and PPID = 34691 and Child ID = 34868
Process 2 with PID = 34868 and PPID = 34867 and Child ID = 34869
Process 3 with PID = 34869 and PPID = 34868 and Child ID = 34870
Process 4 with PID = 34870 and PPID = 34869 and Child ID = 34871
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$
```



Task 2:

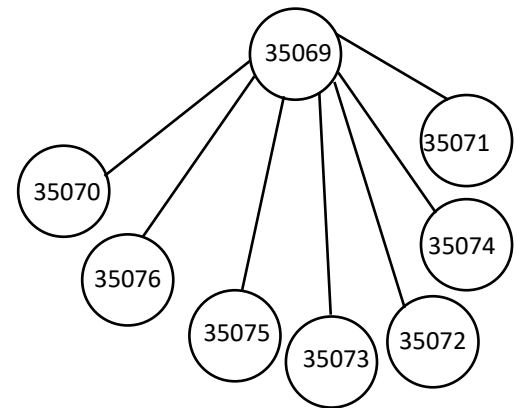
Create process fan as shown in figure 3.1 (a) and fill the figure 3.1 (a) with actual IDs.

Code:

```
task1.c
1 #include<stdio.h>
2 #include<unistd.h>
3 #include<sys/wait.h>
4 #include<stdlib.h>
5
6
7 int main(int argc, char* argv[]){
8     int pid;
9     int r;
10    printf("Parent with PID = %d\n",getpid());
11    for(int i=1; i<=atoi(argv[1]); i++){
12        pid = fork();
13        if(pid == 0){
14            printf("In Child Process %d with PID = %d and PPID = %d\n", i, getpid(), getppid());
15            break;
16        }
17    }
18
19    if(pid > 0){
20        for(int j=0; j<10; j++){
21            r = wait(NULL);
22        }
23        printf("Parent with PID = %d\n", getpid());
24    }
25    return 0;
26 }
27 }
```

Output:

```
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ ./task2.o 7
Parent with PID = 35069
In Child Process 1 with PID = 35070 and PPID = 35069
In Child Process 7 with PID = 35076 and PPID = 35069
In Child Process 6 with PID = 35075 and PPID = 35069
In Child Process 4 with PID = 35073 and PPID = 35069
In Child Process 3 with PID = 35072 and PPID = 35069
In Child Process 5 with PID = 35074 and PPID = 35069
In Child Process 2 with PID = 35071 and PPID = 35069
Parent with PID = 35069
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$
```



Task 3:

Create process tree as shown in figure 3.2 and fill figure 3.2 with actual IDs.

Code:

```
1 #include<stdio.h>
2 #include<unistd.h>
3 #include<sys/wait.h>
4 #include<stdlib.h>
5
6 int main(int argc, char* argv[]){
7     int pid;
8     int r;
9     printf("Parent with PID = %d\n",getpid());
10    for(int i=1; i<=atoi(argv[1]); i++){
11        pid = fork();
12        if(pid == 0){
13            printf("In Child Process %d with PID = %d and PPID = %d\n", i, getpid(), getppid());
14        }
15    }
16
17    if(pid > 0){
18        r = wait(NULL);
19    }
20
21    return 0;
22 }
```

Output:

```
ali@Ubuntu: ~/Desktop/SP Lab/Lab 3
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ ./task3.o 4
Parent with PID = 36317
In Child Process 1 with PID = 36318 and PPID = 36317
In Child Process 3 with PID = 36320 and PPID = 36317
In Child Process 2 with PID = 36319 and PPID = 36317
In Child Process 4 with PID = 36321 and PPID = 36317
In Child Process 4 with PID = 36325 and PPID = 36318
In Child Process 4 with PID = 36324 and PPID = 36320
In Child Process 4 with PID = 36327 and PPID = 36319
In Child Process 3 with PID = 36326 and PPID = 1445
In Child Process 4 with PID = 36328 and PPID = 36326
In Child Process 3 with PID = 36323 and PPID = 36318
In Child Process 2 with PID = 36322 and PPID = 36318
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ In Child Process 4 with PID = 36329 and PPID = 36323
In Child Process 4 with PID = 36331 and PPID = 36322
In Child Process 3 with PID = 36330 and PPID = 1445
In Child Process 4 with PID = 36332 and PPID = 36330
```

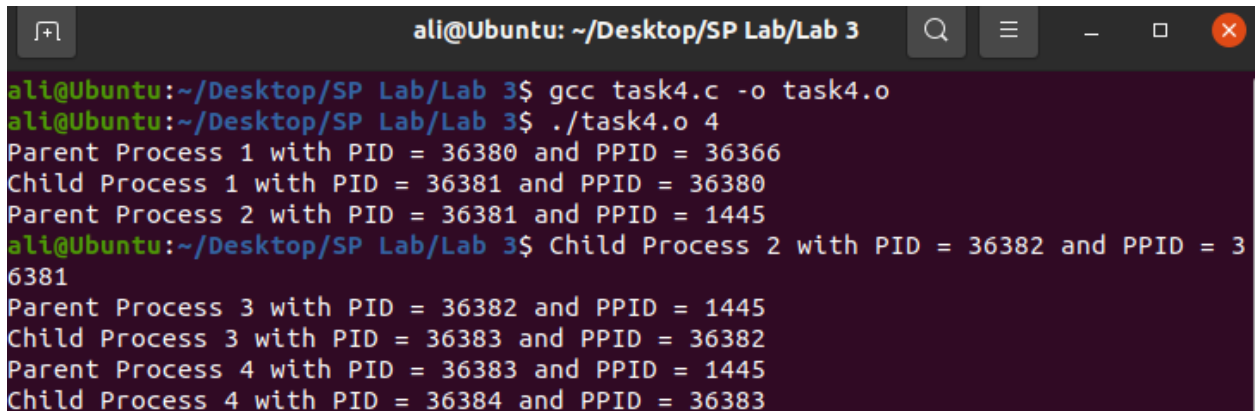
Task 4:

Creates a chain of processes. It takes a single command-line argument that specifies the number of processes to create. Before exiting, each process outputs its *i* value, its process ID, its parent process ID, and the process ID of its child. The parent does not execute wait. If the parent exits before the child, the child becomes an orphan. In this case, the child process is adopted by a special system process (which traditionally is a process, init, with process ID of 1). As a result, some of the processes may indicate a parent process ID of 1.

Code:

```
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
8     int pid,r,status;
9     int i;
10    for(i = 1; i <= atoi(argv[1]); i++){
11        pid = fork();
12        if(pid > 0){
13            printf("Parent Process %d with PID = %d and PPID = %d \n", i , getpid(), getppid());
14            break;
15        }
16
17        if(pid == 0){
18            printf("Child Process %d with PID = %d and PPID = %d \n",i,getpid(),getppid());
19        }
20    }
21
22    return 0;
23 }
24 }
```

Output:



```
ali@Ubuntu: ~/Desktop/SP Lab/Lab 3
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ gcc task4.c -o task4.o
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ ./task4.o 4
Parent Process 1 with PID = 36380 and PPID = 36366
Child Process 1 with PID = 36381 and PPID = 36380
Parent Process 2 with PID = 36381 and PPID = 1445
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ Child Process 2 with PID = 36382 and PPID = 36381
Parent Process 3 with PID = 36382 and PPID = 1445
Child Process 3 with PID = 36383 and PPID = 36382
Parent Process 4 with PID = 36383 and PPID = 1445
Child Process 4 with PID = 36384 and PPID = 36383
```

Task 5:

Write a program that takes N number of integers as argument and displays the factorials of N integers (print 1 only if integers are not less than zero, 0 or 1). Create separate child process for each integer. Make sure no child is orphan/zombie

Code:

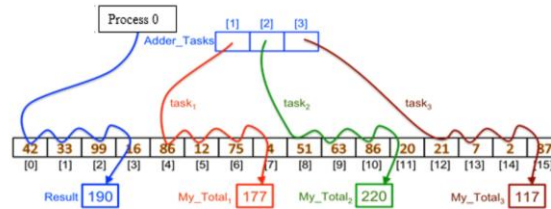
```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<unistd.h>
4 #include<sys/wait.h>
5
6 int main(int argc, char* argv){
7
8     int pid,r,status;
9     int i;
10    for(i = 1; i <= atoi(argv[1]); i++){
11        pid = fork();
12        if(pid > 0){
13            printf("Parent Process %d with PID = %d and PPID = %d \n", i , getpid(), getppid());
14            break;
15        }
16
17        if(pid == 0){
18            printf("Child Process %d with PID = %d and PPID = %d \n",i,getpid(),getppid());
19        }
20    }
21
22    return 0;
23 }
24 }
```

Output:

```
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ gcc task5.c -o task5.o
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ ./task5.o 5
Parent ID = 36395
I am child 2 and my PID is 36397
I am child 1 and my PID is 36396
Child 1 Terminated with factorial = 24
Parent Succesfully Terminated
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$
```

Task 6:

Write a program that creates an array of size 100. Initialize the array with random numbers. Create 10 child processes divide the array between them. Each child will add the portion and return their sum to parent process. Parent will add the results and display a final sum



Code:

```
1 #include<stdio.h>
2 #include<sys/wait.h>
3 #include<unistd.h>
4
5 int main(int argc, char *argv[]){
6     int arr[20] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
7     int status;
8     int pid;
9     int res;
10    int sum;
11    printf("In Parent Process\n");
12
13    for(int i = 0; i<10; i++){
14        pid = fork();
15        if(pid == 0){
16            res = arr[i*2] + arr[2*i+1];
17            printf("In child Process %i\n",i);
18            return res;
19        }
20    }
21
22    if(pid > 0){
23        for(int i = 0; i<10; i++){
24            int r = wait(&status);
25
26            //if(WIFEXITED(status) && !WEXITSTATUS(status))
27            //    printf("Child %d terminated normally\n",i);
28            if(WIFEXITED(status))
29                printf("Child %d terminated with return status %d\n", i, WEXITSTATUS(status));
30            sum += WEXITSTATUS(status);
31        }
32    }
33
34    for(int k = 0; k<10; k++){
35        printf("Sum = %d, ", sum);
36    }
37
38    return 0;
39 }
```

Output:

```
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$ ./task6.o
In Parent Process
In child Process 0
In child Process 3
In child Process 1
Child 0 terminated with return status 3
In child Process 2
Child 1 terminated with return status 11
Child 2 terminated with return status 15
In child Process 8
In child Process 9
In child Process 7
In child Process 4
In child Process 6
In child Process 5
Child 3 terminated with return status 35
Child 4 terminated with return status 7
Child 5 terminated with return status 19
Child 6 terminated with return status 39
Child 7 terminated with return status 27
Child 8 terminated with return status 31
Child 9 terminated with return status 23
ali@Ubuntu:~/Desktop/SP Lab/Lab 3$
```