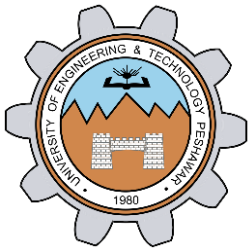
Processes in UNIX

LAB # 03



Fall 2020

CSE302L System Programming Lab

Submitted by: **Shah Raza**

Registration No. : **18PWCSE1658**

Class Section: **B**

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

Student Signature: _____

Submitted to:

Engr. Madiha Sher

Wednesday, December 16th, 2020

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

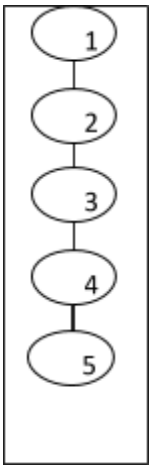
Lab Objective(s):

Understand and use the wait() function.

Understand the difference between Process Chain, Fan and Tree.

Task # 01:

Create a 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).

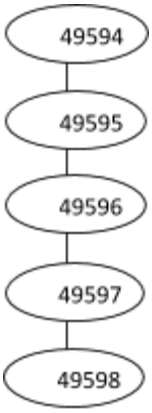


Code:

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

int main(int argc, char *argv[])
{
  int x;
  for (int i=0;i<atoi(argv[1]);i++)
  {
    x= fork();

    if(x>0)
    {
      printf("Pid: %d,  Parent id: %d,  Child id: %d, i: %d\n",getpid(),getppid(),x,i);
      break;
    }
  }
  for(int i=0;i<atoi(argv[1]);i++)
    wait(NULL);
  sleep(15);

}
```

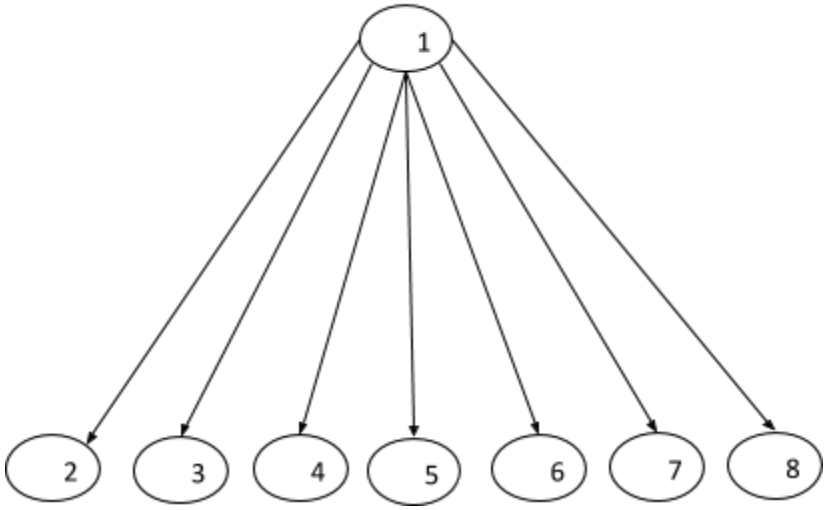
Output:

```
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 1/Process Chain$ ./ProcessChain 4 &
[2] 49594
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 1/Process Chain$ Pid: 49594,  Parent id: 49577,  Child id: 49595, i: 0
Pid: 49595,  Parent id: 49594,  Child id: 49596, i: 1
Pid: 49596,  Parent id: 49595,  Child id: 49597, i: 2
Pid: 49597,  Parent id: 49596,  Child id: 49598, i: 3

ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 1/Process Chain$ pstree -a 49594
ProcessChain 4
  └─ProcessChain 4
      └─ProcessChain 4
          └─ProcessChain 4
              └─ProcessChain 4
```



Task # 02:

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



Code:

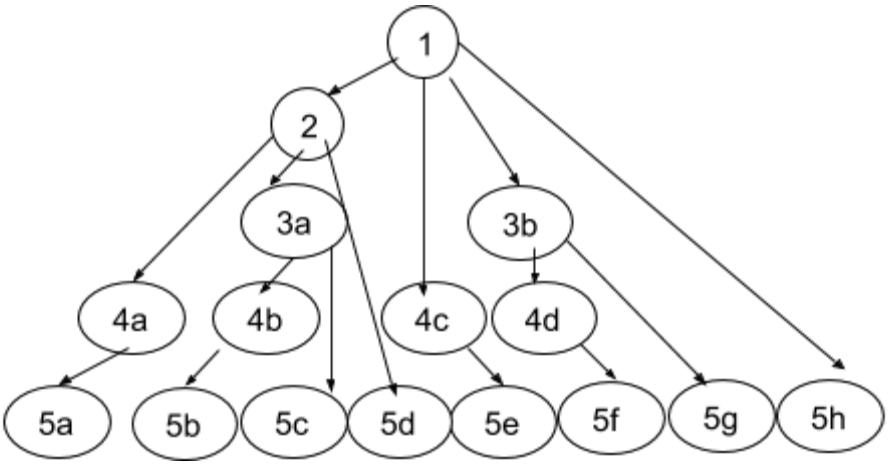
```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

int main(int argc, char *argv[])
{
  int x;
  for (int i=0;i<atoi(argv[1]);i++)
  {
    x= fork();
    if(x==0)
      break;
  }
  for(int i=0;i<atoi(argv[1]);i++)
    wait(NULL);
  printf("Pid: %d, Parent id: %d\n",getpid(),getppid());
  sleep(15);

}
```

Output:

```
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 2$ ./ProcessFan 7 &
[1] 49677
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 2$ Pid: 49684, Parent id: 49677
Pid: 49683, Parent id: 49677
Pid: 49682, Parent id: 49677
Pid: 49681, Parent id: 49677
Pid: 49680, Parent id: 49677
Pid: 49679, Parent id: 49677
Pid: 49678, Parent id: 49677

ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 2$ pstree -ap 49677
ProcessFan,49677 7
  ├─ProcessFan,49678 7
  ├─ProcessFan,49679 7
  ├─ProcessFan,49680 7
  ├─ProcessFan,49681 7
  ├─ProcessFan,49682 7
  ├─ProcessFan,49683 7
  └─ProcessFan,49684 7
```

Task 3:

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

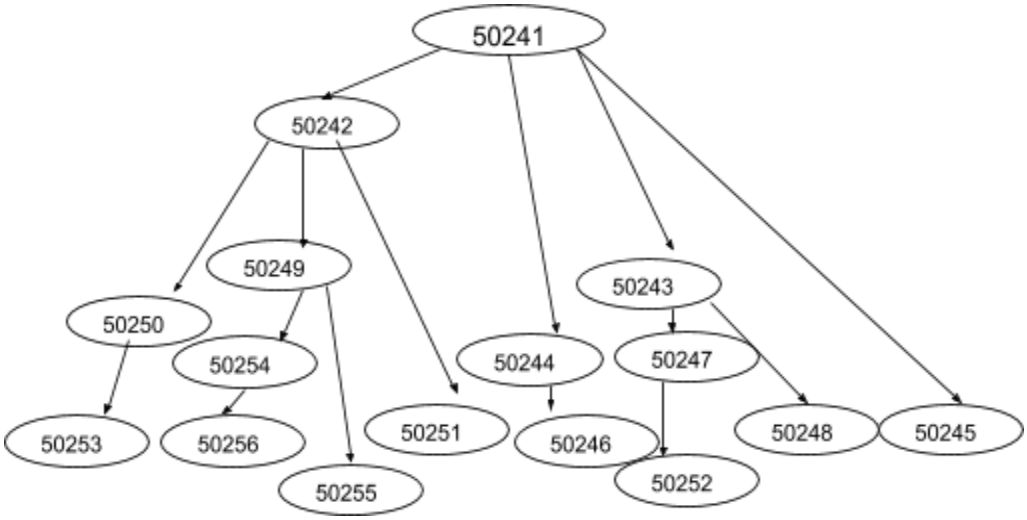


Code:

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

int main(int argc, char *argv[])
{
  int x;
  for (int i=0;i<atoi(argv[1]);i++)
  {
    x= fork();
  }
  for(int i=0;i<atoi(argv[1]);i++)
    wait(NULL);
  printf("Pid: %d, Parent id: %d\n",getpid(),getppid());
  sleep(15);
}
```

Output:

```
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 3$ ./ProcessTree 4 &
[1] 50241
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 3$ Pid: 50245, Parent id: 50241
Pid: 50246, Parent id: 50244
Pid: 50248, Parent id: 50243
Pid: 50251, Parent id: 50242
Pid: 50252, Parent id: 50247
Pid: 50255, Parent id: 50249
Pid: 50253, Parent id: 50250
Pid: 50256, Parent id: 50254

ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 3$ pstree -ap 50241
ProcessTree,50241 4
  ├─ProcessTree,50242 4
  │   ├─ProcessTree,50249 4
  │   │   ├─ProcessTree,50254 4
  │   │   │   └─ProcessTree,50256 4
  │   │   └─ProcessTree,50255 4
  │   ├─ProcessTree,50250 4
  │   │   └─ProcessTree,50253 4
  │   └─ProcessTree,50251 4
  ├─ProcessTree,50243 4
  │   ├─ProcessTree,50247 4
  │   │   └─ProcessTree,50252 4
  │   └─ProcessTree,50248 4
  ├─ProcessTree,50244 4
  │   └─ProcessTree,50246 4
  └─ProcessTree,50245 4
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 3$ Pid: 50244, Parent id: 50241
Pid: 50250, Parent id: 50242
Pid: 50254, Parent id: 50249
Pid: 50247, Parent id: 50243
Pid: 50243, Parent id: 50241
Pid: 50249, Parent id: 50242
Pid: 50242, Parent id: 50241
Pid: 50241, Parent id: 50203
```



Task 4:

This task expands on the process chain of Task 1. The chain is a vehicle for experimenting with `wait` and with sharing of devices. All of the processes in the chain created by Task 1 share standard input, standard output and standard error.

Task 3.1 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.

- **a.** Experiment with different values for the command-line argument to find out the largest number of processes that the program can generate. Observe the fraction that are adopted by `init`.
- **b.** Place `sleep(10);` directly before the final `printf` statement. What is the maximum number of processes generated in this case?

Code:

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  int x;
  for (int i=0;i<atoi(argv[1]);i++)
  {
    x= fork();

    if(x==-1)
    {
      perror("Error Occured");
      return 0;
    }

    if(x>0)
    {
      printf("\nPid: %d,  Parent id: %d,  Child id: %d, i: %d",getpid(),getppid(),x,i);
      break;
    }
  }
}
```

Discussion:

a. Without using the wait/sleep function almost all the processes become orphans and are adopted by the init process. And we can make a chain of as many processes as we want.
b. After putting the sleep function in our code, the maximum number of child processes generated reduces to 3318 on my system, after 3318 child processes are created the OS crashes if we try to fork another child.

Task 5:

Write a program that takes N number of integers as argument and displays the factors of N integers. Create a separate child process for each integer. Make sure no child is an orphan/zombie.

Code:

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

void Factors(int val)
{
  for(int i=1;i<val;i++)
  {
    if(val%i==0)
      printf(" %d ",i);
  }
  printf("\n");
}

int main(int argc, char *argv[])
{
  int x;
  for (int i=1;i<argc;i++)
  {
    x= fork();
    if(x==0)
    {
      printf("Pid: %d, Parent id: %d\nFactors of %d: ",getpid(),getppid(),atoi(argv[i]));
      Factors(atoi(argv[i]));
      break;
    }

  }
  for(int i=0;i<atoi(argv[1]);i++)
    wait(NULL);
}
```

Output:

```
ShahRaza@ubuntu:~/Systems Programming/labs/Lab 3/Task 5$ ./task5 6 34 12
Pid: 50335, Parent id: 50332
Factors of 12:  1  2  3  4  6
Pid: 50334, Parent id: 50332
Factors of 34:  1  2  17
Pid: 50333, Parent id: 50332
Factors of 6:  1  2  3
```

Task 6:

Write a program that creates an array of size 10,000. 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 the parent process. Parent will add the results and display a final sum.

Code:

Using Pipes:

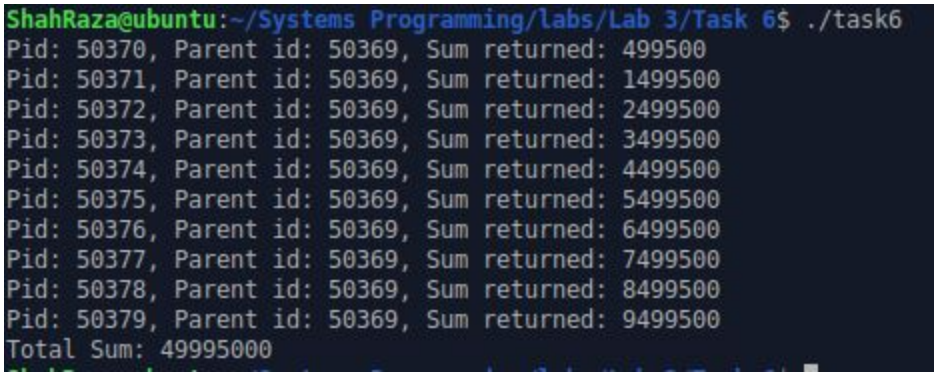
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

int main()
{
  int pipefds[2];
  pipe(pipefds);
  int Array[10000];
  for(int i=0;i<10000;i++)
  {
    Array[i]=i;
  }
  int x;
  int sum[10]={0},total=0;
  int a=0,b=1000;
  for (int i=0;i<10;i++)
  {
    x= fork();
    if(x==0)
    {
      for(int j=a;j<b;j++)
      {
        sum[i]+=Array[j];
      }
      printf("Pid: %d, Parent id: %d, Sum returned: %d\n",getpid(),getppid(),sum[i]);
      write(pipefds[1],&sum[i],sizeof(sum[i]));
      break;
    }
    else if(x>0)
    {
      sleep(1);
      a+=1000;
      b+=1000;
      read(pipefds[0],&sum[i],sizeof(sum[i]));
      total+=sum[i];
    }
  }
  if(x>0)
  {
    sleep(1);
    printf("Total Sum: %d\n",total);
  }
}
```

Using wait and WEXITSTATUS:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>


int main()
{
  int Array[50];
  for(int i=0;i<50;i++)
  {
    Array[i]=i;
  }
  int x;
  int sum[10]={0},total=0,stat;
  int a=0,b=5;
  for (int i=0;i<10;i++)
  {
    x= fork();
    if(x==0)
    {
      for(int j=a;j<b;j++)
      {
        sum[i]+=Array[j];
      }
      printf("Pid: %d, Parent id: %d, Sum returned: %d\n",getpid(),getppid(),sum[i]);
      return sum[i];
      break;
    }
    else if(x>0)
    {
      sleep(1);
      a+=5;
      b+=5;
      wait(&stat);
      total+=WEXITSTATUS(stat);
    }
  }
  if(x>0)
  {
    sleep(1);
    printf("Total Sum: %d\n",total);
  }
}
```

Output:

Pipes version:



WEXITSTATUS version: