# WEEK1

### 1. Write a program to implement single fork() system call. 1.

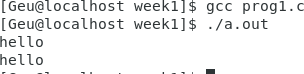
#include<stdio.h> #include<unistd.h> void main()

{

fork(); printf("hello\n");

}

## OUTPUT:



### 2. Write a program to implement multiple fork() system calls.

#include<stdio.h> #include<unistd.h> void main()

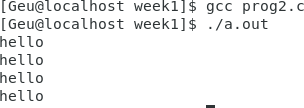
{

fork();

fork(); printf("hello\n");

}

## OUTPUT:



### 3.

#include<stdio.h> #include<unistd.h> void main()

{

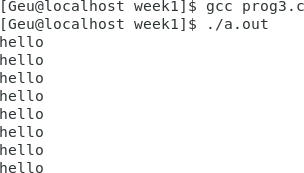
fork();

fork();

fork(); printf("hello\n");

}

## OUTPUT:



### 4.

#include<stdio.h> #include<unistd.h> void main()

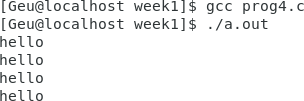
{

if(fork () && fork()) fork ();

printf("hello\n");

}

## OUTPUT:



### 5.

#include<stdio.h> #include<unistd.h> int main()

{

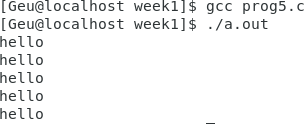
if(fork() || fork()) fork();

printf("hello\n");

return 0;

}

**OUTPUT:**



# WEEK2

## Write a program where parent process compute the sum of even and child process compute the sum of odd using fork ().

#include<unistd.h> #include<stdio.h> void main()

{

int m;

printf("Enter the size of array:"); scanf("%d",&m);

int a[m]; int j=0;

printf("\nEnter the numbers:"); while(j<m)

{

int b; scanf("%d",&b); a[j]=b;

j++;

}

int sume=0,sumo=0; int n=fork();

if(n>0)

{

for(int i=0;i<10;i++)

{

if(a[i]%2==0) sume=sume+a[i];

}

printf("\nsum of even:%d",sume);

}

else

{

for(int i=0;i<10;i++)

{

if(a[i]%2!=0) sumo=sumo+a[i];

}

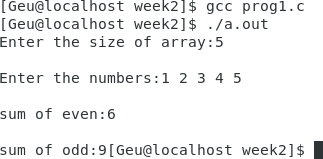
printf("\n");

printf("\nsum of odd:%d",sumo);

}

}

## OUTPUT:



1. **Write a program for check if number is even print parent process and if number is odd print child process.**

#include<unistd.h> #include<stdio.h> int main()

{

int n;

printf("Enter the number:"); scanf("%d",&n);

if(n%2==0)

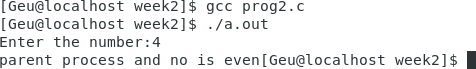
printf("parent process and no is even"); else

printf("child process and no is odd");

return 0;

}

**OUTPUT:**



# WEEK3

## Implementation of wait () system call

### 1.

#include<sys/wait.h> #include<unistd.h> #include<stdio.h>

int main()

{

int pid;

pid =fork(); if(pid==0)

{

printf("Child process\n");

}

else

{

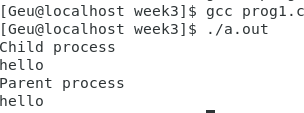
wait(NULL); printf("Parent process\n");

}

printf("hello\n"); return 0;

}

## OUTPUT:



### 2.

#include<sys/wait.h> #include<unistd.h> #include<stdio.h>

int main()

{

int pid;

pid =fork(); if(pid==0)

{

printf("Child process\n");

}

else

{

printf("Parent process\n"); wait(NULL);

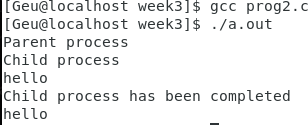
printf("Child process has been completed\n");

}

printf("hello\n"); return 0;

}

**OUTPUT:**



# WEEK4

### 1. Write a program to implement a Zombie process.

#include<unistd.h> #include<stdio.h> void main()

{

int pid=fork(); if(pid==0)

{

printf("child process");

}

else

{

sleep(15);

printf("parent process");

}

}

**OUTPUT:**



**2. Write a program to implement the orphan process.**

#include<unistd.h> #include<stdio.h> void main()

{

int pid=fork(); if(pid==0)

{

sleep(15);

printf("child process\n");

}

else

{

printf("parent process");

}

}

## OUTPUT:



# 3. Write a program to demonstrate the use of wait system call for handling orphan process.

#include<unistd.h>

#include<stdio.h>

#include<sys/wait.h>

void main()

{

int pid=fork();

if(pid==0)

{

sleep(5);

printf("child process\n");

}

else

{

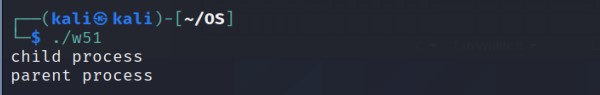
wait(NULL);

printf("parent process");

}

}

**OUTPUT**:-



**4.Write a program to demonstrate the use of wait system call for handling Zombie process**

#include<unistd.h>

#include<stdio.h>

#include<sys/wait.h>

void main()

{

int pid=fork();

if(pid==0)

{

printf("child process\n");

}

else

{

wait(NULL);

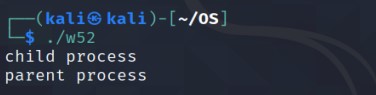
sleep(5);

printf("parent process\n");

}

}

**OUTPUT:-**



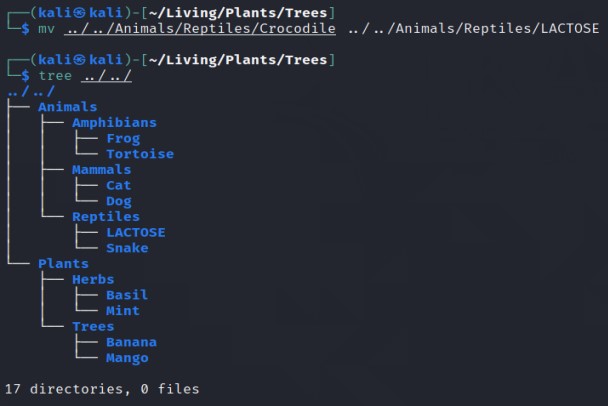
**WEEK5**

**1.Write a sequence of commands to create the following structure.**

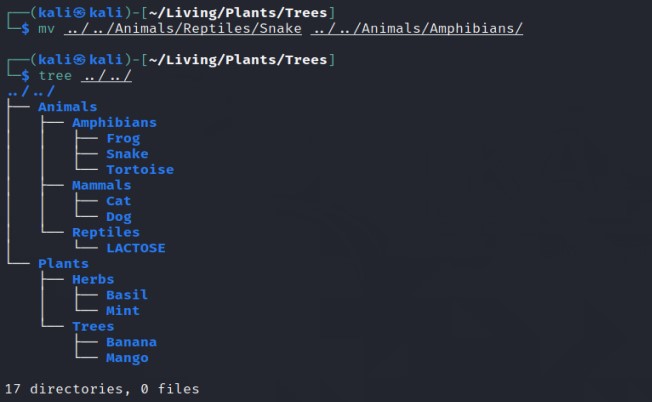
**a.** Display the complete structure in a single step (suppose you are in Trees Directory)

****

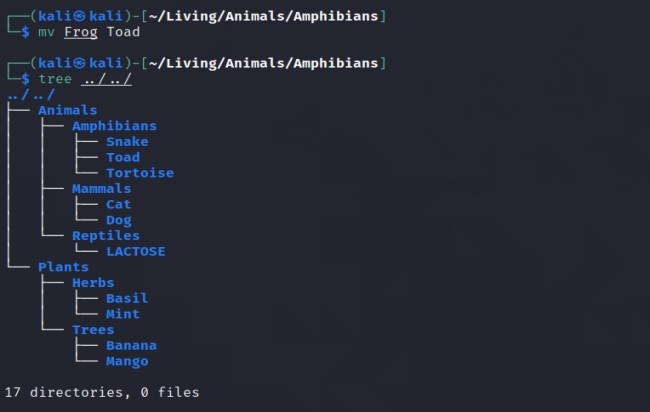
b. Rename the file crocodile by LACOSTE in a single step (suppose you are in Trees Directory)

****

c. Move the file SNAKE to folder Amphibians in a single step (suppose you are in Trees Directory)



d. Rename the file FROG to TOAD in a single step (suppose you are in Amphibians Directory)



e. Display the complete structure in a single step (suppose you are in Plants Directory)



**2. Write a Program to implement the FCFS CPU Scheduling algorithm.**

#include<stdio.h>

#include<math.h>

void main(){

int at[10], bt [10],wt[10],tat[10], temp[10],n,i;

float awt=0, atat=0;

printf("Enter the no. of processes\n");

scanf("%d",&n);

printf("Enter arrival time and burst time respectively\n");

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

scanf("%d %d",&at[i],&bt[i]);

}

temp[0]=0;

printf("Process\t Arrival Time\t Burst time\t Waiting time\t Turn around time \n");

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

tat[i]=0;

wt[i]=0;

temp[i+1]=temp[i]+bt[i];

wt[i]=abs(temp[i]-at[i]);

tat[i]=wt[i]+bt[i];

awt=awt+wt[i];

atat=atat+tat[i];

printf(" %d\t \t%d\t %d\t\t %d\t\t %d\n",i+1,at[i],bt[i],wt[i],tat[i]);

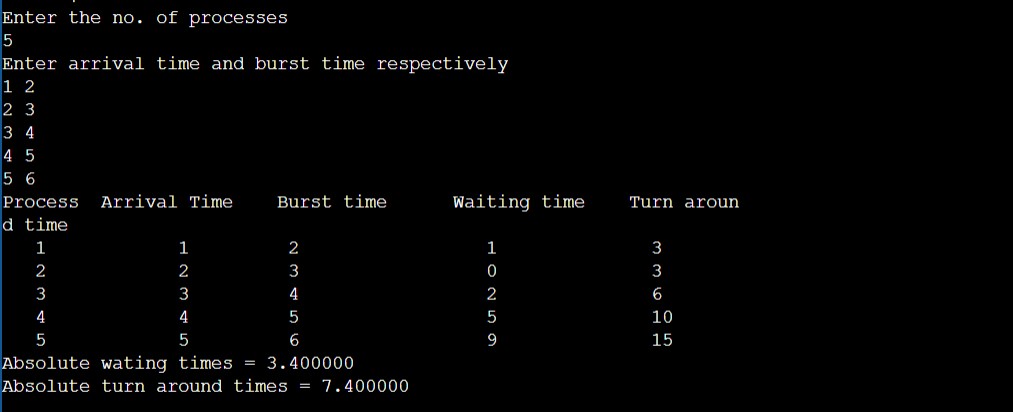
}

printf("Absolute wating times = %f\n",awt/n);

printf("Absolute turn around times = %f",atat/n);

}

**OUTPUT:-**



**3.Write a Program to implement the SJF (Non- Preemptive) CPU Scheduling algorithm.**

#include<iostream>

#include<stdio.h>

using namespace std;

int main()

{

int i,n,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,btime=0;

int burst\_time[10],temp,j,arrival\_time[10],wait\_time[10],turn\_around\_time[10],ta=0,sum=0;

float total\_tat\_time=0,total\_wait\_time=0;

cout<<" -------Shortest Job First Scheduling ( NP )-------\n";

cout<<"\nEnter the No. of processes :";

cin>>n;

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

{

cout<<"\nEnter the burst time of"<<i+1<<"process :";

cin>>burst\_time[i];

cout<<"\nEnter the arrival time of"<<i+1<<"process :";

cin>>arrival\_time[i];

}

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

{

for(j=0;j<n;j++)

{

if(arrival\_time[i]<arrival\_time[j])

{

temp=p[j];

p[j]=p[i];

p[i]=temp;

temp=arrival\_time[j];

arrival\_time[j]=arrival\_time[i];

arrival\_time[i]=temp;

temp=burst\_time[j];

burst\_time[j]=burst\_time[i];

burst\_time[i]=temp;

}

}

}

for(j=0;j<n;j++)

{

btime=btime+burst\_time[j];

min=burst\_time[k];

for(i=k;i<n;i++)

{

if (btime>=arrival\_time[i] && burst\_time[i]<min)

{

temp=p[k];

p[k]=p[i];

p[i]=temp;

temp=arrival\_time[k];

arrival\_time[k]=arrival\_time[i];

arrival\_time[i]=temp;

temp=burst\_time[k];

burst\_time[k]=burst\_time[i];

burst\_time[i]=temp;

}

}

k++;

}

wait\_time[0]=0;

for(i=1;i<n;i++)

{

sum=sum+burst\_time[i-1];

wait\_time[i]=sum-arrival\_time[i];

total\_wait\_time=total\_wait\_time+wait\_time[i];

}

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

{

ta=ta+burst\_time[i];

turn\_around\_time[i]=ta-arrival\_time[i];

total\_tat\_time=total\_tat\_time+turn\_around\_time[i];

}

cout<<"\tPROCESS\t ARRIVAL TIME\t BURST TIME\t WAITING TIME\t TURNAROUND TIME\n";

for(i=0;i<n;i++){ //displaying the burst,wait & turn around time

cout<<"\n\tP"<<p[i]<<"\t\t"<<arrival\_time[i]<<"\t\t"<<burst\_time[i]<<"\t\t"<<wait\_time[i]<<"\t\t"<<turn\_around\_time[i];

}

cout<<"\nAverage Waiting Time --"<<total\_wait\_time/n;

cout<<"\nAverage Turnaround Time --"<<total\_tat\_time/n;

return 0;

}

**OUTPUT:-**

