**OPERATING**

**SYSTEM**

**LAB – 7**

**Process Creation**

**Submitted by: ~**

**Ayushi Kumari**

**RA1911003010865.**

1. **Implement the C program in which the child process calculates the sum of**

**odd numbers and the parent process calculate the sum of even numbers up**

**to the number ‘n’.**

**Objective:**

To write a C program in which the child process calculates the sum of odd

numbers and the parent process calculate the sum of even numbers up to the

number ‘n’.

**Procedure:**

Step 1: Input the value of n.

Step 2: Invoke the fork() system call.

Step 3: Check for the return value of fork() system call. If the value returned is

zero then it is child process. Calculate the sum of odd numbers. Display the

sum of odd numbers.

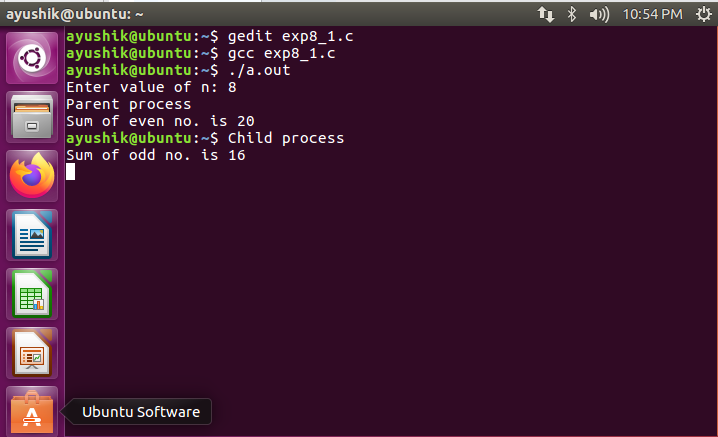
Step 4: If the fork() system call returns a non-zero positive integer, then it is

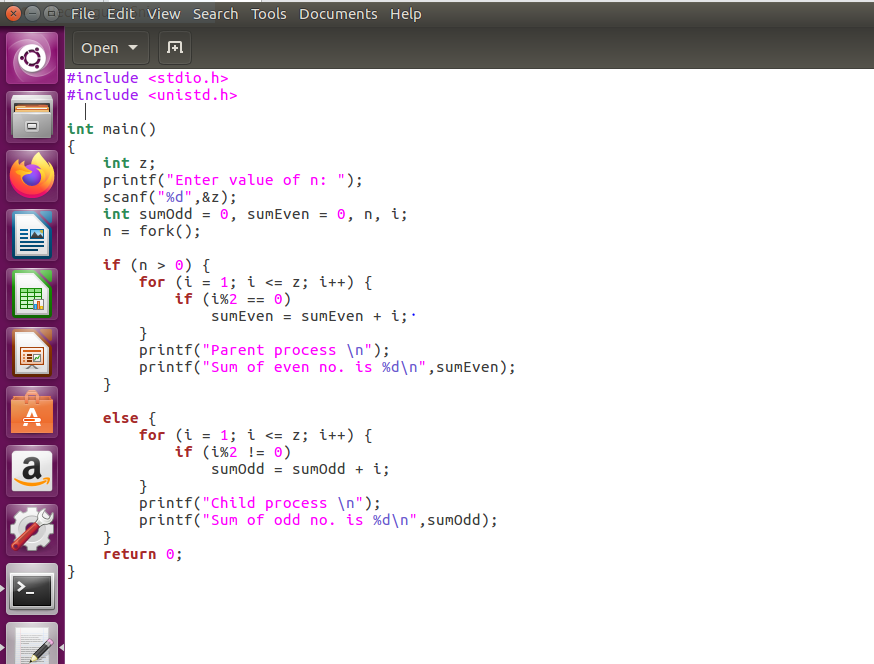
parent process. Calculate the sum of even numbers. Display the sum of even

numbers.

Step 5: stop.

**INPUT/OUTPUT: -**





**CODE: -**

#include <stdio.h>

#include <unistd.h>

int main()

{

int z;

printf("Enter value of n: ");

scanf("%d",&z);

int sumOdd = 0, sumEven = 0, n, i;

n = fork();

if (n > 0) {

for (i = 1; i <= z; i++) {

if (i%2 == 0)

sumEven = sumEven + i;

}

printf("Parent process \n");

printf("Sum of even no. is %d\n",sumEven);

}

else {

for (i = 1; i <= z; i++) {

if (i%2 != 0)

sumOdd = sumOdd + i;

}

printf("Child process \n");

printf("Sum of odd no. is %d\n",sumOdd);

}

return 0;

}

**Outcome:**

Learned and practiced how to use fork() to create child process.

1. **Implement the C program in which main program accepts the integers to**

**be sorted Main program uses the fork system call to create a new process**

**called a child process. Parent process sorts the integers using insertion sort**

**and waits for child process using wait system call to sort the integers using**

**selection sort.**

**Objective:**

To write a C program in which parent process sorts the given integers using

insertion sort and the child process sorts the integers using selection sort. The

parent process must wait until the child process terminates.

**Procedure:**

Step 1: Input the Integers to be sorted

Step 2: Display the Entered Integers

Step 3: Create Child Process

Step 4: Call selection sort method inside child process

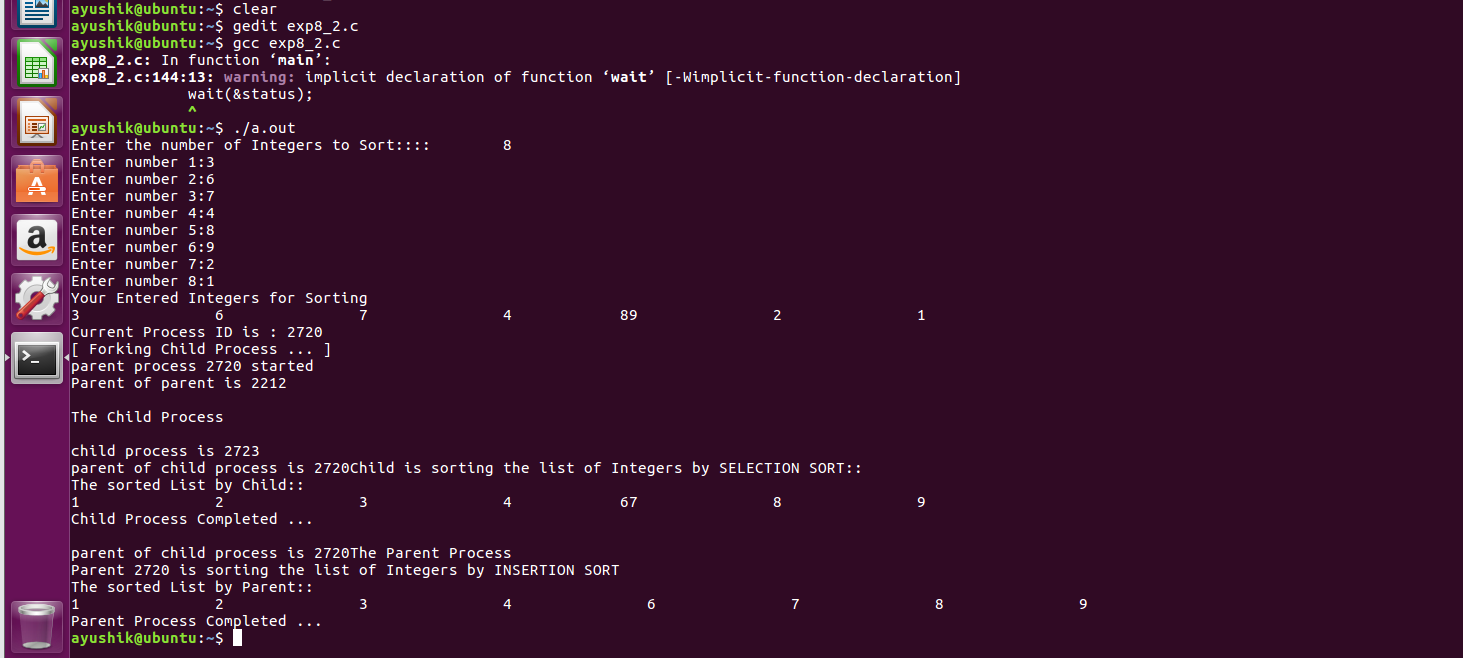
Step 5: Display the sorted Integers using selection sort.

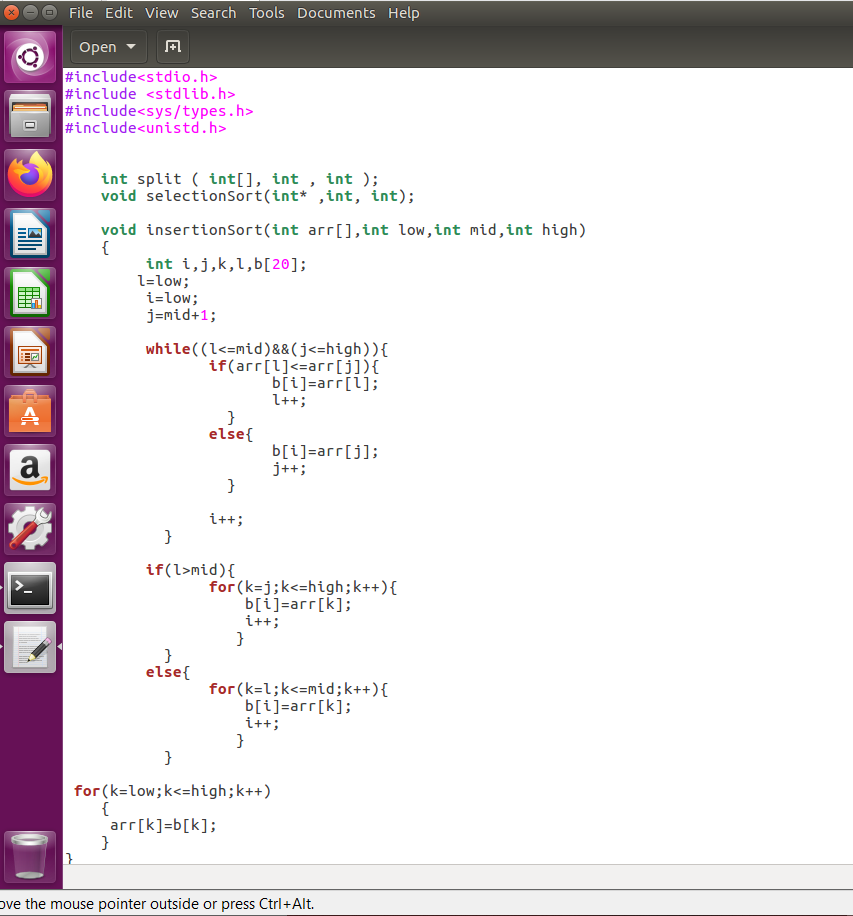
Step 6: Call insertion sort method inside Parent process

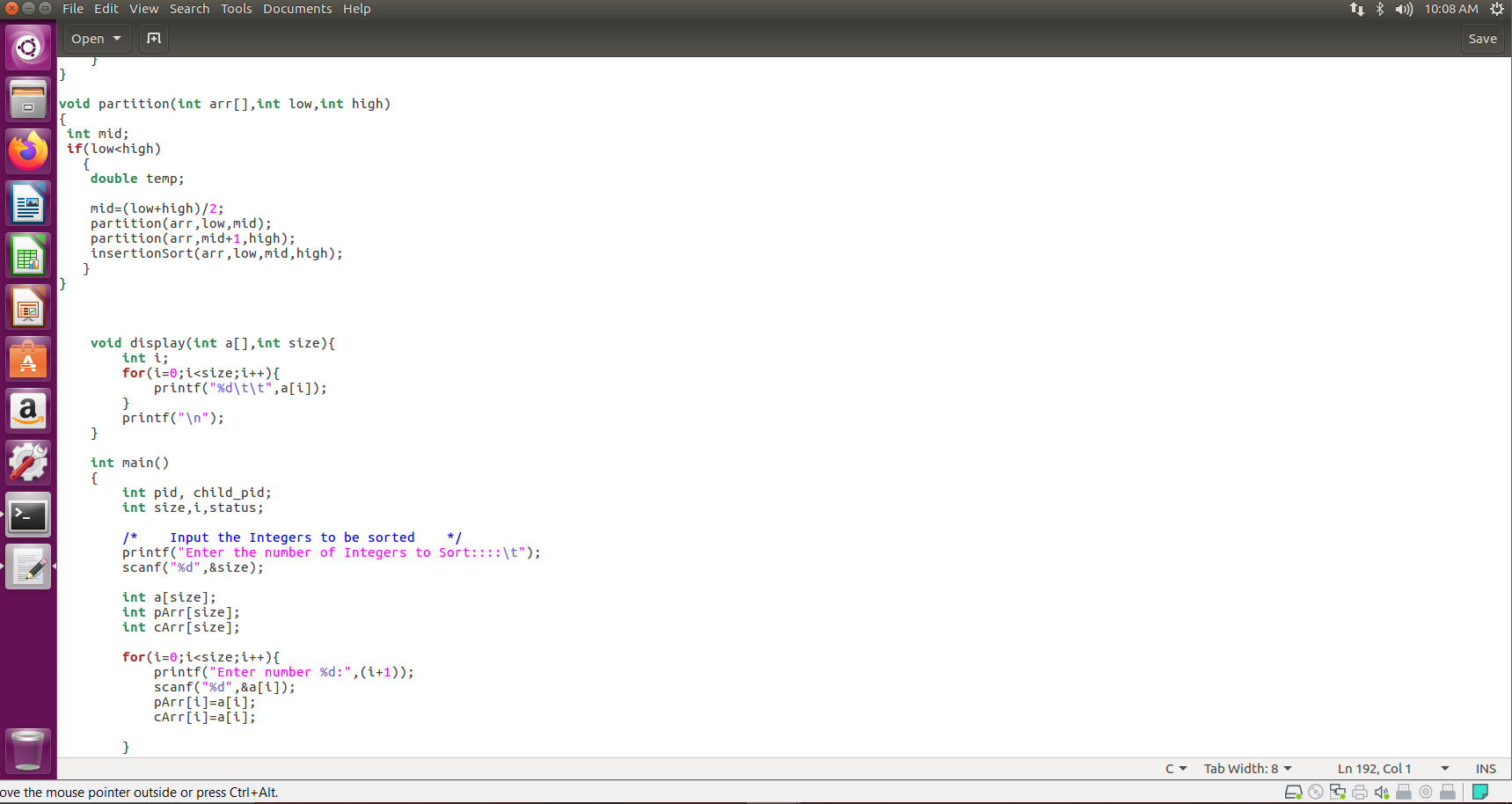
Step 7: Display the sorted Integers using insertion sort sort.

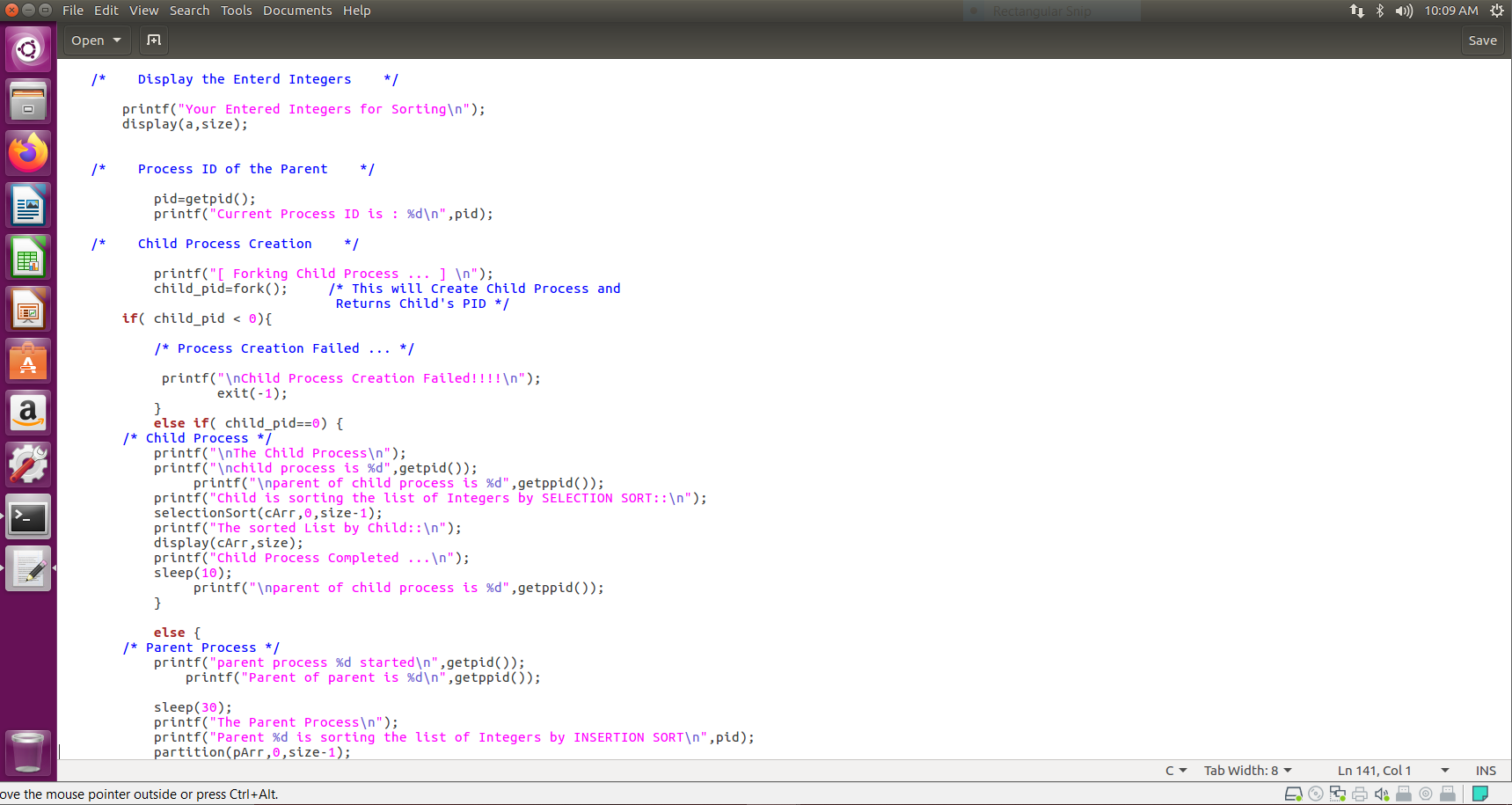
Step 8: stop.

**INPUT/OUTPUT: -**









**CODE: -**

#include<stdio.h>

#include <stdlib.h>

#include<sys/types.h>

#include<unistd.h>

int split ( int[], int , int );

void selectionSort(int\*,int, int);

void insertionSort(int arr[],int low,int mid,int high)

{

int i,j,k,l,b[20];

l=low;

i=low;

j=mid+1;

while((l<=mid)&&(j<=high)){

if(arr[l]<=arr[j]){

b[i]=arr[l];

l++;

}

else{

b[i]=arr[j];

j++;

}

i++;

}

if(l>mid){

for(k=j;k<=high;k++){

b[i]=arr[k];

i++;

}

}

else{

for(k=l;k<=mid;k++){

b[i]=arr[k];

i++;

}

}

for(k=low;k<=high;k++)

{

arr[k]=b[k];

}

}

void partition(int arr[],int low,int high)

{

int mid;

if(low<high)

{

double temp;

mid=(low+high)/2;

partition(arr,low,mid);

partition(arr,mid+1,high);

insertionSort(arr,low,mid,high);

}

}

void display(int a[],int size){

int i;

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

printf("%d\t\t",a[i]);

}

printf("\n");

}

int main()

{

int pid, child\_pid;

int size,i,status;

/\* Input the Integers to be sorted \*/

printf("Enter the number of Integers to Sort::::\t");

scanf("%d",&size);

int a[size];

int pArr[size];

int cArr[size];

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

printf("Enter number %d:",(i+1));

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

pArr[i]=a[i];

cArr[i]=a[i];

}

/\* Display the Enterd Integers \*/

printf("Your Entered Integers for Sorting\n");

display(a,size);

/\* Process ID of the Parent \*/

pid=getpid();

printf("Current Process ID is : %d\n",pid);

/\* Child Process Creation \*/

printf("[ Forking Child Process ... ] \n");

child\_pid=fork(); /\* This will Create Child Process and

Returns Child's PID \*/

if( child\_pid < 0){

/\* Process Creation Failed ... \*/

printf("\nChild Process Creation Failed!!!!\n");

exit(-1);

}

else if( child\_pid==0) {

/\* Child Process \*/

printf("\nThe Child Process\n");

printf("\nchild process is %d",getpid());

printf("\nparent of child process is %d",getppid());

printf("Child is sorting the list of Integers by SELECTION SORT::\n");

selectionSort(cArr,0,size-1);

printf("The sorted List by Child::\n");

display(cArr,size);

printf("Child Process Completed ...\n");

sleep(10);

printf("\nparent of child process is %d",getppid());

}

else {

/\* Parent Process \*/

printf("parent process %d started\n",getpid());

printf("Parent of parent is %d\n",getppid());

sleep(30);

printf("The Parent Process\n");

printf("Parent %d is sorting the list of Integers by INSERTION SORT\n",pid);

partition(pArr,0,size-1);

printf("The sorted List by Parent::\n");

display(pArr,size);

wait(&status);

printf("Parent Process Completed ...\n");

}

return 0;

}

int split ( int a[ ], int lower, int upper )

{

int i, p, q, t ;

p = lower + 1 ;

q = upper ;

i = a[lower] ;

while ( q >= p )

{

while ( a[p] < i )

p++ ;

while ( a[q] > i )

q-- ;

if ( q > p )

{

t = a[p] ;

a[p] = a[q] ;

a[q] = t ;

}

}

t = a[lower] ;

a[lower] = a[q] ;

a[q] = t ;

return q ;

}

void selectionSort(int a[],int lower, int upper){

int i ;

if ( upper > lower )

{

i = split ( a, lower, upper ) ;

selectionSort ( a, lower, i - 1 ) ;

selectionSort ( a, i + 1, upper ) ;

} }

**Outcome:**

Learned and practiced how to use fork() system call to create a child process and

make parent process to wait for child process.

1. **Implement the C program in which main program accepts an integer array.**

**Main program uses the fork system call to create a new process called a**

**child process. Parent process sorts an integer array and passes the sorted**

**array to child process through the command line arguments of execve**

**system call. The child process uses execve system call to load new program**

**that uses this sorted array for performing the binary search to search the**

**item in the array.**

**Objective:**

To write a C program in which parent process sorts the given integers and passes

the sorted integers to the child process through the command line arguments of

execve system call. The child process uses execve system call to load new

program that uses this sorted array for performing the binary search.

**Procedure:**

Step 1: Input the Integers to be sorted

Step 2: Display the Entered Integers

Step 3: Sort the integers.

Step 4: Display the sorted Integers.

Step 5: Input the integer to be searched.

Step 6: Create Child Process

Step 7: Pass the sorted integers to the child process through the command line

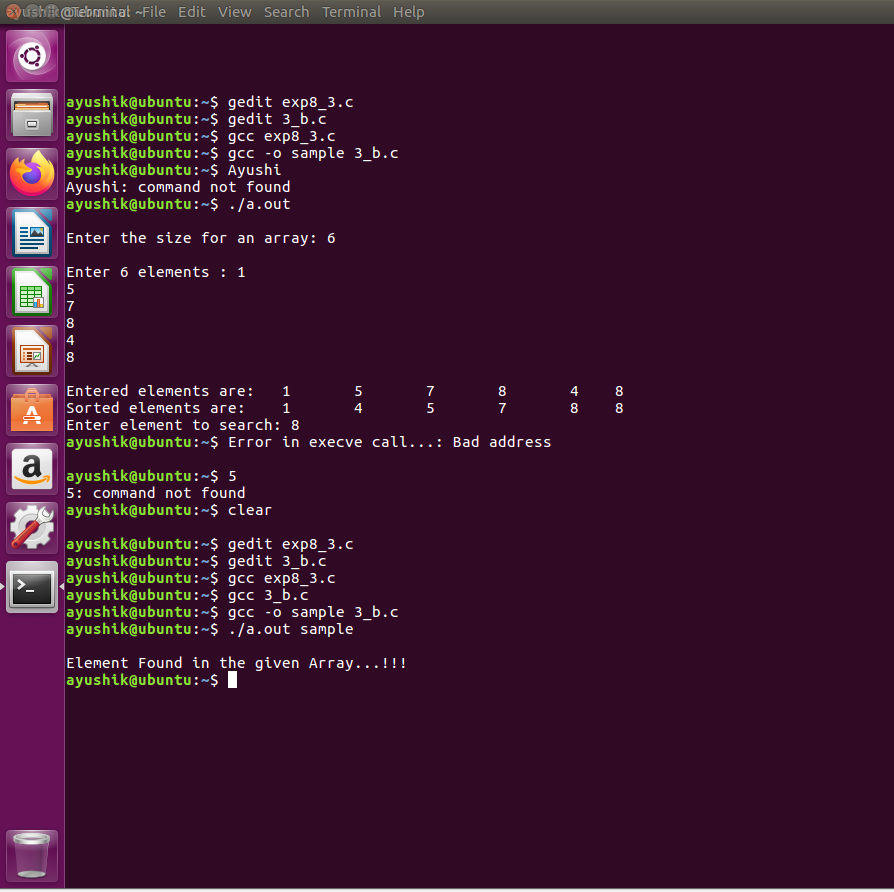
arguments of execve system call.

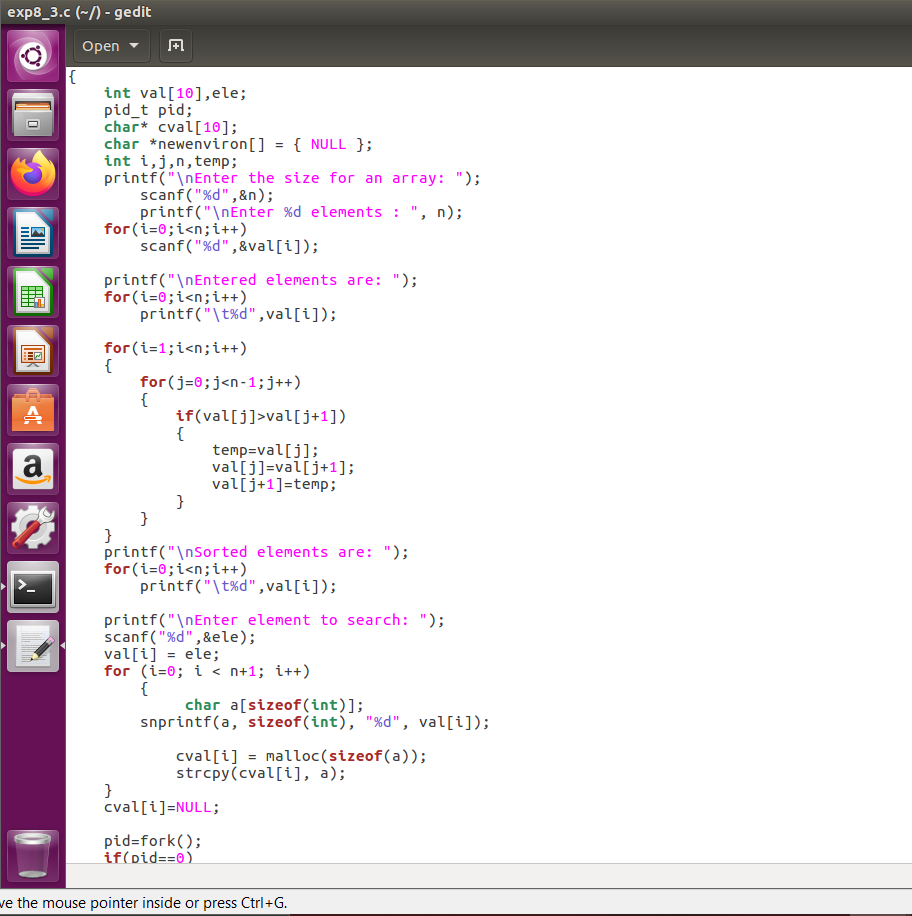
Step 8: Perform binary search.

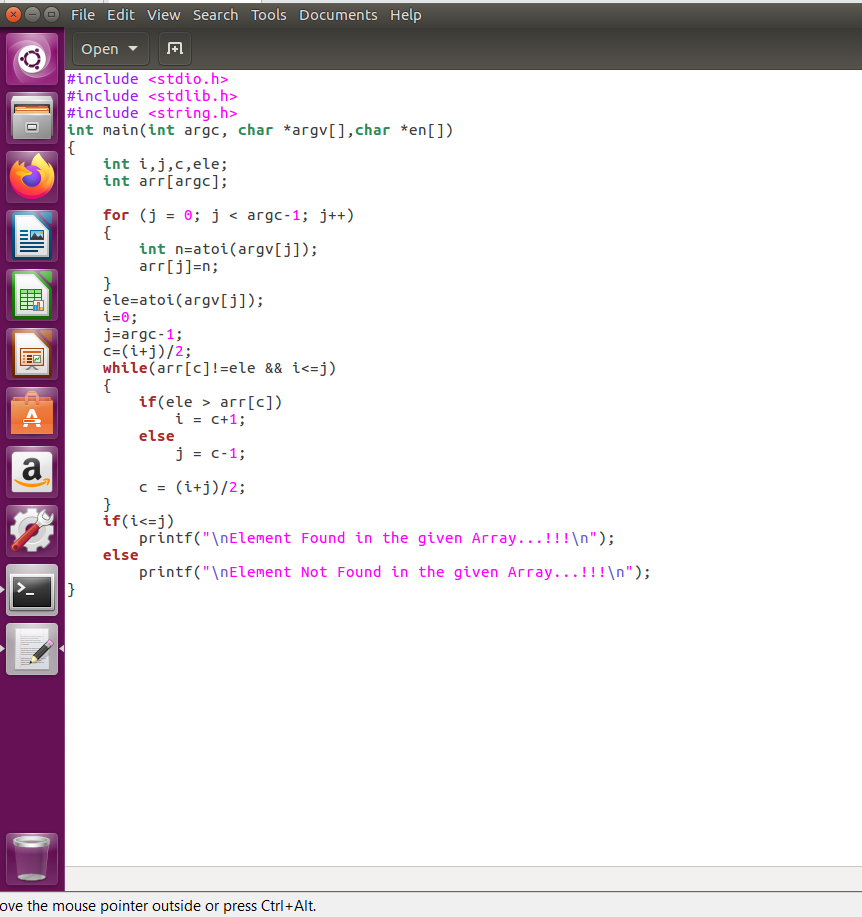
Step 9: Display the result of search.

Step 10: Stop.

**INPUT/OUTPUT: -**







**CODE: -**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int main(int argc, char \*argv[])

{

int val[10],ele;

pid\_t pid;

char\* cval[10];

char \*newenviron[] = { NULL };

int i,j,n,temp;

printf("\nEnter the size for an array: ");

scanf("%d",&n);

printf("\nEnter %d elements : ", n);

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

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

printf("\nEntered elements are: ");

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

printf("\t%d",val[i]);

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

{

for(j=0;j<n-1;j++)

{

if(val[j]>val[j+1])

{

temp=val[j];

val[j]=val[j+1];

val[j+1]=temp;

}

}

}

printf("\nSorted elements are: ");

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

printf("\t%d",val[i]);

printf("\nEnter element to search: ");

scanf("%d",&ele);

val[i] = ele;

for (i=0; i < n+1; i++)

{

char a[sizeof(int)];

snprintf(a, sizeof(int), "%d", val[i]);

cval[i] = malloc(sizeof(a));

strcpy(cval[i], a);

}

cval[i]=NULL;

pid=fork();

if(pid==0)

{

execve(argv[1], cval, newenviron);

perror("Error in execve call...");

}

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*argv[],char \*en[])

{

int i,j,c,ele;

int arr[argc];

for (j = 0; j < argc-1; j++)

{

int n=atoi(argv[j]);

arr[j]=n;

}

ele=atoi(argv[j]);

i=0;

j=argc-1;

c=(i+j)/2;

while(arr[c]!=ele && i<=j)

{

if(ele > arr[c])

i = c+1;

else

j = c-1;

c = (i+j)/2;

}

if(i<=j)

printf("\nElement Found in the given Array...!!!\n");

else

printf("\nElement Not Found in the given Array...!!!\n");

}

**Outcome:**

Learned and practiced how to pass data from parent process to child process in order to solve a problem,

1. **Write a program to print the Child process ID and Parent process ID in**

**both Child and Parent processes.**

**Objective:**

To write a C program to print the Child process ID and Parent process ID in

both Child and Parent processes.

**Procedure:**

Step 1: Print a hello message in Parent process.

Step 2: Invoke the fork() system call.

Step 3: Check for the return value of fork() system call. If the value returned is

zero then it is child process. Print the Child process ID and Parent process ID

using getpid() and getppid() methods.

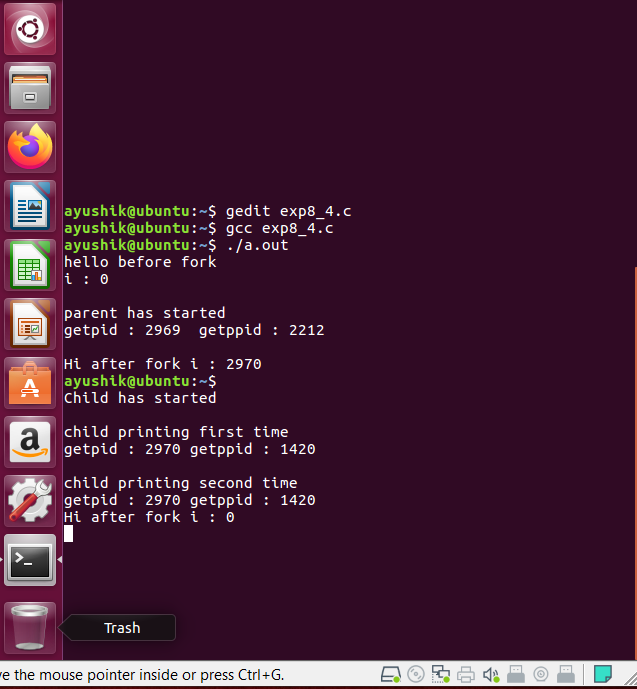
Step 4: If the fork() system call returns a non-zero positive integer, then it is

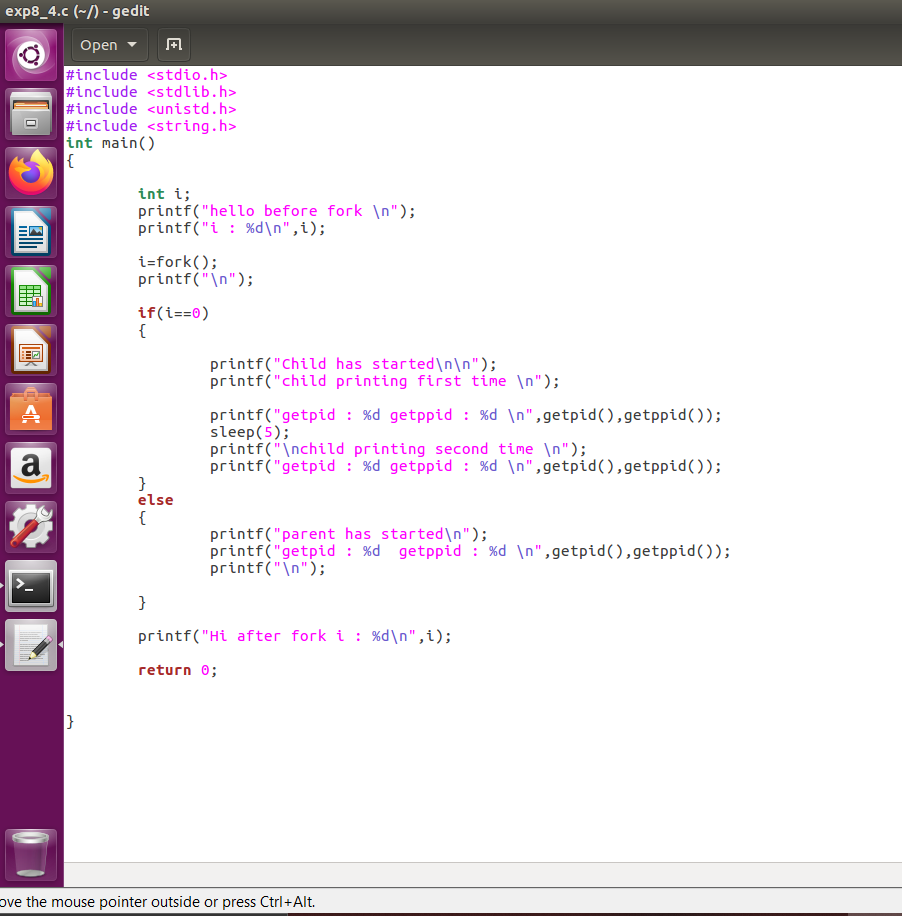
parent process. Print the Child process ID and Parent process ID using getpid()

and getppid() methods.

Step 5: stop.

**INPUT/OUTPUT: -**





**CODE: -**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

int main()

{

int i;

printf("hello before fork \n");

printf("i : %d\n",i);

i=fork();

printf("\n");

if(i==0)

{

printf("Child has started\n\n");

printf("child printing first time \n");

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

sleep(5);

printf("\nchild printing second time \n");

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

}

else

{

printf("parent has started\n");

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

printf("\n");

}

printf("Hi after fork i : %d\n",i);

return 0;

}

**Outcome:**

Learned and practiced how to use getpid() to get the process id and getppid() to

get the parent process id.