**Program 1**

**Aim:** Write a Program to demonstrate the use of atexit() exit handler

**Program :**

#include <stdio.h>

#include <stdlib.h>

#include<fcntl.h>

#include<unistd.h>

//File Discriptors

int fd1;

int fd2;

void functionA () { // Exit Handler

close(fd1);

printf("We are in functionA and closed the file \n");

}

void functionB () { // Exit Handler

close(fd2);

printf("We are in functionB and closed the file \n");

}

int main () {

/\* register the termination function \*/

char buf[10];

fd1=open("kk.txt",O\_RDONLY,742); // kk.txt should present in current folder

fd2=open("kk1.txt",O\_RDONLY,S\_IRWXU|S\_IRGRP|S\_IXOTH);

read(fd1,buf,10);

write(1,buf,10);

printf("\nCompleted the operation A \n");

atexit(functionA );

read(fd2,buf,10);

write(1,buf,10);

printf("\nCompleted the operation B \n");

atexit(functionB );

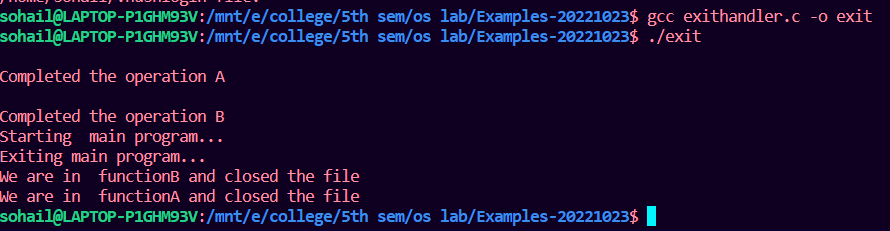
printf("Starting main Program...\n");

printf("Exiting main Program...\n");

return(0);

}

**Output:**



**Program 2**

**Aim:** Write a Program to demonstrate the use of exit() and \_exit() system call

**Program :**

i)use of exit():

#include<stdio.h>

#include<stdlib.h>

void func(){

printf("Exiting\n");

}

int main(){

atexit(func);

exit(10);

}

ii)use of \_exit():

#include<stdio.h>

#include<stdlib.h>

void func(){

printf("Exiting\n");

}

int main(){

printf("Start of the Program\n");

atexit(func);

\_Exit(10);

printf("End of the Program\n");

}

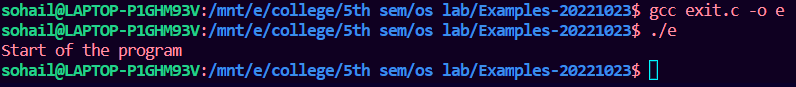
**Output:**

**Exit():**

A blue text on a black background

Description automatically generated

**\_exit():**



1. **WAP to demonstrate Banker’s algorithm (Resource Allocation)**

**Ans.**

**AIM:** A program to demonstrate Banker’s algorithm (Resource Allocation)

**PROCEDURE:**

It is a banker algorithm used to avoid deadlock and allocate resources safely to each process in the computer system. The 'S-State' examines all possible tests or activities before deciding whether the allocation should be allowed to each process. It also helps the operating system to successfully share the resources between all the processes. The banker's algorithm is named because it checks whether a person should be sanctioned a loan amount or not to help the bank system safely simulate allocation resources. For example, suppose the number of account holders in a particular bank is 'n', and the total money in a bank is 'T'. If an account holder applies for a loan; first, the bank subtracts the loan amount from full cash and then estimates the cash difference is greater than T to approve the loan amount. These steps are taken because if another person applies for a loan or withdraws some amount from the bank, it helps the bank manage and operate all things without any restriction in the functionality of the banking system.

Similarly, it works in an [operating system](https://www.javatpoint.com/operating-system). When a new process is created in a computer system, the process must provide all types of information to the [operating system](https://www.javatpoint.com/os-tutorial) like upcoming processes, requests for their resources, counting them, and delays. Based on these criteria, the operating system decides which process sequence should be executed or waited so that no deadlock occurs in a system. Therefore, it is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

**CODE:**

#include <stdio.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

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

    completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

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

{

    printf("\nFor process %d : ", i + 1);

    for(j = 0; j < r; j++)

        scanf("%d", &Max[i][j]);

}

printf("\n\nEnter the allocation for each process : ");

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

{

    printf("\nFor process %d : ",i + 1);

    for(j = 0; j < r; j++)

        scanf("%d", &alloc[i][j]);

}

printf("\n\nEnter the Available Resources : ");

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

        scanf("%d", &avail[i]);

    for(i = 0; i < p; i++)

        for(j = 0; j < r; j++)

            need[i][j] = Max[i][j] - alloc[i][j];

do

{

    printf("\n Max matrix:\tAllocation matrix:\n");

    for(i = 0; i < p; i++)

    {

        for( j = 0; j < r; j++)

            printf("%d  ", Max[i][j]);

        printf("\t\t");

        for( j = 0; j < r; j++)

            printf("%d  ", alloc[i][j]);

        printf("\n");

    }

    process = -1;

    for(i = 0; i < p; i++)

    {

        if(completed[i] == 0)//if not completed

        {

            process = i ;

            for(j = 0; j < r; j++)

            {

                if(avail[j] < need[i][j])

                {

                    process = -1;

                    break;

                }

            }

        }

        if(process != -1)

            break;

    }

    if(process != -1)

    {

        printf("\nProcess %d runs to completion!", process + 1);

        safeSequence[count] = process + 1;

        count++;

        for(j = 0; j < r; j++)

        {

            avail[j] += alloc[process][j];

            alloc[process][j] = 0;

            Max[process][j] = 0;

            completed[process] = 1;

        }

    }

}while(count != p && process != -1);

if(count == p)

{

    printf("\nThe system is in a safe state!!\n");

    printf("Safe Sequence : < ");

    for( i = 0; i < p; i++)

            printf("%d  ", safeSequence[i]);

    printf(">\n");

}

else

    printf("\nThe system is in an unsafe state!!");

}

