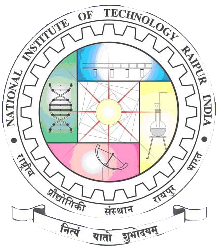
****

**NATIONAL INSTITUTE OF TECHNOLOGY, RAIPUR**

**Branch: Computer Science & Engineering**

**B. Tech Practical File**

**Subject: Operating System & Compiler Design (Lab)**

**Subject Code: CS105402CS**

**Name: Kunal Sachdeva**

**Roll No: 19115045**

**Semester: 5th**

**Date of Submission: 24 – 11 – 2021**

**Submitted to: Mr. Dilip Singh Sisodia**

**Teachers Signature: .................................................**

**Remarks: ................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................**

**Graded By:**

Contents

[**EXPERIMENT - 1** 3](#_Toc88742046)

[**EXPERIMENT - 2** 4](#_Toc88742047)

[**EXPERIMENT - 3** 5](#_Toc88742048)

[**EXPERIMENT - 4** 7](#_Toc88742049)

[**EXPERIMENT - 5** 13](#_Toc88742050)

[**EXPERIMENT - 6** 20](#_Toc88742051)

[**EXPERIMENT - 7** 22](#_Toc88742052)

[**EXPERIMENT - 8** 24](#_Toc88742053)

[**EXPERIMENT - 9** 27](#_Toc88742054)

[**EXPERIMENT - 10** 31](#_Toc88742055)

[**EXPERIMENT - 11** 34](#_Toc88742056)

[**EXPERIMENT - 12** 36](#_Toc88742057)

[**EXPERIMENT - 13** 43](#_Toc88742058)

[**EXPERIMENT - 14** 50](#_Toc88742059)

[**EXPERIMENT - 15** 54](#_Toc88742060)

[**EXPERIMENT - 16** 56](#_Toc88742061)

[**EXPERIMENT - 17** 57](#_Toc88742062)

[**EXPERIMENT - 18** 59](#_Toc88742063)

[**EXPERIMENT - 19** 65](#_Toc88742064)

# **EXPERIMENT - 1**

**Aim:** Working with Different Operating Systems. Study the DOS environment and practice commands for various activities like, File Handling, Text Processing, System Administration, Process Management, Archival, Network, File Systems, Advanced Commands.

**Theory:** Operating System is the resource manager of the system. It acts as an interface between hardware and users. It provides services to the user and programs. The different kinds of operating systems are: MS-DOS, Windows, Linux, Solaris etc.

DOS Environment is a reserved segment of primary memory called Master Environment Block. The environment is used to store system information in the form of environmental variables, or environmental strings. Environmental variables are available to the operating system, batch files, and application programs and are used to modify the behavior of DOS commands and application programs, or to provide them with additional information.

Some important commands:

* cd – change directory
* dir – list files and directories
* copy – copy one file to another
* erase – delete a file
* rmdir – delete a directory
* mkdir – create a new directory
* ren – rename a file
* cls – clear the screen
* ping- test the availability of a network connection to a specified host.
* attrib – display or change file attributes
* break – enable ctrl +c feature
* call – call a batch file from another batch file
* cmd – open command interpreter
* comp – compare files
* compact – compress / uncompress files
* date – view system date
* echo – display messages
* edit – view or edit files
* enable – enable a disables service or drivers
* disable – disable a service or drivers

# **EXPERIMENT - 2**

**Aim:** Work with some configuration commands & create batch files in the DOS environment.

**Theory**: A batch file (also known as a .bat file or batch script) is a **text file** that the Windows **cmd.exe** command line processor **executes as a batch job**.Command Prompt assumes both the role of interpreter and runtime environment. Put simply, a batch file is a computer program or script containing data or tasks that are processed sequentially by Command Prompt.

Batch files (also known as .bat files) are closely associated with Command Prompt. These files contain native commands that cmd.exe uses to process a sequence of commands.

**Procedure:**

**Step 1: Basics**

1. type “Notepad” in the Windows search bar and click on the Notepad icon in the search results
2. you need to know common system commands and understand how they work in batch files. These commands are :

* **ECHO**: Turns on the on-screen text display for executed commands
* **@ECHO OFF**: Turns off the on-screen text display for executed commands
* **START**: Runs a file with its default associated application
* **REM**: Indicates a comment line
* **MKDIR/RMDIR**: Creates or deletes a directory
* **DEL:** Deletes selected file(s)
* **COPY:** Copies selected file(s)
* **TITLE**: Sets the title of the CMD window

**Step 2: Create and save a batch file**

We write a simple script that creates multiple directories on a selected disk on your computer. For example, if you create and run a batch file with the following input, it will create two directories named “*Example1*” and “*Example2*” on drive C:



**Step 3: Run the new batch script**

Either run the script in the familiar Windows Explorer environment or open Command Prompt and run it using a command-line command.

The first option is simpler and easier for beginners because all you have to do is go to the directory where the batch file is located and double-click to run it.

If you want to open the batch file from the command line instead, do the following:

1. Go to the Windows search bar and type cmd.
2. Click Command Prompt to open the command line in the standard way. If you need administrator privileges to run it, right-click Command Prompt and then choose Run as Administrator.
3. Use the “Change directory” command (cd) to go to the directory where the batch file is located.
4. Type the name of the batch script (including the file extension) and press Enter.

**Step 4: Editing batch files**

If you want to add or remove commands or modify directories. To do this, simply go to the folder containing the command line script and right-click it. Then choose Edit:  
  
**Conclusion:** We explained unique features of useful scripts and saw how to create, run, save batch files.

# **EXPERIMENT - 3**

**Aim:** Install and configure Linux and work with Linux commands for the activities given in Experiment 1.

**Theory:** Linux is an open-source operating system developed in 1991 by Linus Torvalds as an idea to improve Unix; but was developed as a standalone operating system. It can be used in phones, laptops, PCs, cars or even in refrigerators. It offers many advantages like getting rid of viruses, malwares, crashes, slowdowns etc.

**Procedure:**

*Installing and Configuring Linux*

1. Partitioning Hard Disk (on Windows 10)
2. Open the Windows search bar.
3. Type “DISKMGMT.MSC” in search bar and press enter.
4. Right click on the main hard disk and select the option shrink volume.
5. Enter the desired amount of partition (atleast 20 GB).
6. Click finally on “shrink”.
7. Making Linux bootable USB
8. Download the Linux distro in ISO format from the distributers’ main website (Ubuntu).
9. Insert the USB device on your computer.
10. Download Rufus.
11. Open Rufus and select your USB drive from the device list.
12. Under Boot section, click Select button and choose the ISO file downloaded earlier.
13. Finally, click on “start” ( If you get a pop-up message asking you to select a mode that you want to use to write the image, choose ISO.)

Then wait for Rufus to mount your ISO file onto your drive. This might take some time.

1. Installing Linux from the USB drive
2. Insert the bootable Linux USB drive.
3. Click on the start menu.
4. Hold the shift key while restarting.
5. Select “Use a device”.
6. Find the device in the list.
7. Select the device and click on it, the computer would now boot Linux.
8. Select “Install Ubuntu” for installing.
9. Complete the installation process as guided.
10. Reboot when prompted.

The Linux would then be installed on your computer and can be used.

*Linux Commands*

***File commands-***

* ls – List the content of current directory
* cp – copy source file to target file
* mv – copy source file to target file and delete source file after that
* rm – remove specified files from the system
* ln – create internal link from source file to target file
* cd – changes the current directory to specified
* mkdir – create a new directory
* rmdir – delete the specified directory
* chown - transfers the ownership of a file to the user with the specified user name
* chgrp – transfer the group ownership of a given file to the group with the specified group name
* chmod – change access permission

***Accessing file content***

* cat – display content of the specified file
* less – browse content of specified file
* grep – find a specific search-string in specified file
* diff – compares content of two files

***File system***

* mount – mount data media (hard disk, CD-ROM, etc.) to directory of Linux file system.
* unmount – unmounts a mounted device from the file device

***System Commands***

* df – disk free; when used without any options, displays information about the total disk space, the disk space currently in use, and the free space on all the mounted drives. If a directory is specified, the information is limited to the drive on which that directory is located.
* free – displays information about RAM and swap space usage, showing the total and the used amount in both categories.
* date – this simple program displays the current system time.

***Processes***

* top – provides a quick overview of the currently running processes
* ps – if run without any options, this command displays a table of all your own programs or processes — those you started
* kill – It sends a TERM signal that instructs the program to shut itself down
* killall - This command is similar to kill, but uses the process name (instead of the process ID) as an argument, causing all processes with that name to be killed

# **EXPERIMENT - 4**

**Aim:** Write a program for the implementation of various CPU scheduling algorithms (FCFS, SJF, Priority).

**Theory:**

The purpose of Scheduling algorithms is:

1. Maximum CPU utilization
2. Fare allocation of CPU
3. Maximum throughput
4. Minimum turnaround time
5. Minimum waiting time
6. Minimum response time

There are three major types of process scheduling algorithms:

1. First Come First Serve (FCFS) Scheduling

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

**Completion Time:** Time at which process completes its execution.

**Turn Around Time(TAT):** Time Difference between completion time and arrival time.

**Turn Around Time:** Completion Time – Arrival Time

**Waiting Time(WT):** Time Difference between turn around time and burst time.

**Waiting Time:** Turn Around Time – Burst Time

1. Shortest Job First (SJF) Scheduling

SJF scheduling algorithm, schedules the processes according to their burst time. In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

1. Priority Scheduling

In Priority scheduling, there is a priority number assigned to each process. In some systems, the lower the number, the higher the priority. While, in the others, the higher the number, the higher will be the priority. The Process with the higher priority among the available processes is given the CPU.

**Source Code - FCFS:**

#include<stdio.h>

#include<conio.h>

void main(){

clrscr();

int bt[10]={0},at[10]={0},tat[10]={0},wt[10]={0},ct[10]={0};

int n,sum=0;

float totalTAT=0,totalWT=0;

printf("Enter number of processes: ");

scanf("%d",&n);

printf("Enter Arrival time and Burst time for each process:\n");

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

     printf("Arrival time of process[%d]: ",i+1);

     scanf("%d",&at[i]);

     printf("Burst time of process[%d]: ",i+1);

     scanf("%d",&bt[i]);

}

//calculate completion time of processes

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

    sum+=bt[j];

    ct[j]+=sum;

}

//calculate turnaround time and waiting times

for(int k=0;k<n;k++){

    tat[k]=ct[k]-at[k];

    totalTAT+=tat[k];

}

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

     wt[k]=tat[k]-bt[k];

     totalWT+=wt[k];

}

printf("P#\t AT\t BT\t CT\t TAT\t WT\t\n\n");

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

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

}

printf("\nAverage Turnaround Time = %f\n", totalTAT/n);

printf("Average WT = %f\n\n", totalWT/n);

getch();

}

**Output - FCFS:**

**Enter number of processes: 5**

**Enter Arrival time and Burst time for each process:**

**Arrival time of process[1]: 0**

**Burst time of process[1]: 4**

**Arrival time of process[2]: 1**

**Burst time of process[2]: 3**

**Arrival time of process[3]: 2**

**Burst time of process[3]: 1**

**Arrival time of process[4]: 3**

**Burst time of process[4]: 2**

**Arrival time of process[5]: 4**

**Burst time of process[5]: 5**

**P#  AT  BT  CT  TAT  WT**

**P1  0   4   4   4    0**

**P2  1   3   7   6    3**

**P3  2   1   8   6    5**

**P4  3   2   10  7    5**

**P5  4   5   15  11   6**

**Average Turnaround Time = 6.800000**

**Average WT = 3.800000**

**Source Code – SJFS:**

#include<stdio.h>

#include<conio.h>

void main(){

      clrscr();

      int n,temp,tt=0,min,d,i,j;

      float atat=0,awt=0,stat=0,swt=0;

      printf("Enter no of process: ");

      scanf("%d",&n);

      int a[10],b[10],e[10],tat[10],wt[10];

      for(i=0;i<n;i++){

    printf("Enter arrival time P[%d]: ",i+1);       //input

    scanf("%d",&a[i]);

      }

      for(i=0;i<n;i++){

    printf("Enter burst time P[%d]: ",i+1);      //input

    scanf("%d",&b[i]);

      }

      for(i=0;i<n;i++){

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

if(b[i]>b[j]){

      temp=a[i];

      a[i]=a[j];

      a[j]=temp;

      temp=b[i];

      b[i]=b[j];

      b[j]=temp;

                }

          }

      }

      min=a[0];

      for(i=0;i<n;i++){

            if(min>a[i])

            {

                  min=a[i];

                  d=i;

            }

      }

      tt=min;

      e[d]=tt+b[d];

      tt=e[d];

      for(i=0;i<n;i++){

            if(a[i]!=min){

                  e[i]=b[i]+tt;

                  tt=e[i];

            }

      }

      for(i=0;i<n;i++){

            tat[i]=e[i]-a[i];

            stat=stat+tat[i];

            wt[i]=tat[i]-b[i];

            swt=swt+wt[i];

      }

      atat=stat/n;

      awt=swt/n;

      printf("Process  Arrival-time(s)  Burst-time(s)  Waiting-time(s)  Turnaround-time(s)\n");

    for(i=0;i<n;i++){

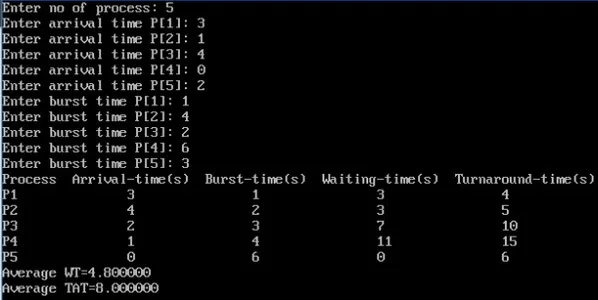
    printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i+1,a[i],b[i],wt[i],tat[i]);

    }

    printf("awt=%f\natat=%f",awt,atat);  //average waiting time and turn around time

    getch();

}

**Output – SJFS:**

**Source Code – Priority:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main(){

    int bt[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],tat[10];

    int totwt=0,tottat=0;

    float awt,atat;

    char pn[10][10],t[10];

    clrscr();

    printf("Enter the number of process:");

    scanf("%d",&n);

    for(i=0; i<n; i++){

     printf("Enter process name:");

     scanf("%s",&pn[i]);

     printf("Enter arrival time:");

     scanf("%d",&at[i]);

     printf("Enter burst time:");

     scanf("%d",&bt[i]);

     printf("Enter priority:");

     scanf("%d",&p[i]);

    }

    for(i=0; i<n; i++)

        for(j=0; j<n; j++){

            if(p[i]<p[j]){

                 temp=p[i];

                 p[i]=p[j];

                 p[j]=temp;

                 temp=at[i];

                 at[i]=at[j];

                 at[j]=temp;

                 temp=bt[i];

                 bt[i]=bt[j];

                 bt[j]=temp;

                 strcpy(t,pn[i]);

                 strcpy(pn[i],pn[j]);

                 strcpy(pn[j],t);

              }

          }

    for(i=0; i<n; i++){

        if(i==0){

           st[i]=at[i];

           wt[i]=st[i]-at[i];

           ft[i]=st[i]+bt[i];

           tat[i]=ft[i]-at[i];

         }

      else{

         st[i]=ft[i-1];

         wt[i]=st[i]-at[i];

         ft[i]=st[i]+bt[i];

         tat[i]=ft[i]-at[i];

       }

     totwt+=wt[i];

     tottat+=tat[i];

    }

    awt=(float)totwt/n;

    atat=(float)tottat/n;

    printf("\nPName\tArrival-time\tBurst-time\tPriority\tWaiting-time\tTat-time");

    for(i=0; i<n; i++)

printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t\%5d\t\t%5d",pn[i],at[i],bt[i],p[i],wt[i],tat[i]);

    printf("\nAverage waiting time is:%f",awt);

    printf("\nAverage turnaroundtime is:%f",atat);

    getch();

}

**Output – Priority:**

# **EXPERIMENT - 5**

**Aim:** To implement various page replacement techniques.

* 1. FIFO
  2. LRU
  3. OPTIMAL

**Theory:** Page replacement algorithms are an important part of virtual memory management and it helps the operating system to decide which memory page can be moved out making space for the currently needed page. However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults.

FIFO - This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

LRU - In this algorithm page will be replaced which is least recently used

OPTIMAL - In this algorithm, pages are replaced which would not be used for the longest duration of time in the future. This algorithm will give us less page fault when compared to other page replacement algorithms.

**A) FIRST IN FIRST OUT SOURCE CODE :**

#include<stdio.h>

#include<conio.h>

int fr[3];

void main() {

void display();

int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};

int flag1=0,flag2=0,pf=0,frsize=3,top=0;

clrscr();

for(i=0;i<3;i++) {

fr[i]= -1;

}

for(j=0;j<12;j++) {

flag1=0;

flag2=0;

for(i=0;i<12;i++) {

if(fr[i]==page[j]) {

flag1=1;

flag2=1;

break;

}

}

if(flag1==0) {

for(i=0;i=frsize)

top=0;

}

display();

}

printf("Number of page faults : %d ",pf+frsize);

getch();

}

void display() {

int i;

printf("\n");

for(i=0;i<3;i++) printf("%d\t",fr[i]);

}

**Output:**

**2 1 1**

**2 3 1**

**2 3 1**

**2 3 1**

**5 3 1**

**5 2 1**

**5 2 4**

**5 2 4**

**3 2 4**

**3 2 4**

**3 5 4**

**3 5 2**

**Number of page faults: 9**

**B) LEAST RECENTLY USED SOURCE CODE :**

#include<stdio.h>

#include<conio.h>

int fr[3];

void main() {

void display();

int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];

int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;

clrscr();

for(i=0;i<3;i++) {

fr[i]=-1;

}

for(j=0;j<12;j++) {

flag1=0,flag2=0;

for(i=0;i<3;i++) {

if(fr[i]==p[j]) {

flag1=1;

flag2=1;

break;

}

}

if(flag1==0) {

for(i=0;i<3;i++) {

if(fr[i]== -1) {

fr[i]=p[j];

flag2=1;

break;

}

}

}

if(flag2==0) {

for(i=0;i<3;i++) fs[i]=0;

for(k=j -1,l=1;l<=frsize -1;l++,k-- ) {

for(i=0;i<3;i++) {

if(fr[i]==p[k]) fs[i]=1;

}

}

for(i=0;i<3;i++) {

if(fs[i]==0) index=i;

}

fr[index]=p[j];

pf++;

}

display();

}

printf(" \n no of page faults :%d",pf+frsize);

getch();

}

void display() {

int i;

printf(" \ n");

for(i=0;i<3;i++) printf(" \ t%d",fr[i]);

}

**Output:**

**2 1 1**

**2 3 1**

**2 3 1**

**2 3 1**

**2 5 1**

**2 5 1**

**2 5 4**

**2 5 4**

**3 5 4**

**3 5 2**

**3 5 2**

**3 5 2**

**No of page faults: 7**

**C) OPTIMAL SOURCE CODE:**

#include<stdio.h>

#include<conio.h>

int fr[3], n, m;

void display();

void main()

{

int i,j,page[20],fs[10];

int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;

float pr;

clrscr();

printf("Enter length of the reference string: ");

scanf("%d",&n);

printf("Enter the reference string: ");

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

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

printf("Enter no of frames: ");

scanf("%d",&m);

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

fr[i]=-1;

pf=m;

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

flag1=0;

flag2=0;

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

if(fr[i]==page[j]) {

flag1=1;

flag2=1;

break;

}

}

if(flag1==0) {

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

if(fr[i] == -1){

fr[i]=page[j];

flag2=1;

break;

}

}

}

if(flag2==0) {

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

lg[i]=0;

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

for(k=j+1;k<=n;k++) {

if(fr[i]==page[k]) {

lg[i]=k-j;

break;

}

}

}

found=0;

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

if(lg[i]==0) {

index=i;

found = 1;

break;

}

}

if(found==0) {

max=lg[0];

index=0;

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

if(max<lg[i]) {

max=lg[i];

index=i;

}

}

}

fr[index]=page[j];

pf++;

}

display();

}

printf("Number of page faults : %d\n", pf);

pr=(float)pf/n\*100;

printf("Page fault rate = %f \n", pr); getch();

}

void display()

{

int i;

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

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

printf("\n");

}

**Output:**

**Enter length of the reference string: 12**

**Enter the reference string: 1 2 3 4 1 2 5 1 2 3 4 5**

**Enter no of frames: 3**

**1 -1 -1**

**1 2 -1**

**1 2 3**

**1 2 4**

**1 2 4**

**1 2 4**

**1 2 5**

**1 2 5**

**1 2 5**

**3 2 5**

**4 2 5**

**4 2 5**

**Number of page faults : 7**

**Page fault rate = 58.333332**

# **EXPERIMENT - 6**

**Aim:** Write a program for the implementation of system calls (fork and vfork) of Unix operating systems.

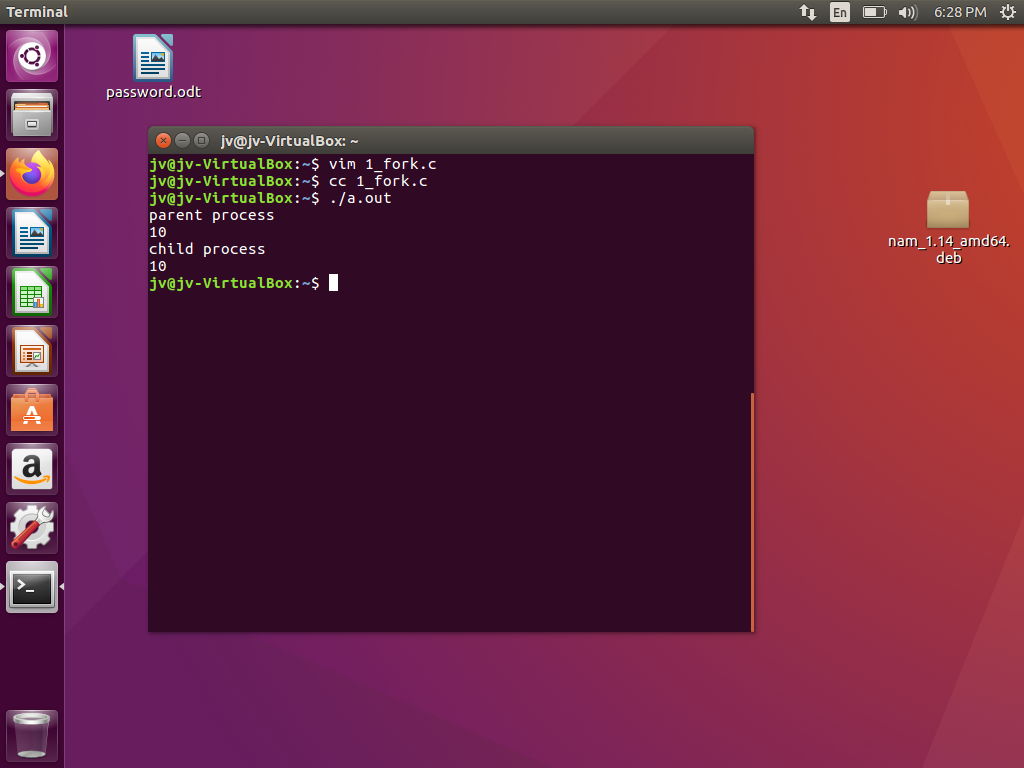
**Theory:**

Fork() is a system call that creates a new process. The new process produced by the fork() system call is referred to as the child process, whereas the process that invoked the fork() system call is referred to as the parent process. The code of a child process is identical to that of its parent. Once a child process has been formed, both the parent and child processes begin execution from the next statement following fork(), and both processes are executed at the same time.

Vfork() is another system call that is used to start a new process. The new process produced by the vfork() system call is known as the child process, and the process that invoked the vfork() system call is known as the parent process. The code of the child process is the same as the code of its parent process. Because both processes share the same address space, the child process suspends the parent process's execution until the child process completes its execution.

**Source Code – fork():**

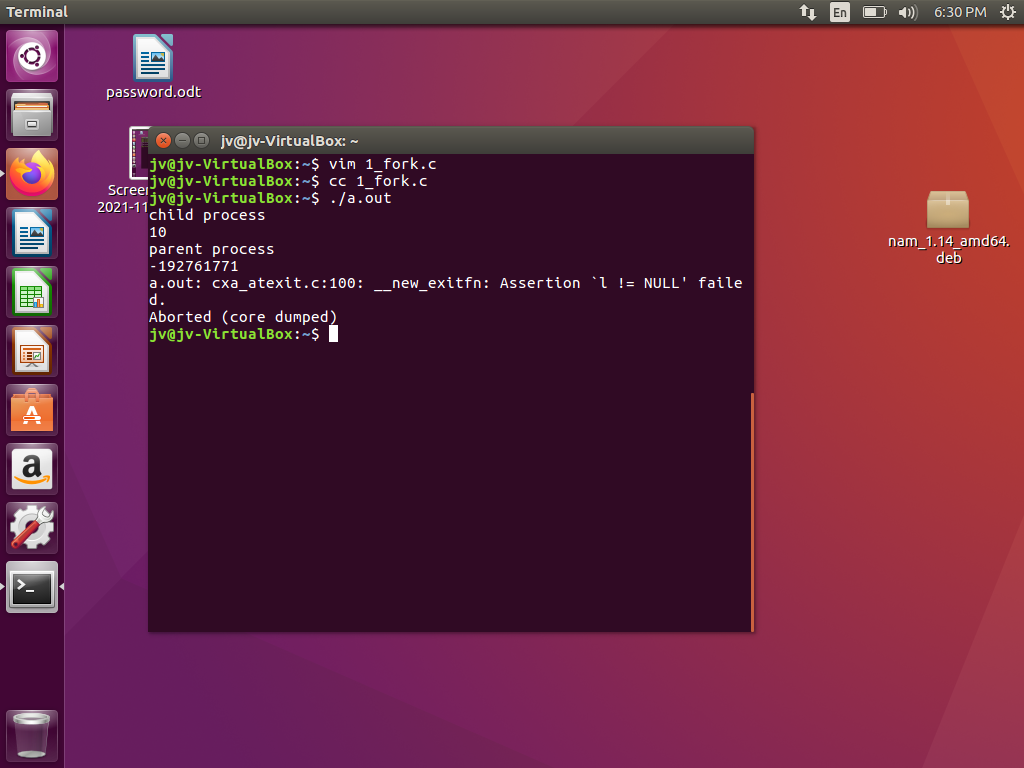
#include<stdio.h>  
#include<unistd.h>  
#include<sys/types.h>  
int main(void)  
{  
int n=10;  
pid\_t pid=fork();  
if(pid==0){  
printf("child process\n");  
}  
else{  
printf("parent process\n");  
}  
printf("%d \n",n);  
return 0;  
}

**Output – fork():**

**Source Code – vfork():**

#include<stdio.h>  
#include<unistd.h>  
#include<sys/types.h>  
int main(void)  
{  
int n=10;  
pid\_t pid=vfork();  
if(pid==0){  
printf("child process\n");  
}  
else{  
printf("parent process\n");  
}  
printf("%d \n",n);  
return 0;  
}

**Output – vfork():**



# **EXPERIMENT - 7**

**Aim:** Write a program for the implementation of Producer Consumer problem.

**Theory:** Producer consumer problem is a synchronization problem. There is a fixed size buffer where the producer produces items and that is consumed by a consumer process. One solution to the producer consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

**Source Code:**

#include<bits/stdc++.h>

using namespace std;

int main() {

    int buffer[10], bufsize, in, out, produce, consume, choice=0;

in = 0;

    out = 0;

    bufsize = 10;

    while(choice != 3) {

        cout<<"\n1.Produce\t2.Consume\t3.Exit";

        cout<<"Enter your choice: ";

        cin>>choice;

        switch(choice) {

            case 1: if((in+1)%bufsize == out)

                        cout<<"\nBuffer is full";

                    else {

                        cout<<"\nEnter the value: ";

                        cin>>produce;

                        buffer[in] = produce;

                        in = (in+1)%bufsize;

                    }

                    break;

            case 2: if(in == out)

                        cout<<"\nBuffer is Empty";

                    else {

                        consume == buffer[out];

                        cout<<"\nThe consumed value is "<<consume;

                        out = (out+1)%bufsize;

                    }

                    break;

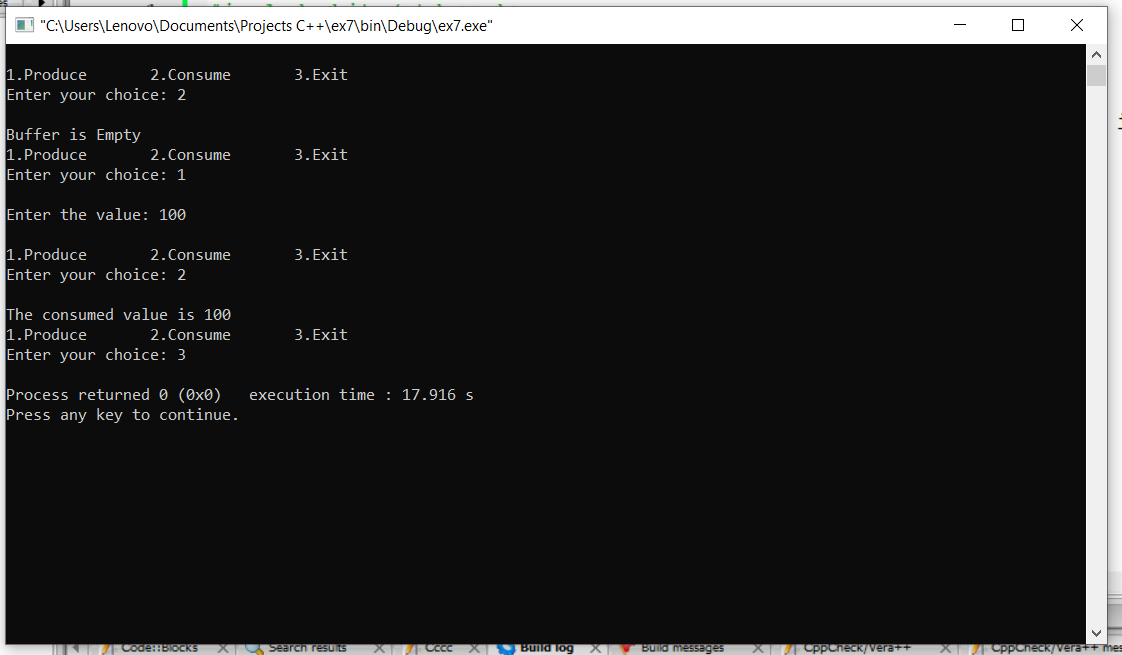
        }

    }

    return 0;

}

**Output:**



# **EXPERIMENT - 8**

**Aim:** Write a program for the implementation of Readers Writers problem.

**Theory:** The readers-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object.

The readers-writers problem is used to manage synchronization so that there are no problems with the object data. For example - If two readers access the object at the same time there is no problem. However, if two writers or a reader and writer access the object at the same time, there may be problems.

To solve this situation, a writer should get exclusive access to an object i.e., when a writer is accessing the object, no reader or writer may access it. However, multiple readers can access the object at the same time. This can be implemented using semaphores.

**Source Code:**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

/\*

This program provides a possible solution for first readers writers problem using mutex and semaphore.

I have used 10 readers and 5 producers to demonstrate the solution.

\*/

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt\*2;

printf("Writer %d modified cnt to %d\n",(\*((int \*)wno)),cnt);

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader++;

if(numreader == 1) {

sem\_wait(&wrt); // If this id the first reader, then it will block the writer

}

pthread\_mutex\_unlock(&mutex);

// Reading Section

printf("Reader %d: read cnt as %d\n",\*((int \*)rno),cnt);

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader--;

if(numreader == 0) {

sem\_post(&wrt); // If this is the last reader, it will wake up the writer.

}

pthread\_mutex\_unlock(&mutex);

}

int main()

{

pthread\_t read[10],write[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&wrt,0,1);

int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering the producer and consumer

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

pthread\_create(&read[i], NULL, (void \*)reader, (void \*)&a[i]);

}

for(int i = 0; i < 5; i++) {

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&a[i]);

}

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

pthread\_join(read[i], NULL);

}

for(int i = 0; i < 5; i++) {

pthread\_join(write[i], NULL);

}

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

**Output:**

**Reader 7: read cnt as 1**

**Reader 8: read cnt as 1**

**Reader 9: read cnt as 1**

**Reader 10: read cnt as 1**

**Writer 1 modified cnt to 2**

**Writer 2 modified cnt to 4**

**Writer 3 modified cnt to 8**

**Writer 4 modified cnt to 16**

**Writer 5 modified cnt to 32**

**Reader 6: read cnt as 32**

**Reader 5: read cnt as 32**

**Reader 4: read cnt as 32**

**Reader 3: read cnt as 32**

**Reader 2: read cnt as 32**

**Reader 1: read cnt as 32**

# **EXPERIMENT - 9**

**Aim:** Write a program for the implementation of Banker’s Algorithm.

**Theory:** Deadlock is a situation where in two or more competing actions are waiting for the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user requests a set of resources, the system must determine whether the allocation of each resource will leave the system in safe state. If it will the resources are allocation; otherwise, the process must wait until some other process release the resources.

Banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” (also called safe state) check to test for possible activities, before deciding whether allocation should be allowed to continue. It uses safety algorithm and resource request algorithm under the hood.

For a process Pi and resource Ri, if the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However, if the state is unsafe, the Pi must wait for Request Ri and the old resource-allocation state is restored.

**Algorithm:**

1. Start the program.

2. Get the values of resources and processes.

3. Get the available value.

4. After allocation, find the need value.

5. Check whether it’s possible to allocate.

6. If it is possible then the system is in safe state.

7. Else system is not in safety state.

8. If the new request comes then check that the system is in safety or not if we choose to allow the request.

10. Stop the program.

11. End

**Source Code:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

int alloc[10][10],max[10][10];

int avail[10],work[10],total[10];

int i,j,k,n,need[10][10];

int m;

int count=0,c=0;

char finish[10];

clrscr();

printf("Enter the no. of processes and resources:");

scanf("%d%d",&n,&m);

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

finish[i]='n';

printf("Enter the claim matrix:\n");

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

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

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

printf("Enter the allocation matrix:\n");

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

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

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

printf("Resource vector:");

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

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

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

avail[i]=0;

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

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

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

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

work[i]=avail[i];

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

work[j]=total[j]-work[j];

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

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

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

A:

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

{

c=0;

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

if((need[i][j]<=work[j])&&(finish[i]=='n'))

c++;

if(c==m)

{

printf("All the resources can be allocated to Process %d", i+1);

printf("\n\nAvailable resources are:");

for(k=0;k<m;k++)

{

work[k]+=alloc[i][k];

printf("%4d",work[k]);

}

printf("\n");

finish[i]='y';

printf("\nProcess %d executed?:%c \n",i+1,finish[i]);

count++;

}

}

if(count!=n)

goto A;

else

printf("\n System is in safe mode");

printf("\n The given state is safe state");

getch();

}

**Output:**

**Enter the no. of processes and resources: 4 3**

**Enter the claim matrix:**

**3 2 2**

**6 1 3**

**3 1 4**

**4 2 2**

**Enter the allocation matrix:**

**1 0 0**

**6 1 2**

**2 1 1**

**0 0 2**

**Resource vector:9 3 6**

**All the resources can be allocated to Process 2**

**Available resources are: 6 2 3**

**Process 2 executed?:y**

**All the resources can be allocated to Process 3 Available resources**

**are: 8 3 4**

**Process 3 executed?:y**

**All the resources can be allocated to Process 4 Available resources**

**are: 8 3 6**

**Process 4 executed?:y**

**All the resources can be allocated to Process 1**

**Available resources are: 9 3 6**

**Process 1 executed?:y**

**System is in safe mode**

**The given state is safe state**

# **EXPERIMENT - 10**

**Aim:** Write a program to simulate the concept of semaphores.

**Theory:** The term "semaphore" refers to an integer variable that is shared by several threads. In a multiprocessing context, this variable is utilised to address the critical section problem and establish process synchronisation.

That is, Semaphore is an integer variable which is used in mutually exclusive manner by various concurrent cooperative processes in order to achieve synchronization.

Semaphores are of two types:

1. Binary Semaphore
2. Counting Semaphore
3. **Binary Semaphore:** It can only have two possible values: 0 and 1. Its value is set to one at the start. It's used to implement several procedures to solve critical section problems.
4. **Counting Semaphore:** Its value can range over an unrestricted domain ie. -∞ to +∞

To solve the critical section problem utilising two atomic operations for process synchronisation, wait and signal.

Wait operation is also called sleep, or down operation, and P and Signal operation is also called signal, wake-up, or up operation and V.

Both operations are atomic and semaphore(s) is always initialized to one. Here atomic means that variable on which read, modify and update happens at the same time/moment with no pre-emption i.e. in-between read, modify and update no other operation is performed that may change the variable.

A critical section is surrounded by both operations to implement process synchronization. The critical section of Process P is in between P and V operation.

**Source Code:**

#include<pthread.h>

#include<stdio.h>

#include<semaphore.h>

#include<unistd.h>

void \*fun1();

void \*fun2();

int shared=1; //shared variable

sem\_t s; //semaphore variable

int main()

{

sem\_init(&s,0,1); //initialize semaphore variable - 1st argument is address of variable, 2nd is number of processes sharing semaphore, 3rd argument is the initial value of semaphore variable

pthread\_t thread1, thread2;

pthread\_create(&thread1, NULL, fun1, NULL);

pthread\_create(&thread2, NULL, fun2, NULL);

pthread\_join(thread1, NULL);

pthread\_join(thread2,NULL);

printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable

}

void \*fun1()

{

int x;

sem\_wait(&s); //executes wait operation on s

x=shared;//thread1 reads value of shared variable

printf("Thread1 reads the value as %d\n",x);

x++; //thread1 increments its value

printf("Local updation by Thread1: %d\n",x);

sleep(1); //thread1 is preempted by thread 2

shared=x; //thread one updates the value of shared variable

printf("Value of shared variable updated by Thread1 is: %d\n",shared);

sem\_post(&s);

}

void \*fun2()

{

int y;

sem\_wait(&s);

y=shared;//thread2 reads value of shared variable

printf("Thread2 reads the value as %d\n",y);

y--; //thread2 increments its value

printf("Local updation by Thread2: %d\n",y);

sleep(1); //thread2 is preempted by thread 1

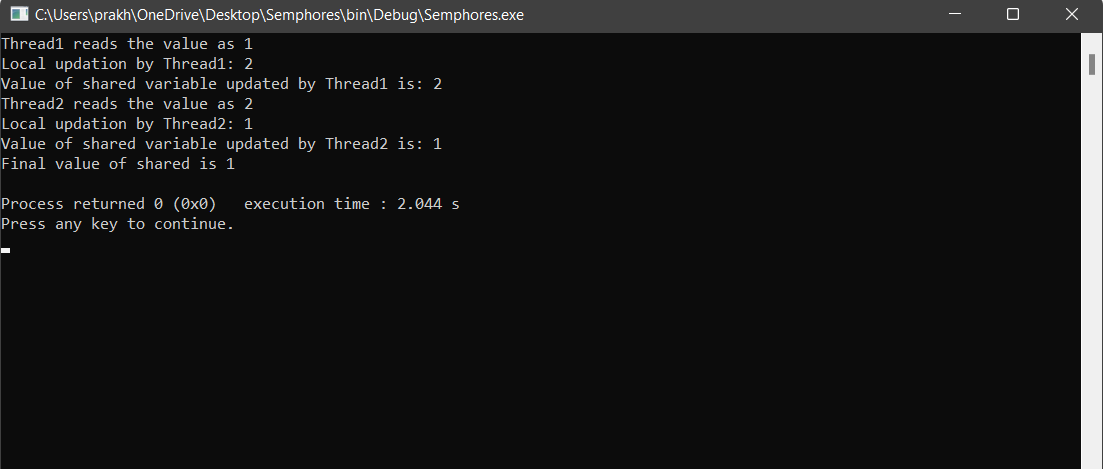
shared=y; //thread2 updates the value of shared variable

printf("Value of shared variable updated by Thread2 is: %d\n",shared);

sem\_post(&s);

}

**Output:**

****

# **EXPERIMENT - 11**

**Aim:** Write a program to simulate the concept of inter process communication

**Theory:** The system call provides an interface to the operating system services. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do.

**Types of System Calls**

There are 5 different categories of system calls:

process control, file manipulation, device manipulation, information maintenance and communication.

* **Process Control:** A running program needs to be able to stop execution either normally or abnormally. When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.
* **File Management :** Some common system calls are create, delete, read, write, reposition, or close. Also, there is a need to determine the file attributes – get and set file attribute. Many times the OS provides an API to make these system calls.
* **Device Management** : Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user process. These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file. User programs request the device, and when finished they release the device. Similar to files, we can read, write, and reposition the device.
* **Information Management :** Some system calls exist purely for transferring information between the user program and the operating system. An example of this is time, or date. The OS also keeps information about all its processes and provides system calls to report this information.
* **Communication :**There are two models of interprocess communication, the message-passing model and the shared memory model. Message-passing uses a common mailbox to pass messages between processes. Shared memory use certain system calls to create and gain access to create and gain access to regions of memory owned by other processes. The two processes exchange information by reading and writing in the shared data.

**Source Code:**

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

int main() {

int pfd[2], i;

pid\_t mypid;

if (pipe(pfd) < 0)

perror("Pipe Error");

if (!fork()) {

char data;

printf("Enter a Number…\n");

scanf("%d", & data);

write(pfd[1], & data, 1);

mypid = getpid();

printf("I am process %d\n", mypid);

printf("My parent is process %d\n", getppid());

printf("Child Exiting…\n");

exit(0);

} else {

char data1;

read(pfd[0], & data1, 1);

printf("Received %d from child \n", data1);

printf("The odd numbers are… \n");

for (i = 1; i <= data1; i += 2) {

printf("%5d", i);

sleep(2);

}

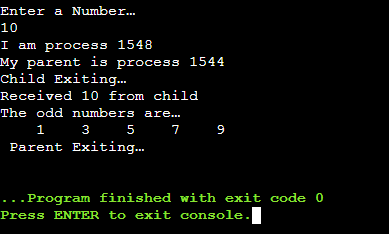
printf("\n Parent Exiting…\n");

exit(0);

}

return (0);

}

**Output:**

# **EXPERIMENT - 12**

**Aim:** Write a program for the implementation of various memory allocation algorithms (First fit, Best fit, and Worst fit).

**Theory:**

**First Fit-** In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

**Best Fit-** The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

**Worst fit-** In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

**Source Code:**

**First fit-**

#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp;

static int bf[max],ff[max];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block %d:",i);

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

}

printf("Enter the size of the files:-\n");

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

{

printf("File %d:",i);

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

}

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

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

{

ff[i]=j;

break;

}

}

}

frag[i]=temp;

bf[ff[i]]=1;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragment");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

**Output:**

**Best fit-**

#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

static int bf[max],ff[max];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block %d:",i);

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

}

printf("Enter the size of the files:-\n");

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

{

printf("File %d:",i);

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

}

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

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1)

{

temp=b[j]-f[i];

if(temp>=0)

if(lowest>temp)

{

ff[i]=j;

lowest=temp;

}

}

}

frag[i]=lowest;

bf[ff[i]]=1;

lowest=10000;

}

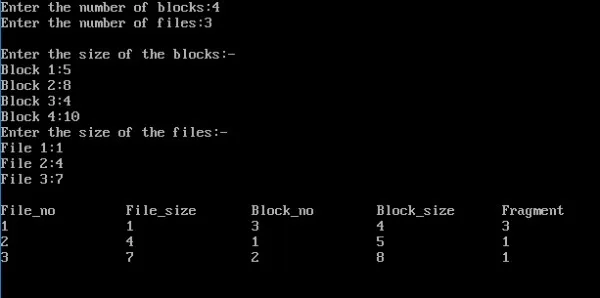
printf("\nFile\_no \tFile\_size \tBlock\_no \tBlock\_size \tFragment");

for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

**Output:**

**Worst fit:**

#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;

static int bf[max],ff[max];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

printf("Block %d:",i);

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

}

printf("Enter the size of the files:-\n");

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

{

printf("File %d:",i);

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

}

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

{

for(j=1;j<=nb;j++)

{

if(bf[j]!=1) //if bf[j] is not allocated

{

temp=b[j]-f[i];

if(temp>=0)

if(highest<temp)

{

ff[i]=j;

highest=temp;

}

}

}

frag[i]=highest;

bf[ff[i]]=1;

highest=0;

}

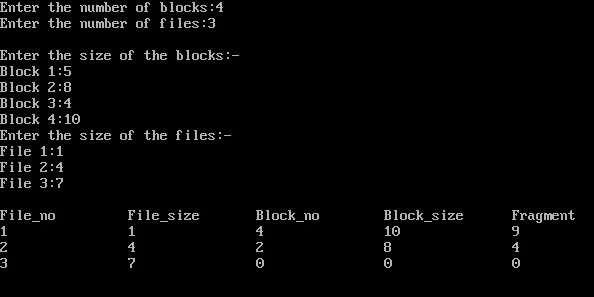
printf("\nFile\_no \tFile\_size \tBlock\_no \tBlock\_size \tFragment");

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

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);

getch();

}

**Output:**

# **EXPERIMENT - 13**

**Aim:** To write a program to implement the following types of Disk Scheduling Algorithms: FCFS, SCAN, SSTF, C-SCAN.

**Theory:** Every process requests the Operating system to access the disk for assigning it the I/O time. The operating system thus has to pick an algorithm to satisfy each request by maintaining the speed and efficiency of the program execution. This technique of choosing the order of process requests is called disk scheduling. The common disk scheduling methods include-

FCFS, SCAN, SSTF, C-SCAN.

**First Come First Serve (FCFS)**

FCFS is the simplest disk scheduling algorithm. As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue.

**SCAN (Elevator) algorithm**

In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk.

**Shortest Seek Time First (SSTF)**

Basic idea is the tracks which are closer to current disk head position should be serviced first in order to minimise the seek operations.

**Circular SCAN (C-SCAN)**

It is a modified version of SCAN disk scheduling algorithm. Like SCAN, it moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip and starts servicing again once reaches the beginning.

**Source Code**:

**(1) FCFS**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

void FCFS(int arr[], int head)

{

    int seek\_count = 0;

    int distance, cur\_track;

    for (int i = 0; i < size; i++) {

        cur\_track = arr[i];

        distance = abs(cur\_track - head);

        seek\_count += distance;

        head = cur\_track;

    }

    cout << "Total number of seek operations = "

         << seek\_count << endl;

    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < size; i++) {

        cout << arr[i] << endl;

    }

}

int main()

{

    int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

    int head = 50;

    FCFS(arr, head);

    return 0;

}

**Output:**

**Total number of seek operations = 510**

**Seek Sequence is**

**176**

**79**

**34**

**60**

**92**

**11**

**41**

**114**

**(2) SCAN**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

int disk\_size = 200;

void SCAN(int arr[], int head, string direction)

{

    int seek\_count = 0;

    int distance, cur\_track;

    vector<int> left, right;

    vector<int> seek\_sequence;

    if (direction == "left")

        left.push\_back(0);

    else if (direction == "right")

        right.push\_back(disk\_size - 1);

    for (int i = 0; i < size; i++) {

        if (arr[i] < head)

            left.push\_back(arr[i]);

        if (arr[i] > head)

            right.push\_back(arr[i]);

    }

    sort(left.begin(), left.end());

    sort(right.begin(), right.end());

    int run = 2;

    while (run--) {

        if (direction == "left") {

            for (int i = left.size() - 1; i >= 0; i--) {

                cur\_track = left[i];

                seek\_sequence.push\_back(cur\_track);

                distance = abs(cur\_track - head);

                seek\_count += distance;

                head = cur\_track;

            }

            direction = "right";

        }

        else if (direction == "right") {

            for (int i = 0; i < right.size(); i++) {

                cur\_track = right[i];

                seek\_sequence.push\_back(cur\_track);

                distance = abs(cur\_track - head);

                seek\_count += distance;

                head = cur\_track;

            }

            direction = "left";

        }

    }

    cout << "Total number of seek operations = "

         << seek\_count << endl;

    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < seek\_sequence.size(); i++) {

        cout << seek\_sequence[i] << endl;

    }

}

int main()

{

    int arr[size] = { 176, 79, 34, 60,

                      92, 11, 41, 114 };

    int head = 50;

    string direction = "left";

    SCAN(arr, head, direction);

    return 0;

}

**Output:**

**Total number of seek operations = 226**

**Seek Sequence is**

**41**

**34**

**11**

**0**

**60**

**79**

**92**

**114**

**176**

**(3) SSTF**

#include <bits/stdc++.h>

using namespace std;

void calculatedifference(int request[], int head,

                         int diff[][2], int n)

{

    for(int i = 0; i < n; i++)

    {

        diff[i][0] = abs(head - request[i]);

    }

}

int findMIN(int diff[][2], int n)

{

    int index = -1;

    int minimum = 1e9;

    for(int i = 0; i < n; i++)

    {

        if (!diff[i][1] && minimum > diff[i][0])

        {

            minimum = diff[i][0];

            index = i;

        }

    }

    return index;

}

void shortestSeekTimeFirst(int request[],

                           int head, int n)

{

    if (n == 0)

    {

        return;

    }

    int diff[n][2] = { { 0, 0 } };

    int seekcount = 0;

    int seeksequence[n + 1] = {0};

    for(int i = 0; i < n; i++)

    {

        seeksequence[i] = head;

        calculatedifference(request, head, diff, n);

        int index = findMIN(diff, n);

        diff[index][1] = 1;

        seekcount += diff[index][0];

        head = request[index];

    }

    seeksequence[n] = head;

    cout << "Total number of seek operations = "

         << seekcount << endl;

    cout << "Seek sequence is : " << "\n";

    for(int i = 0; i <= n; i++)

    {

        cout << seeksequence[i] << "\n";

    }

}

int main()

{

    int n = 8;

    int proc[n] = { 176, 79, 34, 60, 92, 11, 41, 114 };

    shortestSeekTimeFirst(proc, 50, n);

    return 0;

}

**Output:**

**Total number of seek operations = 204**

**Seek Sequence is**

**50**

**41**

**34**

**11**

**60**

**79**

**92**

**114**

**176**

**(4) C-SCAN**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

int disk\_size = 200;

void CSCAN(int arr[], int head)

{

    int seek\_count = 0;

    int distance, cur\_track;

    vector<int> left, right;

    vector<int> seek\_sequence;

    left.push\_back(0);

    right.push\_back(disk\_size - 1);

    for (int i = 0; i < size; i++) {

        if (arr[i] < head)

            left.push\_back(arr[i]);

        if (arr[i] > head)

            right.push\_back(arr[i]);

    }

    sort(left.begin(), left.end());

    sort(right.begin(), right.end());

    for (int i = 0; i < right.size(); i++) {

        cur\_track = right[i];

        seek\_sequence.push\_back(cur\_track);

        distance = abs(cur\_track - head);

        seek\_count += distance;

        head = cur\_track;

    }

    head = 0;

    seek\_count += (disk\_size - 1);

    for (int i = 0; i < left.size(); i++) {

        cur\_track = left[i];

        seek\_sequence.push\_back(cur\_track);

        distance = abs(cur\_track - head);

        seek\_count += distance;

        head = cur\_track;

    }

    cout << "Total number of seek operations = "

         << seek\_count << endl;

    cout << "Seek Sequence is" << endl;

    for (int i = 0; i < seek\_sequence.size(); i++) {

        cout << seek\_sequence[i] << endl;

    }

}

int main()

{

    int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

    int head = 50;

    cout << "Initial position of head: " << head << endl;

    CSCAN(arr, head);

    return 0;

}

**Output:**

**Initial position of head: 50**

**Total number of seek operations = 389**

**Seek Sequence is**

**60**

**79**

**92**

**114**

**176**

**199**

**0**

**11**

**34**

**41**

# **EXPERIMENT - 14**

**Aim**: Design a lexical analyzer for the given language. The lexical analyzer should ignore redundant spaces, tabs and newlines, comments etc.

**Theory :** Lexical analysis is the first phase of a compiler. It takes modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions.

In programming language, keywords, constants, identifiers, strings, numbers, operators and punctuation symbols can be considered as tokens.

**Procedure :**

We make use of the following two functions in the process. look up() – it takes string as an argument and checks its presence in the symbol table. If the string is found then returns the address else it returns NULL. insert() – it takes string as its argument and the same is inserted into the symbol table and the corresponding address is returned.

1. Start

2. Declare an array of characters, an input file to store the input;

3. Read the character from the input file and put it into a character type of variable, say ‘c’.

4. If ‘c’ is blank then do nothing.

5. If ‘c’ is a new line character line=line+1.

6. If ‘c’ is digit, set token Val, the value assigned for a digit and return the ‘NUMBER’.

7. If ‘c’ is a proper token then assign the token value.

8. Print the complete table with Token entered by the user, Associated token value. 9. Stop

**Source Code :**

#include<iostream>

#include<fstream>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

using namespace std;

int isKeyword(char buffer[]){

Char keywords[32][10] = {"auto","break","case","char","const","continue","default",

"do","double","else","enum","extern","float","for","goto",

"if","int","long","register","return","short","signed",

"sizeof","static","struct","switch","typedef","union",

"unsigned","void","volatile","while"};

int i, flag = 0;

for(i = 0; i < 32; ++i){

if(strcmp(keywords[i], buffer) == 0){

flag = 1;

break;

}

}

return flag;

}

int main(){

char ch, buffer[15], operators[] = "+-\*/%=";

ifstream fin("program.txt");

int i,j=0;

if(!fin.is\_open()){

cout<<"error while opening the file\n";

exit(0);

}

while(!fin.eof()){

ch = fin.get();

for(i = 0; i < 6; ++i){

if(ch == operators[i])

cout<<ch<<" is operator\n";

}

if(isalnum(ch)){

buffer[j++] = ch;

}

else if((ch == ' ' || ch == '\n') && (j != 0)){

buffer[j] = '\0';

j = 0;

if(isKeyword(buffer) == 1)

cout<<buffer<<" is keyword\n";

else

cout<<buffer<<" is indentifier\n";

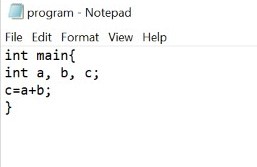
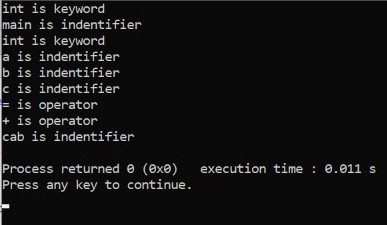
}

}

fin.close();

return 0;

}

**Output :**

# **EXPERIMENT - 15**

**Aim:** Implement the lexical analyzer using JLex, flex or other lexical Analyzer generating tools.

**Theory:**

**Lexical analyser-** First phase of compiler (also known as scanner); it converts the high-level language code into a sequence of tokens and can be implemented with the Deterministic Finite Automata. The output is a series of tokens sent to the syntax analyser.

**Lexical Analyser generating tools-** These are the tools that aid the construction of lexical analyser. Some of these are as follows:

JFlex: It is a lexical analyser generator for Java written in Java.

Flex (fast lexical analyser): It is a lexical analyser generating tool written in C. It is more flexible and produce faster codes.

**Procedure**:

Input: LEX specification file for tokens

Output: Produces source code for lexical analyser with name lex.yy.c and displays token in the input file.

1. Start
2. Open a new file in text editor.
3. Create a Lex specification file that accepts- keywords, identifiers, constants, operators, and relational operators in following format:
4. %{

Definition of constant /header files

%}

1. Regular Expressions

%%

Transition rules

%%

1. Auxiliary Procedure (main( ) function)
2. Save file with extension .l
3. Call lex tool on the terminal e.g. [root@localhost]# lex mylex.l.

This lex tool will convert “.l” file into “.c” language code file i.e., lex.yy.c

1. Compile this file using C/C++ compiler
2. Run file
3. After processing the output will be displayed on the file.
4. Stop

**Source Code:**

DIGIT [0-9]

LETTER [A-Z a-z]

DELIM [ \t\n]

WS { DELIM }+

ID {(LETTER)[LETTER/DIGIT]}+

INTEGER {DIGIT}+

%%

{WS} { printf("\n WS special characters \n"); }

{ID} { printf("\n Identifiers \n"); }

{DIGIT} {printf("\n Intgers\n"); }

if { printf("\n Keywords\n"); }

else { printf("\n keywords\n"); }

">" { printf("\n Relational Operators\n"); }

"<" { printf("\n Relational Operators \n"); }

"<=" { printf("\n Relational Operators \n"); }

"=>" { printf("\n Relational Operators \n"); }

"=" { printf("\n Relational Operators \n"); }

"!=" { printf("\n Logical Operators \n"); }

"&&" { printf("\n Logical Operators \n"); }

"||" { printf("\n Logical Operators \n"); }

"!" { printf("\n Logical Operators \n"); }

"+" { printf("\n Arthmetic Operator\n"); }

"-" { printf("\n Arthmetic Operator\n"); }

"\*" { printf("\n Arthmetic Operator\n"); }

"/" { printf("\n Arthmetic Operator\n"); }

"%" { printf("\n Arthmetic Operator\n"); }

%%

int yywrap(){ }

int main()

{

Printf(‘’ Enter the text : ’’)

yylex();

return 0 ;

}

**Output:**

**[root@localhost]# lex lexprog.l**

**[root@localhost]# cc lex.yy.c**

**[root@localhost]# ./a.out lexprog**

**Test Cases:**

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| **if** | **Keyword** |
| **\*** | **Arithmetic operator** |
| **<=** | **Relational operator** |
| **&&** | **Logical Operator** |

# **EXPERIMENT - 16**

**Aim:** Write a LEX program to convert substring abc to ABC from the given input string

**Theory:** Lex is a computer program that generates lexical analyzers ("scanners" or "lexers"). Lex is commonly used with the yacc parser generator. Lex, originally written by Mike Lesk and Eric Schmidt and described in 1975, is the standard lexical analyzer generator on many Unix systems, and an equivalent tool is specified as part of the POSIX standard.

Lex reads an input stream specifying the lexical analyzer and writes source code which implements the lexical analyzer in the C programming language.

In addition to C, some old versions of Lex could also generate a lexer in Ratfor.

Some major points to keep in mind about LEX is:

* Lex is a program that generates lexical analyzer. It is used with YACC parser generator.
* The lexical analyzer is a program that transforms an input stream into a sequence of tokens.
* It reads the input stream and produces the source code as output through implementing the lexical analyzer in the C program.

**Source Code:**

%{

#include<stdio.h>

#include<string.h>

int i;

%}

%%

[a-z A-Z]\* {

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

{

if((yytext[i]=='a')&&(yytext[i+1]=='b')&&(yytext[i+2]=='c'))

{

yytext[i]='A';

yytext[i+1]='B';

yytext[i+2]='C';

}

}

printf("%s",yytext);

}

[\t]\* return;

.\* {ECHO;}

\n {printf("%s",yytext);}

%%

main()

{

yylex();

}

int yywrap(

{

return 1;

}

**Output:**

**[CSE@localhost ~]$ lex lex1.l**

**[CSE@localhost ~]$ cc lex.yy.c**

**[CSE@localhost ~]$. /a.out**

**abc**

**ABC**

# **EXPERIMENT - 17**

**Aim:** To design and implement an LALR bottom-up parser for checking the syntax of the

statements in the language.

**Theory:** LALR Parser is a lookahead LR parser. It is the most powerful parser which can

handle large classes of grammar. LALR works similarly to CLR. The only difference is, it

combines the similar states of the CLR parsing table into one single state.

**LALR Bottom-up Parser:**

<parser.l>

%{

#include<stdio.h>

#include "y.tab.h"

%}

%%

[0-9]+ {yylval.dval=atof(yytext);

return DIGIT;

}

\n|. return yytext[0];

%%

<parser.y>

%{

/\*This YACC specification file generates the LALR parser for the program

considered in experiment 4.\*/

#include<stdio.h>

%}

%union

{

double dval;

}

%token <dval> DIGIT

%type <dval> expr

%type <dval> term

%type <dval> factor

%%

line: expr '\n' {

;

printf("%g\n",$1);

}

expr: expr '+' term {$$=$1 + $3 ;}

| term

;

term: term '\*' factor {$$=$1 \* $3 ;}

| factor

;

factor: '(' expr ')' {$$=$2 ;}

| DIGIT

;

%%

int main()

{

yyparse();

}

yyerror(char \*s)

{

printf("%s",s);

}

**Output:**

**$lex parser.l**

**$yacc –d parser.y**

**$cc lex.yy.c y.tab.c –ll –lm**

**$./a.out**

**2+3**

**5.0000**

# **EXPERIMENT - 18**

**Aim:** Convert the BNF rules into YACC form and Write code to generate Abstract Syntax Tree.

**Algorithm:**

1. Start

2. Include the header file.

3. In int code.l, declare the variable lie no as integer and assign it to be equal to ‘1’.

4. Start the int code.l with declarative section.

5. In translation rules section define keywords, data types and integer along with their actions.

6. Start the main block. In main block check the statement

7.

1. declarative
2. assignment
3. conditional
4. if and else
5. while assignment

8. Perform the actions of that particular block.

9. In main program declare the parameters arg c as int end \*argv[] as char.

10. In main program open file in read mode.

11. Print the output in a file.

12. End

**Source Code:**

**Lex<Bnf.L>**

%{

#include"y.tab.h"

#include<stdio.h>

#include<string.h>

int LineNo=1;

%}

identifier [a-zA-Z][\_a-zA-Z0-9]\*

number [0-9]+|([0-9]\*\.[0-9]+)

%%

main\(\) return MAIN;

if return IF;

else return ELSE;

while return WHILE;

int |

char |

float return TYPE;

{identifier} {strcpy(yylval.var,yytext);

return VAR;}

{number} {strcpy(yylval.var,yytext);

return NUM;}

\< |

\> |

\>= |

\<= |

== {strcpy(yylval.var,yytext);

return RELOP;}

[ \t] ;

\n LineNo++;

. return yytext[0];

%%

**Yacc <Bnf.Y>**

%{

#include<string.h>

#include<stdio.h>

struct quad

{

char op[5];

char arg1[10];

char arg2[10];

char result[10];

}QUAD[30];

struct stack{

int items[100];

int top;

}stk;

int Index=0,tIndex=0,StNo,Ind,tInd;

extern int LineNo;

%}

%union

{

char var[10];

}

%token <var> NUM VAR RELOP

%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP

%left '-' '+'

%left '\*' '/'

%%

**PROGRAM: MAIN BLOCK**

;

BLOCK: '{' CODE '}'

;

CODE: BLOCK

| STATEMENT CODE

| STATEMENT

;

STATEMENT: DESCT ';'

| ASSIGNMENT ';'

| CONDST

| WHILEST

;

DESCT: TYPE VARLIST

;

VARLIST: VAR ',' VARLIST

| VAR

;

ASSIGNMENT: VAR '=' EXPR{

strcpy(QUAD[Index].op,"=");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,$1);

strcpy($$,QUAD[Index++].result);

};

EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}

| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}

| EXPR '\*' EXPR {AddQuadruple("\*",$1,$3,$$);}

| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}

| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}

| '(' EXPR ')' {strcpy($$,$2);}

| VAR

| NUM

;

CONDST: IFST{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

}

| IFST ELSEST

;

IFST: IF '(' CONDITION ')' {

strcpy(QUAD[Index].op,"==");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"FALSE");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

}

BLOCK {

strcpy(QUAD[Index].op,"GOTO");

strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

};

ELSEST: ELSE{

tInd=pop();

Ind=pop();

push(tInd);

sprintf(QUAD[Ind].result,"%d",Index);

}

BLOCK{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

};

CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);

StNo=Index-1;

}

| VAR

| NUM

;

WHILEST: WHILELOOP{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",StNo);

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

};

WHILELOOP: WHILE '(' CONDITION ')' {

strcpy(QUAD[Index].op,"==");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"FALSE");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

}

BLOCK {

strcpy(QUAD[Index].op,"GOTO");

strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

};

%%

extern FILE \*yyin;

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

FILE \*fp;

int i;

if(argc>1){

fp=fopen(argv[1],"r");

if(!fp){

printf("\n File not found");

exit(0);

}

yyin=fp;

}

yyparse();

printf("\n\n\t\t ----------------------------\n\t\t Pos Operator Arg1 Arg2 Result\n\t\t--------------------");

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

printf("\n\t\t %d\t %s\t %s\t %s\t %s", i, QUAD[i].op, QUAD[i].arg1, QUAD[i].arg2, QUAD[i].result);

}

printf("\n\t\t -----------------------");

printf("\n\n");

return 0;

}

void push(int data){

stk.top++;

if(stk.top==100){

printf("\n Stack overflow\n");

exit(0);

}

stk.items[stk.top]=data;

}

int pop(){

int data;

if(stk.top==-1){

printf("\n Stack underflow\n");

exit(0);

}

data=stk.items[stk.top--];

return data;

}

void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10]){

strcpy(QUAD[Index].op,op);

strcpy(QUAD[Index].arg1,arg1);

strcpy(QUAD[Index].arg2,arg2);

sprintf(QUAD[Index].result,"t%d",tIndex++);

strcpy(result,QUAD[Index++].result);

}

yyerror(){

printf("\n Error on line no:%d",LineNo);

}

**Input:**

main()

{

int a,b,c;

if(a<b){

a=a+b;

}

while(a<b){

a=a+b;

}

if(a<=b){

c=a-b;

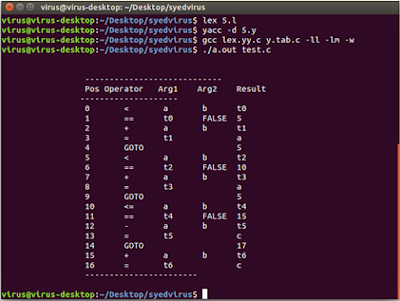
}

else{

c=a+b;

}

}

**Output:**

# **EXPERIMENT - 19**

**Aim:** To write a C Program to Generate Machine Code from the Abstract Syntax Tree using the specified machine instruction formats.

**THEORY:** Abstract Syntax Tree is a kind of tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code.

Machine code is a computer programming language comprising hexadecimal or binary instructions that computers are able to respond to directly. Machine code is written in a machine language. Therefore, a machine, i.e., a computer, can execute it without any translation or conversion.

**Source Code:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int label[20];

int no=0;

int main()

{

FILE \*fp1,\*fp2;

char fname[10],op[10],ch;

char operand1[8],operand2[8],result[8];

int i=0,j=0;

printf("\n Enter filename of the intermediate code");

scanf("%s",&fname);

fp1=fopen(fname,"r");

fp2=fopen("target.txt","w");

if(fp1==NULL || fp2==NULL)

{

printf("\n Error opening the file");

exit(0);

}

while(!feof(fp1))

{

fprintf(fp2,"\n");

fscanf(fp1,"%s",op);

i++;

if(check\_label(i))

fprintf(fp2,"\nlabel#%d",i);

if(strcmp(op,"print")==0)

{

fscanf(fp1,"%s",result);

fprintf(fp2,"\n\t OUT %s",result);

}

if(strcmp(op,"goto")==0)

{

fscanf(fp1,"%s %s",operand1,operand2);

fprintf(fp2,"\n\t JMP %s,label#%s",operand1,operand2);

label[no++]=atoi(operand2);

}

if(strcmp(op,"[]=")==0)

{

fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n\t STORE %s[%s],%s",operand1,operand2,result);

}

if(strcmp(op,"uminus")==0)

{

fscanf(fp1,"%s %s",operand1,result);

fprintf(fp2,"\n\t LOAD -%s,R1",operand1);

fprintf(fp2,"\n\t STORE R1,%s",result);

}

switch(op[0])

{

case '\*': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t MUL R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '+': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t ADD R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '-': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t SUB R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '/': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t DIV R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '%': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n \t LOAD %s,R1",operand2);

fprintf(fp2,"\n \t DIV R1,R0");

fprintf(fp2,"\n \t STORE R0,%s",result);

break;

case '=': fscanf(fp1,"%s %s",operand1,result);

fprintf(fp2,"\n\t STORE %s %s",operand1,result);

break;

case '>': j++;

fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n\t JGT %s,label#%s",operand2,result);

label[no++]=atoi(result);

break;

case '<': fscanf(fp1,"%s %s %s",operand1,operand2,result);

fprintf(fp2,"\n \t LOAD %s,R0",operand1);

fprintf(fp2,"\n\t JLT %s, label#%d",operand2,result);

label[no++]=atoi(result);

break;

}

}

fclose(fp2); fclose(fp1);

fp2=fopen("target.txt","r");

if(fp2==NULL)

{

printf("Error opening the file\n");

exit(0);

}

do

{

ch=fgetc(fp2);

printf("%c",ch);

}while(ch!=EOF);

fclose(fp1);

return 0;

}

int check\_label(int k)

{

int i;

for(i=0;i<no;i++)

{

if(k==label[i])

return 1;

}

return 0;

}

**Input :**

$ vi int.txt

= t1 2

[]= a 0 1

[]= a 1 2

[]= a 2 3

\*t1 6 t2

+ a[2] t2 t3

- a[2] t1 t2

/ t3 t2 t2

uminus t2 t2

print t2

goto t2 t3

= t3 99

uminus 25 t2

\* t2 t3 t3

uminus t1 t1 + t1 t3 t4

print t4

**Output :**

**Enter filename of the intermediate code: int.txt**

**STORE t1, 2**

**STORE a[0], 1**

**STORE a[1], 2**

**STORE a[2], 3**

**LOAD t1, R0**

**LOAD 6, R1**

**ADD R1, R0**

**STORE R0, t3**

**LOAD a[2], R0**

**LOAD t2, R1**

**ADD R1,R0**

**STORE R0,t3**

**LOAD a[t2],R0**

**LOAD t1,R1**

**SUB R1,R0**

**STORE R0,t2**

**LOAD t3,R0**

**LOAD t2,R1**

**DIV R1,R0**

**STORE R0,t2**

**LOAD t2,R1**

**STORE R1,t2**

**LOAD t2,R0**

**JGT 5, label#11**

**Label#11: OUT t2**

**JMP t2, label#13**

**Label#13: STORE t3, 99**

**LOAD 25, R1**

**STORE R1,t2**

**LOAD t2,R0**

**LOAD t3,R1**

**MUL R1,R0**

**STORE R0,t3**

**LOAD t1,R1**

**STORE R1,t1**

**LOAD t1,R0**

**LOAD t3,R1**

**ADD R1,R0**

**STORE R0,t4**

**OUT t4**