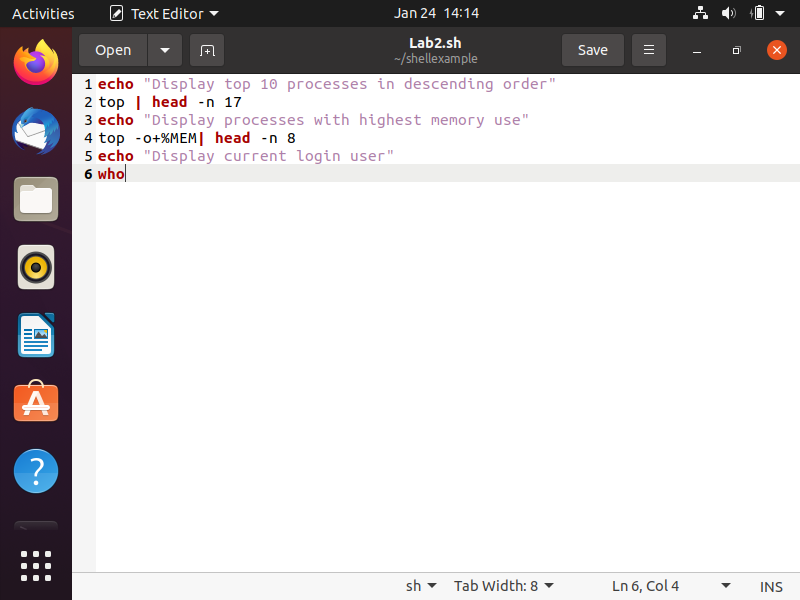
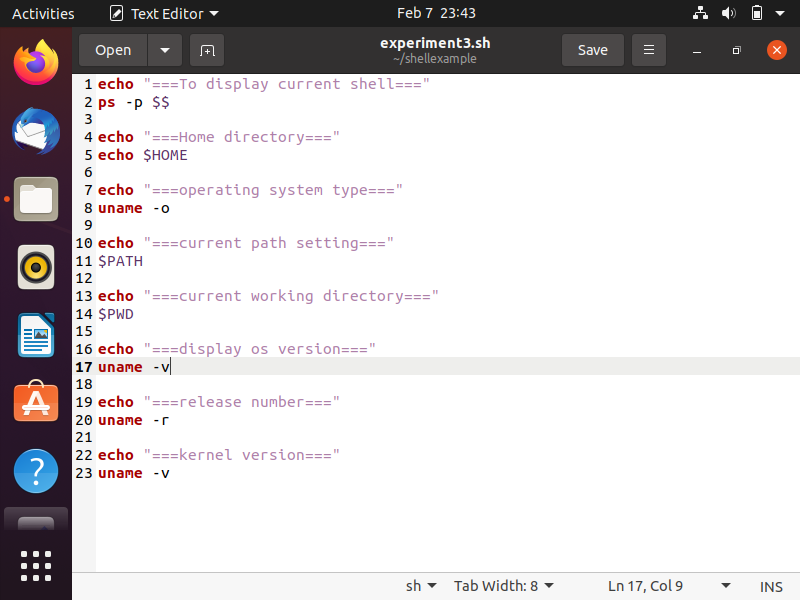
Exp 1- linux commands

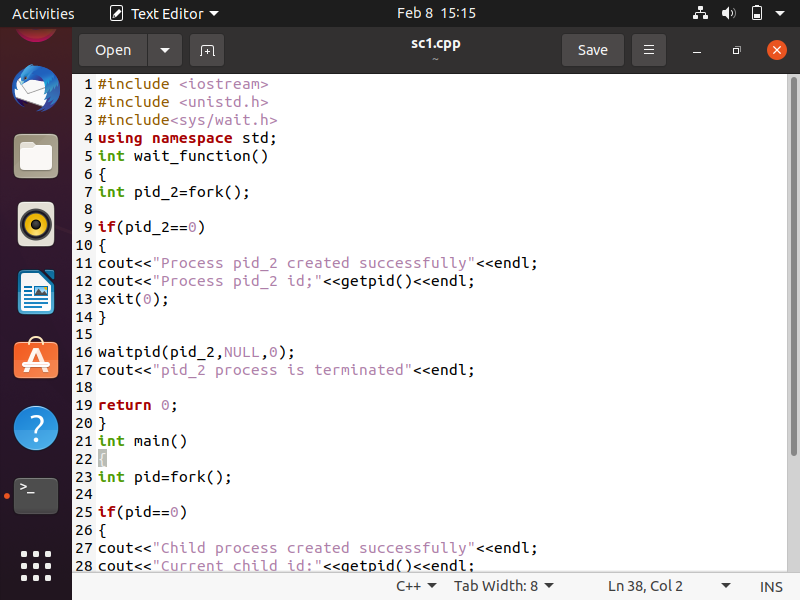
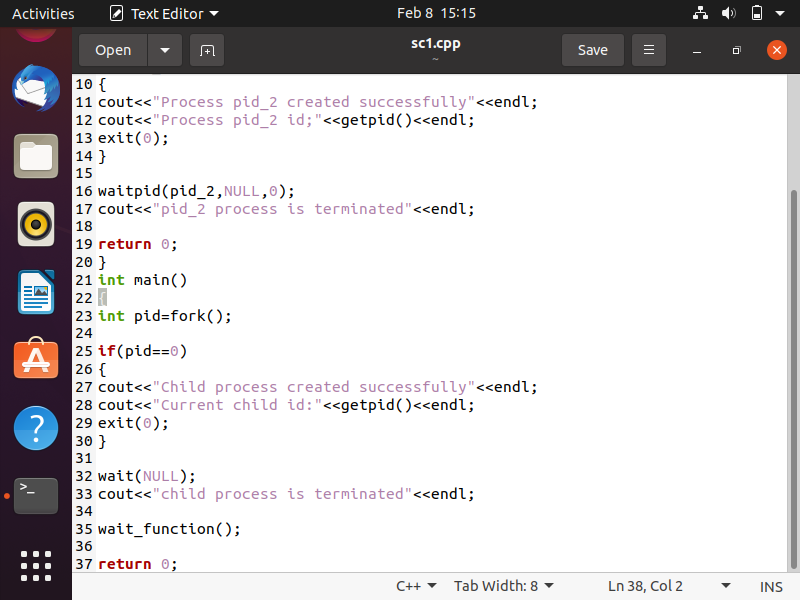
Exp2- display top processes



Exp3- shell script(kernel version)



Exp4- getpid waitpid



Exp5-

#include<iostream>

#include<unistd.h>

#include<sys/wait.h>

#include <fcntl.h>

#define MAX 100

using namespace std;

char Buffer[MAX];

void printBuffer(int len)

{

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

{

printf("%c",Buffer[i]);

}

}

void readFile(int file\_descriptor)

{

cout<<"Reading a file."<<endl;

int statusmsg = read(file\_descriptor,Buffer,MAX);

if(statusmsg > 0)

printBuffer(MAX);

else if (statusmsg == 0)

printf("\nFile is empty");

else

printf("\nError in reading file");

}

void writeFile(int file\_descriptor)

{

cout<<"Writing to file."<<endl;

int statusmsg = write(file\_descriptor,Buffer,MAX);

if(statusmsg > 0)

printf("File written succesfully");

else if (statusmsg == 0)

printf("File is empty");

else

printf("Error in writing file");

}

void process\_function()

{

cout<<"\nValue of pid: "<<getpid()<<endl;

cout<<"Value of uid: "<<getuid()<<endl;

cout<<"Value of gid: "<<getgid()<<endl;

cout<<"Value of egid: "<<getegid()<<endl;

cout<<"Value of euid: "<<geteuid()<<endl;

}

int main()

{

FILE \*f1 = fopen("sampletext.txt","r+");

FILE \*f2 = fopen("sampleout.txt","w+");

int file\_descriptor\_f1 = fileno(f1);

int file\_descriptor\_f2 = fileno(f2);

/\* OPEN FILE \*/

open("sampletext.txt",O\_RDWR);

readFile(file\_descriptor\_f1);

/\* WRITE FILE \*/

writeFile(file\_descriptor\_f2);

/\* CLOSE FILE \*/

close(file\_descriptor\_f1);

close(file\_descriptor\_f2);

process\_function();

return 0;

exp6

1) FCFS

#include<stdio.h>

// Function to find the waiting time for all processes

void findWaitingTime(int processes[], int n, int bt[], int wt[])

{

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

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

wt[i] = bt[i-1] + wt[i-1] ;

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n,int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding-- bt[i] + wt[i]

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

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

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

printf("Processes Burst time Waiting time Turn around time\n");

// Calculate total waiting time and total turn around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

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

}

int s=(float)total\_wt / (float)n;

int t=(float)total\_tat / (float)n;

printf("Average waiting time = %d\n",s);

printf("Average turn around time = %d ",t);

}

int main()

{

//process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

//Burst time of all processes

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

2) SJF

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

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

{

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

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

p[i]=i+1;

}

//sorting of burst times

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

{

pos=i;

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

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

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

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nnProcess\t Burst Time\t Waiting Time\t Turnaround Time");

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

{

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

total+=tat[i];

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

}

avg\_tat=(float)total/n;

printf("\n\nAverage Waiting Time=%0.2f",avg\_wt);

printf("\nAverage Turnaround Time=%0.2f\n",avg\_tat);

}

3) Round Robin

#include<stdio.h>

int main()

{

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

printf(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP; // Assign the number of process to variable y

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

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

temp[i] = bt[i];

}

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0)

{

sum = sum + temp[i];

temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant;

sum = sum + quant;

}

if(temp[i]==0 && count==1)

{

y--;

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-

bt[i]);

wt = wt+sum-at[i]-bt[i];

tat = tat+sum-at[i];

count =0;

}

if(i==NOP-1)

{

i=0;

}

else if(at[i+1]<=sum)

{

i++;

}

else

{

i=0;

}

}

avg\_wt = wt \* 1.0/NOP;

avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt);

printf("\n Average Waiting Time: \t%f\n", avg\_tat);

return 0;

}

Exp7

#include<stdio.h>

#include<unistd.h>

#include<semaphore.h>

#include<pthread.h>

sem\_t mutexlock;

int value = 0;

void\* producer()

{

sem\_wait(&mutexlock);

printf("\nProducer allowed to enter critical section");

//CRITICAL SECTION

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

{

printf("\nvalue: %d",++value);

}

sem\_post(&mutexlock);

printf("\nProducer has set semaphore...");

}

void\* consumer()

{

sem\_wait(&mutexlock);

printf("\nConsumer allowed to enter critical section");

Operating System Experiment - 7 Prasad Jawale - 20

//CRITICAL SECTION

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

{

printf("\nvalue: %d",value--);

}

sem\_post(&mutexlock);

printf("\nConsumer has set semaphore...");

}

int main()

{

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

pthread\_t p1,p2;

pthread\_create(&p1,NULL,&producer,NULL);

pthread\_create(&p2,NULL,&consumer,NULL);

pthread\_join(p1,NULL);

pthread\_join(p2,NULL);

sem\_destroy(&mutexlock);

return 0;

}

**Gcc producer.c -o out**

**./out**

**Exp8 bankers algo**

**n = 5 # Number of processes**

**m = 3 # Number of resources**

**# Allocation Matrix**

**alloc = [[0, 1, 0 ],[ 2, 0, 0 ],**

**[3, 0, 2 ],[2, 1, 1] ,[ 0, 0, 2]]**

**# MAX Matrix**

**max = [[7, 5, 3 ],[3, 2, 2 ],**

**[ 9, 0, 2 ],[2, 2, 2],[4, 3, 3]]**

**avail = [3, 3, 2] # Available Resources**

**f = [0]\*n**

**ans = [0]\*n**

**ind = 0**

**for k in range(n):**

**f[k] = 0**

**need = [[ 0 for i in range(m)]for i in range(n)]**

**for i in range(n):**

**for j in range(m):**

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

**y = 0**

**for k in range(5):**

**for i in range(n):**

**if (f[i] == 0):**

**flag = 0**

**for j in range(m):**

**if (need[i][j] > avail[j]):**

**flag = 1**

**break**

**if (flag == 0):**

**ans[ind] = i**

**ind += 1**

**for y in range(m):**

**avail[y] += alloc[i][y]**

**f[i] = 1**

**print("Following is the SAFE Sequence")**

**for i in range(n - 1):**

**print(" P", ans[i], " ->", sep="", end="")**

**print(" P", ans[n - 1], sep="")**

**exp9**

**First fit**

**#include<stdio.h>**

**void firstfit(int blockSize[],int m,int processorSize[],int n){**

**int allocation[n];**

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

**allocation[i]=-1;**

**}**

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

**for(int j=0; j<m; j++){**

**if(blockSize[j] >= processorSize[i]){**

**allocation[i] = j;**

**blockSize[j] -= processorSize[i];**

**break;**

**}**

**}**

**}**

**printf("Process No. Process Size Block no.\n");**

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

**printf("%d %d ",i + 1,processorSize[i]);**

**if (allocation[i] != -1){**

**printf("%d\n",allocation[i] + 1);**

**}**

**else{**

**printf("Not Allocated");**

**}**

**}**

**}**

**int main(){**

**int b\_size;**

**int p\_size;**

**printf("Enter number of blocks");**

**scanf("%d",&b\_size);**

**printf("Enter number of processes");**

**scanf("%d",&p\_size);**

**int blockSize[b\_size];**

**printf("Enter all the block size\n");**

**for(int i=0; i<b\_size; i++){**

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

**}**

**int processSize[p\_size];**

**printf("Enter all the process size\n");**

**for(int i=0; i<p\_size; i++){**

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

**}**

**firstfit(blockSize, b\_size, processSize, p\_size);**

**return 0;**

**}**

**Worst fit**

**#include<stdio.h>**

**void worstfit(int blockSize[],int m,int processorSize[],int n){**

**int allocation[n];**

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

**allocation[i]=-1;**

**}**

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

**int worst\_id=-1;**

**for(int j=0; j<m; j++){**

**if(blockSize[j] >= processorSize[i]){**

**if(worst\_id==-1){**

**worst\_id=j;**

**}**

**else if(blockSize[worst\_id] < blockSize[j]){**

**worst\_id=j;**

**}**

**}**

**}**

**if(worst\_id!=-1){**

**allocation[i] = worst\_id;**

**blockSize[worst\_id] -= processorSize[i];**

**}**

**}**

**printf("Process No. Process Size Block no.\n");**

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

**printf("%d %d ",i + 1,processorSize[i]);**

**if (allocation[i] != -1){**

**printf("%d\n",allocation[i] + 1);**

**}**

**else{**

**printf("Not Allocated");**

**}**

**}**

**}**

**int main(){**

**int b\_size;**

**int p\_size;**

**printf("Enter number of blocks");**

**scanf("%d",&b\_size);**

**printf("Enter number of processes");**

**scanf("%d",&p\_size);**

**int blockSize[b\_size];**

**printf("Enter all the block size\n");**

**for(int i=0; i<b\_size; i++){**

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

**}**

**int processSize[p\_size];**

**printf("Enter all the process size\n");**

**for(int i=0; i<p\_size; i++){**

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

**}**

**worstfit(blockSize, b\_size, processSize, p\_size);**

**return 0;**

**}**

**Exp11**

**FIFO page replacement :**

**#include<stdio.h>**

**int main()**

**{**

**int i,j,n,a[50],frame[10],no,k,avail,count=0;**

**printf("\n ENTER THE NUMBER OF PAGES:\n");**

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

**printf("\n ENTER THE PAGE NUMBER :\n");**

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

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

**printf("\n ENTER THE NUMBER OF FRAMES :");**

**scanf("%d",&no);**

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

**frame[i]= -1;**

**j=0;**

**printf("\tref string\t page frames\n");**

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

**{**

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

**avail=0;**

**for(k=0;k<no;k++)**

**if(frame[k]==a[i])**

**avail=1;**

**if (avail==0)**

**{**

**frame[j]=a[i];**

**j=(j+1)%no;**

**count++;**

**for(k=0;k<no;k++)**

**printf("%d\t",frame[k]);**

**}**

**printf("\n");**

**}**

**printf("Page Fault Is %d",count);**

**return 0;**

**}**

**OPTIMAL page replacement:**

**#include <stdio.h>**

**#include<stdbool.h>**

**int rs[100],count =0,f[10];**

**bool dataavi(int n)**

**{**

**int i;**

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

**{**

**if(rs[count]==f[i])**

**return 1;**

**}**

**return 0;**

**}**

**int check(int n,int m)**

**{**

**int buf[10],i,j=0;**

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

**{**

**int x=0;**

**for(i=count+1;i<m;i++)**

**{**

**if(f[j]==rs[i])**

**{**

**buf[j]=i;**

**i=m+1;**

**x=1;**

**}**

**}**

**if(x==0)**

**return j;**

**}**

**int max =buf[0],maxi=0;**

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

**{**

**if(max<=buf[i+1])**

**{**

**max=buf[i+1];**

**maxi=i+1;**

**}**

**}**

**return maxi;**

**}**

**void optimal(int n,int m)**

**{**

**int fs=0,i=0,in=0,kom=0;**

**while(count<m)**

**{**

**if(in<n)**

**{**

**if(dataavi(n)&&in>0)**

**{**

**count++;**

**kom=0;**

**}**

**else**

**{**

**f[in++]=rs[count];**

**fs++;**

**count++;**

**kom=1;**

**}**

**}**

**else**

**{**

**if(dataavi(n))**

**{**

**count++;**

**kom=0;**

**}**

**else**

**{**

**int j=check(n,m);**

**f[j]=rs[count++];**

**fs++;**

**kom=1;**

**}**

**}**

**if(kom==1)**

**printf(" Page Fault :");**

**else**

**printf(" :");**

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

**printf(" %d",f[i]);**

**printf("\n");**

**}**

**printf("\n\npage fault =%d",fs);**

**}**

**int main()**

**{**

**int n ,m,i;**

**printf("ENTER THE NO OF FRAME AND REFERENCE STREAM\n");**

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

**printf ("ENTER THE REFERENCE STREAM\n");**

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

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

**optimal(n,m);**

**return 0;**

**}**

**LRU page replacement policy:**

**#include<stdio.h>**

**main()**

**{**

**int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];**

**printf("Enter no of pages:");**

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

**printf("Enter the reference string:");**

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

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

**printf("Enter no of frames:");**

**scanf("%d",&f);**

**q[k]=p[k];**

**printf("\n\t%d\n",q[k]);**

**c++;**

**k++;**

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

**{**

**c1=0;**

**for(j=0;j<f;j++)**

**{**

**if(p[i]!=q[j])**

**c1++;**

**}**

**if(c1==f)**

**{**

**c++;**

**if(k<f)**

**{**

**q[k]=p[i];**

**k++;**

**for(j=0;j<k;j++)**

**printf("\t%d",q[j]);**

**printf("\n");**

**}**

**else**

**{**

**for(r=0;r<f;r++)**

**{**

**c2[r]=0;**

**for(j=i-1;j<n;j--)**

**{**

**if(q[r]!=p[j])**

**c2[r]++;**

**else**

**break;**

**}**

**}**

**for(r=0;r<f;r++)**

**b[r]=c2[r];**

**for(r=0;r<f;r++)**

**{**

**for(j=r;j<f;j++)**

**{**

**if(b[r]<b[j])**

**{**

**t=b[r];**

**b[r]=b[j];**

**b[j]=t;**

**}**

**}**

**}**

**for(r=0;r<f;r++)**

**{**

**if(c2[r]==b[0])**

**q[r]=p[i];**

**printf("\t%d",q[r]);**

**}**

**printf("\n");**

**}**

**}**

**}**

**printf("\nThe no of page faults is %d",c);**

**}**