CSA0477-OPERATING SYSTEM

1. Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
pid_t p;
printf("before fork\n");
p=fork();
if(p==0)
{
printf("I am  having student id %d\n",getpid());
printf("My parent's id is %d\n",getppid());
}
else{
printf("My student's id is %d\n",p);
printf("I am having parent id %d\n",getpid());
}
printf("Common\n");
}
```

2. Identify the system calls to copy the content of one file to another and illustrate the same using a C program.

```
#include<stdio.h>


int main()
{


  int cnt,j,n,t,remain,flag=0,tq;
  int wt=0,tat=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
```

```
scanf("%d",&n);
remain=n;
for(cnt=0;cnt<n;cnt++)
{
  printf("Enter Arrival Time and Burst Time for Process Process Number %d :",cnt+1);
  scanf("%d",&at[cnt]);
  scanf("%d",&bt[cnt]);
  rt[cnt]=bt[cnt];
}
printf("Enter Time Quantum:\t");
scanf("%d",&tq);
printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
for(t=0,cnt=0;remain!=0;)
{
  if(rt[cnt]<=tq && rt[cnt]>0)
  {
    t+=rt[cnt];
    rt[cnt]=0;
    flag=1;
  }
  else if(rt[cnt]>0)
  {
    rt[cnt]-=tq;
    t+=tq;
  }
  if(rt[cnt]==0 && flag==1)
  {
    remain--;
    printf("P[%d]\t|\t%d\t|\t%d\n",cnt+1,t-at[cnt],t-at[cnt]-bt[cnt]);
    wt+=t-at[cnt]-bt[cnt];
    tat+=t-at[cnt];
```

```
      flag=0;
    }
    if(cnt==n-1)
      cnt=0;
    else if(at[cnt+1]<=t)
      cnt++;
    else
      cnt=0;
  }
  printf("\nAverage Waiting Time= %f\n",wt*1.0/n);
  printf("Avg Turnaround Time = %f",tat*1.0/n);


  return 0;
}
```

3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations. a. All processes are activated at time 0. b. Assume that no process waits on I/O devices.

```
#include<stdio.h>
void findWaitingTime(int processes[], int n,int bt[], int wt[])
{
    wt[0] = 0;
    for (int i = 1; i < n ; i++ )
            wt[i] = bt[i-1] + wt[i-1] ;
}
void findTurnAroundTime( int processes[], int n,int bt[], int wt[], int tat[])
{
    for (int i = 0; i < n ; i++)
            tat[i] = bt[i] + wt[i];
}
void findavgTime( int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(processes, n, bt, wt);
    findTurnAroundTime(processes, n, bt, wt, tat);
    printf("Processes Burst time Waiting time Turn around time\n");
    for (int i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf(" %d ",(i+1));
        printf("  %d ", bt[i] );
        printf("  %d",wt[i] );
        printf("  %d\n",tat[i] );
    }
    int s=(float)total_wt / (float)n;
    int t=(float)total_tat / (float)n;
    printf("Average waiting time = %d",s);
    printf("\n");
    printf("Average turn around time = %d ",t);
}
int main()
{
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];
    int burst_time[] = {10, 5, 8};


    findavgTime(processes, n, burst_time);
    return 0;
}
```

4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.

```
#include <stdio.h>
int main()
{
        int A[100][4];
        int i, j, n, total = 0, index, temp;
        float avg_wt, avg_tat;
        printf("Enter number of process: ");
        scanf("%d", &n);
        printf("Enter Burst Time:\n");
        for (i = 0; i < n; i++) {
                printf("P%d: ", i + 1);
                scanf("%d", &A[i][1]);
                A[i][0] = i + 1;
        }
        for (i = 0; i < n; i++) {
                index = i;
                for (j = i + 1; j < n; j++)
                        if (A[j][1] < A[index][1])
                                index = j;
                temp = A[i][1];
                A[i][1] = A[index][1];
                A[index][1] = temp;

                temp = A[i][0];
                A[i][0] = A[index][0];
                A[index][0] = temp;
        }
        A[0][2] = 0;
        for (i = 1; i < n; i++) {
                A[i][2] = 0;
                for (j = 0; j < i; j++)
                        A[i][2] += A[j][1];
                total += A[i][2];
        }
        avg_wt = (float)total / n;
        total = 0;
        printf("P            BT        WT       TAT\n");
        for (i = 0; i < n; i++) {
                A[i][3] = A[i][1] + A[i][2];
                total += A[i][3];
                printf("P%d        %d        %d        %d\n", A[i][0],
                        A[i][1], A[i][2], A[i][3]);
        }
        avg_tat = (float)total / n;
        printf("Average Waiting Time= %f", avg_wt);
        printf("\nAverage Turnaround Time= %f",avg_tat);
}
```

5. Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

```
#include <stdio.h>
int main()
{
```

```
int A[100][4];


int i, j, n, total = 0, index, temp;
float avg_wt, avg_tat;
printf("Enter number of process: ");
scanf("%d", &n);
printf("Enter Burst Time:\n");

for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &A[i][1]);
        A[i][0] = i + 1;
}

for (i = 0; i < n; i++) {
        index = i;
        for (j = i + 1; j < n; j++)
                if (A[j][1] < A[index][1])
                        index = j;
        temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;

        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
}
A[0][2] = 0;

for (i = 1; i < n; i++) {
        A[i][2] = 0;
        for (j = 0; j < i; j++)
                A[i][2] += A[j][1];
        total += A[i][2];
}
avg_wt = (float)total / n;
total = 0;
printf("P        BT      WT      TAT\n");


for (i = 0; i < n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        total += A[i][3];
        printf("P%d     %d      %d      %d\n", A[i][0],
                A[i][1], A[i][2], A[i][3]);
}
avg_tat = (float)total / n;
```

```
        printf("Average Waiting Time= %f", avg_wt);
        printf("\nAverage Turnaround Time= %f",avg_tat);
}
```

6. Construct a C program to implement pre-emptive priority scheduling algorithm.

```
#include<stdio.h>
struct process
{
   int WT,AT,BT,TAT,PT;
};

struct process a[10];

int main()
{
   int n,temp[10],t,count=0,short_p;
   float total_WT=0,total_TAT=0,Avg_WT,Avg_TAT;
   printf("Enter the number of the process\n");
   scanf("%d",&n);
   printf("Enter the arrival time , burst time and priority of the process\n");
   printf("AT BT PT\n");
   for(int i=0;i<n;i++)
   {
      scanf("%d%d%d",&a[i].AT,&a[i].BT,&a[i].PT);
      temp[i]=a[i].BT;
   }
   a[9].PT=10000;

   for(t=0;count!=n;t++)
   {
      short_p=9;
      for(int i=0;i<n;i++)
      {
         if(a[short_p].PT>a[i].PT && a[i].AT<=t && a[i].BT>0)
         {
            short_p=i;
         }
      }
      a[short_p].BT=a[short_p].BT-1;
      if(a[short_p].BT==0)
      {
         count++;
         a[short_p].WT=t+1-a[short_p].AT-temp[short_p];
         a[short_p].TAT=t+1-a[short_p].AT;
         total_WT=total_WT+a[short_p].WT;
         total_TAT=total_TAT+a[short_p].TAT;

      }
   }
```

```
  Avg_WT=total_WT/n;
  Avg_TAT=total_TAT/n;
  printf("ID WT TAT\n");
  for(int i=0;i<n;i++)
  {
    printf("%d %d\t%d\n",i+1,a[i].WT,a[i].TAT);
  }

  printf("Avg waiting time of the process  is %f\n",Avg_WT);
  printf("Avg turn around time of the process is %f\n",Avg_TAT);

return 0;
}
```

7. Construct a C program to implement non-preemptive SJF algorithm.

```
#include<stdio.h>

int main() {
 int time, burst_time[10], at[10], sum_burst_time = 0, smallest, n, i;
 int sumt = 0, sumw = 0;
 printf("enter the no of processes : ");
 scanf("%d", & n);
 for (i = 0; i < n; i++) {
  printf("the arrival time for process P%d : ", i + 1);
  scanf("%d", & at[i]);
  printf("the burst time for process P%d : ", i + 1);
  scanf("%d", & burst_time[i]);
  sum_burst_time += burst_time[i];
 }
 burst_time[9] = 9999;
 for (time = 0; time < sum_burst_time;) {
  smallest = 9;
  for (i = 0; i < n; i++) {
   if (at[i] <= time && burst_time[i] > 0 && burst_time[i] < burst_time[smallest])
    smallest = i;
  }
  printf("P[%d]\t|\t%d\t|\t%d\n", smallest + 1, time + burst_time[smallest] - at[smallest],
time - at[smallest]);
  sumt += time + burst_time[smallest] - at[smallest];
  sumw += time - at[smallest];
  time += burst_time[smallest];
  burst_time[smallest] = 0;
 }
 printf("\n\n average waiting time = %f", sumw * 1.0 / n);
 printf("\n\n average turnaround time = %f", sumt * 1.0 / n);
return 0;
}
```

8. Construct a C program to simulate Round Robin scheduling algorithm with C

```
#include<stdio.h>


    int main()
    {

     int cnt,j,n,t,remain,flag=0,tq;
     int wt=0,tat=0,at[10],bt[10],rt[10];
     printf("Enter Total Process:\t ");
     scanf("%d",&n);
     remain=n;
     for(cnt=0;cnt<n;cnt++)
     {
       printf("Enter Arrival Time and Burst Time for Process Process Number %d :",cnt+1);
       scanf("%d",&at[cnt]);
       scanf("%d",&bt[cnt]);
       rt[cnt]=bt[cnt];
     }
     printf("Enter Time Quantum:\t");
     scanf("%d",&tq);
     printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
     for(t=0,cnt=0;remain!=0;)
     {
       if(rt[cnt]<=tq && rt[cnt]>0)
       {
         t+=rt[cnt];
         rt[cnt]=0;
         flag=1;
       }
       else if(rt[cnt]>0)
       {
         rt[cnt]-=tq;
         t+=tq;
       }
       if(rt[cnt]==0 && flag==1)
       {
         remain--;
         printf("P[%d]\t|\t%d\t|\t%d\n",cnt+1,t-at[cnt],t-at[cnt]-bt[cnt]);
         wt+=t-at[cnt]-bt[cnt];
         tat+=t-at[cnt];
         flag=0;
       }
       if(cnt==n-1)
         cnt=0;
       else if(at[cnt+1]<=t)
         cnt++;
       else
         cnt=0;
```

```
 }
 printf("\nAverage Waiting Time= %f\n",wt*1.0/n);
 printf("Avg Turnaround Time = %f",tat*1.0/n);
return 0;
}
```

9. Illustrate the concept of inter-process communication using shared memory with a C program.

```
#include<stdio.h>
  #include<stdlib.h>
  #include<unistd.h>
  #include<sys/shm.h>
  #include<string.h>
  int main()
  {
  int i;
  void *shared_memory;
  char buff[100];
  int shmid;
  shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
  printf("Key of shared memory is %d\n",shmid);
  shared_memory=shmat(shmid,NULL,0);
  printf("Process attached at %p\n",shared_memory);
  printf("Enter some data to write to shared memory\n");
  read(0,buff,100);
  strcpy(shared_memory,buff);
  printf("You wrote : %s\n",(char *)shared_memory);
}
```

10. Illustrate the concept of inter-process communication using message queue with a C program.

```
#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define MAX 10


// structure for message queue

struct mesg_buffer {

    long mesg_type;

    char mesg_text[100];

} message;


int main()
```

```
{
    key_t key;
    int msgid;


    key = ftok("progfile", 65);


    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;


    printf("Write Data : ");
    fgets(message.mesg_text,MAX,stdin);




    msgsnd(msgid, &message, sizeof(message), 0);




    printf("Data send is : %s \n", message.mesg_text);


    return 0;
}
```

11. Illustrate the concept of multithreading using a C program.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void *myThreadFun(void *vargp)
{
   sleep(1);
   printf("RESHMA IS NAUGHTY GIRL\n");
   return NULL;
}

int main()
{
   pthread_t thread_id;
   printf("Before Thread\n");
   pthread_create(&thread_id, NULL, myThreadFun, NULL);
```

```
  pthread_join(thread_id, NULL);
  printf("After Thread\n");
exit(0);
}
```

12. Design a C program to simulate the concept of Dining-Philosophers problem

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int* ptr;
  int n, i;
  printf("Enter number of elements:");
  scanf("%d",&n);
  printf("Entered number of elements: %d\n", n);
  ptr = (int*)malloc(n * sizeof(int));
  if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
  }
  else {
    printf("Memory successfully allocated using malloc.\n");
    for (i = 0; i < n; ++i) {
      ptr[i] = i + 1;
    }
    printf("The elements of the array are: ");
    for (i = 0; i < n; ++i) {
      printf("%d, ", ptr[i]);
    }
  }

  return 0;
}
```

13. Construct a C program for implementation the various memory allocation strategies.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int* ptr;
  int n, i;
  printf("Enter number of elements:");
  scanf("%d",&n);
  printf("Entered number of elements: %d\n", n);
  ptr = (int*)malloc(n * sizeof(int));
  if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
  }
  else {
    printf("Memory successfully allocated using malloc.\n");
    for (i = 0; i < n; ++i) {
        ptr[i] = i + 1;
    }
    printf("The elements of the array are: ");
    for (i = 0; i < n; ++i) {
        printf("%d, ", ptr[i]);
    }
  }

return 0;
}
```

14. Construct a C program to organize the file using single level directory.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
int main()
{
int nf=0,i=0,j=0,ch;
char mdname[10],fname[10][10],name[10];
printf("Enter the directory name:");
scanf("%s",mdname);
printf("Enter the number of files:");
scanf("%d",&nf);
do
{
printf("Enter file name to be created:");
scanf("%s",name);
for(i=0;i<nf;i++)
{
if(!strcmp(name,fname[i]))
break;
}
if(i==nf)
{
strcpy(fname[j++],name);
nf++;
}
else
printf("There is already %s\n",name);
printf("Do you want to enter another file(yes - 1 or no - 0):");
scanf("%d",&ch);
}
while(ch==1);
printf("Directory name is:%s\n",mdname);
printf("Files names are:");
for(i=0;i<j;i++)
printf("\n%s",fname[i]);
getch();
}
```

15. Design a C program to organize the file using two level directory structure.

```
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir[10];
int main()
{
int i,ch,dcnt,k;
char f[30], d[30];
dcnt=0;
while(1)
{
printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
printf("\n4. Search File\t\t5. Display\t6. Exit\tEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter name of directory -- ");
scanf("%s", dir[dcnt].dname);
dir[dcnt].fcnt=0;
dcnt++;
printf("Directory created");
break;
case 2: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter name of the file -- ");
scanf("%s",dir[i].fname[dir[i].fcnt]);
printf("File created");
break;
}
if(i==dcnt)
printf("Directory %s not found",d);
break;
case 3: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
```

```
printf("Enter name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is deleted ",f);
dir[i].fcnt--;
strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("File %s not found",f);
goto jmp;
}
}
printf("Directory %s not found",d);
jmp : break;
case 4: printf("\nEnter name of the directory -- ");
scanf("%s",d);
for(i=0;i<dcnt;i++)
{
if(strcmp(d,dir[i].dname)==0)
{
printf("Enter the name of the file -- ");
scanf("%s",f);
for(k=0;k<dir[i].fcnt;k++)
{
if(strcmp(f, dir[i].fname[k])==0)
{
printf("File %s is found ",f);
goto jmp1;
}
}
printf("File %s not found",f);
goto jmp1;
}
}
printf("Directory %s not found",d);
jmp1: break;
case 5: if(dcnt==0)
printf("\nNo Directory's ");
else
{
printf("\nDirectory\tFiles");
for(i=0;i<dcnt;i++)
{
printf("\n%s\t\t",dir[i].dname);
```

```
for(k=0;k<dir[i].fcnt;k++)
printf("\t%s",dir[i].fname[k]);
}
}
break;
default:exit(0);
}
}
}
```

16. Develop a C program for implementing random access file for processing the employee details.

```
#include <stdio.h>
   int main () {
     FILE *fp;
     int c;
     fp = fopen("file.txt","w+");
     fputs("This is study.com", fp);

     fseek( fp, 7, SEEK_SET );


     fputs(" C Programming", fp);


     printf("The current position of the file pointer is: %ld\n", ftell(fp));

     rewind(fp);

     printf("The current position of the file pointer is: %ld\n", ftell(fp));
     while(1) {
      c = fgetc(fp);
      if( feof(fp) ) {
       break;
      }
      printf("%c", c);
     }
     fclose(fp);
     return(0);
}
```

17. Illustrate the deadlock avoidance concept by simulating Banker’s algorithm with C.

```
#include<stdio.h>
int main() {
  int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5],
terminate = 0;
  printf("Enter the number of process and resources");
  scanf("%d %d", & p, & c);
  printf("enter allocation of resource of all process %dx%d matrix", p, c);
  for (i = 0; i < p; i++) {
```

```
  for (j = 0; j < c; j++) {
    scanf("%d", & alc[i][j]);
  }
}
printf("enter the max resource process required %dx%d matrix", p, c);
for (i = 0; i < p; i++) {
  for (j = 0; j < c; j++) {
    scanf("%d", & max[i][j]);
  }
}
printf("enter the  available resource");
for (i = 0; i < c; i++)
  scanf("%d", & available[i]);

printf("\n need resources matrix are\n");
for (i = 0; i < p; i++) {
  for (j = 0; j < c; j++) {
    need[i][j] = max[i][j] - alc[i][j];
    printf("%d\t", need[i][j]);
  }
  printf("\n");
}
for (i = 0; i < p; i++) {
  done[i] = 0;
}
while (count < p) {
  for (i = 0; i < p; i++) {
    if (done[i] == 0) {
      for (j = 0; j < c; j++) {
        if (need[i][j] > available[j])
          break;
      }
      if (j == c) {
        safe[count] = i;
        done[i] = 1;
        for (j = 0; j < c; j++) {
          available[j] += alc[i][j];
        }
        count++;
        terminate = 0;
      } else {
        terminate++;
      }
    }
  }
  if (terminate == (p - 1)) {
    printf("safe sequence does not exist");
    break;
  }

 }
 if (terminate != (p - 1)) {
  printf("\n available resource after completion\n");
  for (i = 0; i < c; i++) {
   printf("%d\t", available[i]);
  }
  printf("\n safe sequence are\n");
  for (i = 0; i < p; i++) {
   printf("p%d\t", safe[i]);
  }
 }
return 0;
}
```

18 Construct a C program to simulate producer-consumer problem using semaphores.

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
  int n;
  void producer();
  void consumer();
  int wait(int);
  int signal(int);
  printf("\n1.Producer\n2.Consumer\n3.Exit");
  while(1)
  {
    printf("\nEnter your choice:");
    scanf("%d",&n);
    switch(n)
    {
      case 1:   if((mutex==1)&&(empty!=0))
                  producer();
                else
                  printf("Buffer is full!!");
                break;
      case 2:   if((mutex==1)&&(full!=0))
                  consumer();
                else
                  printf("Buffer is empty!!");
                break;
      case 3:
                exit(0);
                break;
    }
  }
```

```
   return 0;
}
int wait(int s)
{
   return (--s);
}
int signal(int s)
{    return(++s);
}void producer()
{    mutex=wait(mutex);
   full=signal(full);
   empty=wait(empty);
   x++;
   printf("\nProducer produces the item %d",x);
   mutex=signal(mutex);
}void consumer()
{
   mutex=wait(mutex);
   full=wait(full);
   empty=signal(empty);
   printf("\nConsumer consumes item %d",x);
   x--;
   mutex=signal(mutex);
}
```

19.Design a C program to implement process synchronization using mutex locks.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

pthread_t tid[2];
int counter;

void* trythis(void* arg)
{
   unsigned long i = 0;
   counter += 1;
   printf("\n Job %d has started\n", counter);
```

```
    for (i = 0; i < (0xFFFFFFFF); i++)
        ;
    printf("\n Job %d has finished\n", counter);

    return NULL;
}

int main(void)
{
    int i = 0;
    int error;

    while (i < 2) {
        error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
        if (error != 0)
            printf("\nThread can't be created : [%s]", strerror(error));
        i++;
    }

    pthread_join(tid[0], NULL);
    pthread_join(tid[1], NULL);

return 0;
}
```

20. Construct a C program to simulate Reader-Writer problem using Semaphores.

```
#include <iostream>
#include <pthread.h>
#include <unistd.h>
using namespace std;
class monitor {
private:
    int rcnt;
    int wcnt;
    int waitr;
    int waitw;
    pthread_cond_t canread;
    pthread_cond_t canwrite;
    pthread_mutex_t condlock;
  public:
    monitor()
    {
      rcnt = 0;
      wcnt = 0;
      waitr = 0;
      waitw = 0;
      pthread_cond_init(&canread, NULL);
      pthread_cond_init(&canwrite, NULL);
      pthread_mutex_init(&condlock, NULL);
    }
    void beginread(int i)
    {
      pthread_mutex_lock(&condlock);
      if (wcnt == 1 || waitw > 0) {
        waitr++;
        pthread_cond_wait(&canread, &condlock);
        waitr--;
      }
      rcnt++;
      cout << "reader " << i << " is reading\n";
      pthread_mutex_unlock(&condlock);
      pthread_cond_broadcast(&canread);
    }
    void endread(int i)
    {
      pthread_mutex_lock(&condlock);
      if (--rcnt == 0)
        pthread_cond_signal(&canwrite);
      pthread_mutex_unlock(&condlock);
    }
    void beginwrite(int i)
    {
      pthread_mutex_lock(&condlock);
      if (wcnt == 1 || rcnt > 0) {
        ++waitw;
        pthread_cond_wait(&canwrite, &condlock);
        --waitw;
      }
      wcnt = 1;
      cout << "writer " << i << " is writing\n";
      pthread_mutex_unlock(&condlock);
    }
    void endwrite(int i)
    {
        pthread_mutex_lock(&condlock);
        wcnt = 0;
        if (waitr > 0)
            pthread_cond_signal(&canread);
        else
            pthread_cond_signal(&canwrite);
        pthread_mutex_unlock(&condlock);
    }
}
M;
void* reader(void* id)
{
    int c = 0;
    int i = (int)id;

    while (c < 5) {
        usleep(1);
        M.beginread(i);
        M.endread(i);
        c++;
    }
}
void* writer(void* id)
{
    int c = 0;
    int i = (int)id;
    while (c < 5) {
        usleep(1);
        M.beginwrite(i);
        M.endwrite(i);
        c++;
    }
}
int main()
{
    pthread_t r[5], w[5];
    int id[5];
    for (int i = 0; i < 5; i++) {
        id[i] = i;
        pthread_create(&r[i], NULL, &reader, &id[i]);
         pthread_create(&w[i], NULL, &writer, &id[i]);
    }
    for (int i = 0; i < 5; i++) {
        pthread_join(r[i], NULL);
    }
```

```
  for (int i = 0; i < 5; i++) {
    pthread_join(w[i],NULL);
}
}
```

21. Develop a C program to implement worst fit algorithm of memory management.

```
#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)
{
  int allocation[processes];

  for(int i = 0; i < processes; i++){
    allocation[i] = -1;
  }

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

    int indexPlaced = -1;
    for (int j=0; j<blocks; j++)
    {
      if (blockSize[j] >= processSize[i])
      {
        if (indexPlaced == -1)
          indexPlaced = j;

        else if (blockSize[indexPlaced] < blockSize[j])
          indexPlaced = j;
      }
    }

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

      blockSize[indexPlaced] -= processSize[i];
    }
  }

  printf("\nProcess No.\tProcess Size\tBlock no.\n");
  for (int i = 0; i < processes; i++)
  {
    printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
    if (allocation[i] != -1)
      printf("%d\n",allocation[i] + 1);
    else
      printf("Not Allocated\n");
  }
}

int main()
{
   int blockSize[] = {5, 4, 3, 6, 7};
   int processSize[] = {1, 3, 5, 3};
   int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
   int processes = sizeof(processSize)/sizeof(processSize[0]);

   implimentWorstFit(blockSize, blocks, processSize, processes);

    return 0;
}
```

22. Construct a C program to implement best fit algorithm of memory management.

```
#include <stdio.h>

void implimentWorstFit(int blockSize[], int blocks, int processSize[], int processes)
{
   int allocation[processes];

   for(int i = 0; i < processes; i++){
      allocation[i] = -1;
   }

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

      int indexPlaced = -1;
      for (int j=0; j<blocks; j++)
      {
         if (blockSize[j] >= processSize[i])
         {
            if (indexPlaced == -1)
               indexPlaced = j;

            else if (blockSize[indexPlaced] < blockSize[j])
               indexPlaced = j;
         }
      }

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

         blockSize[indexPlaced] -= processSize[i];
      }
   }
```

```
    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < processes; i++)
    {
        printf("%d \t\t\t %d \t\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n",allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }
}

int main()
{
    int blockSize[] = {5, 4, 3, 6, 7};
    int processSize[] = {1, 3, 5, 3};
    int blocks = sizeof(blockSize)/sizeof(blockSize[0]);
    int processes = sizeof(processSize)/sizeof(processSize[0]);

    implimentWorstFit(blockSize, blocks, processSize, processes);

    return 0 ;
}
```

23. Construct a C program to implement first fit algorithm of memory management.

```
#include<stdio.h>


void firstFit(int blockSize[], int m, int processSize[], int n)
{
        int i, j;

        int allocation[n];

        for(i = 0; i < n; i++)
        {
                allocation[i] = -1;
        }

        for (i = 0; i < n; i++)
        {
                for (j = 0; j < m; j++)
                {
                        if (blockSize[j] >= processSize[i])
                        {

                                allocation[i] = j;

                                blockSize[j] -= processSize[i];
                break;
            }
        }
    }

    printf("\nProcess No.\tProcess Size\tBlock no.\n");
    for (int i = 0; i < n; i++)
    {
        printf(" %i\t\t\t", i+1);
        printf("%i\t\t\t\t", processSize[i]);
        if (allocation[i] != -1)
            printf("%i", allocation[i] + 1);
        else
            printf("Not Allocated");
        printf("\n");
    }
}

int main()
{
    int m;
    int n;
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    m = sizeof(blockSize) / sizeof(blockSize[0]);
    n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

24. Design a C program to demonstrate UNIX system calls for file management.

```
#include<unistd.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<stdio.h>

int main()
{
    int n,fd;
    char buff[50];

    printf("Enter text to write in the file:\n");
    n= read(0, buff, 50);

    fd=open("file",O_CREAT | O_RDWR, 0777);
```

```
        write(fd, buff, n);
        write(1, buff, n);

        int close(int fd);

        return 0;
}
```

25. Construct a C program to implement the I/O system calls of UNIX (fcntl, seek, stat, opendir, readdir)

```
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{

        int fd = open("foo.txt", O_RDONLY | O_CREAT);

        printf("fd = %d\n", fd);

        if (fd ==-1)
        {
                printf("Error Number % d\n", errno);
                perror("Program");
        }
        return 0;
}
```

26. Construct a C program to implement the file management operations.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
char character;
FILE *fpointer;
fpointer = fopen("C:\\program.txt","w");
if(fpointer == NULL)
{
printf("Error! The file does not exist.");
exit(0);
}
printf("Enter a character: ");
scanf("%c",&character);
fprintf(fpointer,"%c",character);
fclose(fpointer);
return 0;
}
```

27. Develop a C program for simulating the function of ls UNIX Command.

```
#include<stdio.h>
```

```
#include<dirent.h>
main()
{
char dirname[10];
DIR*p;
struct dirent *d;
printf("Enter directory name\n");
scanf("%s",dirname);
p=opendir(dirname);
if(p==NULL)
  {
  perror("Cannot find directory");
  exit(-1);
  }
while(d=readdir(p))
  printf("%s\n",d->d_name);
}
```

28. Write a C program for simulation of GREP UNIX command

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s [Operating System] [SSE]\n", argv[0]);
        return 1;
    }

    char *search_string = argv[1];
    char *file_name = argv[2];

    FILE *file = fopen(file_name, "r");
    if (!file) {
        printf("Error opening file\n");
        return 1;
    }

    char line[256];
    while (fgets(line, sizeof(line), file)) {
        if (strstr(line, search_string)) {
            printf("%s", line);
        }
    }

    fclose(file);
    return 0;
}
```

29. Write a C program to simulate the solution of Classical Process Synchronization Problem

```
#include <dirent.h>
```

```
#include <stdio.h>

int main(void) {
   DIR *d;
   struct dirent *dir;
   d = opendir(".");
   if (d) {
      while ((dir = readdir(d)) != NULL) {
         printf("%s\n", dir->d_name);
      }
      closedir(d);
   }
   return 0;
}
```

30. Write C programs to demonstrate the following thread related concepts. (i) create (ii) join (iii) equal (iv) exit.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>


void *myThreadFun(void *vargp)
{
            sleep(1);
            printf("Printing GeeksQuiz from Thread \n");
            return NULL;
}

int main()
{
            pthread_t thread_id;
            printf("Before Thread\n");
            pthread_create(&thread_id, NULL, myThreadFun, NULL);
            pthread_join(thread_id, NULL);
            printf("After Thread\n");
            exit(0);
}
```

31. Construct a C program to simulate the First in First Out paging technique of memory management.

```
#include <stdio.h>
#define MAX_PAGE_FRAMES 10
#define MAX_PAGE_REFERENCES 20
int page_frames[MAX_PAGE_FRAMES];
int page_reference_string[MAX_PAGE_REFERENCES] = {1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6};
int page_faults = 0;
int find_page_fault(int page_reference) {
  int i;
  for (i = 0; i < MAX_PAGE_FRAMES; i++) {
    if (page_frames[i] == page_reference) {
      return 0;
    }
  }
  return 1;
}
int main() {
  int i, j, current_page, next_page;
  for (i = 0; i < MAX_PAGE_FRAMES; i++) {
    page_frames[i] = -1;
  }
  for (i = 0; i < MAX_PAGE_REFERENCES; i++) {
    current_page = page_reference_string[i];
    if (find_page_fault(current_page)) {
      page_faults++;
      for (j = 0; j < MAX_PAGE_FRAMES - 1; j++) {
        page_frames[j] = page_frames[j + 1];
      }
      page_frames[MAX_PAGE_FRAMES - 1] = current_page;
    }
    printf("Page frames: ");
    for (j = 0; j < MAX_PAGE_FRAMES; j++) {
      printf("%d ", page_frames[j]);
    }
    printf("\n");
  }
  printf("Total page faults: %d\n", page_faults);
  return 0;
}
```

32. Construct a C program to simulate the Least Recently Used paging technique of memory management.

```
#include <stdio.h>
#define MAX_PAGE_FRAMES 10
#define MAX_PAGE_REFERENCES 20
int page_frames[MAX_PAGE_FRAMES];
int page_reference_string[MAX_PAGE_REFERENCES] = {1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3,
2, 1, 2, 3, 6};
int page_faults = 0;
int last_used[MAX_PAGE_FRAMES];
int find_page_fault(int page_reference) {
  int i;
  for (i = 0; i < MAX_PAGE_FRAMES; i++) {
    if (page_frames[i] == page_reference) {
      last_used[i] = page_faults;
```

```
            return 0;
        }
    }
    return 1;
}

int find_lru() {
    int i, lru = 0;
    for (i = 1; i < MAX_PAGE_FRAMES; i++) {
        if (last_used[i] < last_used[lru]) {
            lru = i;
        }
    }
    return lru;
}

int main() {
    int i, j, current_page, next_page, lru;
    for (i = 0; i < MAX_PAGE_FRAMES; i++) {
        page_frames[i] = -1;
        last_used[i] = -1;
    }
    for (i = 0; i < MAX_PAGE_REFERENCES; i++) {
        current_page = page_reference_string[i];
        if (find_page_fault(current_page)) {
            page_faults++;
            if (page_faults <= MAX_PAGE_FRAMES) {
                for (j = 0; j < MAX_PAGE_FRAMES; j++) {
                    if (page_frames[j] == -1) {
                        page_frames[j] = current_page;
                        last_used[j] = page_faults;
                        break;
                    }
                }
            } else {
                lru = find_lru();
                page_frames[lru] = current_page;
                last_used[lru] = page_faults;
            }
        }
        printf("Page frames: ");
        for (j = 0; j < MAX_PAGE_FRAMES; j++) {
            printf("%d ", page_frames[j]);
        }
        printf("\n");
    }
    printf("Total page faults: %d\n", page_faults);
    return 0;
```

}

33. Construct a C program to simulate the optimal paging technique of memory management

```
#include <stdio.h>
#define MAX_PAGE_FRAMES 10
#define MAX_PAGE_REFERENCES 20
int page_frames[MAX_PAGE_FRAMES];
int page_reference_string[MAX_PAGE_REFERENCES] = {1, 2, 3, 4, 5, 6,  7, 8, 9, 10, 11, 12, 13, 14};
int page_faults = 0;
int find_page_fault(int page_reference) {
    int i;
    for (i = 0; i < MAX_PAGE_FRAMES; i++) {
        if (page_frames[i] == page_reference) {
            return 0;
        }
    }
    return 1;
}
int find_optimal(int current_index) {
    int i, j, max_distance = -1, optimal_index = -1;
    for (i = 0; i < MAX_PAGE_FRAMES; i++) {
        int distance = -1;
        for (j = current_index; j < MAX_PAGE_REFERENCES; j++) {
            if (page_frames[i] == page_reference_string[j]) {
                distance = j - current_index;
                break;
            }
        }
        if (distance > max_distance) {
            max_distance = distance;
            optimal_index = i;
        }
    }
    return optimal_index;
}
int main() {
    int i, j, current_page, next_page, optimal;
    for (i = 0; i < MAX_PAGE_FRAMES; i++) {
        page_frames[i] = -1;
    }
    for (i = 0; i < MAX_PAGE_REFERENCES; i++) {
        current_page = page_reference_string[i];
        if (find_page_fault(current_page)) {
            page_faults++;
            if (page_faults <= MAX_PAGE_FRAMES) {
                for (j = 0; j < MAX_PAGE_FRAMES; j++) {
                    if (page_frames[j] == -1) {
```

```
                page_frames[j] = current_page;
                break;
              }
            }
          } else {
            optimal = find_optimal(i);
            page_frames[optimal] = current_page;
          }
        }
        printf("Page frames: ");
        for (j = 0; j < MAX_PAGE_FRAMES; j++) {
          printf("%d ", page_frames[j]);
        }
        printf("\n");
    }
    printf("Total page faults: %d\n", page_faults);
    return 0;
}
```

34. Consider a file system where the records of the file are stored one after another both physically and logically. A record of the file can only be accessed by reading all the previous records. Design a C program to simulate the file allocation strategy.

```
#include <stdio.h>
#include<conio.h>
void main()
{
int f[50], i, st, len, j, c, k, count = 0;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Files Allocated are : \n");
x: count=0;
printf(“Enter starting block and length of files: ”);
scanf("%d%d", &st,&len);
for(k=st;k<(st+len);k++)
if(f[k]==0)
count++;
if(len==count)
{
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("%d\t%d\n",j,f[j]);
}
if(j!=(st+len-1))
printf(” The file is allocated to disk\n");
}
else
printf(” The file is not allocated \n");
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit();
getch();
}
```

35. Consider a file system that brings all the file pointers together into an index block. The ith entry in the index block points to the ith block of the file. Design a C program to simulate the file allocation strategy.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int f[50], p,i, st, len, j, c, k, a;
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{
for(j=st;j<(st+k);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("%d-------->%d\n",j,f[j]);
}
else
{
printf("%d Block is already allocated \n",j);
k++;
}
}
}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
}
```

36. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file. Each block contains a pointer to the next block. Design a C program to simulate the file allocation strategy.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;
for(i=0;i<50;i++)
f[i]=0;
x:printf("Enter the index block: ");
scanf("%d",&ind);
if(f[ind]!=1)
{
printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);
scanf("%d",&n);
}
else
{
printf("%d index is already allocated \n",ind);
goto x;
}
y: count=0;
for(i=0;i<n;i++)
{
scanf("%d", &index[i]);
if(f[index[i]]==0)
count++;
}
if(count==n)
{
for(j=0;j<n;j++)
f[index[j]]=1;
printf("Allocated\n");
printf("File Indexed\n");
for(k=0;k<n;k++)
printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);
}
else
{
```

```
printf("File in the index is already allocated \n");
printf("Enter another file indexed");
goto y;
}
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
}
```

37.Construct a C program to simulate the First Come First Served disk scheduling algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,n,TotalHeadMoment=0,initial;
   printf("Enter the number of Requests\n");
   scanf("%d",&n);
   printf("Enter the Requests sequence\n");
   for(i=0;i<n;i++)
    scanf("%d",&RQ[i]);
   printf("Enter initial head position\n");
   scanf("%d",&initial);

   for(i=0;i<n;i++)
   {
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
      initial=RQ[i];
   }

   printf("Total head moment is %d",TotalHeadMoment);
   return 0;

}
```

38. Design a C program to simulate SCAN disk scheduling algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i,j,sum=0,n;
int d[20];
int disk;
int temp,max;
int dloc;
printf("enter number of location\t");
scanf("%d",&n);
printf("enter position of head\t");
scanf("%d",&disk);
printf("enter elements of disk queue\n");
for(i=0;i<n;i++)
{
scanf("%d",&d[i]);
}
d[n]=disk;
n=n+1;
for(i=0;i<n;i++)
{
for(j=i;j<n;j++)
{
if(d[i]>d[j])
{
temp=d[i];
d[i]=d[j];
d[j]=temp;
}
}

}
max=d[n];
for(i=0;i<n;i++)
{
if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i>=0;i--)
{
printf("%d -->",d[i]);
}
printf("0 -->");
for(i=dloc+1;i<n;i++)
{
printf("%d-->",d[i]);
}
sum=disk+max;
printf("\nmovement of total cylinders %d",sum);

return 0;
}
```

39. Develop a C program to simulate C-SCAN disk scheduling algorithm.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
   printf("Enter the number of Requests\n");
```

```
scanf("%d",&n);
printf("Enter the Requests sequence\n");
for(i=0;i<n;i++)
 scanf("%d",&RQ[i]);
printf("Enter initial head position\n");
scanf("%d",&initial);
printf("Enter total disk size\n");
scanf("%d",&size);
printf("Enter the head movement direction for high 1 and for low 0\n");
scanf("%d",&move);

for(i=0;i<n;i++)
{
    for( j=0;j<n-i-1;j++)
    {
        if(RQ[j]>RQ[j+1])
        {
            int temp;
            temp=RQ[j];
            RQ[j]=RQ[j+1];
            RQ[j+1]=temp;
        }

    }
}

int index;
for(i=0;i<n;i++)
{
    if(initial<RQ[i])
    {
        index=i;
        break;
    }
}

if(move==1)
{
    for(i=index;i<n;i++)
    {
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
        initial=RQ[i];
    }
    TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
    TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
    initial=0;
    for( i=0;i<index;i++)
    {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];

      }
   }
   else
   {
      for(i=index-1;i>=0;i--)
      {
         TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
         initial=RQ[i];
      }
      TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
      TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
      initial =size-1;
      for(i=n-1;i>=index;i--)
      {
          TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
          initial=RQ[i];

      }
   }

   printf("Total head movement is %d",TotalHeadMoment);
   return 0;
}
```

40. Illustrate the various File Access Permission and different types users in Linux.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char **argv) {
   int result;
   char *filename = (char *)malloc(512);
   if (argc < 2) {
      strcpy(filename, "/usr/bin/adb");
   } else {
      strcpy(filename, argv[1]);
   }
   result = access (filename, R_OK);
   if ( result == 0 ) {
      printf("%s is readable\n",filename);
   } else {
      printf("%s is not readable\n",filename);
   }

   result = access (filename, W_OK);
```

```
    if ( result == 0 ) {
        printf("%s is Writeable\n",filename);
    } else {
        printf("%s is not Writeable\n",filename);
    }

    result = access (filename, X_OK);
    if ( result == 0 ) {
        printf("%s is executable\n",filename);
    } else {
        printf("%s is not executable\n",filename);
    }

    free(filename);
    return 0;
}
```