1. Develop a C program to implement the process system calls `fork()`, `wait()` to create process and terminate process.

→
```c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>

int main(){
        pid_t child_pid;
        child_pid=fork();
        int status;
        if(child_pid<0){
                fprintf(stderr,"fork() failed !\n");
                exit(-1);
        }

        if(child_pid==0){
                printf("Child process (PID : %d) is running ...\n",getpid());
                char *args[]={"/bin/ls","-a",NULL};
                execvp(args[0],args);
                perror("exec() failed !");
                exit(1);
        }else{
                printf("Parent process (PID : %d) is waiting for the child to complete\n",getpid());
                wait(&status);
                if(WIFEXITED(status))
                        printf("Child process (PID : %d) has completed with status %d\n",child_pid,WEX
        }
        return(0);
}
```

2. Simulate the following CPU scheduling algorithm to bind turnaround time waiting time

   a) FCFS
   b) SJF
   c) Round Robin
   d) Priority

   a)
```c
#include<stdio.h>
#include<stdlib.h>

int main(){
        int bt[20],wt[20],tat[20],i,n;
        float avwt,avtat;
        printf("Enter the number of process -- ");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("Enter the burst time for the process %d -- ",i);
                scanf("%d",&bt[i]);
        }
        wt[0]=avwt=0;
        tat[0]=avtat=bt[0];

        for(i=1;i<n;i++){
                wt[i]=wt[i-1]+bt[i-1];
                tat[i]=tat[i-1]+bt[i];
                avwt+=wt[i];
                avtat+=tat[i];
        }

        printf("\nProcess\t Burst time\t Waiting Time\t Turnaround Time\n");
        for(i=0;i<n;i++)
                printf("p%d\t %d\t\t %d\t\t %d\n",i,bt[i],wt[i],tat[i]);
        printf("\nAverage waiting time -- %f",avwt/n);
        printf("\nAverage Turnaround time -- %f\n",avtat/n);
        exit(0);
}
```

b)
```c
#include<stdio.h>
#include<stdlib.h>

int main(){
        int p[20],bt[20],wt[20],tat[20],i,k,n,temp;
        float wtavg,tatavg;
        printf("Enter the number of process : ");
        scanf("%d",&n);

        for(i=0;i<n;i++){
                p[i]=i;
                printf("Enter the burst time for the process %d : ",i);
                scanf("%d",&bt[i]);
        }

        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(bt[i]>bt[k]){
                                temp=bt[i];
                                bt[i]=bt[k];
                                bt[k]=temp;
                                temp=p[i];
                                p[i]=p[k];
                                p[k]=temp;
                        }
        wt[0]=wtavg=0;
        tat[0]=tatavg=bt[0];

        for(i=1;i<n;i++){
                wt[i]=wt[i-1]+bt[i-1];
                tat[i]=tat[i-1]+bt[i];
                wtavg+=wt[i];
                tatavg+=tat[i];
        }

        printf("\nProcess\t Burst time\t Waiting Time\t Turnaround Time\n");
        for(i=0;i<n;i++)
                printf("p%d\t %d\t\t %d\t\t %d\n",p[i],bt[i],wt[i],tat[i]);
        printf("\nAverage waiting time : %f",wtavg/n);
        printf("\nAverage Turnaround time : %f\n",tatavg/n);
        return 0;
}
```

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

int main(){
        int bu[10],wa[10],tat[10],ct[10],i,j,max,n,t;
        float awt,att,temp;
        awt=att=temp=0;

        printf("Enter the number of process : ");
        scanf("%d",&n);

        for(i=0;i<n;i++){
                printf("Enter the burst time for the process %d : ",i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }

        printf("Enter the time slice : ");
        scanf("%d",&t);
        max=bu[0];

        for(i=1;i<n;i++)
                if(max<bu[i])
                        max=bu[i];

        for(j=0;j<(max/t)+1;j++)
                for(i=0;i<n;i++)
                        if(bu[i]!=0){
                                if(bu[i]<=t){
                                        tat[i]=temp+bu[i];
                                        temp+=bu[i];
                                        bu[i]=0;
                                }else{
                                        bu[i]-=t;
                                        temp+=t;
                                }
                        }

        for(i=0;i<n;i++){
                wa[i]=tat[i]-ct[i];
                att+=tat[i];
                awt+=wa[i];
        }

        printf("\nThe Average Turnaround time is : %f",att/n);
        printf("\nThe Average Waiting time is : %f",awt/n);
        printf("\nProcess\t Burst time\t Waiting Time\t Turnaround Time\n");
        for(i=0;i<n;i++)
                printf("%d\t %d\t\t %d\t\t %d\n",i+1,ct[i],wa[i],tat[i]);
        return 0;
}
```

d)
```c
#include<stdio.h>

int main(){
        int p[20],pri[20],bt[20],wt[20],tat[20],i,k,n,temp;
        float wtavg,tatavg;
        printf("Enter the number of process : ");
        scanf("%d",&n);

        for(i=0;i<n;i++){
                p[i]=i;
                printf("Enter the burst time and priority of the process %d : ",i);
                scanf("%d%d",&bt[i],&pri[i]);
        }

        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(pri[i]>pri[k]){
                                temp=p[i];
                                p[i]=p[k];
                                p[k]=temp;
                                temp=bt[i];
                                bt[i]=bt[k];
                                bt[k]=temp;
                                temp=pri[i];
                                pri[i]=pri[k];
                                pri[k]=temp;
                        }

        wt[0]=wtavg=0;
        tat[0]=tatavg=bt[0];

        for(i=1;i<n;i++){
                wt[i]=wt[i-1]+bt[i-1];
                tat[i]=tat[i-1]+bt[i];
                wtavg+=wt[i];
                tatavg+=tat[i];
        }

        printf("\nProcess\t Priority\t Burst time\t Waiting Time\t Turnaround Time\n");
        for(i=0;i<n;i++)
                printf("p%d\t %d\t\t %d\t\t %d\t\t %d\n",p[i],pri[i],bt[i],wt[i],tat[i]);
        printf("\nAverage waiting time : %f",wtavg/n);
        printf("\nAverage Turnaround time : %f\n",tatavg/n);
        return 0;
}
```

3. Develop a C program to simulate producer-consumer problem using semaphores.

→
```c
#include <stdio.h>
#include <stdlib.h>

int mutex=1;
int full=0;
int empty=10;
int x=0;

void producer(){
        --mutex;
        ++full;
        --empty;
        x++;
        printf("\nProducer produces item %d",x);
        ++mutex;
}

void consumer(){
        --mutex;
        --full;
        ++empty;
        printf("\nConsumer consumes item %d",x);
        x--;
        ++mutex;
}

int main(){
        int n,i;
        printf("\n1. Press 1 for Producer\n2. Press 2 for Consumer\n3. Press 3 for Exit");

        for(i=1;i>0;i++){
                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);
                }
        }
        return 0;
}
```

4. Develop a C program which demonstrates interprocess communication between a reader process and a writer process. Use `mkfifo()`, `open()`, `read()`, `write()` and `close()` APIs in your program.

i) Reader Process

```
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#define MAX_BUF 1024

int main(){
        int fd;
        char *myfifo="/tmp/myfifo";
        char buf[MAX_BUF];
        fd=open(myfifo,O_RDONLY);
        read(fd,buf,MAX_BUF);
        printf("Writer: %s\n",buf);
        close(fd);
        return 0;
}
```

ii) Writer Process

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

int main(){
        int fd;
        char buf[1024];
        char *myfifo="/tmp/myfifo";

        mkfifo(myfifo,0666);
        printf("Run Reader process to read the FIFO File\n");
        fd=open(myfifo,O_WRONLY);
        strcpy(buf,"Hello from Writer Process");
        write(fd,buf,sizeof(buf));
        close(fd);
        unlink(myfifo);
        return 0;
}
```

5. Develop a C program to simulate Bankers Algorithm for Dead Lock Avoidance.

→

```c
#include <stdio.h>

int isSafe(int processes,int resources,int max[][resources],int allocated[][resources],int available[]
        int need[processes][resources],finish[processes],i,j;

        for(i=0;i<processes;i++){
                finish[i]=0;

                for(j=0;j<resources;j++)
                        need[i][j]=max[i][j]-allocated[i][j];
        }

        int work[resources];

        for(i=0;i<resources;i++)
                work[i]=available[i];

        while(1){
                int found=0;

                for(i=0;i<processes;i++)
                        if(!finish[i]){
                                int canAllocate=1;

                                for(j=0;j<resources;j++)
                                        if(need[i][j]>work[j]){
                                                canAllocate=0;
                                                break;
                                        }

                                if(canAllocate){
                                        for(j=0;j<resources;j++)
                                                work[j]+=allocated[i][j];
                                        finish[i]=1;
                                        found=1;
                                        break;
                                }
                        }

                if(!found)
                        break;
        }

        for(i=0;i<processes;i++)
                if(!finish[i])
                        return 0;

        return 1;
}

int main(){
        int processes,resources,i,j;

        printf("Enter the number of processes: ");
        scanf("%d",&processes);
        printf("Enter the number of resources: ");
        scanf("%d",&resources);

        int max[processes][resources],allocated[processes][resources],available[resources];

        printf("\nEnter the maximum resource matrix:\n");

        for(i=0;i<processes;i++)
                for(j=0;j<resources;j++)
                        scanf("%d",&max[i][j]);

        printf("\nEnter the allocated resource matrix:\n");

        for(i=0;i<processes;i++)
                for(j=0;j<resources;j++)
                        scanf("%d",&allocated[i][j]);

        printf("\nEnter the available resources:\n");

        for(i=0;i<resources;i++)
                scanf("%d",&available[i]);

        if(isSafe(processes,resources,max,allocated,available))
                printf("\nThe system is in a safe state.\n");
        else
                printf("\nThe system is not in a safe state.\n");


        return 0;
}
```

6. Develop a C program to simulate the following contiguous memory allocation Techniques
   (a) Worst fit
   (b) Best fit
   (c) First fit

i) Worst fit

```
#include<stdio.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];

        printf("\nMemory Management Scheme - Worst Fit");
        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("\nEnter 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 && highest<temp){
                                        ff[i]=j;
                                        highest=temp;
                                }
                        }

                frag[i]=highest;
                bf[ff[i]]=1;
                highest=0;
        }

        printf("\nFile_no\t\tFile_size\tBlock_no\tBlock_size\tFragement\n");
        for(i=1;i<=nf;i++)
                printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

ii) Best fit

```
#include<stdio.h>

void main(){
        int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
        static int barray[20],parray[20];

        printf("\nMemory Management Scheme - Best Fit");
        printf("\nEnter the number of blocks : ");
        scanf("%d",&nb);
        printf("Enter the number of processes : ");
        scanf("%d",&np);

        printf("\nEnter the size of the blocks : \n");
        for(i=1;i<=nb;i++){
                printf("Block no.%d : ",i);
                scanf("%d",&b[i]);
        }

        printf("\nEnter the size of the processes : \n");
        for(i=1;i<=np;i++){
                printf("Process no.%d : ",i);
                scanf("%d",&p[i]);
        }

        for(i=1;i<=np;i++){
                for(j=1;j<=nb;j++)
                        if(barray[j]!=1){
                                temp=b[j]-p[i];
                                if(temp>=0 && lowest>temp){
                                                parray[i]=j;
                                                lowest=temp;
                                        }
                        }

                fragment[i]=lowest;
                barray[parray[i]]=1;
                lowest=10000;
        }

        printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment\n");
        for(i=1;i<=np&&parray[i]!=0;i++)
                printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

iii) First fit

```
#include <stdio.h>
#include <stdlib.h>
#define max 25

void main(){
        int frag[max],b[max],f[max],i,j,nb,nf,temp;
        static int bf[max],ff[max];

        printf("\nMemory Management Scheme-First Fit");
        printf("\nEnter the number of blocks : ");
        scanf("%d",&nb);

        printf("\nEnter 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("\nEnter 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\t\tFile_size\tBlock_no\tBlock_size\tFragement\n");
        for(i=1;i<=nf;i++)
                printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
        exit(0);
}
```

7. Develop a C program to simulate page replacement algorithms
   (a) FIFO
   (b) LRU

i) FIFO

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

int fr[3];

void display(){
        int i;
        printf("\n");

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

void main(){
        int page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
        int i,j,flag1=0,flag2=0,pf=0,frsize=3,top=0;

        for(i=0;i<3;i++)
                fr[i]=-1;

        for(j=0;j<12;j++){
                flag1=flag2=0;

                for(i=0;i<12;i++)
                        if(fr[i]==page[j]){
                                flag1=flag2=1;
                                break;
                        }

                if(flag1==0)
                        for(i=0;i<frsize;i++)
                                if(fr[i]==-1){
                                        fr[i]=page[j];
                                        flag2=1;
                                        break;
                                }

                if(flag2==0){
                        fr[top]=page[j];
                        top++;
                        pf++;
                        if(top>=frsize)
                        top=0;
                }
                display();
        }

        printf("\n\nNumber of page faults : %d\n",pf+frsize);
        exit(0);
}
```

ii) LRU

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

int fr[3];

void display(){
        int i;
        printf("\n");
        for(i=0;i<3;i++)
                printf("\t%d",fr[i]);
}

void main(){
        int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},fs[3];
        int index,i,j,k,l,flag1=0,flag2=0,pf=0,frsize=3;

        for(i=0;i<3;i++)
                fr[i]=-1;

        for(j=0;j<12;j++){
                flag1=flag2=0;

                for(i=0;i<3;i++)
                        if(fr[i]==p[j]){
                                flag1=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\nNo of page faults : %d\n",pf+frsize);
        exit(0);
}
```

8. Simulate following File Organization Techniques

   (a) Single level directory

   (b) Two level directory

→
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FILES 5
#define NAME 20

typedef struct{
        char name[NAME];
        int size;
}file;

typedef struct{
        char name[NAME];
        int file_total;
        file files[FILES];
}dir;

dir sld,tld[FILES];

void init_SLD(){
        strcpy(sld.name,"Root");
        sld.file_total=0;
}

void init_TLD(){
        for(int i=0;i<FILES;++i){
                sprintf(tld[i].name,"Directory%d",i+1);
                tld[i].file_total=0;
        }
}

void display_SLD(){
        printf("Single Level Directory :\nDirectory Name : %s\nNumber of Files : %d\n", \
                sld.name,sld.file_total \
        );

        printf("Files :\n");

        for(int i=0;i<sld.file_total;++i){
                printf( \
                        "File Name : %s, Size : %d KB\n", \
                        sld.files[i].name,sld.files[i].size
                );
        }
        printf("\n");
}

void display_TLD(){
        printf("Two Level Directory:\n");
        for(int i=0;i<FILES;++i){
                printf( \
                        "Directory Name : %s\nNumber of Files : %d\nFiles :\n", \
                        tld[i].name,tld[i].file_total
                );
                for(int j=0;j<tld[i].file_total;++j){
                        printf( \
                                "File Name : %s, Size : %d KB\n", \
                                tld[i].files[j].name,tld[i].files[j].size
                        );
                }
                printf("\n");
        }
}

void add_file_SLD(char name[],int size){
        if(sld.file_total<FILES){
                strcpy(sld.files[sld.file_total].name,name);
                sld.files[sld.file_total].size=size;
                sld.file_total++;
                printf("File '%s' added to Single Level Directory\n",name);
        }else
                printf("Single Level Directory is full,cannot add file '%s'\n",name);
}

void add_file_TLD(char name[],int size,int index){
        if(index>=0 && index<FILES)
                if(tld[index].file_total<FILES){
                        strcpy(tld[index].files[tld[index].file_total].name,name);
                        tld[index].files[tld[index].file_total].size=size;
                        tld[index].file_total++;
                        printf( \
                                "File '%s' added to Directory '%s'\n", \
                                name,tld[index].name
                        );
                }else
                        printf( \
                                "Directory '%s' is full,cannot add file '%s'\n", \
                                tld[index].name,name
                        );
        else
                printf("Invalid Directory Index for Two Level Directory\n");
}

int main(){
        init_SLD();
        init_TLD();
        add_file_SLD("file1.txt",1024);
        add_file_SLD("file2.txt",2048);
        add_file_SLD("file3.txt",3072);
        display_SLD();
        add_file_TLD("file4.txt",4096,0);
        add_file_TLD("file5.txt",5120,1);
        add_file_TLD("file6.txt",6144,2);
        display_TLD();
        return 0;
}
```

9. Develop a C program to simulate the Linked file allocation strategies

→
```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>

int main(){
        int f[50],p,i,j,k,a,st,len,c;
        for(i=0;i<50;i++)
                f[i]=0;

        printf("Enter how many blocks that are already allocated : ");
        scanf("%d",&p);

        printf("\nEnter the blocks no.s that are already allocated : ");
        for(i=0;i<p;i++){
                scanf("%d",&a);
                f[a]=1;
        }

        X:
        printf("Enter the starting index block & length : ");
        scanf("%d%d",&st,&len);
        k=len;

        for(j=st;j<(k+st);j++){
                if(f[j]==0){
                        f[j]=1;
                        printf("\n%d->%d",j,f[j]);
                }else{
                        printf("\n%d->file is already allocated",j);
                        k++;
                }
        }

        printf("\nIf u want to enter one more file ?(yes -1/no -0) : ");
        scanf("%d",&c);
        if (c==1)
                goto X;
        else
                return 0;
}
```

10. Develop a C program to simulate SCAN disk scheduling algorithm

→
```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        int queue[20],head,n,i,j,seekTime=0,direction,maxTrack;

        printf("Enter the number of disk requests : ");
        scanf("%d",&n);

        printf("Enter the disk request queue : ");
        for(i=0;i<n;i++)
                scanf("%d",&queue[i]);

        printf("Enter the initial head position : ");
        scanf("%d",&head);

        printf("Enter the maximum track number : ");
        scanf("%d",&maxTrack);

        printf("Enter the direction(0 for left,1 for right) : ");
        scanf("%d",&direction);

        printf("\n");

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

        int currentTrack=head;
        printf("Seek Sequence :\n");
        if(direction==0){
                for(i=head;i>=0;i--){
                        printf("%d ",i);
                        seekTime+=abs(currentTrack-i);
                        currentTrack=i;
                }
                printf("\n\n0 ");
                seekTime+=currentTrack;

                for(i=1;i<=maxTrack;i++){
                        printf("%d ",i);
                        seekTime+=abs(currentTrack-i);
                        currentTrack=i;
                }
                printf("\n\n");
        }else{
                for(i=head;i<=maxTrack;i++){
                        printf("%d ",i);
                        seekTime+=abs(currentTrack-i);
                        currentTrack=i;
                }
                printf("\n\n%d ",maxTrack);
                seekTime+=abs(currentTrack-maxTrack);
                for(i=maxTrack-1;i>=0;i--){
                        printf("%d ",i);
                        seekTime+=abs(currentTrack-i);
                        currentTrack=i;
                }
        }

        printf("\n\nTotal Seek Time : %d\n",seekTime);
        exit(0);
}
```