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,WEXITSTATUS(status) \
                        );
        }
        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

```c
#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;
}
```