Q.1 Write a program that demonstrates the use of nice() system call. After a child process is started using fork(), assign higher priority to the child using nice() system call.

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

int main() {
    pid_t pid;
    int priority_change = 10;  // Change priority (higher number = lower priority)
    int current_priority;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d)\n", getpid());

        // Get the current priority of the child
        current_priority = nice(0);
        printf("Current priority of child before nice(): %d\n", current_priority);

        // Increase priority (decrease nice value)
        if (nice(priority_change) == -1) {
            perror("Nice failed");
            exit(EXIT_FAILURE);
        }

        // Get the new priority after changing it
        current_priority = nice(0);
        printf("New priority of child after nice(): %d\n", current_priority);

        // Simulating some work
        for (volatile int i = 0; i < 100000000; i++);  // Busy wait
        printf("Child Process (PID: %d) completed.\n", getpid());
        exit(EXIT_SUCCESS);
    } else {
        // Parent process
        printf("Parent Process (PID: %d)\n", getpid());
        wait(NULL);  // Wait for child to finish
        printf("Parent Process completed.\n");
    }

    return 0;
}
```

O/P:- Parent Process (PID: 39912)
Child Process (PID: 39913)
Current priority of child before nice(): 0

New priority of child after nice(): 10
Child Process (PID: 39913) completed.
Parent Process completed.

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames.
Reference String :3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6
Implement FIFO

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n;

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
```

```
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
}

void fifo()
{
 int i,j;

 for(i=0;i<m;i++)
 {
  if(search(ref[i])==-1)
  {
   frames[sp] = ref[i];
   sp = (sp+1)%n;
   faults++;
   for(j=0;j<n;j++)
    mem[j][i] = frames[j];

  }
 }
}

int main()
{
 accept();
 fifo();
 disp();

 return 0;
}
```

//===SLIP NO:-2===//

Q.1 Create a child process using fork(), display parent and child process id. Child process will display the message “Hello World” and the parent process should display “Hi”.

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

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();
```

```
    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process ID: %d\n", getpid());
        printf("Hello World\n");
    } else {
        // Parent process
        printf("Parent Process ID: %d\n", getpid());
        printf("Hi\n");
    }

    return 0;
}
```

Q.2 Write the simulation program using SJF (non-preemptive). The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times.

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1;
 struct process_info *next;
}NODE;

int n;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);

  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
```

```
  scanf("%d",&p->bt);


  p->bt1 = p->bt;
  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;

 p = first;

 printf("pname\tat\tbt\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\n",
   p->pname,p->at,p->bt1);
  p = p->next;
 }
}
```

```
void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
  q=p->next;
  while(q!=NULL)
  {
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);

    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;

    t = p->ct;
    p->ct = q->ct;
    q->ct = t;

    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;

   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;

NODE * get_sjf()
{
 NODE *p,*min_p=NULL;
 int min=9999;

 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0 &&
```

```
   p->bt1<min)
  {
   min = p->bt1;
   min_p = p;
  }
  p=p->next;
 }

 return min_p;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];

int k;

void sjfnp()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  p = get_sjf();

  if(p==NULL)
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;

   prev = time;
   k++;
  }
  else
  {
   time+=p->bt1;
   s[k].start = prev;
   strcpy(s[k].pname, p->pname);
   s[k].end = time;

   prev = time;
   k++;

   p->ct = time;
   p->bt1 = 0;

   n1++;
  }
```

```
  print_input();
  sort();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 sjfnp();
 print_output();
 print_gantt_chart();

 return 0;
}
```

//===SLIP NO:-4===//

Q.1 Write a program to illustrate the concept of orphan process (Using fork() and sleep())

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

int main() {
    pid_t pid;
```

```
    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) is running...\n", getpid());

        // Sleep for a while to simulate work
        sleep(10);
        printf("Child Process (PID: %d) finished execution.\n", getpid());
    } else {
        // Parent process
        printf("Parent Process (PID: %d) will exit...\n", getpid());

        // Exit the parent process
        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

Q.2 Write the program to simulate Non-preemptive Priority scheduling. The arrival time and first CPU burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time..

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1;
 struct process_info *next;
}NODE;

int n;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));
```

```
  printf("Enter process name:");
  scanf("%s",p->pname);

  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);


  p->bt1 = p->bt;
  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;

 p = first;
```

```
 printf("pname\tat\tbt\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\n",
   p->pname,p->at,p->bt1);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
  q=p->next;
  while(q!=NULL)
  {
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);

    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;

    t = p->ct;
    p->ct = q->ct;
    q->ct = t;

    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;

   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;

NODE * get_sjf()
```

```
{
 NODE *p,*min_p=NULL;
 int min=9999;

 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0 &&
   p->bt1<min)
  {
   min = p->bt1;
   min_p = p;
  }
  p=p->next;
 }

 return min_p;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];

int k;

void sjfnp()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  p = get_sjf();

  if(p==NULL)
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;

   prev = time;
   k++;
  }
  else
  {
   time+=p->bt1;
   s[k].start = prev;
   strcpy(s[k].pname, p->pname);
   s[k].end = time;

   prev = time;
```

```
    k++;

    p->ct = time;
    p->bt1 = 0;

    n1++;
   }

   print_input();
   sort();
  }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 sjfnp();
 print_output();
 print_gantt_chart();

 return 0;
}
```

//////////////////////////////////////===SLIP NO:-5 ===//
Q.1 Write a program that demonstrates the use of nice () system call. After a child process is started using fork (), assign higher priority to the child using nice () system call.

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

int main() {
    pid_t pid;
    int nice_value = -10;  // Lowering the nice value to increase priority

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) before nice(): ", getpid());
        printf("Current nice value: %d\n", nice(0));

        // Change nice value to increase priority
        if (nice(nice_value) == -1) {
            perror("Failed to change nice value in child");
            exit(EXIT_FAILURE);
        }

        printf("Child Process (PID: %d) after nice(): ", getpid());
        printf("New nice value: %d\n", nice(0));
    } else {
        // Parent process
        printf("Parent Process (PID: %d) before nice(): ", getpid());
        printf("Current nice value: %d\n", nice(0));

        // Parent can also modify its nice value, if desired
        // nice(5); // Uncomment to change parent's priority as well

        // Wait for the child to finish
        wait(NULL);
        printf("Parent Process (PID: %d) finished.\n", getpid());
    }

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames. Reference String: 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

i. Implement FIFO

```
#include<stdio.h>
#define MAX 20
```

```
int frames[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n;

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }
```

```
 return -1;
}

void fifo()
{
 int i,j;

 for(i=0;i<m;i++)
 {
  if(search(ref[i])==-1)
  {
   frames[sp] = ref[i];
   sp = (sp+1)%n;
   faults++;
   for(j=0;j<n;j++)
    mem[j][i] = frames[j];

  }
 }
}

int main()
{
 accept();
 fifo();
 disp();

 return 0;
}
```

////////////////////////////////////===SLIP NO:-6===//

Q.1 Write a program to find the execution time taken for execution of a given set of instructions (use clock() function)

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

int main() {
    // Start measuring time
    clock_t start, end;
    double cpu_time_used;

    start = clock();  // Record the start time

    // Sample set of instructions: a simple loop
    volatile long sum = 0;  // Using volatile to prevent optimization
    for (long i = 0; i < 1e7; i++) {
        sum += i;  // Perform a simple addition
    }

    end = clock();  // Record the end time

    // Calculate the CPU time used
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    // Print the result
```

```
    printf("Sum: %ld\n", sum);
    printf("Execution time: %f seconds\n", cpu_time_used);

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.

Reference String :3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

Implement FIFO

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n;

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
```

```
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
}

void fifo()
{
 int i,j;

 for(i=0;i<m;i++)
 {
  if(search(ref[i])==-1)
  {
   frames[sp] = ref[i];
   sp = (sp+1)%n;
   faults++;
   for(j=0;j<n;j++)
    mem[j][i] = frames[j];

  }
 }
}

int main()
{
 accept();
 fifo();
 disp();

 return 0;
}
```

//===SLIP NO:-7===//

Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

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

int main() {
    pid_t pid;

    // Create a child process
```

```
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) executing 'ls' command...\n", getpid());

        // Use execl to execute the 'ls' command
        execl("/bin/ls", "ls", NULL);

        // If execl fails
        perror("execl failed");
        exit(EXIT_FAILURE);
    } else {
        // Parent process
        printf("Parent Process (PID: %d) going to sleep...\n", getpid());
        sleep(5);  // Sleep for 5 seconds

        // Wait for the child process to finish
        wait(NULL);
        printf("Parent Process (PID: %d) woke up and finished execution.\n", getpid());
    }

    return 0;
}
```

Q.2 Write the simulation program using FCFS. The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1;
 struct process_info *next;
}NODE;

int n;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

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

```
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);

  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);

  p->bt1 = p->bt;

  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
  avg_wt+=wt;

  printf("%s\t%d\t%d\t%d\t%d\t%d\n",
   p->pname,p->at,p->bt,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;
```

```
 p = first;

 printf("pname\tat\tbt\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\n",
   p->pname,p->at,p->bt1);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
  q=p->next;
  while(q!=NULL)
  {
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);

    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;

    t = p->ct;
    p->ct = q->ct;
    q->ct = t;

    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;
   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;
```

```
NODE * get_fcfs()
{
 NODE *p;

 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0)
   return p;

  p=p->next;
 }

 return NULL;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];

int k;

void fcfs()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  p = get_fcfs();

  if(p==NULL)
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;

   prev = time;
   k++;
  }
  else
  {
   time+=p->bt1;
   s[k].start = prev;
   strcpy(s[k].pname, p->pname);
   s[k].end = time;

   prev = time;
   k++;

   p->ct = time;
```

```
   p->bt1 = 0;

   n1++;
  }

  print_input();
  sort();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 fcfs();
 print_output();
 print_gantt_chart();

 return 0;
}
```

//////////////////////////////////////===SLIP NO:-9===//

Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls”

command.

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

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) executing 'ls' command...\n", getpid());

        // Use execl to execute the 'ls' command
        execl("/bin/ls", "ls", NULL);

        // If execl fails
        perror("execl failed");
        exit(EXIT_FAILURE);
    } else {
        // Parent process
        printf("Parent Process (PID: %d) going to sleep for 5 seconds...\n", getpid());
        sleep(5);  // Sleep for 5 seconds

        // Wait for the child process to finish
        wait(NULL);
        printf("Parent Process (PID: %d) woke up and finished execution.\n", getpid());
    }

    return 0;
}
```

Q.2 Write the program to simulate Round Robin (RR) scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Also give the time quantum as input. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.

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

typedef struct process_info
{
 char pname[20];
 int at,bt,ct,bt1;
 struct process_info *next;
}NODE;
```

```
int n,ts;
NODE *first,*last;

void accept_info()
{
 NODE *p;
 int i;

 printf("Enter no.of process:");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
  p = (NODE*)malloc(sizeof(NODE));

  printf("Enter process name:");
  scanf("%s",p->pname);

  printf("Enter arrival time:");
  scanf("%d",&p->at);

  printf("Enter first CPU burst time:");
  scanf("%d",&p->bt);

  p->bt1 = p->bt;

  p->next = NULL;

  if(first==NULL)
   first=p;
  else
   last->next=p;

  last = p;
 }

 printf("Enter time slice:");
 scanf("%d",&ts);
}

void print_output()
{
 NODE *p;
 float avg_tat=0,avg_wt=0;

 printf("pname\tat\tbt\tct\ttat\twt\n");

 p = first;
 while(p!=NULL)
 {
  int tat = p->ct-p->at;
  int wt = tat-p->bt;

  avg_tat+=tat;
  avg_wt+=wt;
```

```
  printf("%s\t%d\t%d\t%d\t%d\t%d\n",
    p->pname,p->at,p->bt,p->ct,tat,wt);

  p=p->next;
 }

 printf("Avg TAT=%f\tAvg WT=%f\n",
   avg_tat/n,avg_wt/n);
}

void print_input()
{
 NODE *p;

 p = first;

 printf("pname\tat\tbt\n");
 while(p!=NULL)
 {
  printf("%s\t%d\t%d\n",
   p->pname,p->at,p->bt1);
  p = p->next;
 }
}

void sort()
{
 NODE *p,*q;
 int t;
 char name[20];

 p = first;
 while(p->next!=NULL)
 {
  q=p->next;
  while(q!=NULL)
  {
   if(p->at > q->at)
   {
    strcpy(name,p->pname);
    strcpy(p->pname,q->pname);
    strcpy(q->pname,name);

    t = p->at;
    p->at = q->at;
    q->at = t;

    t = p->bt;
    p->bt = q->bt;
    q->bt = t;

    t = p->ct;
    p->ct = q->ct;
    q->ct = t;
```

```
    t = p->bt1;
    p->bt1 = q->bt1;
    q->bt1 = t;
   }

   q=q->next;
  }

  p=p->next;
 }
}

int time;

int is_arrived()
{
 NODE *p;

 p = first;
 while(p!=NULL)
 {
  if(p->at<=time && p->bt1!=0)
   return 1;

  p=p->next;
 }

 return 0;
}

NODE * delq()
{
 NODE *t;

 t = first;
 first = first->next;
 t->next=NULL;

 return t;
}

void addq(NODE *t)
{
 last->next = t;
 last = t;
}

struct gantt_chart
{
 int start;
 char pname[30];
 int end;
}s[100],s1[100];
```

```
int k;

void rr()
{
 int prev=0,n1=0;
 NODE *p;

 while(n1!=n)
 {
  if(!is_arrived())
  {
   time++;
   s[k].start = prev;
   strcpy(s[k].pname,"*");
   s[k].end = time;
   k++;
   prev=time;
  }
  else
  {
   p = first;
   while(1)
   {
    if(p->at<=time && p->bt1!=0)
     break;

    p = delq();
    addq(p);
    p = first;
   }

   if(p->bt1<=ts)
   {
    time+=p->bt1;
    p->bt1=0;
   }
   else
   {
    time+=ts;
    p->bt1-=ts;
   }

   p->ct = time;

   s[k].start = prev;
   strcpy(s[k].pname,p->pname);
   s[k].end = time;

   k++;
   prev = time;

   if(p->bt1==0) n1++;

   p = delq();
   addq(p);
```

```
  }

  print_input();
 }
}

void print_gantt_chart()
{
 int i,j,m;

 s1[0] = s[0];

 for(i=1,j=0;i<k;i++)
 {
  if(strcmp(s[i].pname,s1[j].pname)==0)
   s1[j].end = s[i].end;
  else
   s1[++j] = s[i];
 }

 printf("%d",s1[0].start);
 for(i=0;i<=j;i++)
 {
  m = (s1[i].end - s1[i].start);

  for(k=0;k<m/2;k++)
   printf("-");

  printf("%s",s1[i].pname);

  for(k=0;k<(m+1)/2;k++)
   printf("-");

  printf("%d",s1[i].end);
 }
}

int main()
{
 accept_info();
 sort();
 rr();
 print_output();
 print_gantt_chart();

 return 0;
}
```

////////////////////////////////////===SLIP NO:-10===//

Q.1 Write a program to illustrate the concept of orphan process (Using fork() and sleep())

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

```
int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) is running...\n", getpid());

        // Sleep for a while to simulate some work
        sleep(10);
        printf("Child Process (PID: %d) finished execution.\n", getpid());
    } else {
        // Parent process
        printf("Parent Process (PID: %d) will exit...\n", getpid());

        // Sleep for a shorter duration before exiting
        sleep(2);
        printf("Parent Process (PID: %d) is exiting...\n", getpid());
        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames.
Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8
Implement OPT

```
#include<stdio.h>
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter page reference string: ");

    for(i = 0; i < no_of_pages; ++i){
        scanf("%d", &pages[i]);
    }

    for(i = 0; i < no_of_frames; ++i){
        frames[i] = -1;
    }
```

```
for(i = 0; i < no_of_pages; ++i){
    flag1 = flag2 = 0;

    for(j = 0; j < no_of_frames; ++j){
        if(frames[j] == pages[i]){
              flag1 = flag2 = 1;
              break;
          }
    }

    if(flag1 == 0){
        for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
                faults++;
                frames[j] = pages[i];
                flag2 = 1;
                break;
            }
        }
    }

    if(flag2 == 0){
     flag3 =0;

        for(j = 0; j < no_of_frames; ++j){
         temp[j] = -1;

         for(k = i + 1; k < no_of_pages; ++k){
         if(frames[j] == pages[k]){
         temp[j] = k;
         break;
         }
         }
        }

        for(j = 0; j < no_of_frames; ++j){
         if(temp[j] == -1){
         pos = j;
         flag3 = 1;
         break;
         }
        }

        if(flag3 ==0){
         max = temp[0];
         pos = 0;

         for(j = 1; j < no_of_frames; ++j){
         if(temp[j] > max){
         max = temp[j];
         pos = j;
         }
         }
        }
```

```
frames[pos] = pages[i];
faults++;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

////////////////////////////////////===SLIP NO:-11===//

Q.1 Create a child process using fork(), display parent and child process id. Child process will display the message “Hello World” and the parent process should display “Hi”.

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

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d): Hello World\n", getpid());
    } else {
        // Parent process
        printf("Parent Process (PID: %d): Hi\n", getpid());
    }

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.

Reference String: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Implement FIFO

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n;

void accept()
{
```

```
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
}

void fifo()
{
```

```
 int i,j;

 for(i=0;i<m;i++)
 {
  if(search(ref[i])==-1)
  {
   frames[sp] = ref[i];
   sp = (sp+1)%n;
   faults++;
   for(j=0;j<n;j++)
    mem[j][i] = frames[j];

  }
 }
}

int main()
{
 accept();
 fifo();
 disp();

 return 0;
}
```

//===SLIP NO:-12===//

Q.1 [10] Write a program to illustrate the concept of orphan process (Using fork() and sleep()) .

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

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) is running...\n", getpid());

        // Simulate some work with sleep
        sleep(10);
        printf("Child Process (PID: %d) has finished execution.\n", getpid());
    } else {
        // Parent process
        printf("Parent Process (PID: %d) will exit...\n", getpid());

        // Sleep for a shorter duration before exiting
```

```
        sleep(2);
        printf("Parent Process (PID: %d) is exiting...\n", getpid());
        exit(EXIT_SUCCESS);
    }

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string.
Give input n as the number of memory frames.
Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8
Implement OPT

```
#include<stdio.h>
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter page reference string: ");

    for(i = 0; i < no_of_pages; ++i){
        scanf("%d", &pages[i]);
    }

    for(i = 0; i < no_of_frames; ++i){
        frames[i] = -1;
    }

    for(i = 0; i < no_of_pages; ++i){
        flag1 = flag2 = 0;

        for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == pages[i]){
                flag1 = flag2 = 1;
                break;
            }
        }

        if(flag1 == 0){
            for(j = 0; j < no_of_frames; ++j){
                if(frames[j] == -1){
                    faults++;
                    frames[j] = pages[i];
                    flag2 = 1;
                    break;
                }
            }
        }

        if(flag2 == 0){
```

```
        flag3 =0;

          for(j = 0; j < no_of_frames; ++j){
           temp[j] = -1;

           for(k = i + 1; k < no_of_pages; ++k){
           if(frames[j] == pages[k]){
           temp[j] = k;
           break;
           }
           }
          }

          for(j = 0; j < no_of_frames; ++j){
           if(temp[j] == -1){
           pos = j;
           flag3 = 1;
           break;
           }
          }

          if(flag3 ==0){
           max = temp[0];
           pos = 0;

           for(j = 1; j < no_of_frames; ++j){
           if(temp[j] > max){
           max = temp[j];
           pos = j;
           }
           }
          }
frames[pos] = pages[i];
faults++;
       }

       printf("\n");

       for(j = 0; j < no_of_frames; ++j){
          printf("%d\t", frames[j]);
       }
   }

   printf("\n\nTotal Page Faults = %d", faults);

   return 0;
}
```

////////////////////////////////////===SLIP NO:-14===//

Q.1 Write a program to find the execution time taken for execution of a given set of instructions (use clock() function)

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

int main() {
```

```
    // Start measuring time
    clock_t start, end;
    double cpu_time_used;

    start = clock();  // Record the start time

    // Sample set of instructions: a simple loop
    volatile long sum = 0;  // Using volatile to prevent optimization
    for (long i = 0; i < 1e7; i++) {
        sum += i;  // Perform a simple addition
    }

    end = clock();  // Record the end time

    // Calculate the CPU time used
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    // Print the result
    printf("Sum: %ld\n", sum);
    printf("Execution time: %f seconds\n", cpu_time_used);

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Implement FIFO

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n;

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;
```

```
 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
}

void fifo()
{
 int i,j;

 for(i=0;i<m;i++)
 {
  if(search(ref[i])==-1)
  {
   frames[sp] = ref[i];
   sp = (sp+1)%n;
   faults++;
   for(j=0;j<n;j++)
    mem[j][i] = frames[j];

  }
 }
}

int main()
{
 accept();
 fifo();
```

```
 disp();

 return 0;
}
```
////////////////////////////////////===SLIP NO:-15===//
Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

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

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) executing ’ls’ command...\n", getpid());

        // Use execl to execute the ’ls’ command
        execl("/bin/ls", "ls", NULL);

        // If execl fails
        perror("execl failed");
        exit(EXIT_FAILURE);
    } else {
        // Parent process
        printf("Parent Process (PID: %d) going to sleep for 5 seconds...\n", getpid());
        sleep(5);  // Sleep for 5 seconds

        // Wait for the child process to finish
        wait(NULL);
        printf("Parent Process (PID: %d) woke up and finished execution.\n", getpid());
    }

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n as the number of memory frames.

Reference String :7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2

Implement LRU

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
```

```
 sp,m,n,time[MAX];

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
```

```
}

int get_lru()
{
 int i,min_i,min=9999;

 for(i=0;i<n;i++)
 {
  if(time[i]<min)
  {
   min = time[i];
   min_i = i;
  }
 }

 return min_i;
}


void lru()
{
 int i,j,k;

 for(i=0;i<m && sp<n;i++)
 {
  k=search(ref[i]);
  if(k==-1)
  {
   frames[sp]=ref[i];
   time[sp]=i;
   faults++;
   sp++;

   for(j=0;j<n;j++)
    mem[j][i]=frames[j];
  }
  else
   time[k]=i;

 }

 for(;i<m;i++)
 {
  k = search(ref[i]);
  if(k==-1)
  {
   sp = get_lru();
   frames[sp] = ref[i];
   time[sp] = i;
   faults++;

   for(j=0;j<n;j++)
    mem[j][i] = frames[j];
  }
  else
```

```
   time[k]=i;
 }
}
 int main()
{
 accept();
 lru();
 disp();

 return 0;
}
```

////////////////////////////////////===SLIP NO:-16===//

Q.1 Write a program to find the execution time taken for execution of a given set of instructions (use clock()function)

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

int main() {
    // Start measuring time
    clock_t start, end;
    double cpu_time_used;

    // Record the start time
    start = clock();

    // Sample set of instructions: a simple loop
    long sum = 0;  // Variable to hold the sum
    for (long i = 0; i < 1e7; i++) {
        sum += i;  // Perform a simple addition
    }

    // Record the end time
    end = clock();

    // Calculate the CPU time used
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

    // Print the result
    printf("Sum: %ld\n", sum);
    printf("Execution time: %f seconds\n", cpu_time_used);

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames.

Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

Implement OPT

```
#include<stdio.h>
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
```

```
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);

printf("Enter number of pages: ");
scanf("%d", &no_of_pages);

printf("Enter page reference string: ");

for(i = 0; i < no_of_pages; ++i){
    scanf("%d", &pages[i]);
}

for(i = 0; i < no_of_frames; ++i){
    frames[i] = -1;
}

for(i = 0; i < no_of_pages; ++i){
    flag1 = flag2 = 0;

    for(j = 0; j < no_of_frames; ++j){
        if(frames[j] == pages[i]){
              flag1 = flag2 = 1;
              break;
          }
    }

    if(flag1 == 0){
        for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == -1){
                faults++;
                frames[j] = pages[i];
                flag2 = 1;
                break;
            }
        }
    }

    if(flag2 == 0){
     flag3 =0;

        for(j = 0; j < no_of_frames; ++j){
         temp[j] = -1;

         for(k = i + 1; k < no_of_pages; ++k){
         if(frames[j] == pages[k]){
         temp[j] = k;
         break;
         }
         }
        }

        for(j = 0; j < no_of_frames; ++j){
         if(temp[j] == -1){
         pos = j;
         flag3 = 1;
```

```
            break;
            }
           }

           if(flag3 ==0){
            max = temp[0];
            pos = 0;

            for(j = 1; j < no_of_frames; ++j){
            if(temp[j] > max){
            max = temp[j];
            pos = j;
            }
            }
           }
frames[pos] = pages[i];
faults++;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

////////////////////////////////////===SLIP NO:-17===//

Q.1 Write the program to calculate minimum number of resources needed to avoid deadlock.

```
#include <stdio.h>

int main() {
    int num_processes;
    int num_resources;

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

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

    // Calculate the minimum number of resources needed to avoid deadlock
    int min_resources_needed = num_resources + (num_processes - 1);

    // Display the result
    printf("Minimum number of resources needed to avoid deadlock: %d\n", min_resources_needed);

    return 0;
```

}
Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames.
Reference String : 12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8
Implement OPT

```
#include<stdio.h>
int main()
{
    int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
    printf("Enter number of frames: ");
    scanf("%d", &no_of_frames);

    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter page reference string: ");

    for(i = 0; i < no_of_pages; ++i){
        scanf("%d", &pages[i]);
    }

    for(i = 0; i < no_of_frames; ++i){
        frames[i] = -1;
    }

    for(i = 0; i < no_of_pages; ++i){
        flag1 = flag2 = 0;

        for(j = 0; j < no_of_frames; ++j){
            if(frames[j] == pages[i]){
                  flag1 = flag2 = 1;
                  break;
              }
        }

        if(flag1 == 0){
            for(j = 0; j < no_of_frames; ++j){
                if(frames[j] == -1){
                    faults++;
                    frames[j] = pages[i];
                    flag2 = 1;
                    break;
                }
            }
        }

        if(flag2 == 0){
         flag3 =0;

            for(j = 0; j < no_of_frames; ++j){
             temp[j] = -1;
```

```
            for(k = i + 1; k < no_of_pages; ++k){
            if(frames[j] == pages[k]){
            temp[j] = k;
            break;
            }
            }
           }

           for(j = 0; j < no_of_frames; ++j){
            if(temp[j] == -1){
            pos = j;
            flag3 = 1;
            break;
            }
           }

           if(flag3 ==0){
            max = temp[0];
            pos = 0;

            for(j = 1; j < no_of_frames; ++j){
            if(temp[j] > max){
            max = temp[j];
            pos = j;
            }
            }
           }
frames[pos] = pages[i];
faults++;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

////////////////////////////////////===SLIP NO:-20===//

Q.1 Write a program to create a child process using fork().The parent should goto sleep state and child process should begin its execution. In the child process, use execl() to execute the “ls” command.

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

int main() {
    pid_t pid;
```

```
    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        perror("Fork failed");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Child process
        printf("Child Process (PID: %d) executing 'ls' command...\n", getpid());

        // Use execl to execute the 'ls' command
        execl("/bin/ls", "ls", NULL);

        // If execl fails
        perror("execl failed");
        exit(EXIT_FAILURE);
    } else {
        // Parent process
        printf("Parent Process (PID: %d) going to sleep for 5 seconds...\n", getpid());
        sleep(5);  // Sleep for 5 seconds

        // Wait for the child process to finish
        wait(NULL);
        printf("Parent Process (PID: %d) woke up and finished execution.\n", getpid());
    }

    return 0;
}
```

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames.

Reference String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2

i. Implement LRU

```
#include<stdio.h>
#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,
 sp,m,n,time[MAX];

void accept()
{
 int i;

 printf("Enter no.of frames:");
 scanf("%d", &n);

 printf("Enter no.of references:");
 scanf("%d", &m);

 printf("Enter reference string:\n");
 for(i=0;i<m;i++)
 {
```

```
  printf("[%d]=",i);
  scanf("%d",&ref[i]);
 }
}

void disp()
{
 int i,j;

 for(i=0;i<m;i++)
  printf("%3d",ref[i]);

 printf("\n\n");

 for(i=0;i<n;i++)
 {
  for(j=0;j<m;j++)
  {
   if(mem[i][j])
    printf("%3d",mem[i][j]);
   else
    printf("   ");
  }
  printf("\n");
 }

 printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
 int i;

 for(i=0;i<n;i++)
 {
  if(frames[i]==pno)
   return i;
 }

 return -1;
}

int get_lru()
{
 int i,min_i,min=9999;

 for(i=0;i<n;i++)
 {
  if(time[i]<min)
  {
   min = time[i];
   min_i = i;
  }
 }
```

```
 return min_i;
}


void lru()
{
 int i,j,k;

 for(i=0;i<m && sp<n;i++)
 {
  k=search(ref[i]);
  if(k==-1)
  {
   frames[sp]=ref[i];
   time[sp]=i;
   faults++;
   sp++;

   for(j=0;j<n;j++)
    mem[j][i]=frames[j];
  }
  else
   time[k]=i;

 }

 for(;i<m;i++)
 {
  k = search(ref[i]);
  if(k==-1)
  {
   sp = get_lru();
   frames[sp] = ref[i];
   time[sp] = i;
   faults++;

   for(j=0;j<n;j++)
    mem[j][i] = frames[j];
  }
  else
   time[k]=i;
 }
}


int main()
{
 accept();
 lru();
 disp();

 return 0;
}
```