FCFS

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define max 30
void main()
{
Int
i,j,n,bt[max],at[max],wt[max],tat[
max],temp[max];
   float awt=0,atat=0;
   printf("Enter the number of
process:");
   scanf("%d",&n);
   printf("Enter the burst time:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&bt[i]);
   }
   printf("Enter the arrival
time:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&at[i]);
   }
   temp[0]=0;
   printf("\nGantt Chart:\n");
   for (i = 0; i < n; i++) {
      printf("| P%d ", i + 1);
   }
   printf("|\n");
   printf("Process\tBurst
Time\tArrival Time\tWaiting
Time\tTurnaround Time\n");
   for(i=0;i<n;i++)
   {
      wt[i]=0;
      tat[i]=0;
      temp[i+1]=temp[i]+bt[i];
      wt[i]=temp[i]-at[i];
      tat[i]=wt[i]+bt[i];
      awt=awt+wt[i];
      atat=atat+tat[i];

printf("%d\t\t%d\t\t%d\t\t%d\t\t
%d\n",i+1,bt[i],at[i],wt[i],tat[i]);
   }
   awt = awt/n;
   atat = atat/n;
   printf("Average Waiting Time =
%f\n",awt);
   printf("Average Turnaround
Time = %f\n",atat);

}
```

Round Robin

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
   int
i,n,qt,count=0,temp,sq=0,bt[10],
wt[10],tat[10],rem_bt[10];
   float awt=0,atat=0;
   printf("Enter the number of
process:");
   scanf("%d",&n);
   printf("Enter the burst time:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&bt[i]);
      rem_bt[i]=bt[i];
   }
   printf("Enter the quantum
time:");
   scanf("%d",&qt);
   while(1)
   {
      for(i=0,count=0;i<n;i++)
      {
         temp=qt;
         if(rem_bt[i]==0)
         {
            count++;
            continue;
         }
         if(rem_bt[i]>qt)
            rem_bt[i]=rem_bt[i]-qt;
         else
           if(rem_bt[i]>=0)
           {
              temp=rem_bt[i];
              rem_bt[i]=0;
           }
           sq=sq+temp;
           tat[i]=sq;
      }
      if(n==count)
         break;
   }

printf("\nProcess\tBurstTime\tTu
rnAroundTime\tWaitingTime\n");
   for(i=0;i<n;i++)
   {
      wt[i]=tat[i]-bt[i];
      awt=awt+wt[i];
      atat=atat+tat[i];
printf("\n%d\t%d\t\t%d\t\t%d",i
+1,bt[i],tat[i],wt[i]);
   }
   awt=awt/n;
   atat=atat/n;
   printf("\nAverage waiting
time=%f",awt);
   printf("\nAverage turn around
time=%f",atat);
}
```

SJF

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define max 30
void main()
{
   int
i,j,n,temp,p[max],at[max],bt[max
],wt[max],tat[max];
   float awt=0,atat=0;
   printf("Enter the number of
process:");
   scanf("%d",&n);
   printf("Enter the process
number : ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&p[i]);
   }
   printf("Enter the arrival time
for each process:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&at[i]);
   }
   printf("Enter the burst time for
each process:");
   for(i=0;i<n;i++)
   {
      scanf("%d",&bt[i]);
   }
   for(i=0;i<n;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
         if(bt[j]>bt[j+1])
         {
            temp=bt[j];
            bt[j]=bt[j+1];
            bt[j+1]=temp;
            temp=at[j];
            at[j]=at[j+1];
            at[j+1]=temp;
```

```
            temp=p[j];
            p[j]=p[j+1];
            p[j+1]=temp;
          }
       }
    }
    printf("\nGantt Chart:\n");
    for (i = 0; i < n; i++) {
       printf("| P%d ", i + 1);
    }
    printf("|\n");
    printf("\n");
    printf("process \t arrival time
\t burst time \t waiting time \t
turn around time\n");
    for(i=0;i<n;i++)
    {
       wt[i]=0;
       tat[i]=0;
       for(j=0;j<i;j++)
       {
          wt[i]=wt[i]+bt[j];
       }
       tat[i]=wt[i]+bt[i];
       awt=awt+wt[i];
       atat=atat+tat[i];
       printf("%d\t\t %d\t\t %d\t\t
%d\t\t
%d\n",p[i],at[i],bt[i],wt[i],tat[i]);
    }
    awt=awt/n;
    atat=atat/n;
    printf("Average waiting
time=%f \n",awt);
    printf("Average turn around
time=%f \n",atat);
}
```

FIFO

```
#include <stdio.h>
int main()
{
    int i, j, n, ref_string[50],
frame[10], no, k, avail, count = 0;
    printf("\n ENTER THE NUMBER
OF PAGES:\n");
    scanf("%d", &n);
    printf("\n ENTER THE PAGE
NUMBER :\n");
    for (i = 1; i <= n; i++)
       scanf("%d", &ref_string[i]);
    printf("\n ENTER THE NUMBER
OF FRAMES :");
    scanf("%d", &no);
    for (i = 0; i < no; i++)
       frame[i] = -1;
    j = 0;
    printf("\tref string\t page
frames\n");
    for (i = 1; i <= n; i++)
    {
       printf("%d\t\t", ref_string[i]);
       avail = 0;
       for (k = 0; k < no; k++)
          if (frame[k] ==
ref_string[i])
             avail = 1;
       if (avail == 0)
       {
          frame[j] = ref_string[i];
          j = (j + 1) % no;
          count++;
          for (k = 0; k < no; k++)
             printf("%d\t", frame[k]);
       }
       printf("\n");
    }
    printf("Page Fault Is %d",
count);
    return 0;
}
```

LFU

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

#define MAX_PAGES 100

int main() {
    int n, i, j, page_faults = 0;

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

    if (n <= 0) {
       printf("Invalid number of
frames. Exiting...\n");
       return 1;
    }

    int frames[n], counters[n],
reference[MAX_PAGES];

    printf("Enter the reference
string size: ");
    int reference_size;
    scanf("%d", &reference_size);

    printf("Enter the reference
string (space-separated): ");
    for (i = 0; i < reference_size;
i++) {
       scanf("%d", &reference[i]);
    }

    for (i = 0; i < n; i++) {
       frames[i] = -1; // Initialize
frames with -1 to indicate empty
frame
       counters[i] = 0; // Initialize
counters to 0
    }

    printf("\nPage Scheduling:\n");

    for (i = 0; i < reference_size;
i++) {
       int page = reference[i];
       int page_found = 0;
       for (j = 0; j < n; j++) {
          if (frames[j] == page) {
             page_found = 1;
             counters[j]++;
             break;
          }
       }
       if (!page_found) {
          int min_count =
counters[0];
          int min_index = 0;
          for (j = 1; j < n; j++) {
             if (counters[j] <
min_count) {
                min_count =
counters[j];
                min_index = j;
             }
          }

          frames[min_index] = page;
          counters[min_index] = 1;
page_faults++;
          printf("Page %d loaded
into frame %d\n", page,
min_index);
       }
    }

    printf("\nTotal Page Faults:
%d\n", page_faults);

    return 0;
}
```

LRU

```
#include <stdio.h>
main()
{
   int q[20], p[50], c = 0, c1, d, f, i,
j, k = 0, n, r, t, b[20], c2[20];
   printf("Enter no of pages:");
   scanf("%d", &n);
   printf("Enter the reference
string:");
   for (i = 0; i < n; i++)
      scanf("%d", &p[i]);
   printf("Enter no of frames:");
   scanf("%d", &f);
   q[k] = p[k];
   printf("\n\t%d\n", q[k]);
   c++;
   k++;
   for (i = 1; i < n; i++)
   {
      c1 = 0;
      for (j = 0; j < f; j++)
      {
         if (p[i] != q[j])
            c1++;
      }
      if (c1 == f)
      {
         c++;
         if (k < f)
         {
            q[k] = p[i];
            k++;
            for (j = 0; j < k; j++)
               printf("\t%d", q[j]);
            printf("\n");
         }
         else
         {
            for (r = 0; r < f; r++)
            {
               c2[r] = 0;
               for (j = i - 1; j < n; j--)
               {
                  if (q[r] != p[j])
                     c2[r]++;
                  else
                     break;
               }
            }
            for (r = 0; r < f; r++)
               b[r] = c2[r];
            for (r = 0; r < f; r++)
            {
               for (j = r; j < f; j++)
               {
                  if (b[r] < b[j])
                  {
                     t = b[r];
                     b[r] = b[j];
                     b[j] = t;
                  }
               }
            }
            for (r = 0; r < f; r++)
            {
               if (c2[r] == b[0])
                  q[r] = p[i];
               printf("\t%d", q[r]);
            }
            printf("\n");
         }
      }
   }
   printf("\nThe no of page faults
is %d", c);
}
```

MFU

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

void main()
{
   int hit = 0, miss = 0, i, j,
noPages, noFrames, min;
   int frames[10], pages[20];
   int flag = 0, flag1 = 0, flag2 = 0;
   int flagFound = 0;
   int count = 0;
   int frameAge[50],
frameFREQ[50];

   printf("enter number of
frames\n");
   scanf("%d", &noFrames);
   printf("enter number of
pages\n");
   scanf("%d", &noPages);

   printf("enter the page string ");

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

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

   for (j = 0; j < noFrames; j++)
      frameFREQ[j] = 0;

   for (j = 0; j < noPages; j++)
   {
      int index;
      printf(" page:%d   ",
pages[j]);
      flagFound = 0, flag = 0, flag2
= 0;

      for (i = 0; i < noFrames; i++)
      {
         if (frames[i] == pages[j])
         {
            flagFound = 1;
            flag = 1;
            index = i;
            printf("hit ");

            hit++;

            break;
         }
      }

      if (flagFound == 0) // if frame
not found and empty frame
avalible
      {
         for (i = 0; i < noFrames;
i++)
         {
            if (frames[i] == -1)
            {
               frames[i] = pages[j];
               flag = 1;
               count++;
               frameAge[i] = count;

               miss++;
               frameFREQ[i] = 1;
               printf("miss ");
               break;
            }
         }

         if (flag == 0)
         {
            int bestmfu = 0;
            for (i = 0; i < noFrames;
i++)
            {
               if (frameFREQ[i] >
frameFREQ[bestmfu])
                  bestmfu = i;
```

```
            }
            frames[bestmfu] =
pages[j];
            miss++;
            printf("miss ");
            frameFREQ[bestmfu] =
1;
         }

      }       else
      {
         frameFREQ[index]++;
      }

      for (i = 0; i < noFrames; i++)
      {
         printf("   %d ", frames[i]);
      }
      printf("\n");
   }

   printf("number of hits %d\n",
hit);
   printf("number of miss %d\n",
miss);
}
```

MRU

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

void main()
{
   int hit = 0, miss = 0, i, j,
noPages, noFrames, max;
   int frames[10], pages[20];
   int flag = 0, flag1 = 0, flag2 = 0;
   int flagFound = 0;
   int count = 0;
   int frameAge[10];

   printf("enter number of
frames\n");
   scanf("%d", &noFrames);
   printf("enter number of
pages\n");
   scanf("%d", &noPages);

   printf("enter the page string ");

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

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

   printf("\nPageNo    miss/hit
frames");
   printf("\n----------------------------
----------------------\n");
   for (j = 0; j < noPages; j++)
   {
      printf("%3d      ", pages[j]);
      flagFound = 0, flag = 0, flag2
= 0;

      for (i = 0; i < noFrames; i++)
      {
         if (frames[i] == pages[j])
         {
            flagFound = 1;
            flag = 1;
            count++;
            frameAge[i] = count;
            // age frame
            hit++;
            printf("hit \t");
            break;
         }
      }

      if (flagFound == 0) // if frame
not found and empty frame
avalible
      {
         for (i = 0; i < noFrames;
i++)
         {

            if (frames[i] == -1)
            {
               frames[i] = pages[j];

               flag = 1;
               count++;
               frameAge[i] = count;
               printf("miss\t");
               miss++;
               break;
            }
         }
      }

      if (flag == 0) // if frame not
found
      {
         // printf("check\n");
         max = frameAge[0];
         int m = 0;
         for (i = 0; i < noFrames;
i++)
         {
            if (frameAge[i] > max) //
only this line changes for mru
compared to lru
            {
               max = frameAge[i];
               m = i;
            }
         }

         frames[m] = pages[j];
         count++;
         frameAge[m] = count;
         miss++;
         printf("miss\t");
      }

      for (i = 0; i < noFrames; i++)
      {
         printf("   %3d ", frames[i]);
      }
      printf("\n");
   }

   printf("\n----------------------------
----------------------\n");

   printf("number of hits %d\n",
hit);
   printf("number of miss %d\n",
miss);
}
```

SLIP 21

```
// Write a C Program to create a
child process using fork (), display
parent and
// child process id. Child process
will display the message “I am
Child Process"
// and the parent process should
display "I am Parent Process".

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

int main()
{
   pid_t pid;

   pid = fork();
```

```
   if (pid < 0)
   {
      fprintf(stderr, "Fork
failed\n");
      return 1;
   }
   else if (pid == 0)
   {
      printf("I am Child
Process\n");
      printf("Child Process ID:
%d\n", getpid());
   }
   else
   {
      printf("I am Parent
Process\n");
      printf("Parent Process ID:
%d\n", getpid());
   }

   return 0;
}
```

SLIP 22

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

int main()
{
   pid_t pid;

   pid = fork();

   if (pid < 0)
   {
      perror("Fork failed");
      exit(EXIT_FAILURE);
   }
   else if (pid == 0)
   {
      printf("Child process ID:
%d\n", getpid());
      int nice_val = nice(10); // Set
a higher priority (lower nice
value)
      if (nice_val == -1)
      {
         perror("Nice failed");
         exit(EXIT_FAILURE);
      }

      printf("Child process nice
value: %d\n", nice_val);
      printf("Child process is doing
some work...\n");
      sleep(5);

      printf("Child process
exiting.\n");
      exit(EXIT_SUCCESS);
   }
   else
   {
      wait(NULL);
      printf("Parent process ID:
%d\n", getpid());
      printf("Parent process
exiting.\n");
      exit(EXIT_SUCCESS);
   }

   return 0;
}
```

SLIP 23

// Write a C program to illustrate the concept of orphan process. Parent process. creates a child and terminates before child has finished its task. So child process becomes orphan process. (Use fork(), sleep(), getpid(), getppid()).

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

int main() {
   pid_t pid = fork();
   if (pid < 0) {
      perror("Fork failed");
      exit(EXIT_FAILURE);
   }

   if (pid > 0) {
      printf("Parent process (PID:
%d) is sleeping for 2 seconds and
then terminating.\n", getpid());
      sleep(2);
      printf("Parent process is
terminating.\n");
      exit(EXIT_SUCCESS);
   } else {
      printf("Child process (PID:
%d) is doing some task and
sleeping for 5 seconds.\n",
getpid());
      sleep(5);
      printf("Child process is done
with its task.\n");
      printf("Child process (PID:
%d) is now an orphan process
with PPID: %d.\n", getpid(),
getppid());

   }

   return 0;
}
```

SLIP 24

// Write a C program to accept n integers to be sorted. Main function
// creates child process using fork system call. Parent process sorts the integers
// using bubble sort and waits for child process using wait system call. Child
// process sorts the integers using insertion sort.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
void bubbleSort(int arr[], int n);
void insertionSort(int arr[], int n);

int main() {
   int n;

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

   int arr[n];

   printf("Enter %d integers:\n",
n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   pid_t pid = fork();

   if (pid < 0) {
      fprintf(stderr, "Fork
failed\n");
```

```
    exit(EXIT_FAILURE);
  } else if (pid == 0) {
    // Child process
    printf("Child process sorting
using Insertion Sort.\n");
    insertionSort(arr, n);
    printf("Sorted array in child
process:\n");
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
    }
    printf("\n");
  } else {
    wait(NULL);
    printf("Parent process
sorting using Bubble Sort.\n");
    bubbleSort(arr, n);
    printf("Sorted array in
parent process:\n");
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
    }
    printf("\n");
  }

  return 0;
}
Sort
void bubbleSort(int arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        // Swap arr[j] and
arr[j+1]
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}
void insertionSort(int arr[], int n)
{
  int i, key, j;
  for (i = 1; i < n; i++) {
    key = arr[i];
    j = i - 1;

    // Move elements of arr[0..i-
1] that are greater than key to
one position ahead of their
current position
    while (j >= 0 && arr[j] > key)
{
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
}
```

SLIP 25

```
// Write a C program to accept n
integers to be sorted. Main
function
// creates child process using fork
system call. Parent process sorts
the integers
// using bubble sort and waits for
child process using wait system
call. Child
// process sorts the integers
using insertion sort.

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

void bubbleSort(int arr[], int n);
void insertionSort(int arr[], int n);

int main() {
  int n;

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

  int arr[n];

  printf("Enter %d integers:\n",
n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
  }

  pid_t pid = fork();

  if (pid < 0) {
    fprintf(stderr, "Fork
failed\n");
    exit(EXIT_FAILURE);
  } else if (pid == 0) {
    printf("Child process sorting
using Insertion Sort.\n");
    insertionSort(arr, n);
    printf("Sorted array in child
process:\n");
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
    }
    printf("\n");
  } else {
    wait(NULL);
    printf("Parent process
sorting using Bubble Sort.\n");
    bubbleSort(arr, n);
    printf("Sorted array in
parent process:\n");
    for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]);
    }
    printf("\n");
  }

  return 0;
}
void bubbleSort(int arr[], int n) {
  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        int temp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = temp;
      }
    }
  }
}
void insertionSort(int arr[], int n)
{
  int i, key, j;
  for (i = 1; i < n; i++) {
    key = arr[i];
    j = i - 1;
    while (j >= 0 && arr[j] > key)
{
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
}
```