slip 1

1->

```
#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 the number of frames: ");
   scanf("%d", &n);
   printf("Enter the number of references: ");
   scanf("%d", &m);
   printf("Enter the 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 lfu()
{
   int i, j, k, count[20];
   for (i = 0; i < m && sp < n; i++)
   {
      k = search(ref[i]);
      if (k == -1)
      {
         frames[sp] = ref[i];
         time[sp] = 1;
         count[sp] = 1;
         faults++;
         sp++;
         for (j = 0; j < n; j++)
            mem[j][i] = frames[j];
      }
      else
      {
         time[k] = i;
         count[k]++;
      }
   }
   for (; i < m; i++)
   {
      k = search(ref[i]);
      if (k == -1)
      {
         int min_i = 0, min = 9999;
         for (j = 0; j < n; j++)
         {
            if (count[j] < min)
            {
               min = count[j];
               min_i = j;
            }
         }
         sp = min_i;
         frames[sp] = ref[i];
         time[sp] = i;
         count[sp] = 1;
         faults++;
         for (j = 0; j < n; j++)
            mem[j][i] = frames[j];
      }
      else
      {
         time[k] = i;
         count[k]++;
      }
   }
}
int main()
{
   accept();
   lfu();
   disp();
   return 0;
}
```

2->

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char *s, char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, " ");
   }
   tok[i] = NULL;
}
int main()
{
   char buff[80], *args[10];
   int pid;
   while (1)
   {
      printf("myshell$ ");
      fflush(stdin);
      fgets(buff, 80, stdin);
      buff[strlen(buff) - 1] = '\0';
      make_toks(buff, args);
      if (strcmp(args[0], "typeline") == 0)
      {
         if (args[1][0] == '+')
         {
            int n = atoi(args[1] + 1);

            FILE *fp = fopen(args[2], "r");
            if (fp == NULL)
            {

printf("Error opening file %s.\n", args[2]);
               return 1;
            }

            char line[80];
            int count = 0;
            while (count < n && fgets(line, 80, fp) != NULL)
            {
               printf("%s", line);
               count++;
            }

            fclose(fp);
         }

         else if (strcmp(args[2], "-a") == 0)
         {

            FILE *fp = fopen(args[2], "r");
            if (fp == NULL)
```

```
        {
printf("Error opening
file %s.\n", args[2]);
            return 1;
        }

        char line[80];
        while
(fgets(line, 80, fp) !=
NULL)
            printf("%s",
line);

        fclose(fp);
      }
      else
      {
        printf("Invalid
option.\n");
      }
    }
  }
}
```

slip 2

1->

```
#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;
}
```

2->

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
void make_toks(char
*s, char *tok[])
{
  int i = 0;
  char *p;
  p = strtok(s, " ");
  while (p != NULL)
  {
    tok[i++] = p;
    p = strtok(NULL, "
");
  }
  tok[i] = NULL;
}
void list(char
*dirname, char op)
{
  DIR *dir;
  struct dirent *entry;
  int count = 0;

  dir =
opendir(dirname);
  if (dir == NULL)
  {
    printf("Directory
%s not found.\n",
dirname);
    return;
  }
  while ((entry =
readdir(dir)) != NULL)
  {
    switch (op)
    {
    case 'f':
      printf("%s\n",
entry->d_name);
      break;
    case 'n':
      count++;
      break;
    }
  }

  closedir(dir);
  if (op == 'n')
  {
    printf("Number of
entries: %d\n", count);
  }
}
int main()
{
  char buff[80],
*args[10];
  int pid;
  while (1)
  {
    printf("myshell$
");
    fflush(stdin);
    fgets(buff, 80,
stdin);
    buff[strlen(buff) -
1] = '\0';
    make_toks(buff,
args);
    if (strcmp(args[0],
"list") == 0)
    {
      list(args[2],
args[1][0]);
    }
    else
    {
      pid = fork();
      if (pid > 0)
        wait();
      else
      {
        if
(execvp(args[0], args)
== -1)
          printf("Bad
command.\n");
      }
    }
  }
```

```
   return 0;
}

slip 3
1->
#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;
}

2->
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char
*s, char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void count(char *fn,
char op)
{
   FILE *fh;
   int cc = 0, wc = 0, lc =
0;
   char c;
   fh = fopen(fn, "r");
   if (fh == NULL)
   {
      printf("File %s not
found.\n", fn);
      return;
   }
   while ((c = fgetc(fh))
!= EOF)
   {
      if (c == ' ')
         wc++;
      else if (c == '\n')
      {
         wc++;
         lc++;
      }
      cc++;
   }
   fclose(fh);
   switch (op)
   {
   case 'c':
      printf("No.of
characters:%d\n", cc -
1);
      break;
   case 'w':
      printf("No.of
words:%d\n", wc);
      break;
   case 'l':
      printf("No.of
lines:%d\n", lc + 1);
      break;
   }
}
int main()
{
   char buff[80],
      *args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"count") == 0)
```

```
        count(args[2],
args[1][0]);
      else
      {
        pid = fork();
        if (pid > 0)
          wait();
        else
        {
          if
(execvp(args[0], args)
== -1)
            printf("Bad
command.\n");
        }
      }
   }
   return 0;
}
```

slip 4

1->

```
#include <stdio.h>
#define MAX 20
int frames[MAX],
ref[MAX],
mem[MAX][MAX],
faults,
   sp, m, n,
count[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_mfu(int sp)
{
   int i, max_i, max = -
9999;
   i = sp;
   do
   {
      if (count[i] > max)
      {
        max = count[i];
        max_i = i;
      }
      i = (i + 1) % n;
   } while (i != sp);
   return max_i;
}
void mfu()
{
   int i, j, k;
   for (i = 0; i < m &&
sp < n; i++)
   {
      k = search(ref[i]);
      if (k == -1)
      {
        frames[sp] =
ref[i];
        count[sp]++;
        faults++;
        sp++;
        for (j = 0; j < n;
j++)
          mem[j][i] =
frames[j];
      }
      else
        count[k]++;
   }
   sp = 0;
   for (; i < m; i++)
   {
      k = search(ref[i]);
      if (k == -1)
      {
        sp =
get_mfu(sp);
        frames[sp] =
ref[i];
        count[sp] = 1;
        faults++;
        sp = (sp + 1) %
n;
        for (j = 0; j < n;
j++)
          mem[j][i] =
frames[j];
      }
      else
        count[k]++;
   }
}

int main()
{
   accept();
   mfu();
   disp();
   return 0;
}
```

2->

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
   void
   make_toks(char *s,
char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void search(char *fn,
char op, char *pattern)
{
   FILE *fh;
   int count = 0;
   char line[80];
   fh = fopen(fn, "r");
   if (fh == NULL)
   {
      printf("File %s not
found.\n", fn);
      return;
   }
   while (fgets(line, 80,
fh) != NULL)
   {
      if (strstr(line,
pattern) != NULL)
      {
        count++;
        if (op == 'a')
          printf("%s",
line);
      }
   }
   fclose(fh);
   if (op == 'c')
      printf("Number of
occurrences: %d\n",
count);
}
int main()
{
   char buff[80],
*args[10];
```

```
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"search") == 0)
         search(args[2],
args[1][0], args[3]);
      else
      {
         pid = fork();
         if (pid > 0)
            wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}
```

slip 5

1->

```
#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;
}
```

2->

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
   void
   make_toks(char *s,
char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void search(char *fn,
char op, char *pattern)
{
   FILE *fh;
   int count = 0;
   char line[80];
   fh = fopen(fn, "r");
   if (fh == NULL)
   {
```

```
      printf("File %s not
found.\n", fn);
      return;
   }
   if (op == 'f')
   {

      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {

            printf("%s",
line);
            break;
         }
      }
   }
   else
   {

      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {
            count++;
            if (op == 'a')
               printf("%s",
line);
         }
      }
      if (op == 'c')
         printf("Number
of occurrences: %d\n",
count);
   }
   fclose(fh);
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"search") == 0)
         search(args[2],
args[1][0], args[3]);
      else
      {
         pid = fork();
         if (pid > 0)
            wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}
```

slip 6

1->

```
#include <stdio.h>
#define MAX 20
int frames[MAX],
ref[MAX],
mem[MAX][MAX],
faults,
   sp, m, n,
count[MAX],
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_mru()
{
   int i, max_i, max = 0;
   for (i = 0; i < n; i++)
   {
      if (time[i] > max)
      {
         max = time[i];
         max_i = i;
      }
   }
   return max_i;
}
void mru()
{
   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 = 0; i < m; i++)
   {
      k = search(ref[i]);
      if (k == -1)
      {
         sp = get_mru();
         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();
   mru();
   disp();
   return 0;
}
```

2->

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

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
   void
   make_toks(char *s,
char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void search(char *fn,
char op, char *pattern)
{
   FILE *fh;
   int count = 0;
   char line[80];
   fh = fopen(fn, "r");
   if (fh == NULL)
   {
      printf("File %s not
found.\n", fn);
      return;
   }
   if (op == 'f')
   {
      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {
            // Print the
line and exit the loop
            printf("%s",
line);
            break;
         }
      }
   }
   else
   {
      // Continue with
the existing search
implementation
      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {
            count++;
            if (op == 'a')
               printf("%s",
line);
         }
      }
      if (op == 'c')
         printf("Number
of occurrences: %d\n",
count);
   }
   fclose(fh);
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"search") == 0)
         search(args[2],
args[1][0], args[3]);
      else
      {
         pid = fork();
         if (pid > 0)
            wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}
```

slip 7

1->

```
#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;
}
```

2->

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
 void
 make_toks(char *s,
char *tok[])
{
 int i = 0;
 char *p;
 p = strtok(s, " ");
 while (p != NULL)
 {
   tok[i++] = p;
   p = strtok(NULL, "
");
 }
 tok[i] = NULL;
}
void search(char *fn,
char op, char *pattern)
{
 FILE *fh;
 int count = 0;
 char line[80];
 fh = fopen(fn, "r");
 if (fh == NULL)
 {
   printf("File %s not
found.\n", fn);
   return;
 }
 if (op == 'f')
 {
   while (fgets(line,
80, fh) != NULL)
   {
     if (strstr(line,
pattern) != NULL)
     {
       // Print the
line and exit the loop
       printf("%s",
line);
       break;
     }
   }
 }
 else
 {
   while (fgets(line,
80, fh) != NULL)
   {
     if (strstr(line,
pattern) != NULL)
     {
       count++;
       if (op == 'a')
         printf("%s",
line);
     }
   }
   if (op == 'c')
     printf("Number
of occurrences: %d\n",
count);
 }
 fclose(fh);
}
int main()
{
 char buff[80],
*args[10];
 int pid;
 while (1)
 {
   printf("myshell$
");
   fflush(stdin);
   fgets(buff, 80,
stdin);
   buff[strlen(buff) -
1] = '\0';
   make_toks(buff,
args);
   if (strcmp(args[0],
"search") == 0)
     search(args[2],
args[1][0], args[3]);
   else
   {
     pid = fork();
     if (pid > 0)
       wait();
     else
     {
       if
(execvp(args[0], args)
== -1)
         printf("Bad
command.\n");
     }
   }
 }
 return 0;
}
```

slip 8

1->

```
#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 the
number of frames: ");
 scanf("%d", &n);
 printf("Enter the
number of references:
");
 scanf("%d", &m);
 printf("Enter the
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 lfu()
{
   int i, j, k, count[20];
   for (i = 0; i < m &&
sp < n; i++)
   {
      k = search(ref[i]);
      if (k == -1)
      {
         frames[sp] =
ref[i];
         time[sp] = 1;
         count[sp] = 1;
         faults++;
         sp++;
         for (j = 0; j < n;
j++)
            mem[j][i] =
frames[j];
      }
      else
      {
         time[k] = i;
         count[k]++;
      }
   }
   for (; i < m; i++)
   {
      k = search(ref[i]);
      if (k == -1)
      {
         int min_i = 0,
min = 9999;
         for (j = 0; j < n;
j++)
         {
            if (count[j] <
min)
            {
               min =
count[j];
               min_i = j;
            }
         }
         sp = min_i;
         frames[sp] =
ref[i];
         time[sp] = i;
         count[sp] = 1;
         faults++;
         for (j = 0; j < n;
j++)
            mem[j][i] =
frames[j];
      }
      else
      {
         time[k] = i;
         count[k]++;
      }
   }
}
int main()
{
   accept()
      lfu();
   disp();
   return 0;
}

2->
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char
*s, char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void search(char *fn,
char op, char *pattern)
{
   FILE *fh;
   int count = 0;
   char line[80];
   fh = fopen(fn, "r");
   if (fh == NULL)
   {
      printf("File %s not
found.\n", fn);
      return;
   }
   if (op == 'f')
   {
      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {
            // Print the
line and exit the loop
            printf("%s",
line);
            break;
         }
      }
   }
   else
   {
      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {
            count++;
            if (op == 'a')
               printf("%s",
line);
         }
      }
      if (op == 'c')
         printf("Number
of occurrences: %d\n",
count);
   }
   fclose(fh);
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"search") == 0)
         search(args[2],
args[1][0], args[3]);
      else
      {
         pid = fork();
         if (pid > 0)
            29345436
wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}



slip 9
1->
#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;
}

2->
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
   void
   make_toks(char *s,
char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void search(char *fn,
char op, char *pattern)
{
   FILE *fh;
   int count = 0;
   char line[80];
   fh = fopen(fn, "r");
   if (fh == NULL)
   {
      printf("File %s not
found.\n", fn);
      return;
   }
   if (op == 'f')
   {
      // Search for the
first occurrence of the
pattern
      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {
            // Print the
line and exit the loop
            printf("%s",
line);
            break;
         }
      }
   }
   else
   {
      while (fgets(line,
80, fh) != NULL)
      {
         if (strstr(line,
pattern) != NULL)
         {
            count++;
            if (op == 'a')
               printf("%s",
line);
         }
      }
      if (op == 'c')
         printf("Number
of occurrences: %d\n",
count);
   }
   fclose(fh);
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"search") == 0)
         search(args[2],
args[1][0], args[3]);
      else
      {
         pid = fork();
         if (pid > 0)
            wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}


slip 10
1->
#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;
}

2->
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
   void
   make_toks(char *s,
char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void list(char
*dirname, char op)
{
   DIR *dir;
   struct dirent *entry;
   int count = 0;
   dir =
opendir(dirname);
   if (dir == NULL)
   {
      printf("Directory
%s not found.\n",
dirname);
      return;
   }
   // Iterate through
the entries in the
directory
   while ((entry =
readdir(dir)) != NULL)
   {
      switch (op)
      {
      case 'f':
         // Display the
filename
         printf("%s\n",
entry->d_name);
         break;
      case 'n':
         count++;
         break;
      case 'i':
         printf("%s:
%lu\n", entry-
>d_name, entry-
>d_ino);
         break;
      }
   }

   closedir(dir);
   if (op == 'n')
   {
      printf("Number of
entries: %d\n", count);
   }
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"list") == 0)
      {
         list(args[2],
args[1][0]);
      }
      else
      {
         pid = fork();
         if (pid > 0)
            wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}


slip 11
1->
#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 the
number of frames: ");
   scanf("%d", &n);
   printf("Enter the
number of references:
");
   scanf("%d", &m);
   printf("Enter the
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 lfu()
{
  int i, j, k, count[20];
  for (i = 0; i < m &&
sp < n; i++)
  {
    k = search(ref[i]);
    if (k == -1)
    {
      frames[sp] =
ref[i];
      time[sp] = 1;
      count[sp] = 1;
      faults++;
      sp++;
      for (j = 0; j < n;
j++)
        mem[j][i] =
frames[j];
    }
    else
    {
      time[k] = i;
      count[k]++;
    }
  }
  for (; i < m; i++)
  {
    k = search(ref[i]);
    if (k == -1)
    {
      int min_i = 0,
min = 9999;
      for (j = 0; j < n;
j++)
      {
        if (count[j] <
min)
        {
          min =
count[j];
          min_i = j;
        }
      }
      sp = min_i;
      frames[sp] =
ref[i];
      time[sp] = i;
      count[sp] = 1;
      faults++;
      for (j = 0; j < n;
j++)
        mem[j][i] =
frames[j];
    }
    else
    {
      time[k] = i;
      count[k]++;
    }
  }
}
int main()
{
  accept();
  lfu();
  disp();
  return 0;
}

2->
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
  void
  make_toks(char *s,
char *tok[])
{
  int i = 0;
  char *p;
  p = strtok(s, " ");
  while (p != NULL)
  {
    tok[i++] = p;
    p = strtok(NULL, "
");
  }
  tok[i] = NULL;
}
void list(char
*dirname, char op)
{
  DIR *dir;
  struct dirent *entry;
  int count = 0;
  // Open the
directory
  dir =
opendir(dirname);
  if (dir == NULL)
  {
    printf("Directory
%s not found.\n",
dirname);
    return;
  }
  while ((entry =
readdir(dir)) != NULL)
  {
    switch (op)
    {
    case 'f':
      printf("%s\n",
entry->d_name);
      break;
    case 'n':
      count++;
      break;
    case 'i':
      printf("%s:
%lu\n", entry-
>d_name, entry-
>d_ino);
      break;
    }
  }
  closedir(dir);
  if (op == 'n')
  {
    printf("Number of
entries: %d\n", count);
  }
}
int main()
{
  char buff[80],
*args[10];
  int pid;
  while (1)
  {
    printf("myshell$
");
    fflush(stdin);
    fgets(buff, 80,
stdin);
    buff[strlen(buff) -
1] = '\0';
    make_toks(buff,
args);
    if (strcmp(args[0],
"list") == 0)
    {
      list(args[2],
args[1][0]);
    }
    else
    {
      pid = fork();
      if (pid > 0)
        wait();
      else
      {
        if
(execvp(args[0], args)
== -1)
          printf("Bad
command.\n");
      }
    }
  }
  return 0;
}


slip 12
1->

#include <stdio.h>
#define MAX 20
int
frames[MAX],ref[MAX]
,mem[MAX][MAX],faul
ts,
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 = 0; 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;
}
```

2->

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
   void
   make_toks(char *s,
char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
void list(char
*dirname, char op)
{
   DIR *dir;
   struct dirent *entry;
   int count = 0;
   dir =
opendir(dirname);
   if (dir == NULL)
   {
      printf("Directory
%s not found.\n",
dirname);
      return;
   }
   while ((entry =
readdir(dir)) != NULL)
   {
      switch (op)
      {
      case 'f':
         // Display the
filename
         printf("%s\n",
entry->d_name);
         break;
      case 'n':
         // Increment
the count
         count++;
         break;
      case 'i':
         // Display the
filename and inode
number
         printf("%s:
%lu\n", entry-
>d_name, entry-
>d_ino);
         break;
      }
   }

   closedir(dir);

   if (op == 'n')
   {
      printf("Number of
entries: %d\n", count);
   }
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"list") == 0)
      {
         list(args[2],
args[1][0]);
```

```
    }
    else
    {
      pid = fork();
      if (pid > 0)
        wait();
      else
      {
        if
(execvp(args[0], args)
== -1)
          printf("Bad
command.\n");
      }
    }
  }
  return 0;
}


slip 13
1->
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char
*s, char *tok[])
{
  int i = 0;
  char *p;
  p = strtok(s, " ");
  while (p != NULL)
  {
    tok[i++] = p;
    p = strtok(NULL, "
");
  }
  tok[i] = NULL;
}
int main()
{
  char buff[80],
*args[10];
  int pid;
  while (1)
  {
    printf("myshell$
");
    fflush(stdin);
    fgets(buff, 80,
stdin);
    buff[strlen(buff) -
1] = '\0';
    make_toks(buff,
args);
    if (strcmp(args[0],
"typeline") == 0)
    {
      if (args[1][0] ==
'+')
      {
        // Read the
line count
        int n =
atoi(args[1] + 1);
        // Open the
file
        FILE *fp =
fopen(args[2], "r");
        if (fp == NULL)
        {

printf("Error opening
file %s.\n", args[2]);
          return 1;
        }
        // Read and
print the first n lines
        char line[80];
        int count = 0;
        while (count <
n && fgets(line, 80, fp)
!= NULL)
        {
          printf("%s",
line);
          count++;
        }
        // Close the
file
        fclose(fp);
      }
      else if
(strcmp(args[2], "-a")
== 0)
      {
        // Open the
file
        FILE *fp =
fopen(args[2], "r");
        if (fp == NULL)
        {

printf("Error opening
file %s.\n", args[2]);
          return 1;
        }
        char line[80];
        while
(fgets(line, 80, fp) !=
NULL)
          printf("%s",
line);
        fclose(fp);
      }
      else
      {
        printf("Invalid
option.\n");
      }
    }
  }
}

2->
#include <stdio.h>
#include <string.h>
struct job
{
  char name[20];
  int at, bt, ct, tat, wt,
st, tbt;
} job[10];
int n, i, j, tq;
float avg_tat = 0;
float avg_wt = 0;
// to accept the info
about processes
take_input()
{
  printf("Enter the no
of jobs : ");
  scanf("%d", &n);
  printf("Enter the
time Quantum: ");
  scanf("%d", &tq);
  for (i = 0; i < n; i++)
  {
    printf("Enter the
name of the job: ");
    scanf("%s",
&job[i].name);
    printf("Enter the
arrival time of the job :
");
    scanf("%d",
&job[i].at);
    printf("Enter the
burst time of the job:
");
    scanf("%d",
&job[i].bt);
    job[i].tbt =
job[i].bt;
    printf("\n\n");
  }
}

// to sort the
processes by arriaval
time
sort()
{
  struct job temp;
  for (i = 0; i < n; i++)
  {
    for (j = i + 1; j < n;
j++)
    {
      if (job[i].at >
job[j].at)
      {
        temp = job[i];
        job[i] = job[j];
        job[j] = temp;
      }
    }
  }
}
// to calculate the tat n
wt
void process()
{
  int jno = 0, cnt = 0,
time = job[0].at;

printf("\n**********
Gantt
chart*********\n");
  while (1)
  {
    if (job[jno].tbt !=
0)
    {
      printf("|%d %s
", time,
job[jno].name);
```

```
        if (job[jno].tbt
>= tq)
        {
           job[jno].tbt =
job[jno].tbt - tq;
           time = time +
tq;
        }
        else
        {
           time = time +
job[jno].tbt;
           job[jno].tbt =
0;
        }
        printf("%d |",
time);
        if (job[jno].tbt
== 0)
        {
           job[jno].ct =
time;
           job[jno].tat =
time - job[jno].at;
           job[jno].wt =
job[jno].tat -
job[jno].bt;
           cnt++;
        }
     }
     jno++;
     if (jno == n)
     {
        jno = 0;
     }
     if (cnt == n)
        break;
  }
}
// to print the output
table
void print_output()
{
  printf("\n\n");
  printf("\n----------
------------------------");
  printf("\n pname
at     bt     tat     wt ");
  printf("\n----------
------------------------");
  for (i = 0; i < n; i++)
  {
     printf("\n%s    %d
%d     %d     %d ",
job[i].name, job[i].at,
job[i].bt, job[i].tat,
job[i].wt);
     avg_tat = avg_tat
+ (float)(job[i].tat);
     avg_wt =
(float)avg_wt +
(float)(job[i].wt);
  }
  printf("\n----------
--------------------------");
  printf("\nThe avg of
the turn around time is
%f", avg_tat / n);
  printf("\nThe avg of
the waiting time is %f",
avg_wt / n);
}
main()
{
  take_input();
  process();
  print_output();
  for (i = 0; i < n; i++)
  {
     job[i].tbt =
job[i].bt = rand() % 10
+ 1;
     job[i].at = job[i].ct
+ 2;
  }
  process();
  print_output();
}

slip 14
1->
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char
*s, char *tok[])
{
  int i = 0;
  char *p;
  p = strtok(s, " ");
  while (p != NULL)
  {
     tok[i++] = p;
     p = strtok(NULL, "
");
  }
  tok[i] = NULL;
}
int main()
{
  char buff[80],
*args[10];
  int pid;
  while (1)
  {
     printf("myshell$
");
     fflush(stdin);
     fgets(buff, 80,
stdin);
     buff[strlen(buff) -
1] = '\0';
     make_toks(buff,
args);
     if (strcmp(args[0],
"typeline") == 0)
     {
        // Check if the
'+n' option is specified
        if (args[1][0] ==
'+')
        {
           // Read the
line count
           int n =
atoi(args[1] + 1);
           // Open the
file
           FILE *fp =
fopen(args[2], "r");
           if (fp == NULL)
           {

printf("Error opening
file %s.\n", args[2]);
              return 1;
           }
           // Read and
print the first n lines
           char line[80];
           int count = 0;
           while (count <
n && fgets(line, 80, fp)
!= NULL)
           {
              printf("%s",
line);
              count++;
           }
           // Close the
file
           fclose(fp);
        }
        // If the '-a'
option is specified
        else if
(strcmp(args[2], "-a")
== 0)
        {
           // Open the
file
           FILE *fp =
fopen(args[2], "r");
           if (fp == NULL)
           {

printf("Error opening
file %s.\n", args[2]);
              return 1;
           }
           // Read and
print the entire file
           char line[80];
           while
(fgets(line, 80, fp) !=
NULL)
              printf("%s",
line);
           // Close the
file
           fclose(fp);
        }
        else
        {
           printf("Invalid
option.\n");
        }
     }
  }
}

2->
#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 15
1->
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
void make_toks(char
*s, char *tok[])
{
    int i = 0;
    char *p;
    p = strtok(s, " ");
    while (p != NULL)
    {
        tok[i++] = p;
        p = strtok(NULL, "
");
    }
    tok[i] = NULL;
}
void list(char
*dirname, char op)
{
    DIR *dir;
    struct dirent *entry;
    int count = 0;
    // Open the
directory
    dir =
opendir(dirname);
    if (dir == NULL)
    {
        printf("Directory
%s not found.\n",
dirname);
        return;
    }
    // Iterate through
the entries in the
directory
    while ((entry =
readdir(dir)) != NULL)
    {
        switch (op)
        {
        case 'f':
            // Display the
filename
            printf("%s\n",
entry->d_name);
            break;
        case 'n':
            // Increment
the count
            count++;
            break;
        case 'i':
            // Display the
filename and inode
number
            printf("%s:
%lu\n", entry-
>d_name, entry-
>d_ino);
            break;
        }
    }
    // Close the
directory
    closedir(dir);
    if (op == 'n')
    {
        printf("Number of
entries: %d\n", count);
    }
}
int main()
{
    char buff[80],
*args[10];
    int pid;
    while (1)
    {
        printf("myshell$
");
        fflush(stdin);
        fgets(buff, 80,
stdin);
        buff[strlen(buff) -
1] = '\0';
        make_toks(buff,
args);
        if (strcmp(args[0],
"list") == 0)
        {
            list(args[2],
args[1][0]);
        }
        else
        {
            pid = fork();
            if (pid > 0)
                wait();
            else
            {
                if
(execvp(args[0], args)
== -1)
                    printf("Bad
command.\n");
            }
        }
    }
    return 0;
}

2->
#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\t\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 sjfp()
{
   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++;
         s[k].start = prev;

strcpy(s[k].pname, p->pname);
         s[k].end = time;
         prev = time;
         k++;
         p->ct = time;
         p->bt1--;
         if (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();
   sjfp();
   print_output();
   print_gantt_chart();
   return 0;
}
```

slip 16

1->

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char *s, char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, " ");
   }
   tok[i] = NULL;
```

```
}
void count(char *fn,
char op)
{
   FILE *fh;
   int cc = 0, wc = 0, lc =
0;
   char c;
   fh = fopen(fn, "r");
   if (fh == NULL)
   {
      printf("File %s not
found.\n", fn);
      return;
   }
   while ((c = fgetc(fh))
!= EOF)
   {
      if (c == ' ')
         wc++;
      else if (c == '\n')
      {
         wc++;
         lc++;
      }
      cc++;
   }
   fclose(fh);
   switch (op)
   {
   case 'c':
      printf("No.of
characters:%d\n", cc -
1);
      break;
   case 'w':
      printf("No.of
words:%d\n", wc);
      break;
   case 'l':
      printf("No.of
lines:%d\n", lc + 1);
      break;
   }
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"count") == 0)
         count(args[2],
args[1][0]);
      else
      {
         pid = fork();
         if (pid > 0)
            wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}

Q .2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
   typedef struct
process_info
{
   char pname[20];
   int at, bt, ct, bt1, p;
   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);
      printf("Enter
priority:");
      scanf("%d", &p-
>p);
      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\
tp\ttct\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\t%d\n",
            p->pname, p-
>at, p->bt, p->p, 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\
tp\n");
   while (p != NULL)
   {

printf("%s\t%d\t%d\t%
d\n",
            p->pname, p-
>at, p->bt1, p->p);
      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;
            t = p->p;
            p->p = q->p;
            q->p = t;
         }
         q = q->next;
      }

      p = p->next;
   }
}
int time;
NODE *get_p()
{
   NODE *p, *min_p = NULL;
   int min = 9999;
   p = first;
   while (p != NULL)
   {
      if (p->at <= time && p->bt1 != 0 &&
         p->p < min)
      {
         min = p->p;
         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 pnp()
{
   int prev = 0,
      n1 = 0;
   NODE *p;
   while (n1 != n)
   {
      p = get_p();
      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();
   pnp();
   print_output();
   print_gantt_chart();
   return 0;
}
```

slip 17

1->

```
#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;
}

2->
#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 18
1->
#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 = 0; 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;
}

2->

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
   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 19

1->

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
void make_toks(char
*s, char *tok[])
{
int i=0;
char *p;
p = strtok(s," ");
while(p!=NULL)
{
 tok[i++]=p;
 p=strtok(NULL," ");
}
tok[i]=NULL;
}
void list(char
*dirname, char op)
{
 DIR *dir;
 struct dirent *entry;
 int count = 0;
 // Open the directory
 dir =
opendir(dirname);
 if (dir == NULL)
 {
 printf("Directory %s
not found.\n",
dirname);
 return;
 }
 // Iterate through the
entries in the directory
 while ((entry =
readdir(dir)) != NULL)
 {
 switch (op)
 {
 case 'f':
 // Display the
filename
 printf("%s\n", entry-
>d_name);
 break;
 case 'n':
 // Increment the
count
 count++;
 break;
 case 'i':
 // Display the
filename and inode
number
 printf("%s: %lu\n",
entry->d_name, entry-
>d_ino);
 break;
 }
 }
 closedir(dir);
// If the 'n' option was
specified, print the
count
 if (op == 'n')
 {
 printf("Number of
entries: %d\n", count);
 }
}
int main()
{
char
buff[80],*args[10];
int pid;
while(1)
{
 printf("myshell$ ");
 fflush(stdin);
 fgets(buff,80,stdin);
 buff[strlen(buff)-
1]='\0';
 make_toks(buff,args);
if (strcmp(args[0],
"list") == 0)
{
 list(args[2],
args[1][0]);
}
else
{
 pid = fork();
 if (pid > 0)
 wait();
 else
 {
 if (execvp(args[0],
args) == -1)
 printf("Bad
command.\n");
 }
}
}
return 0;
}
```

2->

```
#include <stdio.h>
#include <string.h>
struct job
{
   char name[20];
   int at, bt, ct, tat, wt,
st, tbt;
} job[10];
int n, i, j, tq;
float avg_tat = 0;
float avg_wt = 0;
// to accept the info
about processes
take_input()
{
   printf("Enter the no
of jobs : ");
   scanf("%d", &n);
   printf("Enter the
time Quantum: ");
   scanf("%d", &tq);
   for (i = 0; i < n; i++)
   {
      printf("Enter the
name of the job: ");
      scanf("%s",
&job[i].name);
      printf("Enter the
arrival time of the job :
");
      scanf("%d",
&job[i].at);
      printf("Enter the
burst time of the job:
");
      scanf("%d",
&job[i].bt);
      job[i].tbt =
job[i].bt;
      printf("\n\n");
   }
}

// to sort the
processes by arriaval
time
sort()
{
   struct job temp;
   for (i = 0; i < n; i++)
   {
      for (j = i + 1; j < n;
j++)
      {
         if (job[i].at >
job[j].at)
         {
            temp = job[i];
            job[i] = job[j];
            job[j] = temp;
```

```
        }
      }
   }
}
// to calculate the tat n
wt
void process()
{
   int jno = 0, cnt = 0,
time = job[0].at;

printf("\n**********
Gantt
chart*********\n");
   while (1)
   {
      if (job[jno].tbt !=
0)
      {
         printf("|%d %s
", time,
job[jno].name);
         if (job[jno].tbt
>= tq)
         {
            job[jno].tbt =
job[jno].tbt - tq;
            time = time +
tq;
         }
         else
         {
            time = time +
job[jno].tbt;
            job[jno].tbt =
0;
         }
         printf("%d |",
time);
         if (job[jno].tbt
== 0)
         {
            job[jno].ct =
time;
            job[jno].tat =
time - job[jno].at;
            job[jno].wt =
job[jno].tat -
job[jno].bt;
            cnt++;
         }
      }
      jno++;
      if (jno == n)
      {
         jno = 0;
      }
      if (cnt == n)
         break;
   }
}
// to print the output
table
void print_output()
{
   printf("\n\n");
   printf("\n----------
----------");
   printf("\n pname
at     bt     tat     wt ");
   printf("\n----------
----------");
   for (i = 0; i < n; i++)
   {
      printf("\n%s    %d
%d     %d     %d ",
job[i].name, job[i].at,
job[i].bt, job[i].tat,
job[i].wt);
      avg_tat = avg_tat
+ (float)(job[i].tat);
      avg_wt =
(float)avg_wt +
(float)(job[i].wt);
   }
   printf("\n----------
----------");
   printf("\nThe avg of
the turn around time is
%f", avg_tat / n);
   printf("\nThe avg of
the waiting time is %f",
avg_wt / n);
}
main()
{
   take_input();
   process();
   print_output();
   for (i = 0; i < n; i++)
   {
      job[i].tbt =
job[i].bt = rand() % 10
+ 1;
      job[i].at = job[i].ct
+ 2;
   }
   process();
   print_output();
}

slip 20
1->
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char
*s, char *tok[])
{
   int i = 0;
   char *p;
   p = strtok(s, " ");
   while (p != NULL)
   {
      tok[i++] = p;
      p = strtok(NULL, "
");
   }
   tok[i] = NULL;
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"typeline") == 0)
      {
         if (args[1][0] ==
'+')
         {
            // Read the
line count
            int n =
atoi(args[1] + 1);
            // Open the
file
            FILE *fp =
fopen(args[2], "r");
            if (fp == NULL)
            {

printf("Error opening
file %s.\n", args[2]);
               return 1;
            }
            // Read and
print the first n lines
            char line[80];
            int count = 0;
            while (count <
n && fgets(line, 80, fp)
!= NULL)
            {
               printf("%s",
line);
               count++;
            }
            // Close the
file
            fclose(fp);
         }
         // If the '-a'
option is specified
         else if
(strcmp(args[2], "-a")
== 0)
         {
            // Open the
file
            FILE *fp =
fopen(args[2], "r");
            if (fp == NULL)
            {

printf("Error opening
file %s.\n", args[2]);
               return 1;
            }
            // Read and
print the entire file
            char line[80];
            while
(fgets(line, 80, fp) !=
NULL)
```

```
            printf("%s",
line);
            // Close the
file
            fclose(fp);
         }
         else
         {
            printf("Invalid
option.\n");
         }
      }
   }
}

2->
#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;
}
```

1->

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
 int pid=fork();
 if(pid>0){
 printf("I am parent
process\n");
 printf("PID =
%d\n\n",getpid());
 }
 else if(pid==0){
 printf("I am child
process\n");
 printf("PID =
%d\n",getpid());
 }
 else{
 printf("Failed to
create child process");
 }
 return 0;
}
```

2->

```
#include <stdio.h>
#include <string.h>
struct job
{
   char name[20];
   int at, bt, ct, tat, wt,
st, tbt, p;
} job[10];
int jno, n, i, j;
float avg_tat = 0;
float avg_wt = 0;

take_input()
{
   printf("Enter the no
of jobs : ");
   scanf("%d", &n);
   for (i = 0; i < n; i++)
   {
      printf("Enter the
arrival time of the job:
");
      scanf("%d",
&job[i].at);
      printf("Enter the
burst time of the job:
");
      scanf("%d",
&job[i].bt);
      printf("Enter the
name of the job: ");
      scanf("%s",
job[i].name);
      printf("Enter the
priority: ");
      scanf("%d",
&job[i].p);
      job[i].tbt =
job[i].bt;
      printf("\n\n");
   }
}
// to calculate the tat n
wt
void process()
{
   int time = 0, cnt = 0,
i;
   char prev_job[10],
cur_job[10];
   while (1)
   {
      jno = getjob(time);
      job[jno].tbt--;
      strcpy(cur_job,
job[jno].name);
      if (strcmp(cur_job,
prev_job) != 0)
         printf("%d -->
%s", time,
job[jno].name);
      time++;
      if (job[jno].tbt ==
0)
      {
         job[jno].ct =
time;
         job[jno].tat =
time - job[jno].at;
         job[jno].wt =
job[jno].tat -

job[jno].bt;

         cnt++;
      }
      strcpy(prev_job,
cur_job);
      if (cnt == n)
         break;
   }
   printf("End Time
%d", time);
}

int getjob(int time)
{
   int i, min = 0, k;
   for (i = 0; i < n; i++)
      if (job[i].at <= time
&& job[i].tbt != 0)
      {
         if (min <
job[i].p)
         {
            min = job[i].p;
            k = i;
         }
      }
   return k;
}
// to print the output
table
void print_output()
{
   printf("\n\n");
   printf("\n----------
----------");
   printf("\n pname at
bt tat wt ");
   printf("\n----------
----------");
   for (i = 0; i < n; i++)
   {
      printf("\n%s %d
%d %d %d ",
job[i].name, job[i].at,
job[i].bt, job[i].tat,
job[i].wt);
      avg_tat = avg_tat
+ (float)(job[i].tat);
      avg_wt =
(float)avg_wt +
(float)(job[i].wt);
   }
   printf("\n----------
----------");
```

```
   printf("\nThe avg of
the turn around time is
%f", avg_tat / n);
   printf("\nThe avg of
the waiting time is %f",
avg_wt / n);
}
main()
{
   take_input();
   process();
   print_output();
   for (i = 0; i < n; i++)
   {
      job[i].tbt =
job[i].bt = rand() % 10
+ 1;
      job[i].at = job[i].ct
+ 2;
   }
   process();
   print_output();
}



slip 22
1->
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   int pid = fork();
   int retnice;
   if (pid == 0)
   {
      retnice = nice(-
15);
      printf("\nI am
child process, PID =
%d\n", getpid());
      printf("\nChild
gets higher priority
%d\n", retnice);
   }
   else if (pid > 0)
   {
      retnice = nice(15);
      printf("\nI am
parent process, PID =
%d\n", getpid());
      printf("\nParent
gets lower priority
%d\n", retnice);
   }
   return 0;
}
Q .2
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
   typedef struct
process_info
{
   char pname[20];
   int at, bt, ct, bt1, p;
   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);
      printf("Enter
priority:");
      scanf("%d", &p-
>p);
      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\
tp\ttct\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\t%d\n",
            p->pname,
            p->at, p->bt,
p->p, 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\
tp\n");
   while (p != NULL)
   {

printf("%s\t%d\t%d\t%
d\n",
            p->pname, p-
>at, p->bt1, p->p);
      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;

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

      p = p->next;
   }
}
int time;
NODE *get_p()
{
   NODE *p, *min_p =
NULL;
```

```
   int min = 9999;
   p = first;
   while (p != NULL)
   {
      if (p->at <= time
&& p->bt1 != 0 &&
         p->p < min)
      {
         min = p->p;
         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 pnp()
{
   int prev = 0, n1 = 0;
   NODE *p;
   while (n1 != n)
   {
      p = get_p();
      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();
   pnp();
   print_output();
   print_gantt_chart();
   return 0;
}
```

slip 23

1->

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   int pid;
   pid = getpid();
   printf("Current
Process ID is : %d\n",
pid);
   printf("\n[Forking
Child Process ... ] \n");
   pid = fork();
   if (pid < 0)
   {
      printf("\nProcess
can not be created ");
   }
   else
   {
      if (pid == 0)
      {
         printf("\nChild
Process is Sleeping
...");
         sleep(5);

printf("\nOrphan
Process ID : %d",
getppid());
      }
      else
      { /* Parent
Process */
         printf("\nParent
Process Completed
...");
      }
   }
   return 0;
}
```

Q .2

```
#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 24

1->

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
void bubblesort(int
arr[30], int n)
{
  int i, j, temp;
  for (i = 0; i < n; i++)
  {
    for (j = 0; j < n - 1;
j++)
    {
      if (arr[j] > arr[j +
1])
      {
        temp = arr[j];
        arr[j] = arr[j +
1];
        arr[j + 1] =
temp;
      }
    }
  }
}
void insertionsort(int
arr[30], int n)
{
  int i, j, temp;
  for (i = 1; i < n; i++)
  {
    temp = arr[i];
    j = i - 1;

    while (j >= 0 &&
arr[j] > temp)
    {
      arr[j + 1] =
arr[j];
      j--;
    }
    arr[j + 1] = temp;
  }
}
void fork1()
{
  int arr[25], arr1[25],
n, i, status;
  printf("\nEnter the
no of values in array
:");
  scanf("%d", &n);
  printf("\nEnter the
array elements :");
  for (i = 0; i < n; i++)
    scanf("%d",
&arr[i]);
  int pid = fork();
  if (pid == 0)
  {
    sleep(10);
    printf("Child
process PID = %d\n",
getpid());
    printf("\nElements
Sorted Using
insertionsort:");
    insertionsort(arr,
n);
    printf("\n");
    for (i = 0; i < n; i++)
      printf("%d,",
arr[i]);
    printf("\b");
    printf("\nparent
process id=%d\n",
getppid());
    system("ps -x");
  }
  else
  {
    printf("\nParent
process PID = %d\n",
getppid());
    printf("Elements
Sorted Using
bubblesort:");
    bubblesort(arr, n);
    printf("\n");
    for (i = 0; i < n; i++)
      printf("%d,",
arr[i]);
    printf("\n\n\n");
  }
}
int main()
{
  fork1();
  return 0;
}
```

Q .2

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
  void make_toks(char
*s, char *tok[])
{
  int i = 0;
  char *p;
  p = strtok(s, " ");
  while (p != NULL)
  {
    tok[i++] = p;
    p = strtok(NULL, "
");
  }
  tok[i] = NULL;
}
void count(char *fn,
char op)
{
  FILE *fh;
```

```
   int cc = 0, wc = 0, lc =
0;
   char c;
   fh = fopen(fn, "r");
  if (fh == NULL)
   {
      printf("File %s not
found.\n", fn);
      return;
   }
   while ((c = fgetc(fh))
!= EOF)
   {
      if (c == ' ')
         wc++;
      else if (c == '\n')
      {
         wc++;
         lc++;
      }
      cc++;
   }
   fclose(fh);
   switch (op)
   {
   case 'c':
      printf("No.of
characters:%d\n", cc -
1);
      break;
   case 'w':
      printf("No.of
words:%d\n", wc);
      break;
   case 'l':
      printf("No.of
lines:%d\n", lc + 1);
      break;
   }
}
int main()
{
   char buff[80],
*args[10];
   int pid;
   while (1)
   {
      printf("myshell$
");
      fflush(stdin);
      fgets(buff, 80,
stdin);
      buff[strlen(buff) -
1] = '\0';
      make_toks(buff,
args);
      if (strcmp(args[0],
"count") == 0)
         count(args[2],
args[1][0]);
      else
      {
         pid = fork();
         if (pid > 0)
            wait();
         else
         {
            if
(execvp(args[0], args)
== -1)
               printf("Bad
command.\n");
         }
      }
   }
   return 0;
}
```

slip 25

1->

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

// Function to sort an
integer array using
bubble 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]) {
 int temp = arr[j];
 arr[j] = arr[j + 1];
 arr[j + 1] = temp;
 }
 }
 }
}

// Function to perform
binary search on a
sorted integer array
int binarySearch(int
arr[], int low, int high,
int key) {
 while (low <= high) {
 int mid = low + (high -
low) / 2;
 if (arr[mid] == key) {
 return mid;
 } else if (arr[mid] <
key) {
 low = mid + 1;
 } else {
 high = mid - 1;
 }
 }
 return -1; // Key not
found
}

int main() {
 int arr[] = {5, 2, 8, 12,
3};
 int n = sizeof(arr) /
sizeof(arr[0]);

 // Fork a child process
 pid_t pid = fork();

 if (pid == 0) {
 // Child process
 char *args[3];
 args[0] =
"./binary_search"; //
Name of the new
program to be
executed
 args[1] = (char
*)malloc(10 *
sizeof(char)); //
Allocate memory for
the sorted array
 args[2] = NULL; // End
of arguments

 // Sort the array
 bubbleSort(arr, n);

 // Convert the sorted
array to a string
 sprintf(args[1], "%d",
arr[0]);
 for (int i = 1; i < n; i++)
{
 sprintf(args[1], "%s
%d", args[1], arr[i]);
 }

 // Execute the new
program with the
sorted array as
command line
arguments
 execve(args[0], args,
NULL);
 } else if (pid > 0) {
 // Parent process
 wait(NULL); // Wait
for the child process to
finish
 printf("Child process
completed.\n");
 } else {
 // Forking failed
 printf("Forking
failed.\n");
 return 1;
 }

 return 0;
}
```

2->

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
void make_toks(char
*s, char *tok[])
{
int i=0;
char *p;
p = strtok(s," ");
while(p!=NULL)
{
 tok[i++]=p;
 p=strtok(NULL," ");
}
tok[i]=NULL;
}
void search(char *fn,
char op, char *pattern)
{
FILE *fh;
```

```
int count = 0;
char line[80];
fh = fopen(fn, "r");
if (fh == NULL)
{
printf("File %s not
found.\n", fn);
return;
}
while (fgets(line, 80,
fh) != NULL)
{
if (strstr(line, pattern)
!= NULL)
{
count++;
if (op == 'a')
printf("%s", line);
}
}
fclose(fh);
if (op == 'c')
printf("Number of
occurrences: %d\n",
count);
}
int main()
{
char
buff[80],*args[10];
int pid;
while(1)
{
 printf("myshell$ ");
 fflush(stdin);
 fgets(buff,80,stdin);
 buff[strlen(buff)-
1]='\0';
 make_toks(buff,args);
if (strcmp(args[0],
"search") == 0)
search(args[2],
args[1][0], args[3]);
else
 {
 pid = fork();
 if(pid>0)
 wait();
  else
 {

if(execvp(args[0],args)
==-1)
 printf("Bad
command.\n");
 }
 }
}
return 0;
}
```