Lab Exercises:

1. Write shell commands for the following.
   i) To create a directory in your home directory having 2 subdirectories.
   ii) In the first subdirectory create 3 different files with different content in each of them.
   iii) Copy the first file from the first subdirectory to the second subdirectory.
   iv) Create one more file in the second subdirectory which has the output of the number of users and number of files.
   v) To list all the files which starts with either a or A.
   vi) To count the number of files in the current directory.
   vii) Display the output if the compilation of a program succeeds.
2. Execute the following commands in sequence: i) date ii) ls iii) pwd
3. Write a C program to simulate the **grep** command.

Additional Exercises:

1. Write shell commands for the following.
   i. To Display an error message if the compilation of a program fails.
   ii. To write a text block into a new file.
   iii. List all the files.
   iv. To count the number of users logged on to the system.

2. Write a C program to simulate **wc** command to count number of characters, words and lines in a file.

[OBSERVATION SPACE – LAB 1]

1. (i) mkdir hello
cd hello
mkdir sem5 roll4.

(ii). cd sem5
cat > fil1.txt
cat > fil2.txt
cat > fil3.txt

(iii) cp fil1.txt ../roll4

LAB NO.:

[OBSERVATION SPACE – LAB 1]

(iv). cd..
cd roll4
who|wc -l > fil4.txt && ls|wc -l >> fil4.txt

(v) ls [Aa]*

(vi) ls|wc -l

(vii) cc addi.c && ./a.out
output: 2+3=5

addi.c:

```
# include<stdio.h>
void main
{ int a=2,b=3,c;
  c=a+b;
  printf("%d+%d=
  "%d",a,b,c);
}
```

2. (i) Thu Jul 27 02:45:16 IST 2017

(ii) addi.c, Documents, hello, lab1.txt, Pictures, Templates, Desktop, a.out, Downloads, hello.c, Music, Public, Videos, examples.desktop, here.sh, os.txt, roll4

(iii) /home/OS_A1.

Lab Exercises:

Write Shell Scripts to do the following:

1. To list all the files under the given input directory, whose extension has only character?
2. Write a shell script which accepts two command line parameters. First parameter indicates the directory and the second parameter indicates a regular expression. The script should display all the files and directories in the directory specified in the first argument matching the format specified in the second argument.
3. Count the number of users logged on to the system. Display the output as Number of users logged into the system.
4. Count the only the number of files in the current directory.
5. Echo today's date along with the string "The date today is:"
6. Write a shell script which accepts two command line arguments. First argument indicates format of file and the second argument indicates the destination directory. The script should copy all the files as specified in the first argument to the location indicated by the second argument. Also, try the script where the destination directory name has space in it.

Additional Exercises:

1. Write Shell Scripts to do the following
   i) To list all the .c files in any given input subdirectory.
   ii) Write a script to include n different commands.

[OBSERVATION SPACE – LAB 2]

1.
```
#!/bin/bash
cd $1
ls *.?
```
Output: add.c

2
```
#!/bin/bash
cd $1
ls *$2
```

Output:

```
$./lab2_2.sh /home/OS_A1/Desktop/1809050018/week2 "*.*"
access.sh  ex1.sh  lab2_2.sh  26  lab2_4.sh  lab2_6.sh
create.sh  lab2_1.sh  lab2_3.sh  lab2_5.sh  script.sh
```

3. #!/bin/bash
u='who|wc -l'
echo Number of users logged into the system = $u

Output:
$ ./lab2_3.sh
Number of users logged into the system = 1

4. #!/bin/bash
u='ls *.*|wc -l'
echo Number of files in current directory = $u

Output:
$ ./lab2_4.sh
Number of files in current directory = 11

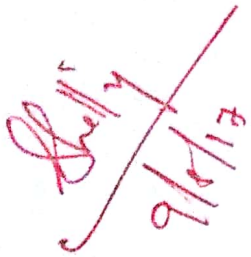
5. #!/bin/bash
echo The date today is 'date'

Output:
$ ./lab2_5.sh
The date today is Thu Aug 3 03:03:12 BST 2017

[OBSERVATION SPACE – LAB 2]

6. #!/bin/bash

cp *$1 $2



The above example lists the names of all files under / (the root directory)

Lab Exercises:

1. Write a shell script which accepts a directory name. The shell script should display the largest file in the given directory.
2. Write a shell script which accepts a directory name. The script should delete the oldest file in the given input directory.
3. Assume there is a file named TOC.txt which contains list of file names. Write a script which accepts a directory name as argument. The script should check each file name in the directory whether it exists in the TOC.txt file. If not exists, it should append the filename to TOC.txt.
4. Write a shell script which takes two sorted numeric files as input and produce a single sorted numeric file.

Additional Exercises

1. Write Shell scripts to do the following
   i) Find whether a input string name is a file, directory or sub-directory.
   ii) Find the file which is the latest modified file in the given directory.

[OBSERVATION SPACE – LAB 3]

1.
```
#!/bin/bash
echo "Enter name: "
read name
ls -S $name | head -1
```

Output:

```
Enter name:
/home/OS_A1/Desktop/160905018/36 week3
exmp4.sh.
```

[OBSERVATION SPACE – LAB 3]

2.
```
#!/bin/bash
echo "Enter name:"
read name
e='ls -t $name|tail -1'
rm $e
```

Output:

```
Enter name:
/home/OS_A1/Desktop/150905018/week3
```

3.
```
#!/bin/bash
for i in 'ls -a $1'
do
  if grep -E $i TOC.txt
  then
  echo "File name present!!"
  else
    echo $i >> TOC.txt
  fi
done
```

Output:

```
$ ./q3.sh /home/OS_A1/Desktop/150905018/week3
File name present!
File name present!
```

4.

```
#!/bin/bash
echo "Enter first file:"
read f1
echo "Enter second file:"
read f2
cat $f2 >> $f1
sort -n $f1 > sort.txt
```

Output:

Enter first file:
s1.txt
Enter second file:
s2.txt

1
2
3
4
5
6
7
8

16/8/17

Lab Exercises:

1. Write a C program to block a parent process until child completes using wait system call.
2. Write a C program to load the binary executable of the previous program in a child process using exec system call.
3. Write a program to create a child process. Display the process IDs of the process, parent and child (if any) in both the parent and child processes.
4. Create a zombie (defunct) child process (a child with exit() call, but no corresponding wait() in the sleeping parent) and allow the init process to adopt it (after parent terminates). Run the process as background process and run "ps" command.
5. Write a C program to create a file and write contents to it.
6. Write a C program to copy the content of one file to other.

Additional Exercises:

1. Create a orphan process (parent dies before child – adopted by "init" process) and display the PID of parent of child before and after it becomes orphan. Use sleep(n) in the child to delay the termination.
2. Modify the program in the previous question to include wait (&status) in the parent and to display the exit return code (left most byte of status) of the child.
3. Use lseek() to copy different parts (initial, middle and last) of the file to other. (For lseek() refer to man pages)
4. Create a child process which returns a 0 exit status when the minute of time is odd and returning a non-zero (can be 1) status when the minute of time is even.

1.
```
#include<sys/types.h>
#include<sys/wait.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main()
{
    int status;
    pid_t pid;
```

```
pid = fork(); [OBSERVATION SPACE – LAB 4]
if(pid==-1)
  printf("\n Error, Child not created");
else if (pid==0)
{  printf("\nI am the child");
   exit(0);
}
else
{
   wait(&status);
   printf("\nI am the parent");
   printf("\n Child returned: %d\n", status);
}  }
```

Output:

I am the child
I am the parent
Child returned: 0

```
#include<sys/types.h>
#include<sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include<stdlib.h>
int main()
{
   pid_t pid;
   pid= fork();
   if(pid<0)
   {  fprintf(stderr, "fork failed");
      exit(0); }
```

[OBSERVATION SPACE – LAB 4]

```
    else if (pid == 0)
        execlp("/home/OS_A1/Desktop/1509050l8/week4/exmp2", "exmp2", NULL);
    else
    {
        wait(NULL);
        printf("Child complete\n");
        exit(0);
    }
}
```

Output :

Child complete.

3.

```
#include<sys/types.h>
#include<sys/wait.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
    pid_t pid;
    pid = fork();
    if(pid<0)
    {
        fprintf(stderr, "Fork Failed");
        exit(0);
    }
    else if (pid == 0)
    {
        pid_t child_id = getpid();
        pid_t parent_id = getppid();
        printf("From child process: \n Parent id = %d\n Child id
        in parent process< %d\n Process id = %d\n", parent_id,
```

```
else
{   pid_t cur_id = getpid();
    pid_t parent_id =getppid();
    printf("From parent process :\n Parent id = %d\n Child id
    in parent process = %d\nProcess id = %d\n", parent_id,
    pid, cur_id);
    wait(NULL);
    exit(0);
}
}
```

Output:

```
From parent process:
Parent id = 2712
Child id in parent process = 3636
Process id = 3635
From child process:
Parent id = 3635
Child id in child process = 3636
```

4.

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
#include<sys/wait.h>
int main()
{
    pid_t pid;
    int n;
    char *msg;
    pid = fork();
    switch(pid)
    {
        case -1: fprintf(stderr, "fork error");
                 exit(0);
```

```
    case 0: msg="Child Process";
            n=3;
            break;
    default: msg="Parent Process";
            n=10;
            break;
  }
  for(;n>0;n--)
    sleep(1);
  exit(0);
}
```

o/p ?

5.
```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/stat.h>
int main()
{
  int fp;
  fp=creat("hello.txt",S_IRWXU);
  if(fp<0)
  {
    printf("Error creating file");
    exit(0);
  }
  char a[]="Hello, Welcome";
  write(fp, a, sizeof(a)-1);
  return 0;
}
```

o/p ?

```
#include<stdio.h>
#include <stdlib.h>
#include < fcntl.h>
#include <sys/stat.h>
int main()
{
    int fp;
    fp = open("hello.txt", O_RDONLY);
    char a[15];
    read (fp, a, sizeof(a));
    int fp1;
    fp1 = creat("copy.txt", S_IRWXU);
    write (fp1, a, sizeof(a));
    return 0;
}
```

Setting the Attributes for Pthreads

The attributes for a thread are set when the thread is created. To set the initial attributes, first create a thread attributes structure, and then set the appropriate attributes in that structure, before passing the structure into the pthread_create() call.

```
#include <pthread.h>
...
int main()
{
        pthread_t thread;
        pthread_attr_t attributes;
        pthread_attr_init( &attributes );
        pthread_create( &thread, &attributes, child_routine, 0 );
}
```

Lab Exercises:

1. Write multithreaded program that generates the Fibonaaci series. The program should work as follows: The user will enter on the command line the number of Fibonacci numbers that the program is to generate. The program then will create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that is shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution the parent will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish.
2. Write a multithreaded program that calculates summation of non-negative integers in a separate thread and passes the result to main thread.
3. Write a multithreaded program for generating prime numbers from a given starting number to the given ending number.

Additional Exercises:

1. Write a multithreaded program for matrix multiplication.

1.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
int res[30], upper;
void *fibo(void *param)
{
    int i=0, j=1, k, l=2;
    upper = atoi(param);
    res[0] = i;
    if(upper==0)
      return;
    res[1] = j;
    if(upper==1)
      return;
    else
    {
      while(l<=upper)
      {
        k=i+j;
        i=j;
        j=k;
        res[l]=k;
        l++;
}  }  }
int main(int argc, char *argv[])
{
  pthread_t thread;
  pthread_attr_t attr;
  pthread_attr_init(&attr);
  pthread_create(&thread, &attr, fibo, argv[1]);
  pthread_join(thread, NULL);
  for(int i=0; i<upper; i++)
    printf("%d ", res[i]); }
```

Output:

```
gcc q1.c -o q1 -lpthread
./q1 10
0 1 1 2 3 5 8 13 21 34
```

2.

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>
int sum;
void *runner(void *param)
{
    int i, upper = atoi(param);
    sum = 0;
    for(i=1; i<=upper; i++)
      sum += i;
    pthread_exit(0);
}
int main(int argc, char *argv[])
{
    pthread_t thread;
    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_create(&thread, &attr, runner, argv[1]);
    pthread_join(thread, NULL);
    printf("Sum is: %d\n", sum);
    return 0;
}
```

Output:

```
./q2 5
Sum is: 15
```

3. [OBSERVATION SPACE – LAB 5]

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int arr[15], k, a[2];
void *prime (void *param)
{
    int flag, i, start, end;
    start = a[0];
    end = a[1];
    k = 0;
    for (i = start; i <= end; i++)
    {
        flag = 0;
        for(int j = 2; j <= sqrt(i); j++)
        {
            if ((i % j) == 0)
            {
                flag = 1;
                break;
            }
        }
        if(flag == 0)
            arr[k++] = i;
} }
Int main (int argc, char *argv[])
{
    pthread_t thread;
    pthread_attr_t attr;
    pthread_attr_init (&attr);
    a[0] = atoi (argv[1]);65
    a[1] = atoi (argv[2]);
```

```
pthread_create(&thread, &attr, prime, 0);
pthread_join(thread, NULL);
for(int i=0; i<k; i++)
    printf("%d", arr[i]);
return 0;
}
```

Output:

./a3 2 10

2 3 5 7

23/1/17

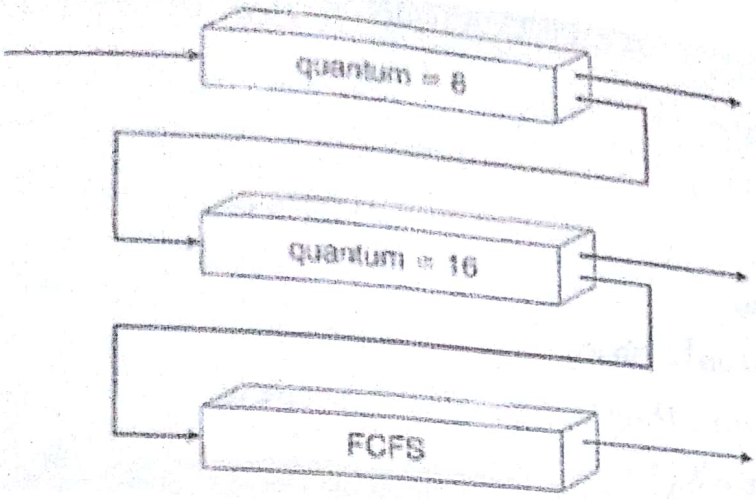


Figure 6.2: Multilevel feedback queues

Lab Exercises:

1. Consider the following set of processes, with length of the CPU burst given in milliseconds:

Process	Arrival time	Burst time	Priority
P1	0	60	3
P2	3	30	2
P3	4	40	1
P4	9	10	4

Write a C program to simulate the following CPU scheduling algorithms. Display Gantt chart showing the order of execution of each process. Compute waiting time and turnaround time for each process. Hence compute average waiting time and average turnaround time.

(i) FCFS (ii) SRTF (iii) non-preemptive priority (iv) Round-Robin (quantum = 10)

Additional Exercises:

1. Write a C program to simulate the following CPU scheduling algorithms. Display Gantt chart showing the order of execution of each process. Compute waiting time and turnaround time for each process. Hence compute average waiting time and average turnaround time.
   (i) SJF (ii) preemptive priority
2. Write a C program to simulate multi-level queue scheduling algorithm.
3. Write a C program to simulate multi-level feedback queue scheduling algorithm.

1. (i)

```
#include<stdio.h>
#include<stdlib.h>
struct PRO
{
    int pno;
    int arrival_time;
    int burst_time;
    int priority;
};
int main()
{
    int p;
    printf("Enter number of processes: ");
    scanf("%d", &p);
    struct PRO process[p];
    struct PRO temp;
    printf("Enter details: ");
    for (int i=0; i<p; i++)
    {
        printf("\n Process P%d ", i+1);
        process[i].pno=i+1;
        printf("\nArrival time: ");
        scanf("%d", &process[i].~~burst~~arrival_time);
        printf("\nBurst time: ");
        scanf("%d", &process[i].burst_time);
        printf("Priority: ");
        scanf("%d", &process[i].priority);
    }
    for (int i=0; i<p; i++)
        for (int j=0; j<p-i-1; j++)
            if (process[j].arrival_time > process[j+1].arrival_time)
            {
                temp = process[j];
                process[j] = process[j+1];
```

```
        process[j+1] = temp;
    }
}
int wait[p], turn[p], avg_wait=0, avg_turn=0;
for(int i=0; i<p; i++)
{ int j= i-1;
  wait[i]=0;
  while (j>=0)
  { wait[i]+= process[j].burst_time;
    j--;
  }
  wait[i]-= process[i].arrival_time;
  avg_wait+= wait[i];
  turn[i] = process[i].burst_time+wait[i];
  avg_turn+= turn[i];
}
avg_wait/= p;
avg_turn/= p;
printf("Entered processes: ");
printf("\nProcess\tArrival time\tBurst time\tPriority\t Waiting time
\tTurnaround time");
for(int i=0; i<p; i++)
  printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d", process[i].pno,
  process[i].arrival_time, process[i].burst_time, process[i].priority,
  wait[i], turn[i];
printf("Average waiting time = %d", avg_wait);
printf("Average turnaround time = %d", avg_turn);
return 0;
}
```

Output:

```
Enter number of processes: 4
Enter details:
Process P1:
Arrival time: 0
Burst time: 60
Priority: 3
Process P2:
Arrival time: 3
Burst time: 30
Priority = 2
Process P3
Arrival time: 9
Burst time: 10
Priority: 4
Process P4
Arrival time: 4
Burst time: 40
Priority: 1
Entered items:
```

Process	arrival time	Burst time	Priority	waiting time	Turnaround tim.
P_1	0	60	3	0	60
P_2	3	30	2	57	87
P_4	4	40	1	86	126
P_3	9	10	4	121	131

Average waiting time = 66 Average turnaround time = 101

(iii)

```
#include<stdio.h>
#include<stdlib.h>
struct PRO
{
    int pno;
    int arrival_time;
    int burst_time;
    int priority;
};
int main()
{
  int p;
  int wait[p], turn[p], avg_wait = 0, avg_turn = 0;
  printf("Enter number of processes: ");
  scanf("%d", &p);
  struct PRO process[p], temp;
```

```
printf("Enter details: ");
for(int i=0; i<p; i++)
{   printf("\nProcess P%d", i+1);
    proces[i].pno = i+1;
    printf("\nArrival time: ");
    scanf("%d", &proces[i].arrival_time);
    printf("Burst time: ");
    scanf("%d", &proces[i].burst_time);
    printf("Priority: ");
    scanf("%d", &proces[i].priority);
}
for(int i=0; i<p; i++)
   for(int j=0; j<p-i-1; j++)
      if(proces[j].arrival_time > proces[j+1].arrival_time)
      {   temp = proces[j];
          proces[j] = proces[j+1];
          proces[j+1] = temp;
      }
for(int i=0; i<p; i++)
wait[0] = 0;
turn[0] = proces[0].burst_time;
for(int i=1; i<p; i++)
   for(int j=1; j<p; j++)
      if(proces[j].priority > proces[j+1].priority)
      {   temp = proces[j];
          proces[j] = proces[j+1];
          proces[j+1] = temp;
      }
for(int i=1; i<p; i++)
{   int j=i-1;
```

```
        wait[i]=0;
        while(j>=0)
        {   wait[i]+= process[j].burst_time;
            j--;
        }
        wait[i]-=process[i].arrival_time;
        avg_wait+=wait[i];
        turn[i]=process[i].burst_time+wait[i];
        avg_turn+=turn[i];
    }
    avg_wait+=wait[0];
    avg_turn+=turn[0];
    avg_wait/=p;
    avg_turn/=p;
    printf("\nProcess\tArrival time\tBurst time\t Priority\t Waiting time\t
    Turnaround time");
    for(int i=0; i<p; i++)
        printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d",
        process[i].pno, process[i].arrival_time, process[i].burst_time,
        process[i].priority, wait[i], turn[i]);
    printf("\nAverage waiting time = %d", avg_wait);
    printf("\nAverage turnaround time= %d", avg_turn);
    return 0;
}
```

Output:

Enter number of processes: 4

Process P1:
Arrival time: 0
Burst time: 60
Priority: 3

Process P2
Arrival time= 3
Burst time= 30
Priority= 2

Process P3
Arrival time: 4
Burst time: 40
Priority: 1

Process P4
Arrival time: 9
Burst time: 10
Priority: 4

[OBSERVATION SPACE – LAB 6]

LAB NO.: 6

Process	Arrival time	Burst time	Priority	Waiting time	Turnaround time
P1	0	60	3	0	60
P2	4	80	1	51	96
P3	3	30	2	97	127
P4	9	10	4	121	131

Average waiting time = 68

Average turnaround time = 103.

```
    }
    printf("Process %d finished, %d bytes read\n", getpid(), bytes_read);
    exit(EXIT_SUCCESS);
}
```

The Readers-Writers Problem:

- Concurrent processes share a file, record, or other resources
- Some may read only (readers), some may both read and write (writers)
- Two concurrent reads have no adverse effects
- Problems if
  - concurrent reads and writes
  - multiple writes

Two Variations

- First Readers-Writers problem: No reader be kept waiting unless a writer has already obtained exclusive write permissions (Readers have high priority)
- Second Readers-Writers problem: If a writer is waiting/ready , no new readers may start reading (Writers have high priority)

Lab Exercises:

1. Write a first reader-writer program in C in which writer program writes a string into a pipe and then the reader program reads the string and displays.
2. Write a producer and consumer program in C using FIFO queue. The producer should write a set of 4 integers into the FIFO queue and the consumer should display the 4 integers.
3. Implement a parent process which sends an English alphabet to child process using shared memory. Child process responds back with next English alphabet to the parent. Parent displays the reply from the Child.
4. Write a producer-consumer program in C in which producer writes a set of words into shared memory and then consumer reads the set of words from the shared memory. The shared memory need to be detached and deleted after use.

Demonstrate creation of a process which writes through a pipe while the parent process reads from it.

Write a program which creates a message queue and writes message into queue which contains number of users working on the machine along with observed time in hours and minutes. This is repeated for every 10 minutes. Write another program which reads this information form the queue and calculates on average in each hour how many users are working.

[OBSERVATION SPACE – LAB 7]

1.
```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
#include<unistd.h>
#include <sys/wait.h>
#define READ 0
#define WRITE 1
char *phrase = "Hello... this is sample data";
int main()
{
    int fd[2], bytesread;
    char message[100] = " ";
    pipe(fd);
    if(fork() == 0)
    {
        close(fd[READ]);
        write(fd[WRITE], phrase, strlen(phrase));
        close(fd[WRITE]);
    }
    else
    {
        close(fd[WRITE]);
        read(fd[READ], message, sizeof(message));
        close(fd[READ]);
        wait(NULL);
```

```
        printf("Message: %s\n", message);
    }
    return 0;
}
```

Output: Message : Hello... this is sample data

8.

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<unistd.h>
int main()
{
    int segment_id;
    char *shared_memory;
    char a[2];
    const int size=4096;
    segment_id = shmget(IPC_PRIVATE, size, 0666|IPC_CREAT);
    shared_memory=(char*)shmat(segment_id, NULL, 0);
    printf("Enter a character: ");
    gets(a);
    sprintf(shared_memory, a);
    pid_t pid;
    pid=fork();
    if(pid==0)
    {
        shared_memory = (char*) shmat(segment_id, NULL, 0);
        char c;
        c=shared_memory[0];
        c=c+1;
        shared_memory[0]=c;
```

[OBSERVATION SPACE – LAB 7]

```
        shmdt (shared_memory);
        exit (0);

    }
    else
    {   shared_memory = (char*) shmat (segment_id, NULL, 0);
        wait (NULL);
        printf ("Updated character: %s\n", shared_memory);

    }
    shmdt (shared_memory);
    shmctl (segment_id, IPC_RMID, NULL);
    return 0;

}
```

Output:

Enter a character: r

Updated character: s

2. Producer:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys\stat.h>
int main ()
{   int fd, i=0;
    char buf[]={'1', '2', '3', '4'};
    char *myfifo = "/tmp/myfifo";
    mkfifo (myfifo, 0666);
    fd = open (myfifo, O_WRONLY);
```

```
    if (fd1 != -1)
        write (fd, buf, sizeof(buf));
    close(fd);
    printf("Finished producing");
    return 0;
}
```

Consumer:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
    int fd,
    char buf[100];
    char * mififo = "/tmp/myfifo"
    fd = open( myfifo, O_RDONLY );
    if (fd1 != -1)
        read (fd, buf, sizeof(buf));
    close(fd);
    printf("Consumed data: %s\n", buf);
    return 0;
}
```

Output:

Finished producing

Consumed data: 1, 2, 3, 4

LAB NO.: 7

550
1...n approach

[OBSERVATION SPACE – LAB 7]

LAB NO.: 7

shm_com.h :

```
#define TEXT_SZ 2048
struct shared_use_st
{  int written_by_you;
   char some_text[TEXT_SZ];
};
```

Consumer process :

```
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include"shm_com.h"
int main()
{  int running=1; shmid;
   void *shared_memory=(void *)0;
   struct shared_use_st *shared_stuff;
   srand((unsigned int)getpid());
   shmid=shmget((key_t)1234, sizeof(struct shared_use_st), 0666|
   IPC_CREAT);
   if(shmid==-1)
   {  fprintf(stderr, "shmget failed\n");
      exit(EXIT_FAILURE);
   }
   shared_memory=shmat(shmid, (void *)0, 0);
   if(shared_memory==(void *)-1)
   {  fprintf(stderr, "shmat failed\n");
      exit(EXIT_FAILURE);
   }
   printf("Memory attached at %X\n", (int)shared_memory);
   shared_stuff=(struct shared_use_st *)shared_memory;
```

[OBSERVATION SPACE – LAB 7]

```
shared_stuff->written_by_you = 0;
while (running)
{
    if (shared_stuff->written_by_you)
    {
        printf ("You wrote : %s", shared_stuff->some_text);
        sleep (rand()%4);
        shared_stuff->written_by_you = 0;
        if (strncmp (shared_stuff->some_text, "end", 3) == 0)
        {
            running=0; } }
        if (shmdt (shared_memory==-1)
        {
            fprintf (stderr, "shmdt failed\n");
            exit (EXIT_FAILURE);
        }
        if (shmctl (shmid, IPC_RMID,0)==-1)
        {
            fprintf (stderr, "shmctl(IPC_RMID) failed\n");
            exit (EXIT_FAILURE);
        }
        exit(EXIT_SUCCESS);
    }.
```

Producer:

```
#include<stdio.h>          #include<sys/types.h>
#include<stdlib.h>         #include<sys/ipc.h>
#include<unistd.h>         #include<sys/shm.h>
#include<string.h>         #include "shm_com.h"
int main()
{ int running=1, shmid;
  void *shared_memory=(void *)0;
  struct shared_use_st *shared_stuff;
  char buffer[BUFSIZE];
  shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666|IPC_CREAT);
  if (shmid==-1)
  { fprintf(stderr, "shmget failed\n");
    exit(EXIT_FAILURE);
  }
  shared_memory=shmat(shmid, (void*)0, 0);
```

[OBSERVATION SPACE – LAB 7]

```
if (shared_memory == (void *)-1)
{ fprintf(stderr, "shmat failed\n");
  exit (EXIT_FAILURE);
{printf("Memory attached at %X\n", (int)shared_memory);
shared_stuff = (struct shared_use_st *) shared_memory;
while (running)
{   while (shared_stuff->written_by_you == 1)
    {  sleep (1);
       printf(" waiting for client...\n");
    {
    printf ("Enter some text: ");
    fgets (buffer, BUFSIZ, stdin);
    strncpy (shared_stuff -> some_temt, buffer, TEXT_SZ);
    shared_stuff ->written_by_you = 1;
     if (strncmp(buffer, "end", 3) == 0)
     {   running = 0; {
{
if (shmdt(shared_memory) == -1)
{   fprintf(stderr, "shmdt failed\n");
    exit (EXIT_FAILURE);
{
exit (EXIT_SUCCESS);
{
```

- A philosopher needs two chopsticks to eat
  - philosophers must share chopsticks to eat
- No interaction occurs while thinking

The situation of the dining philosophers is shown in Fig. 8.1

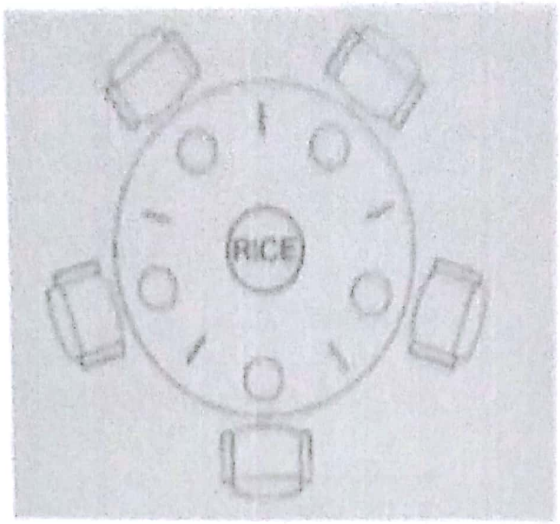


Figure 8.1

Lab Exercises:

1. Modify the above Producer-Consumer program so that, a producer can produce at the most 10 items more than what the consumer has consumed.
2. Write a C program for first readers-writers problem using semaphores.

Additional Exercises:

1. Write a C program for Dining-Philosophers problem using monitors.

[OBSERVATION SPACE – LAB 8]

2.
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex, wrt;
int data=0, readcount=0;
void *reader(void *param)
{  sem_wait(&mutex);
```

```
    readcount++;
    if(readcount==1)
        sem_wait(&wrt);
    sem_post(&mutex);
    printf("Data read by reader is %d\n", data);
    sleep(1);
    sem_wait(&mutex);
    readcount--;
    if(readcount==0)
        sem_post(&wrt);
    sem_post(&mutex);
}
void *writer(void *param)
{
    sem_wait(&wrt);
    data++;
    printf("Data written by the writer is %d\n", data);
    sleep(1);
    sem_post(&wrt);
}
void main()
{
    pthread_t rtid, wtid;
    sem_init(&mutex, 0, 1);
    sem_init(&wrt, 0, 1);
    pthread_create(&wtid, NULL, writer, 0);
    pthread_create(&wtid, NULL, reader, 0);
    pthread_join(wtid, NULL);
    pthread_join(rtid, NULL);
    sem_destroy(&mutex);
    sem_destroy(&wrt);
}
```

[OBSERVATION SPACE – LAB 8]

Output:
Data written by writter is 1
Data read ~~from~~ by reader is 1

1.
```
#include<pthread.h>
#include<stdio.h>
#include <semaphore.h>
sem_t semaphore;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int queue[500], queueLength;
void *producer (void *param)
{
    for(int i=0; i<500; i++)
    {
        pthread_mutex_lock (&mutex);
        queue [queueLength++] = i+1 ;
        printf ("Produced %i\n" , i);
        pthread_mutex_unlock (&mutex);
        if (queueLength > 10)
            sem_post (&semaphore);
} }
Void *consumer (void *param)
{
    for(int i=0; i<500; i++)
    {
        int item;
        if (queueLength == 0)
            sem_wait(&semaphore);
        pthread_mutex_lock (&mutex);
        item = queue[--queueLength];
        printf("Received %i\n", item);
        pthread_mutex_unlock (&mutex); } }
```

[OBSERVATION SPACE - LAB 8]


```
int main()
{
    pthread_t Threads[2];
    sem_init(&semaphore, 0, 0);
    pthread_create(&Threads[0], 0, producer, 0);
    pthread_create(&Threads[1], 0, consumer, 0);
    pthread_join(Threads[0], 0);
    pthread_join(Threads[1], 0);
    sem_destroy(&semaphore);
}
```

Output:

```
Produced 0
Produced 1
Produced 2
Produced 3
Produced 4
Produced 5
Produced 6
Produced 7
Produced 8
Produced 9
Produced 10
Received 10
Received 9
Received 8
Received 7
Received 6
Received 5
Received 4
Received 3
Received 2
```

Shelly 17/9/17

Let *Work* and *Finish* be ... respectively. Initialize:
Work = *Available*. For $i = 0,2, \ldots, n\text{-}1$, if $Allocation_i \neq 0$, then *Finish*[i] = *false*; otherwise, *Finish*[i] = *true*.
Find an index *i* such that both:
(a) *Finish*[*i*] == *false*
(b) $Request_i \leq Work$
If no such *i* exists, go to step 4.
Work = *Work* + $Allocation_i$
Finish[*i*] = *true*
go to step 2.
If *Finish*[*i*] == false, for some *i*, $0 \leq i < n$, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then process P_i is deadlocked

The above algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Exercises:

Consider the following snapshot of the system. Write C program to implement Banker's algorithm for deadlock avoidance.

(a) What is the content of the matrix *Need*?
(b) Is the system in a safe state?
(c) If a request from process P4 arrives for (3, 3, 0), can the request be granted immediately?
(d) If a request from process P0 arrives for (0, 2, 0), can the request be granted immediately?

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Consider the following snapshot of the system. Write C program to implement deadlock detection algorithm.

(a) Is the system in a safe state?
(b) Suppose that process P2 make one additional request for instance of type C, can the system still be in a safe state?

	Allocation	Request	Available
	A B C	A B C	A B C
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

Additional Exercises:

1. Write a multithreaded program that implements the banker's algorithm. Create n threads that request and release resources from the bank. The banker will grant the request only if it leaves the system in a safe state. You may write this program using either Pthreads. It is important that shared data be safe from concurrent access. To ensure safe access to shared data, you can use mutex locks, which are available in the Pthreads libraries.

[OBSERVATION SPACE – LAB 9]

1.
```
#include <stdio.h>
#include <stdlib.h>
void checksafe(int **allocation, int available[], int **need, int n, int m)
{
    int finish[n], work[m], flag, i, j;
    for(i=0; i<m; i++)
    {
        work[i] = available[i];
        finish[i]=0;
    }
    while (i<n)
    {
        finish[i]=0;
        i++;
    }
    int flag1, flag2, count=0;
    i=0;
    while(i<n)
    {
        count++;
        if(i==0)
            flag2=0;
```

LAB NO.: 9

```
flag1 = 0;
if (finish[i] == 0)
{
    for (j=0; j<m; j++)
        if (need[i][j] > work[j])
        {
            flag = 1;
            break;
        }
    if (flag1 == 0)
    {
        finish[i] = 1;
        for (int k=0; k<m; k++)
            work[k] += allocation[i][k];
        printf("Work %d: ", i);
        for (j=0; j<m; j++)
            printf("%d", work[j]);
        printf("\n");
    }
    else
        flag2 = 1;
}
i++;
if (count == n*n)
{
    flag = 0;
    break;
}
if (i == n)
{
    if (flag2 == 1)
        i = 0;
    else
        flag = 1;
}
```

[OBSERVATION SPACE – LAB 9]

```
    }
  if(flag)
    printf("System is in safe state!\n");
  else
    printf("System is not in safe state!\n");
}
int main()
{
    int n, m, i, j;
    printf("Enter number of processes and resources: ");
    scanf("%d %d", &n, &m);
    int **allocation, max[n][m], **need, available[m];
    allocation = (int **) calloc (n, sizeof(int *));
    need = (int **) calloc (n, sizeof(int *));
    for(i=0; i<n; i++)
    {
        allocation[i] = (int *) calloc (m, sizeof(int));
        need[i] = (int *) calloc (m, sizeof(int));
    }
    printf("Enter allocation matrix\n");
    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
            scanf("%d", &allocation[i][j]);
    printf("Enter max matrix\n");
    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
            scanf("%d", &max[i][j]);
    printf("Enter Available vector\n");
    for(i=0; i<m; i++)
        scanf("%d", &available[i]);
    for(i=0; i<n; i++)
```

CN -

Computer Network

(8c) - 425

- 355


```
for (j=0; j<m; j++)
    need[i][j] = max[i][j] - allocation[i][j];
int choice, pno, temp[m];
do
{
printf("1. Request \n 2. Check saftey \n 3. Exit \n Enter your choice");
scanf("%d", & choice);
switch (choice)
{
    case 1 : printf("Enter process number: ");
             scanf("%d", &pno);
             printf("Enter request resource: ");
             for (i=0; i<m; i++)
                 scanf("%d", &temp[i]);
             for (i=0; i<m; i++)
             {   available[i] -= temp[i];
                 allocation[pno-1][i] += temp[i];
                 need[pno-1][i] -= temp[i];
             }
             printf("Need matrix:\n");
             for(i=0; i<n; i++)
             {   for(j=0; j<m; j++)
                     printf("%d", need[i][j]);
                 printf("\n");
             }
             printf("Available vector:\n");
             for (i=0; i<m; i++)
                 printf("%d", available[i]);
             printf("\n");
             checksafe(allocation, available, need, n, m);
             break;
```


Scanned by CamScanner

[OBSERVATION SPACE – LAB 9]

```
case 2: printf("Need matrix : \n");
        for (i=0; i<n;i++)
        {   for (j=0; j<m;j++)
              printf("%d", need[i][j]);
            printf("\n");
        }
        printf("Available vector: \n");
        for(i=0;i<m;i++)
           printf("%d ", available[i]);
        printf("\n");
        checksafe(allocation, available, need, m,n);
        break;
case 3: exit(1);
default: printf("Invalid choice!.");
    }
}while(choice!=3);
}
```

Output

```
Enter number of processes and resources: 5 3
Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter available matrix:
3 3 2
Allocation matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Max matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
```

[OBSERVATION SPACE – LAB 9]

```
1. Request
2. Check saftey
3. Exit
Enter your choice:2
Need matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
Available vector:
3 2 2
Work 1: 5 3 2
Work3 : 7 4 3
Work4: 7 4 5
Work 0: 7 5 5
Work 2: 10 5 7
System is in safe state!!
1. Request
2. Check saftey
3 Exit
Enter your choice: 1
Enter process no.: 5
Enter requested resource : 3 3 1
Need matrix:
7 4 3
1 2 2
6 0 0
0 1 1
1 0 0
Available vector:
0 0 1
System is not in safe state
1. Request
2. Check saftey
3. Exit
Enter your choice 3.
```

[OBSERVATION SPACE – LAB 9]

(c) Unsafe

(d) Safe

11/10/21

417K, 112K, and 426K (in order)? Which algorithm makes efficient use of memory?

2. Assuming a page size of 32 bytes and there are total of 8 such pages totaling 256 bytes. Write a C program to simulate this memory mapping. The program should read the logical memory address and display the page number and page offset in decimal. How many bytes do you need to represent the address in this scenario? Display the page number and offset to reference the following logical addresses.
(i) 204 byte (ii) 56 byte

Additional Exercises:

1. We have five segments numbered 0 through 4. The segments are stored in physical memory as shown in the following Fig 10.1. Write a C program to create segment table. Write methods for converting logical address to physical address. Compute the physical address for the following.
(i) 53 byte of segment 2 (ii) 852 byte of segment 3 (iii) 1222 byte of segment 0

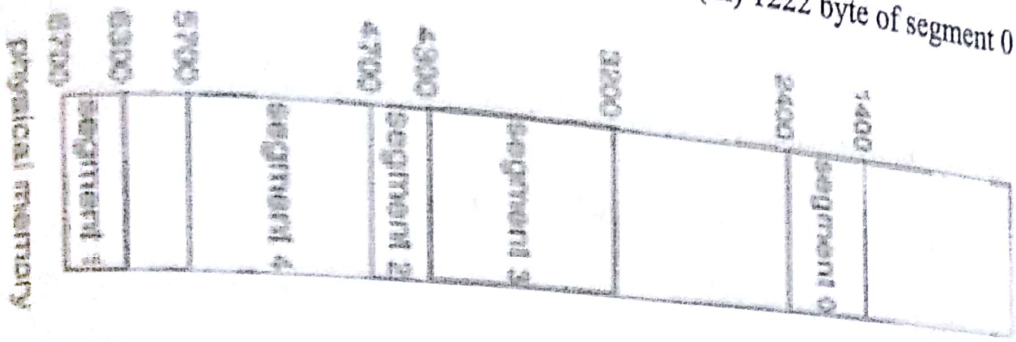


Figure 10.1: Physical memory

[OBSERVATION SPACE – LAB 10]

```
1. #include <stdio.h>
   #include <stdlib.h>
   void firstfit (int c[], int b[], int n, int m)
   {
       int i, j, temp=0, finish[m], a[n];
       for (i=0; i<n; i++)
          a[i]=c[i];
       for (i=0; i<m; i++)
       {
          finish[i]=0;
          for (j=0; j<n; j++)
             if(b[i]<a[j])
             {  printf("\n%d allocated in %d", b[i], a[j]);
```

[OBSERVATION SPACE – LAB 10]

```
                a[j] -= b[i];
                finish[i] = 1;
                break;
            }
        }
        for (i=0; i<m; i++)
            if (!finish[i])
                printf("\nNo space to allocate %d ", b[i]);
        for (i=0; i<n; i++)
            temp += a[i];
        printf("\nMemory remaining is : %d \n", temp);
}
void bestfit(int c[], int b[], int n, int m)
{
    int i, j, temp=0, finish[m], a[n], t;
    for (i=0; i<n; i++)
        a[i] = c[i];
    for (i=0; i<n; i++)
        for (j=0; j<n-i-1; j++)
            if (a[j] > a[j+1])
            {   t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
    for (i=0; i<m; i++)
    {
        finish[i] = 0;
        for (j=0; j<n; j++)
            if (b[i] < a[j])
            {   printf("\n%d allocated in %d ", b[i], a[j]);
```

[OBSERVATION SPACE - LAB 10]

```
            a[j] -= b[i];
            finish[i] = 1;
            break;
        }
    }
    for (i=0; i<m; i++)
        if (!finish[i])
            printf("\nNo space to allocate %d ", b[i]);
    for (i=0; i<n; i++)
        temp += a[i];
    printf("\nMemory remaining is : %d\n", temp);
}
void worstfit(int c[], int b[], int n, int m)
{
    int i, j, temp = 0, finish[m], a[n], t;
    for(i=0; i<n; i++)
        a[i] = c[i];
    for(i=0; i<n; i++)
        for(j=0; j<n-i-1; j++)
            if (a[j] < a[j+1])
            {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
    for(i=0; i<m; i++)
    {
        finish[i] = 0;
        for (j=0; j<n; j++)
            if(b[i] < a[j])
            {
                printf("\n%d allocated in %d ", b[i], a[j]);
                a[j] -= b[i];
```

[OBSERVATION SPACE – LAB 10]

```
                finish[i]=1;
                break;
        }   }
        for(i=0; i<m;i++)
          if(!finish[i])
            printf("\nNo space to allocate %d ", b[i]);
        for(i=0;i<n;i++)
          temp+=a[i];
        printf("\nMemory remaining is: %d\n", temp);
}
int main()
{
    int i, n, m;
    printf("Enter number of memory partitions: ");
    scanf("%d", &n);
    int a[n];
    printf("Enter partitions: ");
    for(i=0; i<n; i++)
      scanf("%d", &a[i]);
    printf("Enter number of user memory partitions: ");
    scanf("%d", &m);
    int b[m];
    printf("Enter memory for processes: ");
    for(i=0; i<m; i++)
      scanf("%d",&b[i]);
    printf("\nFirst fit: ");
    firstfit(a, b, n, m);
    printf("\nBest fit: ");
    bestfit(a, b, n, m);
```

```
    printf("\nWorst fit: ");
    worstfit (a, b, m, n);
    return 0;
}
```

Output:

```
Enter number of memory partitions: 5
Enter partitions: 100 500 200 300 600
Enter number of user memory partitions: 4
Enter memory for processes: 212 417 112 426
First fit:
212 allocated in 500
417 allocated in 600
112 allocated in 288
No space to allocate 426.
Memory remaining is: 959
Best fit:
212 allocated in 300
417 allocated in 500
112 allocated in 200
426 allocated in 600
Memory remaining is: 533
Worst fit:
212 allocated in 600
417 allocated in 500
112 allocated in 388
No space to allocate 426
Memory remaining is: 959
```

CN → Behrouz A Forouzan

LAB NO.: 10

[OBSERVATION SPACE – LAB 10]

2.
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main (int argc, char *argv[])
{
    int psize, pno, pgno, offset, logaddr;
    printf("Enter page size: ");
    scanf("%d", &psize);
    printf("Enter total pages: ");
    scanf("%d", &pno);
    logaddr = atoi(argv[1]);
    pgno = logaddr / psize;
    offset = logaddr % psize;
    printf("Page number: %d\n", pgno);
    printf("Page offset: %d\n", offset);
    return 0;
}
```

Output:

```
a/a.out 204
Enter page size: 32
Enter total pages: 8
Page number = 6
Page offset = 12
```

Counting-Based Page Replacement:

Keep a counter of the number of references that have been made to each page, and develop the following two schemes:

- **The least frequently used (LFU) Algorithm:** replaces page with smallest count
  - Suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.
- **The most frequently used (MFU) Algorithm:** replaces the page with the largest count
  - Based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Lab Exercises:

1. Write a C program to simulate the following page replacement algorithms
   (i) FIFO (ii) Optimal (iii) LRU
2. Write a C program to simulate LRU approximation page replacement using second chance algorithm.

Additional Exercises:

1. Write a C program to simulate the following page replacement algorithms.
   (i) LFU (ii) MFU
2. Write a C program to simulate the following LRU approximation page replacement algorithms. (a) Additional Reference byte algorithm (b) Enhanced second chance algorithm.

[OBSERVATION SPACE – LAB 11]

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
   int p;
   printf("Enter page size: ");
   scanf("%d", &p);
   int a[p], i, count=0, n, marker=0, flag;
   for(i=0;i<p;i++)
     a[i]=-1;
   do
   {
```

[OBSERVATION SPACE – LAB 11]

```
flag = 0;
printf("Enter page number (-1 to stop): ");
scanf("%d", &n);
for (i=0; i<p; i++)
    if (a[i] == n)
    {
        flag=1;
        printf("Page found at position %d", i+1);
        break;
    }
If ((!flag) && (n!=-1))
{   a[marker] = n;
    count++;
    marker+=1;
    if (marker==p)
        marker=0;
    for(i=0; i<p; i++)
        printf("%d", a[i]);
}
{while(n!=-1);
printf("Total page faults= %d\n", count);
}
```

Output:

```
Enter page size: 3
Enter page number (-1 to stop): 7
7 -1 -1
Enter page number (-1 to stop): 0
7 0 -1
Enter page number (-1 to stop)148: 1
7 0 1
```

[OBSERVATION SPACE – LAB 11]

```
Enter page number (-1 to stop): 2
2 0 1
Enter page number (-1 to stop): 0
Page found at position 2
Enter page number (-1 to stop): -1
Total page faults = 4
```

(iv).

```
#include<stdio.h>
#include<stdlib.h>
void main()
{  int p;
   printf("Enter page size: ");
   scanf("%d", &p);
   int a[p], count[p], i, marker=0, n, counter=0, flag, max;
   for(i=0; i<p; i++)
   {  a[i]=-1;
      count[i]=0;
   }
   do
   {
      flag=max=0;
      printf("Page number (-1 to stop): ");
      scanf("%d", &n);
      for(i=0; i<p; i++)
         if(a[i]==n)
         {  flag=1;
            printf("Page found at position %d", i+1);
            break;
         }
      if(flag)
      {
```

[OBSERVATION SPACE – LAB 11]

```
        count[i]=0;
        for(int j=0;j<p;j++)
            if(j!=i)
                count[j]++;
    }
    else if((!flag)&&(n!=-1))
    {
        for(i=p-1;i>=0;i--)
            if(count[i]>=max)
            {
                max=count[i];
                marker=i;
            }
        a[marker]=n;
        count[marker]=0;
        for(int j=0;j<p;j++)
            if(j!=marker)
                count[j]++;
        counter++;
        for(i=0;i<p;i++)
            printf(" %d",a[i]);
    }
}while(n!=-1);
printf("Total page faults = %d\n",counter);
}
```

Output:

```
Enter page size: 3
Page number(-1 to stop): 7
7 -1 -1
Enter number (-1 to stop): 0
7 0 -1
```

[OBSERVATION SPACE – LAB 11] LAB NO: 11

```
Page number (-1 to stop): 1
1 0 1
Page number (-1 to stop): 2
2 0 1
Page number (-1 to stop): 0
Page found at position: 2
Page number (-1 to stop) -1
Total page faults: 4
```

2

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int p;
    printf("Page size: ");
    scanf("%d", &p);
    int a[p], ref[p], counter=0, i, flag, n, marker=0;
    for(i=0;i<p;i++)
    {   a[i]=-1;
        ref[i]=1;
    }
    do
    {
        flag=0;
        printf("\n Page number (-1 to stop): ");
        scanf("%d", &n);
        for(i=0; i<p;i++)
            if(a[i]==n)
            {   ref[i] = flag=1;
                if(marker==1)
```

[OBSERVATION SPACE – LAB 11]

```
            marker++;
            if (marker == p)
                marker = 0;
        }
        printf("Page found at position %d", i+1);
        break;
    }
if((!flag) && (n!=-1))
{
    while (marker <p)
    {
        if(ref[marker] == 1)
        {
            ref[marker]=0;
            marker++;
        }
        else if (ref[marker]==0)
            break;
        if (marker==p)
            marker=0;
}
{
a[marker]=n;
ref[marker]=1;
counter++;
printf("Page frame: ");
for(i=0; i<p; i++)
    printf("%d", a[i]);
printf("Reference: ");
for(i=0; i<p; i++)
    printf("%d", ref[i]);
marker++;
if (marker==p)
    marker=0;
```

[OBSERVATION SPACE – LAB 11]

```
    }
    while (n!=-1);
    printf("Total page faults= %d\n", counter);

}
```

Output:

```
Page size: 3
Page number (-1 to stop): 7
Pages: 7 -1 -1
Reference: 1 0 0
Page number (-1 to stop): 0
Pages: 7 0 -1
Reference: 1 1 0
Page number (-1 to stop): 1
Pages: 7 0 1
Reference: 1 1 1
Page number (-1 to stop): 2
Pages: 2 0 1
Reference: 1 0 0
Page number (-1 to stop): 0
Page found at position 2
Page number (-1 to stop): -1
Total page faults = 4.
```

Result
25/10/23

C-LOOK

This scheduling algorithm is Circular LOOK is like a C-SCAN which uses a return sweep before processing a set of disk requests. It does not reach the end of the tracks unless there is a request, either read or write on such disk location similar with the LOOK algorithm. Using the same sets of example in FCFS the solutions are as follows:
THM = |65-53|+|67-65|+|98-67|+|122-98|+|124-122|+|183-124|+|37-14| =
12+2+31+24+2+59+23 = 153 tracks

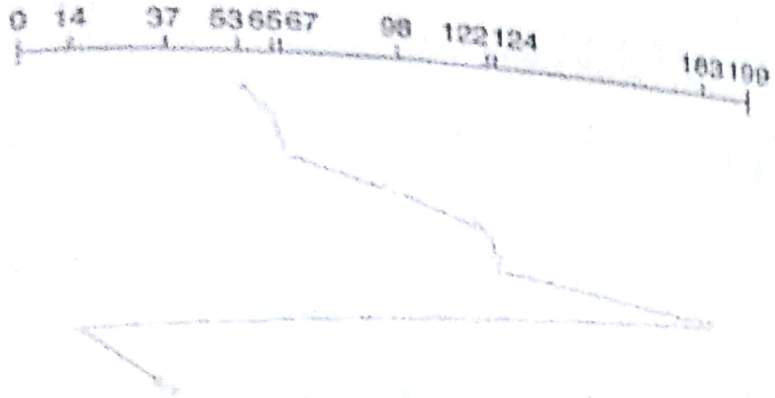


Figure 12.5: C-LOOK disk scheduling

Lab Exercises:

1. Write a C Program to simulate the following algorithms find the total no. of cylinder movements for various input requests
   (i) FCFS (ii) SSTF (iii) SCAN (iv) C-SCAN

Additional Exercises:

1. Write a C Program to simulate the following algorithms find the total no. of cylinder movements for various input requests
   (i) LOOK (ii) C-LOOK

[OBSERVATION SPACE – LAB 12]

1. (i)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
Void main()
{
    int head, diskqueue[30], n, THM=0;
    Printf("Enter head position: ");
    scanf("%d", &head);
    Printf("Enter number of elements in diskqueue: ");
    scanf("%d", &n);
```

[OBSERVATION SPACE – LAB 12]

```
    printf("Enter elements: ");
    for (int i=0; i<n; i++)
        scanf("%d", &diskqueue[i]);
    THM = abs(head - diskqueue[0]);
    for (int i=1; i<n; i++)
        THM += abs(diskqueue[i] - diskqueue[i-1]);
    printf("Total Head Movements = %d", THM);
}
```

Output:

```
Enter head position: 53
Enter number of elements in disk queue: 8
Enter elements: 98 183 37 122 14 124 65 67
Total Head Movements = 640.
```

(ii)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <limits.h>
void main()
{
    int head, i, diskqueue[30], n, THM=0, visited[30];
    printf("Enter head position:");
    scanf("%d", &head);
    printf("Enter number of elements in disk queue: ");
    scanf("%d", &n);
    printf("Enter elements: ");
    for (i=0; i<n; i++)
        scanf("%d", diskqueue[i]);
    memset(visited, 0, n*sizeof(int));
```

```
int min = INT_MAX; next;
for (i=0; i<n; i++)
    if (abs (head - diskqueue[i]) < min)
    {   min = abs (head - diskqueue[i]);
        next = i;
    }
visited [next] = 1;
THM = min;
i = 1;
head = diskqueue [next];
while (i<n)
{   min = INT_MAX;
    for (int j=0; j<n; j++)
    {   if ((!visited [j]) && (abs (head - diskqueue[j]) < min))
        {   min = abs (head - diskqueue[j]);
            next = j;
        }
    }
    head = diskqueue [next];
    visited [next] = 1;
    THM += min;
    i++;
}
printf("Total Head Movements = %d\n", THM);
}
```

Output:

```
Enter head position: 53
Enter number of elements in diskqueue: 8
Enter elements: 98 183 37 122 14 124 65 67
Total Head Movements = 036. 161
```

LAB NO.: 12

[OBSERVATION SPACE – LAB 12]

(iii)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void main()
{
    int head, diskqueue[30], n, THM=0, i, j, temp;
    printf("Enter head position: ");
    scanf("%d", &head);
    printf("Enter number of elements in disk queue: ");
    scanf("%d", &n);
    printf("Enter elements: ");
    for(i=0; i<n; i++)
        scanf("%d", &diskqueue[i]);
    for(i=0; i<n; i++)
        for(j=0; j<n-1-i; j++)
            if(diskqueue[j] > diskqueue[j+1])
            {
                temp = diskqueue[j];
                diskqueue[j] = diskqueue[j+1];
                diskqueue[j+1] = temp;
            }
    for(i=0; i<n; i++)
        if(diskqueue[j] > head)
            break;
    for(j=i-1; j>=0; j--)
    {
        THM+ = abs(head - diskqueue[j]);
        head = diskqueue[j];
    }
    THM+ = head;
    head = diskqueue[i];
    THM+ = head;
```

[OBSERVATION SPACE – LAB 12]

```
for(j=i+1; j<n; j++)
{   THM+=abs(head-diskqueue[j]);
    head=diskqueue[j];

}
printf("Total Head Movements = %d\n", THM);

}
```

Output:

```
Enter head position: 53
Enter number of elements in disk queue: 8
Enter elements: 98 183 37 122 14 124 65 67
Total Head Movements = 236.
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
void main()
{
    int head, diskqueue[30], n, THM=0, i, j, temp;
    printf("Enter head position: ");
    scanf("%d", &head);
    printf("Enter number of elements in disk queue: ");
    scanf("%d", &n);
    printf("Enter elements: ");
    for(i=0; i<n; i++)
        scanf("%d", &diskqueue[i]);
    for(i=0; i<n; i++)
        for(j=0; j<n-i-1; j++)
            if(diskqueue[j]>diskqueue[j+1])
            {   temp=diskqueue[j];
```

[OBSERVATION SPACE – LAB 12]

```
                diskqueue[j] = diskqueue[j+1];
                diskqueue[j+1] = temp;
            }
        }
    for(i=0; i<n; i++)
        if(diskqueue[i]>head)
            break;
    for(j=i; j<n; j++)
    {   THM+=abs(head - diskqueue[j]);
        head = diskqueue[j];
    }
    THM+=199-head;
    head=0;
    for(j=0; j<i; j++)
    {
        THM+=abs(head-diskqueue[j]);
        head = diskqueue[j];
    }
    printf("Total Head Movements = %d\n", THM);
}
```

Output:

```
Enter head position: 53
Enter number of elements in diskqueue: 8
Enter elements: 98 183 37 122 14 124 65 67
Total Head Movements = 183
```

25/10