CST 206 - OPERATING SYSTEMS LAB

SOURCE CODE AND OUTPUT

Emmanuel Jojy

53 - S4 CSE A

104/19

Experiment 1

Basic Linux Commands

1. Create an empty file without using text editor.
   ```
   $ touch f1.txt
   ```

2. Create three empty files without using text editor.
   ```
   $ touch f1.txt, f2.txt, f3.txt
   ```

3. Create a file with content “hello, World!”, and print its content to the screen without using text editor.
   ```
   $ echo “hello, World!” > f1.txt
   ```

4. Command which translates lower case vowels to uppercase.
   ```
   $ tr aeiou AEIOU
   ```

5. Command to read from standard input and write to standard output and one or more files.
   ```
   $ tee f.txt
   ```

6. Create a directory structure “ktu/fisat/cseb/student”
   ```
   $ mkdir -p ktu/fisat/cseb/stud
   ```

7. Print current working directory.
   ```
   $ pwd
   ```

8. Print the location of bash.
   ```
   $ whereis bash
   ```

9. Print file type of bash.
   ```
   $ file /usr/bin/bash
   ```

10. Create a C program that prints “hello, World!”. Print its current permission, change permission to executable and print the permissions again.

```
$ nano hello.c
$ ls -l hello.c
$ chmod +x hello.c
$ ls -l hello.c
```

11. Print file type of above created file.

```
$ file hello.c
```

12. Command to print “hello, World!”.

```
$ echo “hello, World!”
```

13. Command to print previously executed commands.

```
$ history
```

14. Move two levels upward from current working directory.

```
$ cd ../..
```

15. Print count of number of files in directory.

```
$ ls | wc -l
```

Experiment 2

Familiarization of Shell Script

1. Write shell script equivalent of the following pseudocode.

```
# pseudocode
function gcd(a,b)
   while a ≠ b
       if a > b
            a := a – b
       else
            b := b – a
   return a
```

```
# Shell Script
a=$1
b=$2
echo -n "GCD($a, $b) = "
while ((a!=b))
do
   if ((a>b))
   then
       ((a=a-b))
   else
       ((b=b-a))
   fi
done
echo "$a"
```

```
# Output
$ bash p1_gcd.sh 10 5
GCD(10, 5) = 5
```

2. Write shell script equivalent of the following pseudocode.

```
# pseudocode
function gcd(a,b)
   while b ≠ 0
       t := b
       b := a mod b
       a := t
   return a
```

```
# Shell Script
a=$1
b=$2
echo -n "GCD($a, $b) = "
while ((b!=0))
do
   ((t=b))
   ((b=a%b))
   ((a=t))
done
echo "$a"
```

```
# Output
$ bash p2_gcd.sh 10 5
GCD(4, 3) = 1
```

3. Write shell script equivalent of the following pseudocode.

```
# pseudocode
Read number
 Fact = 1
 i = 1
```

```
while i <= number
    Fact = Fact * i
    i = i + 1
end while
write Fact
```

```
# Shell Script
read number
fact=1

i=1
while((i<=number))
do
   ((fact*=i))
   ((i++))
done

echo "fact($number) = $fact"
```

```
# Output
$ bash p3_fact.sh
5
fact(5) = 120
```

4. Write a shell script that reads two numbers and an arithmetic operator, and display the result.

```
# Shell Script
echo -n "a: "
read a
echo -n "b: "
read b
```

```
echo -n "operator: "
read op

if [[ "$op" == "+" ]]
then
    ((res = a + b))
    echo "$res"
elif [[ "$op" == '-' ]]
then
    ((res = a - b))
    echo "$res"
elif [[ "$op" == '*' ]]
then
    ((res = a * b))
    echo "$res"
elif [[ "$op" == '/' ]]
then
    ((res = a / b))
    echo "$res"
else
    echo "Invalid operator"
fi
```

```
# output
$ bash p4_op.sh
a: 5
b: 10
operator: +
15
```

5. Write a shell script to convert temperature from Celsius to Fahrenheit.

```
# Shell Script
echo -n "T: "
read t
```

```
((res = t * 9/5))
((res = res + 32))

echo "$res"
```

```
# output
$ bash p5_temp.sh
T: 0
32
```

6. Write a shell script to find largest among three numbers inputted.

```
# Shell Script
echo -n "a: "
read a
echo -n "b: "
read b
echo -n "c: "
read c

if ((a > b && a > c))
then
    echo "$a is greatest"
elif ((b > a && b > c))
then
    echo "$b is greatest"
else
    echo "$c is greatest"
fi
```

```
# output
$ bash p6_great.sh
a: 5
```

```
b: 10
c: 15
15 is greatest
```

7. Write a shell script to perform the following task according to user choice menu.
   a. Area of Circle
   b. Circumference of Circle
   c. Area of Rectangle
   d. Area of Square

```
# Shell Script
echo "1. Area of Circle"
echo "2. Circumference of Circle"
echo "3. Area of Recatangle"
echo "4. Area of Square"

echo -n "Enter Choice: "
read ch

if ((ch == 1))
then
    echo -n "R = "
    read r
    ((a = 22 / 7 * r * r))
    echo "Area = $a"

elif ((ch == 2))
then
    echo -n "R = "
    read r
    ((a = 2 * 22 / 7 * r))
    echo "Circumference = $a"

elif ((ch == 3))
```

```
then
    echo -n "L = "
    read l
    echo -n "B = "
    read b
    ((a = l * b))
    echo "Area = $a"

elif ((ch == 4))
then
    echo -n "x = "
    read r
    ((a = r * r))
    echo "Area = $a"

else
    echo "Invalid Choice"
fi
```

```
# output
$ bash p7_ch.sh
1. Area of Circle
2. Circumference of Circle
3. Area of Recatangle
4. Area of Square
Enter Choice: 4
x = 5
Area = 25
```

8. Write a shell script to generate Fibonacci series up to n.

```
# Shell Script
echo -n "n: "
read n
a=0
```

```
b=1
echo -n "fib series: $a $b "
((x = a + b))

while ((x < n))
do
    echo -n "$x "
    ((a = b))
    ((b = x))
    ((x = a + b))
done

echo ""
```

```
# output
$ bash p8_fib.sh
n: 30
fib series: 0 1 1 2 3 5 8 13 21
```

9. Create an empty file without using text editor.

```
$ touch f.txt
```

10. Write a shell script to print given three alphabets in alphabetical order. Input should be taken from command line.

```
# Shell Script
a=$1
b=$2
c=$3

if [[ $a > $c ]]
then
   temp=$a
   a=$c
   c=$temp
```

```
fi
if [[ $a > $b ]]
then
   temp=$a
   a=$b
   b=$temp
fi

if [[ $b > $c ]]
then
   temp=$b
   b=$c
   c=$temp
fi
echo "Alphabetic Order -> $a, $b, $c"
```

```
# output
$ bash p9_alpha.sh d a
Alphabetic Order -> a, d, z
```

11. Write a shell script to print given three strings in dictionary order.

```
# Same Script as previous applies

# Shell Script
a=$1
b=$2
c=$3

if [[ $a > $c ]]
then
   temp=$a
   a=$c
   c=$temp
fi
```

```
if [[ $a > $b ]]
then
   temp=$a
   a=$b
   b=$temp
fi

if [[ $b > $c ]]
then
   temp=$b
   b=$c
   c=$temp
fi

echo "Alphabetic Order -> $a, $b, $c"
```

```
# output
$ bash p9_alpha.sh emmi emmanuel emma
Alphabetic Order -> emma, emmanuel, emmi
```

12. Write a shell script to do the following in order:
    a. Check whether a directory by the name “bash” exist
    b. If not create one.
    c. List all the directories.

```
# Shell Script
if [[ -d "bash" ]]
then
   echo "a. Directory bash exist"
else
   echo "a. Directory does not exist"
   mkdir bash
```

```
  echo "b. Directory bash created"
fi
echo "c. Listing"
ls
```

```
# output
$ bash p10_exist.sh
a. Directory does not exist
b. Directory bash created
c. Listing
bash  p11_greet.sh  p14_odd.sh  p3_fact.sh
p6_great.sh  p9_alpha.sh   test.sh
file_exist.sh  p12_avg.sh  p1_gcd.sh  p4_op.sh
p7_ch.sh  readme.txt  p10_exist.sh  p13_op.sh
p2_gcd.sh  p5_temp.sh  p8_fib.sh  redirect.sh
```

13. Write a Bash script which accepts as input your name and displays the greeting “Hello name”.

```
# Shell Script
echo -n "Name: "
read name
echo "Hello, $name"
```

```
# output
$ bash p11_greet.sh
Name: Emmanuel Jojy
Hello, Emmanuel Jojy
```

14. Given n integers, compute their average.

```
# Shell Script
echo -n "N: "
read n
sum=0
echo "Enter Numbers: "
for ((i=0; i<n; i++))
do
    read a
    ((sum += a))
done
((avg = sum / n))
echo "Average = $avg"
```

```
# output
$ bash p12_avg.sh
N: 4
Enter Numbers:
10
20
30
40
Average = 25
```

15. Given two integers A and B, find their sum, difference, product, and quotient.

```
# Shell Script
a=$1
b=$2
echo "A = $a, B = $b"

((res = a + b))
echo "Sum = $res"

((res = a - b))
echo "Dif = $res"
```

```
((res = a * b))
echo "Pdt = $res"

((res = a / b))
echo "Div = $res"
```

```
# output
$ bash p13_op.sh 10 5
A = 10, B = 5
Sum = 15
Dif = 5
Pdt = 50
Div = 2
```

16. Use for loops to display only odd natural numbers from 0 to 99.

```
# Shell Script
echo "Displaying Odd Numbers in [0 99]"
for ((i=1; i<=99; i+=2))
do
   echo -n "$i "
done
```

```
# output
$ bash p14_odd.sh
Displaying Odd Numbers in [0 99]
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37
39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71
73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

Experiment 3

System Calls

1. Write a C program to create a new child process and print "I'm child" in child process and "I'm Parent" in parent process.

```c
// Source Code

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

int main(){
   pid_t p = fork();
   if(p < 0){
      printf("Error in fork");
   }
   else if(p == 0){
      printf("I'm Child.");
   }
   else{
      printf("I'm Parent.");
   }

   printf("\n");
   return 0;
}
```

```
# output

$ gcc -g -o out p1.c && ./out
I'm Parent.
I'm Child.
```

2. Write a C program that prints 1...10 in both parent and child process. Explain the output.

```
// Source Code

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

int main(){
   printf("P = Parent Process, C = Child Process\n");
   pid_t p = fork();
   if(p < 0){
      printf("Error in fork");
   }
   else if(p == 0){
      for(int i=1;i<=10;i++){
          printf("C%d ", i);
      }
   }
   else{
      for(int i=1;i<=10;i++){
          printf("P%d ", i);
      }
      wait(NULL);
      printf("\n");
   }
   return 0;
}
```

```
# output

$ gcc -g -o out p2.c && ./out
P = Parent Process, C = Child Process
P1 P2 P3 P4 P5 P6 C1 P7 P8 C2 P9 C3 P10 C4 C5 C6 C7
C8 C9 C10
```

3. Write a C program stat.c, that prints file type and mode of a given file.

```
// Source Code

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

long int tobin(int);
void analyse(long int);

void main(){
   char name[25], loc[50];
   struct stat buf;
   int mode;

   printf("Enter file name (in current directory):
");
   scanf("%s", name);

   int status = stat(name, &buf);
   if(status == -1){
      printf("File not found.\n");
      return;
   }
   mode = buf.st_mode;
   printf("MODE = %d\n", mode);
   printf("TYPE = ");
   if(S_ISDIR(mode))
      printf("DIRECTORY\n");
   if(S_ISREG(mode))
      printf("REGULAR FILE\n");
   analyse(tobin(buf.st_mode));
   printf("\n");
}

long int tobin(int n){
    long int temp = 0, b = 0;
    while(n != 0){
        temp = (temp * 10) + (n % 2);
        n /= 2;
    }
    while(temp != 0){
        b = (b * 10) + (temp % 10);
        temp /= 10;
    }
    return b;
}

void analyse(long int n){
    char s[4] = {'X', 'W', 'R'};
    int i, j, p;
    printf("\nPermission\n");
    printf("\tWORLD  GROUP  ROOT\n\t");
    if(n == 1){
        printf("- - -  - - -  - - -\n");
        return;
    }
    for(i = 0; i < 3; i++){
        p = n % 1000;
        n = n / 1000;
        for(j = 0; j < 3; j++, p /= 10){
            if(p%10 == 1)
                printf("%c ", s[j]);
            else
                printf("- ");
        }
        printf(" ");
    }
}
```

```
# output

$ gcc -g -o out p3.c && ./out
Enter file name (in current directory): out
```

```
MODE = 33279
TYPE = REGULAR FILE

Permission
        WORLD  GROUP  ROOT
        X W R  X W R  X W R
```

4. Write a C program that checks whether a directory by name “FISAT” exists in the current directory.

```
// Source Code

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

void main(){
   struct stat buf;
   if(stat("FISAT", &buf) == -1 &&
!S_ISDIR(buf.st_mode)){
      printf("DIRECTORY FISAT does not exist.\n");
      return;
   }
   printf("DIRECTORY FISAT exist.\n");
}
```

```
# output

$ gcc -g -o out p4.c && ./out
DIRECTORY FISAT exist.
```

5. Write a C program that prints PID of itself and its parent. *You should create a child process, and PID of itself and its parent should be printed.*

```
// Source Code
```

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

int main(){
   pid_t p = fork();
   if(p < 0){
      printf("Error in fork");
   }
   else if(p == 0){
      printf("Child Process  -> PID: %d, PPID: %d",
getpid(), getppid());
   }
   else{
      printf("Parent Process -> PID: %d, PPID: %d",
getpid(), getppid());
   }
   printf("\n");
   return 0;
}
```

```
# output

$ gcc -g -o out p5.c && ./out
Parent Process -> PID: 107, PPID: 9
Child Process  -> PID: 108, PPID: 107
```

6. Write a C program with 4 calls to fork() and print ”Hello, World!” after that. Explain the output.

```
// Source Code

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

```
int main(){
  fork(); fork(); fork(); fork();
  printf("hello, World!\n");
  wait(NULL);
  return 0;
}
```

```
# output

$ gcc -g -o out p6.c && ./out
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
hello, World!
```

7. Create a C program that does the following:
    a. Takes a filename as argument.
    b. Create a child process.
    c. Execute `cat` command in the child.
    d. Call `wait()` so that parent is blocked until child terminates.

```
// Source Code

#include <stdio.h>
```

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

void main(int argc, char *argv[]){
   if(argc != 2){
      printf("Argument Error.\n");
      return;
   }
   char *name = argv[1];
   int status;

   pid_t p = fork();
   if(p < 0)
      printf("Fork Error.\n");
   else if(p == 0)
      wait(&status);
   else
      execlp("/usr/bin/cat", "cat", name, NULL);
}
```

```
# output

$ gcc -g -o out p7.c && ./out a.txt
*************
hello, World!
Emmanuel Jojy
Bye.
*************
```

8. Create a C program to list directories in the given directory.

```
// Source Code

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

```
#include <dirent.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>

void main(){
   DIR *d;
   int type;
   struct dirent *de;
   d = opendir(".");
   printf("Listing Directories:\n");
   while(de = readdir(d)){
       type = de->d_type;
       if(type == 4)
           printf("%s\n", de->d_name);
   }
}
```

```
# output

$ gcc -g -o out p8.c && ./out
Listing Directories:
.
..
demo
```

9. Create a C program to list files in the given directory.

```
// Source Code

#include <stdio.h>
#include <unistd.h>
#include <dirent.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <sys/wait.h>
void main(){
   DIR *d;
   int type;
   struct dirent *de;
   d = opendir(".");
   printf("Listing Files:\n");
   while(de = readdir(d)){
       type = de->d_type;
       if(type == 8)
           printf("%s\n", de->d_name);
   }
}
```

```
# output

$ gcc -g -o out p9.c && ./out
Listing Files:
a.txt
b.txt
fileop.c
filewr.c
p1
p1.c
p2
p2.c
p3
p3.c
p4
p4.c
start.c
```

10. Write a C program that does the following.
    a. Takes two filenames as argument.
    b. Creates a file with name as second argument.
    c. Copies content from ***file_1*** to ***file_2***.
    d. First line of ***file_2*** should be **START** and last line should be **STOP**.

```
// Source Code

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

#include <unistd.h>
#include<fcntl.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>

void main(int argc, char *argv[]){
   if(argc != 3){
       printf("Argument Error.\n");
       return;
   }

   char *a = argv[1], *b = argv[2];
   char buf[128] = "";
   int rfd, wfd, stat;

   // only read
   rfd = open(a, O_RDONLY);

   // only write
   wfd = open(b, O_WRONLY);
   if(wfd == -1)
       wfd = open(b, O_WRONLY | O_CREAT);
   write(wfd, "START\n", 6);
   while(read(rfd, buf, 1) > 0){
       write(wfd, buf, strlen(buf));
   }
   write(wfd, "\nSTOP\n", 6);

   close(rfd);
   close(wfd);
```

```
  printf("Copied Contents from %s -> %s\n", a, b);
  return;
}
```

```
# output
# The source and target files were verified.

$ gcc -g -o out p10.c && ./out a.txt b.txt
Copied Contents from a.txt -> b.txt
```

Experiment 4

Inter-Process Communication

1. Write a program where first process sends a number to second process and calculates the factorial of that number. (Use shared memory concept)

```
// Source code
// Sender Process Program - fact_a.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>

void main()
{
   key_t key = 12345;
   printf("Writing to Shared Memory (Key = %d)\n",
key);

   int shmid = shmget(key, sizeof(int), 0666 |
IPC_CREAT);
   printf("shmid:\t%d\n", shmid);

   void *shmad = shmat(shmid, NULL, 0);
   printf("shmad:\t%p\n", shmad);

   int n;
   printf("\nFactorial:\n");
   printf("N = ");
   scanf("%d", &n);
   sprintf(shmad, "%d", n);
   printf("\nWrite '%d' to SHM complete.\n\n", n);
}
```

```
// Source code
// Receiver Process Program – fact_b.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/shm.h>

void main()
{
   key_t key = 12345;

   printf("Reading from Shared Memory (Key = %d)\n",
key);

   int shmid = shmget(key, sizeof(int), 0666);
   printf("shmid:\t%d\n", shmid);

   if (shmid == -1)
   {
       printf("Error accessing shared memory.
Failure.\n");
       return;
   }

   void *shmad = shmat(shmid, NULL, 0);
   printf("shmad:\t%p\n", shmad);

   int n = atoi((char *)shmad);
   printf("\nRead '%d' from SHM complete.\n", n);

   shmdt(shmad);
   shmctl(shmid, IPC_RMID, 0);
   printf("SHM destroyed.\n\n");

   long int res = 1;
   for (int i = 1; i <= n; i++)
   {
      res *= i;
   }

   printf("fact(%d) = %ld\n\n", n, res);
}
```

```
# output

$ gcc -g -o write fact_a.c && ./write
Writing to Shared Memory (Key = 12345)
shmid:  1
shmad:  0x7f3920714000
Factorial:
N = 10
Write '10' to SHM complete.

$ gcc -g -o read fact_b.c && ./read
Reading from Shared Memory (Key = 12345)
shmid:  1
shmad:  0x7f99c7ec3000
Read '10' from SHM complete.
SHM destroyed.
fact(10) = 3628800
```

2. Write a program that implements inter-process communication using shared memory.

```
// Source code
// Sender Process Program - sender.c

#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

void main()
{
    char s[7] = "/oslab2";
    printf("Writting to SHM (name = '%s')\n", s);

    int fd = shm_open(s, O_CREAT|O_RDWR, 0666);
    printf("File Descriptor: %d\n", fd);

    ftruncate(fd, 2*sizeof(int));

    int *shmad = mmap(NULL, 2*sizeof(int),
PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
    printf("Virtual Address: %p\n", shmad);

    printf("\nA: ");
    scanf("%d", &shmad[0]);
    printf("B: ");
    scanf("%d", &shmad[1]);
    printf("\nValues written to SHM.");

    munmap(shmad, 2*sizeof(int));
    printf("\nMemory Unmapped.\n\n");
    close(fd);
}
```

```
// Source code
// Receiver Process Program – receiver.c

#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

void main()
{
    char s[7] = "/oslab2";
    printf("Reading from SHM (name = '%s')\n", s);

    int fd = shm_open(s, O_RDONLY, 0666);
    printf("File Descriptor: %d\n", fd);
    if (fd == -1)
   {
      printf("Error accessing shared memory.
Failure.\n");
      return;
   }

    int *shmad = mmap(NULL, 2*sizeof(int),
PROT_READ, MAP_SHARED, fd, 0);
    printf("Virtual Address: %p\n", shmad);

    printf("\n'%d' and '%d' read from SHM.\n",
shmad[0], shmad[1]);

    munmap(shmad, 2*sizeof(int));
    printf("\nMemory Unmapped.\n");
    shm_unlink(s);
    printf("Memory Unlinked.\n\n");
    close(fd);
}
```

```
# output

$ gcc -g -o sen sender.c -lrt && ./sen
Writting to SHM (name = '/oslab2')
File Descriptor: 3
Virtual Address: 0x7f872ecb9000

A: 25
B: 125

Values written to SHM.
Memory Unmapped.

$ gcc -g -o rec reciever.c -lrt && ./rec
Reading from SHM (name = '/oslab2')
File Descriptor: 3
```

```
Virtual Address: 0x7fa0eb0dd000

'25' and '125' read from SHM.

Memory Unmapped.
Memory Unlinked.
```

Experiment 5

Semaphores

1. Implement semaphores using the classic synchronization problem, Producer-Consumer

```
// Source code
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <unistd.h>
#include <time.h>
sem_t mutex, empty, full;
int buffer[5], in = 0, out = 0, item = 0;
void *producer(int *arg) {
    do {
        sem_wait(&empty);
        sem_wait(&mutex);

        buffer[in] = item;
        printf("\n[PRODUCER %d]: buffer[%d] <- %d",
*arg, in, item);
        in = (in + 1) % 5;
        item++;

        sem_post(&full);
        sem_post(&mutex);
        sleep(1);
    } while (1);
}
void *consumer(int *arg) {
    do {
        sem_wait(&full);
        sem_wait(&mutex);

        printf("\n[CONSUMER %d]: buffer[%d] -> %d",
*arg, out, buffer[out]);
        out = (out + 1) % 5;

        sem_post(&empty);
        sem_post(&mutex);
        sleep(1);
    } while (1);
}

void main() {
    int p, c;
    printf("\nBUFFER SIZE = 5\n");
    printf("Number of Producers: ");
    scanf("%d", &p);
    pthread_t prod[p];
    int np[p];
    printf("Number of Consumers: ");
    scanf("%d", &c);
    pthread_t cons[10];
    int nc[c];
    sem_init(&mutex, 0, 1);
    sem_init(&empty, 0, 5);
    sem_init(&full, 0, 0);
    int i;
    for (i = 0; i < p; i++) {
        np[i] = i;
        pthread_create(&prod[i], NULL, (void
*)producer, &np[i]);
    }
    for (i = 0; i < c; i++) {
        nc[i] = i;
        pthread_create(&cons[i], NULL, (void
*)consumer, &nc[i]);
    }
    for (i = 0; i < p; i++)
        pthread_join(prod[i], NULL);
    for (i = 0; i < c; i++)
        pthread_join(cons[i], NULL);
```

```
      printf("\n\n");
      sem_destroy(&mutex);
      sem_destroy(&full);
      sem_destroy(&empty);
  }
```

```
# output

$ gcc -g -o out prodcons.c -pthread && ./out

BUFFER SIZE = 5
Number of Producers: 5
Number of Consumers: 5

[PRODUCER 0]: buffer[0] <- 0
[PRODUCER 2]: buffer[1] <- 1
[PRODUCER 3]: buffer[2] <- 2
[PRODUCER 4]: buffer[3] <- 3
[PRODUCER 1]: buffer[4] <- 4
[CONSUMER 1]: buffer[0] -> 0
[CONSUMER 2]: buffer[1] -> 1
[CONSUMER 3]: buffer[2] -> 2
[CONSUMER 4]: buffer[3] -> 3
^C
```

Experiment 6/7

Scheduling Algorithms

1. Write a program to implement process scheduling algorithms:
   a. FCFS b. SJF c. Priority d. Round Robin

```
// Source code

#include <stdio.h>
#include <stdlib.h>
struct pcb {
    int i, a, b, br, s, p, c, t, w;
};
struct pcb *p;
struct qnode {
    struct pcb *node;
    struct qnode *link;
};
struct qnode *front = NULL, *rear = NULL;
struct gantt {
    int id;
    int time;
};
struct gantt *g;
int gmax = -1;
int algo, tot, tq = 0, flag = 0;
void addGantt(int id, int time) {
    gmax++;
    g[gmax].id = id;
    g[gmax].time = time;
}
void swap(struct pcb **a, struct pcb **b) {
    struct pcb *temp = *a;
    *a = *b;
    *b = temp;
}
```

```
void hLine(int tab) {
    int i;
    printf("\n");
    for (i = 0; i <= tab * 8; i++)
        printf("-");
}
void tabulate() {
    float tt = 0, wt = 0;
    int i, j, flag = 0;
    printf("\nGantt Chart\n");
    hLine(gmax * 2);
    printf("\n|\t");
    for (i = 0; i < gmax; i++) {
        if (flag == 1 && g[i].id == -1)
            continue;
        else if (g[i].id == -1) {
            flag = 1;
            printf("--\t|\t");
        }
        else {
            flag = 0;
            printf("P%d\t|\t", g[i].id);
        }
    }
    hLine(gmax * 2);
    printf("\n");
    for (i = 0; i <= gmax; i++) {
        if (flag == 1 && g[i].id == -1)
            continue;
        if (g[i].id == -1)
            flag = 1;
        else
            flag = 0;
        printf("[%d]\t\t", g[i].time);
    }
    printf("\n\t ");
    for (i = 0; i < tot; i++) {
        p[i].t = p[i].c - p[i].a;
        p[i].w = p[i].t - p[i].b;
        tt += p[i].t;
        wt += p[i].w;
    }
    printf("\n\nFINAL TABULATION (TIME QUANTUM = %d)",
tq);
    hLine(16);

printf("\n|\tID\t|\tAT\t|\tBT\t|\tPR\t||\tST\t|\tCT\t|
\tTT\t|\tWT\t|");
    hLine(16);
    for (i = 0; i < tot; i++) {
        struct pcb t = p[i];

printf("\n|\tP%d\t|\t%d\t|\t%d\t|\t%d\t||\t%d\t|\t%d\t
|\t%d\t|\t%d\t|", t.i, t.a, t.b, t.p, t.s, t.c, t.t,
t.w);
    }
    hLine(16);

printf("\n\t\t\t\t\t\t\t\t\t\t\t\t\t|\t%d\t|\t%d\t|",
(int)tt, (int)wt);
    hLine(16);
    printf("\n\nAverage TT = %f ms", tt / tot);
    printf("\nAverage WT = %f ms\n", wt / tot);
    printf("\n* (ID - PID, AT - Arrival, BT - Burst,
PR - Priority, ST - Start, CT - Completion, TT -
Turnaround, WT - Wait)\n");
    if (algo == 3)
        printf("* Assumption: low number represents
high priority (only for priority schedule).");
    printf("\n\n");
}
int complete() {
    int i;
    for (i = 0; i < tot; i++)
        if (p[i].br != 0)
            return 0;
    flag = 1;
    return 1;
}
void push(struct pcb *item) {
    struct qnode *p = malloc(sizeof(struct qnode));
    p->node = item;
    p->link = NULL;
    if (front == NULL) {
        front = p;
        rear = p;
    }
    else {
        rear->link = p;
        rear = p;
    }
}
struct pcb *pop() {
    if (front == NULL)
        return NULL;
    struct qnode *temp = front;
    struct pcb *item = front->node;
    if (front == rear)
    {
        front = NULL;
        rear = NULL;
    }
    else
    {
        front = front->link;
    }
    free(temp);
    return item;
}
void sort() {
    struct qnode *i, *j;
    for (i = front; i != NULL; i = i->link)
    {
        for (j = front; j->link != NULL; j = j->link)
        {
            if ((algo == 3) && j->node->br > j->link-
>node->br)
                swap(&j->node, &j->link->node);
            if ((algo == 4) && j->node->p > j->link-
>node->p)
                swap(&j->node, &j->link->node);
        }
    }
}
// (1) First Come First Serve
void fcfs()
{
    int pulse, i, flag = 0;
    struct pcb *current = NULL;
    for (pulse = 0; complete() == 0; pulse++) {
        for (i = 0; i < tot; i++)
            if (p[i].a == pulse)
                push(&p[i]);
        if (current != NULL) {
            current->br -= 1;
            if (current->br == 0) {
                current->c = pulse;
                current = NULL;
            }
        }
        if (current == NULL) {
            current = pop();
            if (current == NULL)
                addGantt(-1, pulse);
            else {
                addGantt(current->i, pulse);
                current->s = pulse;
            }
        }
    }
}
// (2) Shortest Job First
```

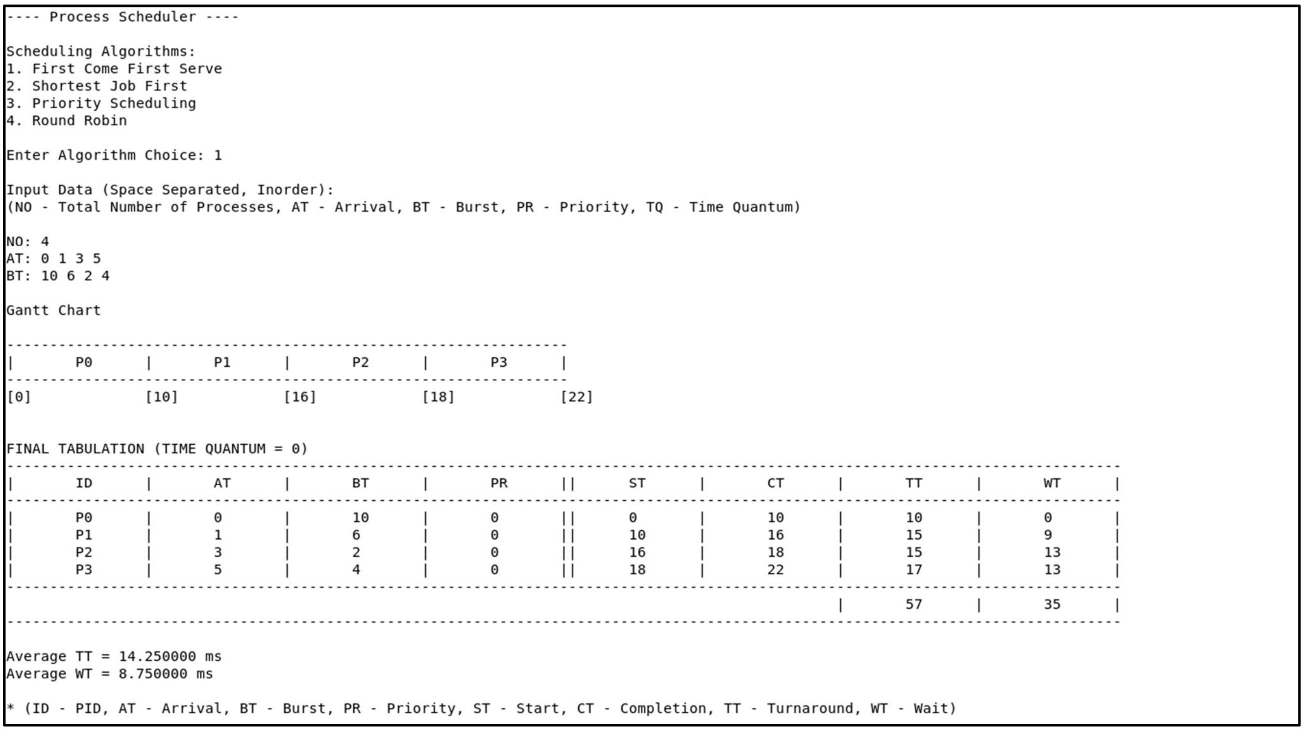
```
void sjf() {
    int pulse, i, flag = 0;
    struct pcb *current = NULL;
    for (pulse = 0; complete() == 0; pulse++) {
        for (i = 0; i < tot; i++)
            if (p[i].a == pulse)
                push(&p[i]);
        if (current != NULL) {
            current->br -= 1;
            if (current->br == 0) {
                current->c = pulse;
                current = NULL;
            }
        }
        if (current == NULL) {
            sort();
            current = pop();
            if (current == NULL)
                addGantt(-1, pulse);
            else {
                addGantt(current->i, pulse);
                current->s = pulse;
            }
        }
    }
}
// (3) Priority Scheduling
void pr() {
    int pulse, i, flag = 0;
    struct pcb *current = NULL;
    for (pulse = 0; complete() == 0; pulse++) {
        for (i = 0; i < tot; i++)
            if (p[i].a == pulse)
                push(&p[i]);
        if (current != NULL) {
            current->br -= 1;
            if (current->br == 0) {
                current->c = pulse;
                current = NULL;
            }
        }
        if (current == NULL) {
            sort();
            current = pop();
            if (current == NULL)
                addGantt(-1, pulse);
            else {
                addGantt(current->i, pulse);
                current->s = pulse;
            }
        }
    }
}
// (4) Round Robin
void rr() {
    int pulse, i, tqq = tq;
    struct pcb *current = NULL;
    for (pulse = 0; complete() == 0; pulse++) {
        for (i = 0; i < tot; i++)
            if (p[i].a == pulse)
                push(&p[i]);
        if (current != NULL) {
            current->br -= 1;
            tqq--;
            if (current->br == 0) {
                current->c = pulse;
                current = NULL;
            }
        }
        if (tqq == 0 && current != NULL) {
            tqq = tq;
            push(current);
            current = NULL;
        }
        if (current == NULL) {
            current = pop();
            tqq = tq;
            if (current == NULL)
                addGantt(-1, pulse);
            else {
                addGantt(current->i, pulse);
                current->s = pulse;
            }
        }
    }
}
int input() {
    int i;
    printf("\nInput Data (Space Separated,
Inorder):");
    printf("\n(NO - Total Number of Processes, AT -
Arrival, BT - Burst, PR - Priority, TQ - Time
Quantum)\n\n");
    printf("NO: ");
    scanf("%d", &tot);
    if (tot < 1) {
        printf("\n~ Invalid Number of Processes.\n");
        return -1;
    }
    p = malloc(sizeof(struct pcb) * tot);
    g = malloc(sizeof(struct gantt) * 100);
    printf("AT: ");
    for (i = 0; i < tot; i++) {
        scanf("%d", &p[i].a);
        if (p[i].a < 0) {
            printf("\n~ Invalid Arrival Time(s).\n");
            return -1;
        }
        p[i].i = i;
        p[i].p = 0;
    }
    printf("BT: ");
    for (i = 0; i < tot; i++) {
        scanf("%d", &p[i].b);
        if (p[i].b < 1) {
            printf("\n~ Invalid Burst Time(s).\n");
            return -1;
        }
        p[i].br = p[i].b;
    }
    if (algo == 3) {
        printf("PR: ");
        for (i = 0; i < tot; i++)
            scanf("%d", &p[i].p);
    }
    if (algo == 4) {
        printf("TQ: ");
        scanf("%d", &tq);
        if (tq < 1) {
            printf("\n~ Invalid Time Slice.\n");
            return -1;
        }
    }
    return 0;
}
void main() {
    printf("---- Process Scheduler ---- \n\n");
    printf("Scheduling Algorithms: \n");
    printf("1. First Come First Serve\n");
    printf("2. Shortest Job First\n");
    printf("3. Priority Scheduling\n");
    printf("4. Round Robin\n\n");
    printf("Enter Algorithm Choice: ");
    scanf("%d", &algo);
    if (algo < 1 || algo > 4) {
        printf("\n~ Invalid Choice.\n");
        return;
    }
    if (input() != -1) {
        switch (algo) {
        case 1: fcfs(); break;
        case 2: sjf(); break;
```

```
            case 4: pr(); break;
            case 6: rr(); break;
            }
            tabulate();
        }
        free(p);
        free(g);
        return;
    }
```

# output

# First Come First Serve

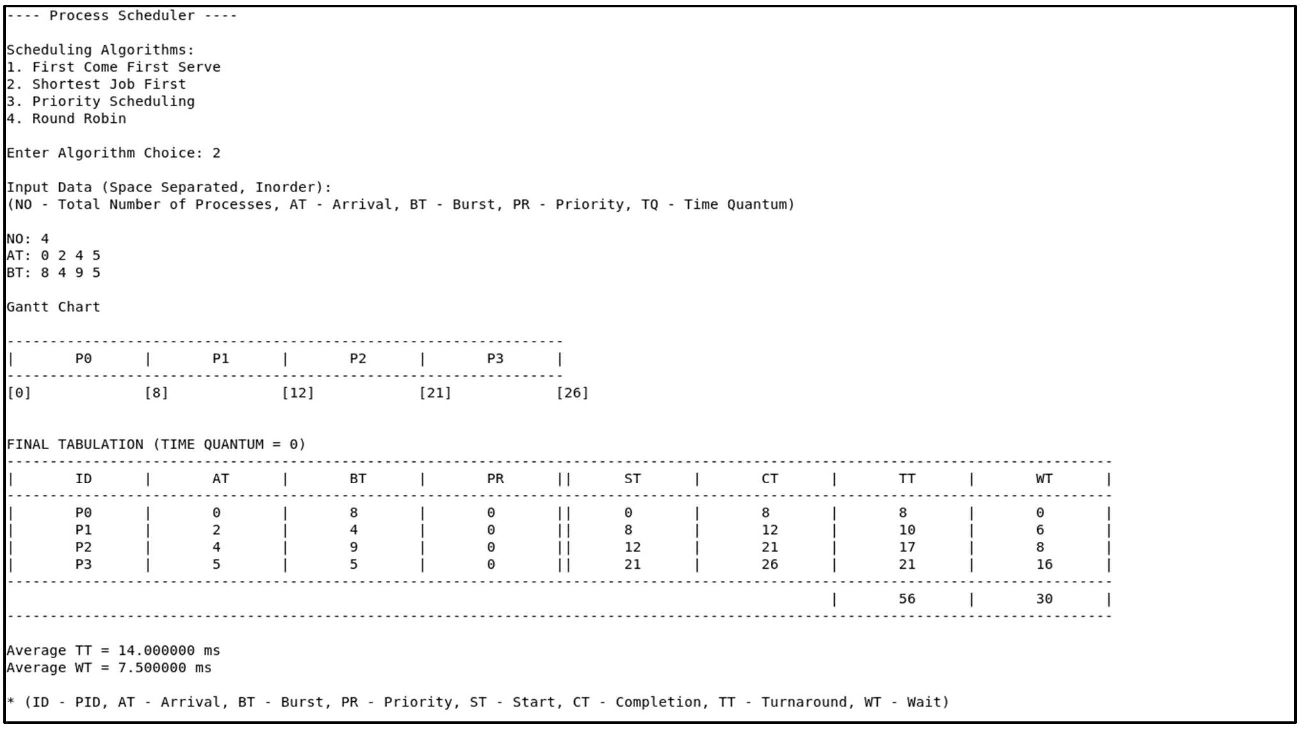
```
$ gcc -g -o out scheduler.c && ./out
```



```
---- Process Scheduler ----

Scheduling Algorithms:
1. First Come First Serve
2. Shortest Job First
3. Priority Scheduling
4. Round Robin

Enter Algorithm Choice: 1

Input Data (Space Separated, Inorder):
(NO - Total Number of Processes, AT - Arrival, BT - Burst, PR - Priority, TQ - Time Quantum)

NO: 4
AT: 0 1 3 5
BT: 10 6 2 4

Gantt Chart

-----------------------------------------------------------------
|       P0      |       P1      |       P2      |       P3      |
-----------------------------------------------------------------
[0]             [10]            [16]            [18]            [22]


FINAL TABULATION (TIME QUANTUM = 0)
---------------------------------------------------------------------------------------------------------------------------------
|       ID      |       AT      |       BT      |       PR      ||      ST      |       CT      |       TT      |       WT      |
---------------------------------------------------------------------------------------------------------------------------------
|       P0      |       0       |       10      |       0       ||      0       |       10      |       10      |       0       |
|       P1      |       1       |       6       |       0       ||      10      |       16      |       15      |       9       |
|       P2      |       3       |       2       |       0       ||      16      |       18      |       15      |       13      |
|       P3      |       5       |       4       |       0       ||      18      |       22      |       17      |       13      |
---------------------------------------------------------------------------------------------------------------------------------
                                                                                                |       57      |       35      |
---------------------------------------------------------------------------------------------------------------------------------

Average TT = 14.250000 ms
Average WT = 8.750000 ms

* (ID - PID, AT - Arrival, BT - Burst, PR - Priority, ST - Start, CT - Completion, TT - Turnaround, WT - Wait)
```

# Shortest Job First

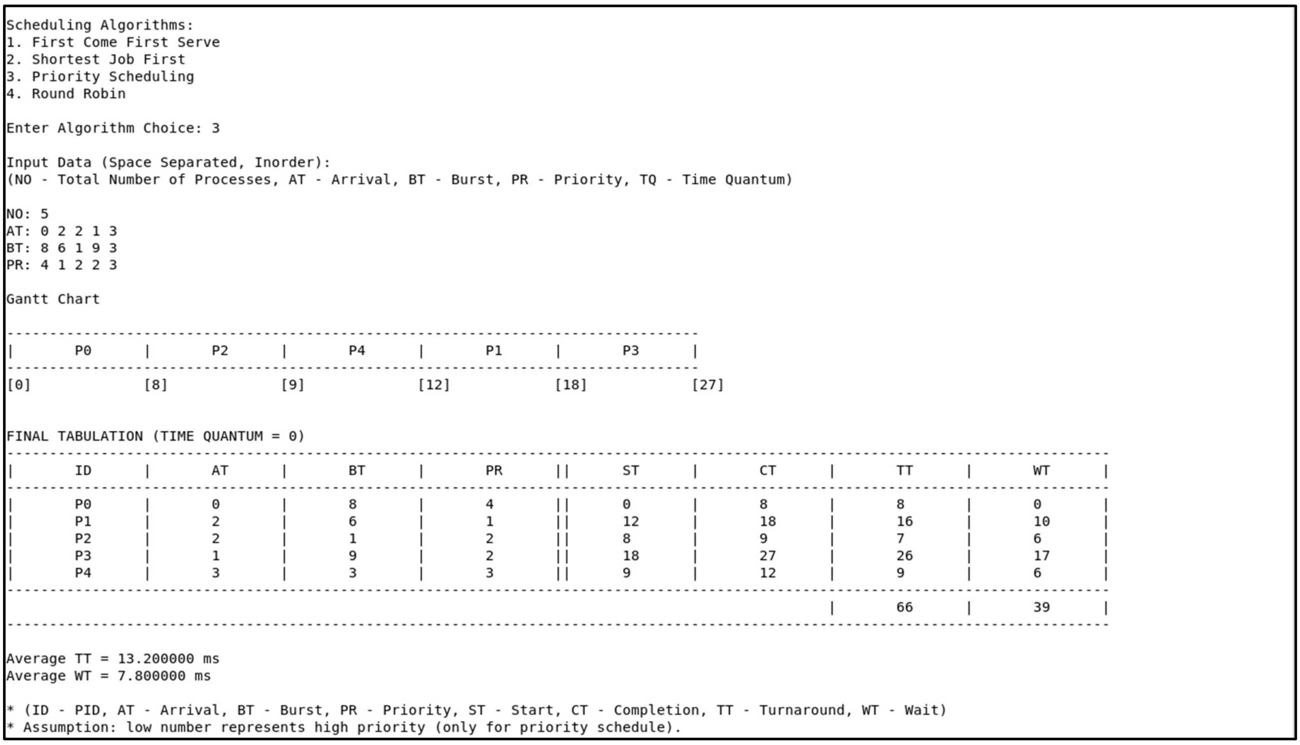
```
$ gcc -g -o out scheduler.c && ./out
```



```
---- Process Scheduler ----

Scheduling Algorithms:
1. First Come First Serve
2. Shortest Job First
3. Priority Scheduling
4. Round Robin

Enter Algorithm Choice: 2

Input Data (Space Separated, Inorder):
(NO - Total Number of Processes, AT - Arrival, BT - Burst, PR - Priority, TQ - Time Quantum)

NO: 4
AT: 0 2 4 5
BT: 8 4 9 5

Gantt Chart

-----------------------------------------------------------------
|       P0      |       P1      |       P2      |       P3      |
-----------------------------------------------------------------
[0]             [8]             [12]            [21]            [26]


FINAL TABULATION (TIME QUANTUM = 0)
---------------------------------------------------------------------------------------------------------------------------------
|       ID      |       AT      |       BT      |       PR      ||      ST      |       CT      |       TT      |       WT      |
---------------------------------------------------------------------------------------------------------------------------------
|       P0      |       0       |       8       |       0       ||      0       |       8       |       8       |       0       |
|       P1      |       2       |       4       |       0       ||      8       |       12      |       10      |       6       |
|       P2      |       4       |       9       |       0       ||      12      |       21      |       17      |       8       |
|       P3      |       5       |       5       |       0       ||      21      |       26      |       21      |       16      |
---------------------------------------------------------------------------------------------------------------------------------
                                                                                                |       56      |       30      |
---------------------------------------------------------------------------------------------------------------------------------

Average TT = 14.000000 ms
Average WT = 7.500000 ms

* (ID - PID, AT - Arrival, BT - Burst, PR - Priority, ST - Start, CT - Completion, TT - Turnaround, WT - Wait)
```

# Priority Scheduling

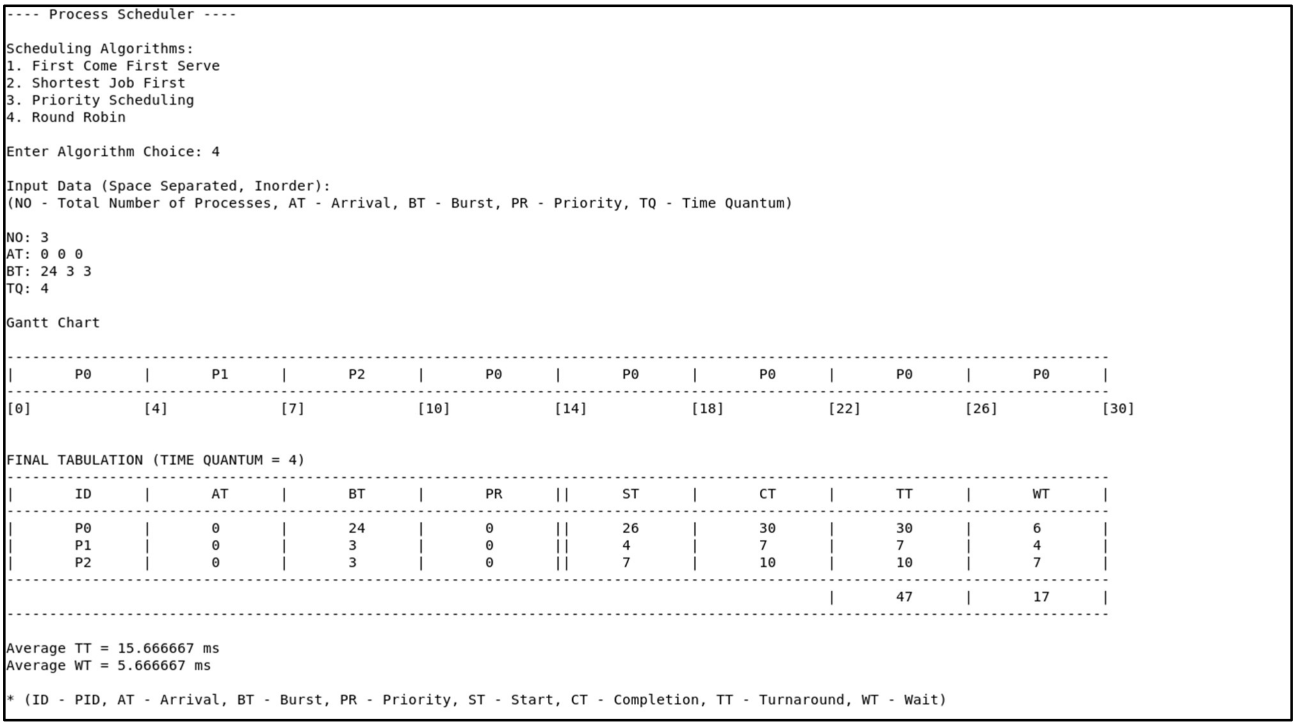
```
$ gcc -g -o out scheduler.c && ./out
```



```
Scheduling Algorithms:
1. First Come First Serve
2. Shortest Job First
3. Priority Scheduling
4. Round Robin

Enter Algorithm Choice: 3

Input Data (Space Separated, Inorder):
(NO - Total Number of Processes, AT - Arrival, BT - Burst, PR - Priority, TQ - Time Quantum)

NO: 5
AT: 0 2 2 1 3
BT: 8 6 1 9 3
PR: 4 1 2 2 3

Gantt Chart

---------------------------------------------------------------------------------
|       P0      |       P2      |       P4      |       P1      |       P3      |
---------------------------------------------------------------------------------
[0]             [8]             [9]             [12]            [18]            [27]


FINAL TABULATION (TIME QUANTUM = 0)
---------------------------------------------------------------------------------------------------------------------------------
|       ID      |       AT      |       BT      |       PR      ||      ST      |       CT      |       TT      |       WT      |
---------------------------------------------------------------------------------------------------------------------------------
|       P0      |       0       |       8       |       4       ||      0       |       8       |       8       |       0       |
|       P1      |       2       |       6       |       1       ||      12      |       18      |       16      |       10      |
|       P2      |       2       |       1       |       2       ||      8       |       9       |       7       |       6       |
|       P3      |       1       |       9       |       2       ||      18      |       27      |       26      |       17      |
|       P4      |       3       |       3       |       3       ||      9       |       12      |       9       |       6       |
---------------------------------------------------------------------------------------------------------------------------------
                                                                                                |       66      |       39      |
---------------------------------------------------------------------------------------------------------------------------------

Average TT = 13.200000 ms
Average WT = 7.800000 ms

* (ID - PID, AT - Arrival, BT - Burst, PR - Priority, ST - Start, CT - Completion, TT - Turnaround, WT - Wait)
* Assumption: low number represents high priority (only for priority schedule).
```

Round Robin

```
$ gcc -g -o out scheduler.c && ./out
```



```
---- Process Scheduler ----

Scheduling Algorithms:
1. First Come First Serve
2. Shortest Job First
3. Priority Scheduling
4. Round Robin

Enter Algorithm Choice: 4

Input Data (Space Separated, Inorder):
(NO - Total Number of Processes, AT - Arrival, BT - Burst, PR - Priority, TQ - Time Quantum)

NO: 3
AT: 0 0 0
BT: 24 3 3
TQ: 4

Gantt Chart

---------------------------------------------------------------------------------------------------------------------------------
|       P0      |       P1      |       P2      |       P0      |       P0      |       P0      |       P0      |       P0      |
---------------------------------------------------------------------------------------------------------------------------------
[0]             [4]             [7]             [10]            [14]            [18]            [22]            [26]            [30]


FINAL TABULATION (TIME QUANTUM = 4)
---------------------------------------------------------------------------------------------------------------------------------
|       ID      |       AT      |       BT      |       PR      ||      ST      |       CT      |       TT      |       WT      |
---------------------------------------------------------------------------------------------------------------------------------
|       P0      |       0       |       24      |       0       ||      26      |       30      |       30      |       6       |
|       P1      |       0       |       3       |       0       ||      4       |       7       |       7       |       4       |
|       P2      |       0       |       3       |       0       ||      7       |       10      |       10      |       7       |
---------------------------------------------------------------------------------------------------------------------------------
                                                                                                |       47      |       17      |
---------------------------------------------------------------------------------------------------------------------------------

Average TT = 15.666667 ms
Average WT = 5.666667 ms

* (ID - PID, AT - Arrival, BT - Burst, PR - Priority, ST - Start, CT - Completion, TT - Turnaround, WT - Wait)
```

Experiment 8

Banker’s Algorithm

1. Write a program to implement Bankers Algorithm for deadlock avoidance.

```
// Source code
#include <stdio.h>
void main() {
    int n, m;
    printf("Number of process: ");
    scanf("%d", &n);
    printf("Number of resources: ");
    scanf("%d", &m);
    printf("\n");
    int max[n][m], alloc[n][m], need[n][m], ava[m],
complete[n];
    int i, j, flag, valid;
    for (i = 0; i < n; i++) {
        printf("\nPID = %d", i);
        complete[i] = 0;
        printf("\n   Allocation: ");
        for (j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);
        printf("   Maximum Re: ");
        for (j = 0; j < m; j++) {
            scanf("%d", &max[i][j]);
            need[i][j] = max[i][j] - alloc[i][j];
        }
    }
    printf("\nAvailable: ");
    for (j = 0; j < m; j++)
        scanf("%d", &ava[j]);
    printf("\nSpecial Request (1/0): ");
    scanf("%d", &flag);
    if (flag) {
        int req[m];
        printf("PID: ");
        scanf("%d", &i);
        printf("Allocation: ");
        for (j = 0; j < m; j++) {
            scanf("%d", &req[j]);
            if (req[j] > need[i][j] || req[j] >
ava[j]){
                printf("\nInconsistent Request
Input\n");
                return;
            }
        }
        for (j = 0; j < m; j++) {
            alloc[i][j] += req[j];
            need[i][j] -= req[j];
            ava[j] -= req[j];
        }
    }
    printf("\n--- Input Values ---");
    printf("\nInitial Available: ");
    for (j = 0; j < m; j++)
        printf("%d ", ava[j]);
    printf("\n\nPID\tALLOC\tMAX\tNEED\t\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t", i);
        for (j = 0; j < m; j++)
            printf("%d ", alloc[i][j]);
        printf("\t");
        for (j = 0; j < m; j++)
            printf("%d ", max[i][j]);
        printf("\t");
        for (j = 0; j < m; j++)
            printf("%d ", need[i][j]);
        printf("\n");
    }
    printf("\nSafe Sequence: ");
    flag = 1;
    while (flag) {
        flag = 0;
        for (i = 0; i < n; i++) {
            valid = 1;
            if (complete[i] == 1)
                continue;
            for (j = 0; j < m; j++)
                if (need[i][j] > ava[j])
                    valid = 0;
            if (valid) {
                flag = 1;
                for (j = 0; j < m; j++)
                    ava[j] += alloc[i][j];
                printf("P%d  ", i);
                complete[i] = 1;
            }
        }
    }
    for (i = 0; i < n; i++) {
        if (complete[i] == 0) {
            printf("\nComplete safe sequence could
not be found.");
            printf("\nSystem in Unsafe State.");
            break;
        }
    }
    printf("\n\n");
}
```

```
# output
$ gcc -g -o out banker.c && ./out
Number of process: 5
Number of resources: 3



PID = 0
   Allocation: 0 1 0
   Maximum Re: 7 5 3

PID = 1
   Allocation: 2 0 0
   Maximum Re: 3 2 2

PID = 2
   Allocation: 3 0 2
   Maximum Re: 9 0 2

PID = 3
   Allocation: 2 1 1
   Maximum Re: 2 2 2

PID = 4
   Allocation: 0 0 2
   Maximum Re: 4 3 3

Available: 3 3 2

Special Request (1/0): 1
PID: 1
Allocation: 1 0 2

--- Input Values ---
Initial Available: 2 3 0

PID ALLOC   MAX NEED
P0  0 1 0   7 5 3   7 4 3
P1  3 0 2   3 2 2   0 2 0
P2  3 0 2   9 0 2   6 0 0
P3  2 1 1   2 2 2   0 1 1
P4  0 0 2   4 3 3   4 3 1

Safe Sequence: P1  P3  P4  P0  P2
```

Experiment 9

Page Replacement Algorithm

1. Write a program to implement page replacement algorithms:
   a. FIFO b. LRU c. LFU

```
// Source code
#include <stdio.h>
#include <stdlib.h>
int *pages, *frames;
int np, nf, ch;
int miss = 0, hit = 0;
void display(int type, int pg) {
   printf("Page = %d\t", pg);
   for(int i = 0; i < nf; i++) {
      if(frames[i] == -1)
          printf("x ");
      else
          printf("%d ", frames[i]);
   }
   if(type == 1)
      printf("\t[HIT]\n");
   else
      printf("\t[MISS]\n");
}
int isIn(int pg) {
   for(int i = 0; i < nf; i++)
      if(frames[i] == pg)
          return 1;
   return 0;
}
void insF(int pg) {
   int i;
   for(i = 0; i < nf; i++)
      if(frames[i] == -1) {
          frames[i] = pg;
          return;
      }
   for(i = 0; i < nf - 1; i++)
      frames[i] = frames[i + 1];
   frames[nf - 1] = pg;
}
void fcfs() {
   int i, j, pg;
   for(i = 0; i < np; i++) {
      pg = pages[i];
      if(isIn(pg) == 1) {
          hit++;
          display(1, pg);
      }
      else {
          miss++;
          insF(pg);
          display(0, pg);
      }
   }
}
void insL(int pg, int pos) {
   int i, j;
   int maxR = -1, maxP = -1, r;

   for(i = 0; i < nf; i++)
      if(frames[i] == -1) {
          frames[i] = pg;
          return;
      }
   for(i = 0; i < nf; i++) {
      r = 0;
      for(j = pos - 1; j > -1; j--) {
          r++;
          if(pages[j] == frames[i])
              break;
      }
      if(r > maxR) {
          maxR = r;
          maxP = i;
      }
   }
   frames[maxP] = pg;
}
void lru() {
   int i, j, pg;
   int minR, minP;
   for(i = 0; i < np; i++) {
      pg = pages[i];
      if(isIn(pg) == 1) {
          hit++;
          display(1, pg);
      }
      else {
          miss++;
          insL(pg, i);
          display(0, pg);
      }
   }
}
void insLFU(int pg, int pos) {
   int i, j;
   int minR = 100, minP = 100, r;
   for(i = 0; i < nf; i++)
      if(frames[i] == -1) {
          frames[i] = pg;
          return;
      }
   for(i = 0; i < nf; i++) {
      r = 0;
      for(j = 0; j < pos; j++) {
          if(pages[j] == frames[i])
              r++;
      }
      if(r < minR) {
          minR = r;
          minP = i;
      }
   }
   for(i = minP; i < nf - 1; i++)
      frames[i] = frames[i + 1];
   frames[nf - 1] = pg;
}
void lfu() {
   int i, j, pg;
   int minR, minP;
   for(i = 0; i < np; i++) {
      pg = pages[i];
      if(isIn(pg) == 1) {
          hit++;
          display(1, pg);
      }
      else {
          miss++;
          insLFU(pg, i);
          display(0, pg);
      }
   }
}

void main() {
   int i;
   printf("--- Page Replacement Algorithm ---\n");
   printf("1. FCFS\t2. LRU\t3. LFU\n");
   printf("Choose Algorithm: ");
   scanf("%d", &ch);
   printf("Number of Frames: ");
   scanf("%d", &nf);
   frames = malloc(nf * sizeof(int));
   for(i = 0; i < nf; i++)
      frames[i] = -1;
   printf("Number of Pages : ");
   scanf("%d", &np);
   pages = malloc(np * sizeof(int));
```

```
    printf("Page Requests: ");
    for(i = 0; i < np; i++)
        scanf("%d", &pages[i]);
    printf("\n");
    switch(ch) {
        case 1: fcfs(); break;
        case 2: lru(); break;
        case 3: lfu(); break;

    }
    printf("\nTotal Page Faults: %d\n", miss);
    printf("Total Page Hit(s): %d\n\n", hit);
}
```

```
# Output:

$ gcc -g -o out page.c && ./out
--- Page Replacement Algorithm ---
1. FCFS 2. LRU  3. LFU
Choose Algorithm: 1
Number of Frames: 4
Number of Pages : 6
Page Requests: 5 6 4 1 2 3

Page = 5        5 x x x         [MISS]
Page = 6        5 6 x x         [MISS]
Page = 4        5 6 4 x         [MISS]
Page = 1        5 6 4 1         [MISS]
Page = 2        6 4 1 2         [MISS]
Page = 3        4 1 2 3         [MISS]

Total Page Faults: 6
Total Page Hit(s): 0

$ gcc -g -o out page.c && ./out
--- Page Replacement Algorithm ---
1. FCFS 2. LRU  3. LFU
Choose Algorithm: 2
```

```
Number of Frames: 3
Number of Pages : 20
Page Requests: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page = 7        7 x x   [MISS]
Page = 0        7 0 x   [MISS]
Page = 1        7 0 1   [MISS]
Page = 2        2 0 1   [MISS]
Page = 0        2 0 1   [HIT]
Page = 3        2 0 3   [MISS]
Page = 0        2 0 3   [HIT]
Page = 4        4 0 3   [MISS]
Page = 2        4 0 2   [MISS]
Page = 3        4 3 2   [MISS]
Page = 0        0 3 2   [MISS]
Page = 3        0 3 2   [HIT]
Page = 2        0 3 2   [HIT]
Page = 1        1 3 2   [MISS]
Page = 2        1 3 2   [HIT]
Page = 0        1 0 2   [MISS]
Page = 1        1 0 2   [HIT]
Page = 7        1 0 7   [MISS]
Page = 0        1 0 7   [HIT]
Page = 1        1 0 7   [HIT]

Total Page Faults: 12
Total Page Hit(s): 8

$ gcc -g -o out page.c && ./out
--- Page Replacement Algorithm ---
1. FCFS 2. LRU  3. LFU
Choose Algorithm: 3
Number of Frames: 4
Number of Pages : 9
Page Requests: 5 0 1 3 2 4 1 0 5

Page = 5        5 x x x         [MISS]
Page = 0        5 0 x x         [MISS]
Page = 1        5 0 1 x         [MISS]
Page = 3        5 0 1 3         [MISS]
Page = 2        0 1 3 2         [MISS]
Page = 4        1 3 2 4         [MISS]
Page = 1        1 3 2 4         [HIT]
Page = 0        1 2 4 0         [MISS]
Page = 5        1 4 0 5         [MISS]

Total Page Faults: 8
Total Page Hit(s): 1
```

Experiment 10

Disk Scheduling Algorithm

1. Write a program to implement disk scheduling algorithms:
   a. FCFS   b. SCAN   c. C-SCAN

```
// Source code
#include <stdio.h>
#include <stdlib.h>
int *request;
int nr, st, max, seek = 0;
int mod(int x) {
   if(x < 0)
      x *= -1;
   return x;
}
void sort() {
   int i, j;
   for(i = 0; i < nr; i++) {
      for(j =0 ; j < nr - i - 1; j++) {
          if(request[j] > request[j + 1]) {
              int temp = request[j];
              request[j] = request[j + 1];
              request[j + 1] = temp;
          }
      }
   }
}
void scan() {
   int i, last = st;
   int mid = -1;
   sort();
   printf("\nScheduler: ");
   for(i = 0 ; i < nr; i++) {
      if(request[i] < st) {
          mid = i;
          continue;
      }
      printf("%d ", request[i]);
   }
   for(i = mid; i > -1; i--)
      printf("%d ", request[i]);
   if (mid != -1)
      seek = mod(st - max) + mod(max - request[0]);
   else
      seek = mod(st - request[nr - 1]);
}
void cscan() {
   int i, last = st;
   int mid = -1;
   sort();
   printf("\nScheduler: ");
   for(i = 0 ; i < nr; i++) {
      if(request[i] < st) {
          mid = i;
          continue;
      }
      printf("%d ", request[i]);
   }
   for(i = 0; i <= mid; i++)
      printf("%d ", request[i]);
   if (mid != -1)
      seek = mod(st - max) + max + request[mid];
   else
      seek = mod(st - request[nr - 1]);
}
void fcfs() {
   int i, last = st;;
   printf("\nScheduler: ");
   for(i = 0; i < nr; i++) {
      printf("%d ", request[i]);
      seek += mod(last - request[i]);
      last = request[i];
   }
}
```

```c
void main() {
    int ch, i;
    printf("--- Disk Scheduling ---");
    printf("\n1. FCFS\t2. Scan\t3. C-Scan\n");
    printf("Scheduling Choice: ");
    scanf("%d", &ch);
    printf("Number of Request: ");
    scanf("%d", &nr);
    request = malloc(nr * sizeof(int));
    printf("Enter all Request: ");
    for(i = 0; i < nr; i++)
        scanf("%d", &request[i]);
    if(ch != 1) {
        printf("Enter Range (start 0): ");
        scanf("%d", &max);
    }
    printf("Initial Cylinder : ");
    scanf("%d", &st);
    switch(ch) {
        case 1: fcfs(); break;
        case 2: scan(); break;
        case 3: cscan(); break;
    }
    printf("\nSeek Time: %d\n\n", seek);
    free(request);
}
```

```
# Output:

$ gcc -g -o out disk.c && ./out

--- Disk Scheduling ---
1. FCFS 2. Scan 3. C-Scan
Scheduling Choice: 1
Number of Request: 7
Enter all Request: 82 170 43 140 24 16 190
Initial Cylinder : 50

Scheduler: 82 170 43 140 24 16 190
```

```
Seek Time: 642

$ gcc -g -o out disk.c && ./out

--- Disk Scheduling ---
1. FCFS 2. Scan 3. C-Scan
Scheduling Choice: 2
Number of Request: 7
Enter all Request: 82 170 43 140 24 16 190
Enter Range (start 0): 199
Initial Cylinder : 50

Scheduler: 82 140 170 190 43 24 16
Seek Time: 332

$ gcc -g -o out disk.c && ./out

--- Disk Scheduling ---
1. FCFS 2. Scan 3. C-Scan
Scheduling Choice: 3
Number of Request: 7
Enter all Request: 82 170 43 140 24 16 190
Enter Range (start 0): 199
Initial Cylinder : 50

Scheduler: 82 140 170 190 16 24 43
Seek Time: 391
```

Experiment 11

Memory Allocation Algorithm

1. Write a program to implement disk scheduling algorithms for fixed size memory partition:
   a. First Fit b. Worst Fit c. Worst Fit

```
// Source code
#include <stdio.h>
#include <stdlib.h>
int *mem, *req;
int np, nr;
void display(int r, int flag, int pos) {
   int i;
   if(flag == 1)
      printf("Request %d Allocated @ [%d]\tCurrent
Partition: ", r, pos);
   else
      printf("Request %d Not - Allocated\tCurrent
Partition: ", r);
   for(i = 0; i < np; i++)
          printf("%d ", mem[i]);
   printf("\n");
}
void f() {
   int i, j, flag;
   for(i = 0; i < nr; i++) {
      flag = 0;
      for(j = 0; j < np; j++) {
          if(req[i] <= mem[j]) {
              mem[j] -= req[i];
              flag = 1;
              break;
          }
      }
      display(req[i], flag, j);
   }
```

```
}
void w() {
   int i, j, flag = 0;
   int max;
   for(i = 0; i < nr; i++) {
      max = 0;
      flag = 0;
      for(j = 0; j < np; j++) {
          if(mem[max] < mem[j])
              max = j;
      }
      if(req[i] <= mem[max]) {
          mem[max] -= req[i];
          flag = 1;
      }
      display(req[i], flag, max);
   }
}
void b() {
   int i, j, flag = 0;
   int max;
   for(i = 0; i < nr; i++) {
      max = 0;
      flag = 0;
      for(j = 0; j < np; j++) {
          if(mem[max] < mem[j])
              max = j;
      }
      for(j = 0; j < np; j++) {
          if(mem[max] > mem[j] && req[i] <= mem[j])
              max = j;
      }
      if(req[i] <= mem[max]) {
          mem[max] -= req[i];
          flag = 1;
      }
      display(req[i], flag, max);
   }
```

```
}
void main() {
   int ch, i;
   printf("--- Memory Allocation ---");
   printf("\n1. First Fit\t2. Worst Fit\t3. Best
Fit\n");
   printf("Choose Algorithm: ");
   scanf("%d", &ch);
   printf("\nNumber of Partitions: ");
   scanf("%d", &np);
   mem = malloc(np * sizeof(int));
   printf("Enter all Partitions: ");
   for(i = 0; i < np; i++)
       scanf("%d", &mem[i]);
   printf("\nNumber of Request(s): ");
   scanf("%d", &nr);
   req = malloc(nr * sizeof(int));
   printf("Enter all Request(s): ");
   for(i = 0; i < nr; i++)
       scanf("%d", &req[i]);
   printf("\n");
   switch(ch) {
       case 1: f(); break;
       case 2: w(); break;
       case 3: b(); break;
   }
   printf("\n");
   free(mem);
   free(req);
}
```

```
# Output:

$ gcc -g -o out mem.c && ./out

--- Memory Allocation ---
1. First Fit    2. Worst Fit    3. Best Fit
Choose Algorithm: 1

Number of Partitions: 5
```

```
Enter all Partitions: 100 500 200 300 600

Number of Request(s): 4
Enter all Request(s): 212 417 112 426

Request 212 Allocated @ [1]     Current Partition: 100 288 200 300 600
Request 417 Allocated @ [4]     Current Partition: 100 288 200 300 183
Request 112 Allocated @ [1]     Current Partition: 100 176 200 300 183
Request 426 Not - Allocated     Current Partition: 100 176 200 300 183

$ gcc -g -o out mem.c && ./out

1. First Fit    2. Worst Fit    3. Best Fit
Choose Algorithm: 2

Number of Partitions: 5
Enter all Partitions: 100 500 200 300 600

Number of Request(s): 4
Enter all Request(s): 212 417 112 426

Request 212 Allocated @ [4]     Current Partition: 100 500 200 300 388
Request 417 Allocated @ [1]     Current Partition: 100 83 200 300 388
Request 112 Allocated @ [4]     Current Partition: 100 83 200 300 276
Request 426 Not - Allocated     Current Partition: 100 83 200 300 276

$ gcc -g -o out mem.c && ./out

--- Memory Allocation ---
1. First Fit    2. Worst Fit    3. Best Fit
Choose Algorithm: 3

Number of Partitions: 5
Enter all Partitions: 100 500 200 300 600

Number of Request(s): 4
Enter all Request(s): 212 417 112 426

Request 212 Allocated @ [3]     Current Partition: 100 500 200 88 600
Request 417 Allocated @ [1]     Current Partition: 100 83 200 88 600
Request 112 Allocated @ [2]     Current Partition: 100 83 88 88 600
Request 426 Allocated @ [4]     Current Partition: 100 83 88 88 174
```