4. SHELL SCRIPT

4.1 Write a shell script to print Fibonacci series:

Answer: gedit Fibonacci.sh

```
echo "Enter number of terms:"
read n

a=0
b=1

echo "Fibonacci Series:"
for (( i=1; i<=n; i++ ))
do
echo -n "$a "
c=$((a + b))
a=$b
b=$c
done
echo
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit fibonacci.sh
suman@LAPTOP-R96B2LVV:~$ bash fibonacci.sh
Enter number of terms:
5
Fibonacci Series:
0 1 1 2 3
```

4.2 Write a shell script to display all prime numbers from 1 to 100

Answer: **gedit primeNo.sh**

```
echo "Prime numbers from 1 to 100
are:"

for (( num=2; num<=100; num++ ))
do
   flag=1
   for (( i=2; i<=num/2; i++ ))
   do
      if [ $((num % i)) -eq 0 ]
      then
         flag=0
         break
      fi
   done

   if [ $flag -eq 1 ]
   then
      echo -n "$num "
   fi
done
echo
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit primeNo.sh
suman@LAPTOP-R96B2LVV:~$ bash primeNo.sh
Prime numbers from 1 to 100 are:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
suman@LAPTOP-R96B2LVV:~$
```

4.3 Write a shell script to display all ARMSTRONG numbers from 1 to 10,000

Answer: **gedit primeNo.sh**

```
echo "Armstrong numbers from 1 to
10000 are:"

for (( num=1; num<=10000; num++ ))
do
   temp=$num
   sum=0
   digits=${#num}

   while [ $temp -gt 0 ]
   do
      rem=$((temp % 10))
      sum=$((sum + rem**digits))
      temp=$((temp / 10))
   done

   if [ $sum -eq $num ]
   then
      echo -n "$num "
   fi
done
echo
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit armstrong.sh
suman@LAPTOP-R96B2LVV:~$ bash armstrong.sh
Armstrong numbers from 1 to 10000 are:
1 2 3 4 5 6 7 8 9 153 370 371 407 1634 8208 9474
```

4.4 Write a shell script to sort n number of elements

Answer: gedit sortNo.sh

```
echo "Enter number of elements:"
read n

echo "Enter elements:"
for (( i=0; i<n; i++ ))
do
   read arr[$i]
done

for (( i=0; i<n; i++ ))
do
   for (( j=i+1; j<n; j++ ))
   do
      if [ ${arr[$i]} -gt ${arr[$j]} ]
      then
         temp=${arr[$i]}
         arr[$i]=${arr[$j]}
         arr[$j]=$temp
      fi
   done
done

echo "Sorted elements:"
for (( i=0; i<n; i++ ))
do
   echo -n "${arr[$i]} "
done
echo
```

OUTPUT:

5: FILE ORGANISATION SHELL SCRIPT

5.1 Shell script to check whether a string is palindrome or not

ANSWER: **gedit palindrome.sh**

```
echo "Enter a string:"
read str

rev=$(echo "$str" | rev)

if [ "$str" = "$rev" ]; then
    echo "The string is a palindrome"
else
    echo "The string is not a palindrome"
fi
```

OUTPUT:

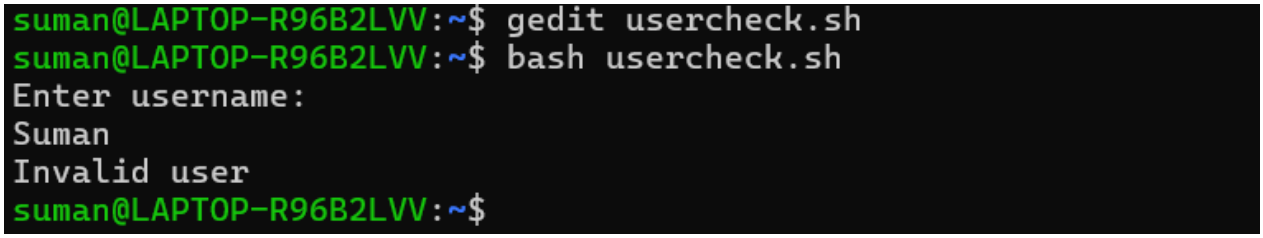
```
suman@LAPTOP-R96B2LVV:~$ gedit palindrome.sh
suman@LAPTOP-R96B2LVV:~$ bash palindrome.sh
Enter a string:
suman
The string is not a palindrome
suman@LAPTOP-R96B2LVV:~$ bash palindrome.sh
Enter a string:
madam
The string is a palindrome
suman@LAPTOP-R96B2LVV:~$
```

5.2 Shell script to check whether a username is valid or not

ANSWER: gedit usercheck.sh

```
echo "Enter username:"
read user

if id "$user" &>/dev/null; then
    echo "Valid user"
else
    echo "Invalid user"
fi
```

OUTPUT:

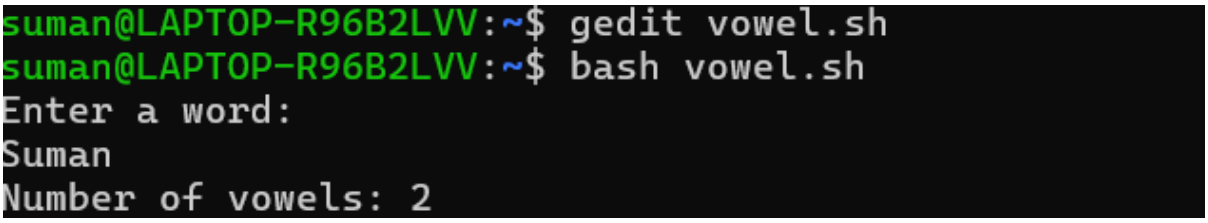


5.3 Shell script to count the number of vowels in a word

ANSWER: gedit vowel.sh

```
echo "Enter a word:"
read word

count=$(echo "$word" | grep -o -i
"[aeiou]" | wc -l)

echo "Number of vowels: $count"
```

OUTPUT:



5.4 Shell script to compare two files and delete second file if same

ANSWER: gedit filecompare.sh

```
if [ $# -ne 2 ]; then
    echo "Usage: $0 file1 file2"
    exit 1
fi

if cmp -s "$1" "$2"; then
    echo "Files are same. Deleting
second file."
    rm "$2"
else
    echo "Files are different."
fi
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit filecompare.sh
suman@LAPTOP-R96B2LVV:~$ bash filecompare.sh
Usage: filecompare.sh file1 file2
suman@LAPTOP-R96B2LVV:~$ chmod +x filecompare.sh
suman@LAPTOP-R96B2LVV:~$ ./filecompare.sh file1.txt file2.txt
Files are different.
```

5.5 Shell script to check if string has at least 10 characters

ANSWER: gedit filecompare.sh

```
echo "Enter a string:"
read str

len=${#str}

if [ $len -lt 10 ]; then
    echo "String does not have at least
10 characters"
else
    echo "String has 10 or more
characters"
fi
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit length.sh
suman@LAPTOP-R96B2LVV:~$ bash length.sh
Enter a string:
meghnadsahainstitute
String has 10 or more characters
```

ASSIGNMENT-6

6.1 Write a C program to display UID, PID and PPID of a current process.

ANSWER: **gedit uid_pid_ppid.c**

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

int main()
{
   printf("User ID (UID)  : %d\n",
getuid());
   printf("Process ID (PID): %d\n",
getpid());
   printf("Parent PID     : %d\n",
getppid());

   return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit uid_pid_ppid.c
suman@LAPTOP-R96B2LVV:~$ gcc uid_pid_ppid.c -o uid_pid_ppid
suman@LAPTOP-R96B2LVV:~$ ./uid_pid_ppid
User ID (UID)  : 1000
Process ID (PID): 6373
Parent PID     : 2150
```

6.2 write a C program to implement Zombie and orphan process.

A) Zombie Process Program:

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

int main()
{
    int pid = fork();

    if (pid > 0)
    {
        // Parent process
        sleep(10);   // Parent sleeps, child
exits
        printf("Parent process\n");
    }
    else if (pid == 0)
    {
        // Child process
        printf("Child process exiting\n");
    }

    return 0;
}
```

OUTPUT:

B) Orphan Process Program:

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

int main()
{
   int pid = fork();

   if (pid > 0)
   {
      // Parent process
      printf("Parent process exiting\n");
   }
   else if (pid == 0)
   {
      // Child process
      sleep(5);
      printf("Child process running after
parent exit\n");
      printf("New Parent PID: %d\n",
getppid());
   }

   return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit orphan.c
suman@LAPTOP-R96B2LVV:~$ gcc orphan.c -o orphan
suman@LAPTOP-R96B2LVV:~$ ./orphan
Parent process exiting
suman@LAPTOP-R96B2LVV:~$ Child process running after parent exit
New Parent PID: 2146
```

ASSIGNMENT:7

7. Write a C program to implement execl() system call.

ANSWER:

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

int main() {
    printf("Before execl() call\n");

    execl("/bin/ls", "ls", "-l", NULL);

    perror("execl failed"); // only runs if
execl fails
    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit syscall.c
suman@LAPTOP-R96B2LVV:~$ gcc syscall.c -o syscall
suman@LAPTOP-R96B2LVV:~$ ./syscall
Before execl() call
total 116
-rw-r--r-- 1 suman suman   338 Dec 18 12:03 armstrong.sh
-rw-r--r-- 1 suman suman   164 Dec 18 11:48 fibonacci.sh
-rwxr-xr-x 1 suman suman   198 Dec 18 12:39 filecompare.sh
-rw-r--r-- 1 suman suman   178 Dec 18 12:46 length.sh
-rwxr-xr-x 1 suman suman 16128 Dec 18 13:01 orphan
-rw-r--r-- 1 suman suman   383 Dec 18 13:01 orphan.c
-rw-r--r-- 1 suman suman   176 Dec 18 12:18 palindrome.sh
-rw-r--r-- 1 suman suman   306 Dec 18 11:57 primeNo.sh
-rw-r--r-- 1 suman suman   430 Dec 18 12:08 sortNo.sh
-rwxr-xr-x 1 suman suman 16048 Dec 18 13:09 syscall
-rw-r--r-- 1 suman suman   178 Dec 18 13:08 syscall.c
-rwxr-xr-x 1 suman suman 16096 Dec 18 12:54 uid_pid_ppid
-rw-r--r-- 1 suman suman   214 Dec 18 12:54 uid_pid_ppid.c
-rw-r--r-- 1 suman suman   120 Dec 18 12:26 usercheck.sh
-rw-r--r-- 1 suman suman   118 Dec 18 12:33 vowel.sh
-rwxr-xr-x 1 suman suman 16048 Dec 18 12:58 zombie
-rw-r--r-- 1 suman suman   339 Dec 18 12:58 zombie.c
```

ASSIGNMENT-8

Write C programs to simulate the following CPU Scheduling algorithms

(i) FCFS and SJF

(ii) (ii) Priority Scheduling and Round Robin Algorithm

(i) FCFS (First-Come, First-Served):

CODE:

```
#include <stdio.h>

int main() {
    int n, i;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    int bt[n], wt[n], tat[n];
    int total_wt = 0, total_tat = 0;

    printf("Enter burst times for each process:\n");
    for(i=0; i<n; i++){
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
    }

    // FCFS Waiting Time
    wt[0] = 0;
    for(i=1; i<n; i++){
        wt[i] = wt[i-1] + bt[i-1];
    }

    // Turnaround Time
    for(i=0; i<n; i++){
        tat[i] = bt[i] + wt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProcess\tBurst Time\tWaiting
Time\tTurnaround Time\n");
    for(i=0; i<n; i++){
        printf("P%d\t%d\t\t%d\t\t%d\n", i+1, bt[i], wt[i],
tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n",
(float)total_wt/n);
    printf("Average Turnaround Time: %.2f\n",
(float)total_tat/n);

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit fcfs.c
suman@LAPTOP-R96B2LVV:~$ gcc fcfs.c -o fcfs
suman@LAPTOP-R96B2LVV:~$ ./fcfs
Enter number of processes: 4
Enter burst times for each process:
P1: 5
P2: 3
P3: 8
P4: 6

Process Burst Time      Waiting Time    Turnaround Time
P1      5               0               5
P2      3               5               8
P3      8               8               16
P4      6               16              22

Average Waiting Time: 7.25
Average Turnaround Time: 12.75
```

(ii) SJF (Shortest Job First, Non-preemptive):

CODE:

```
#include <stdio.h>

int main() {
    int n, i, j;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int bt[n], wt[n], tat[n], p[n], temp;
    int total_wt=0, total_tat=0;

    printf("Enter burst times for each process:\n");
    for(i=0; i<n; i++){
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
        p[i] = i+1;
    }

    // Sort processes by burst time
    for(i=0; i<n-1; i++){
        for(j=i+1; j<n; j++){
            if(bt[i] > bt[j]){
                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                temp = p[i]; p[i] = p[j]; p[j] = temp;
            }
        }
    }

    // Waiting Time
    wt[0] = 0;
    for(i=1; i<n; i++){
        wt[i] = wt[i-1] + bt[i-1];
    }

    // Turnaround Time
    for(i=0; i<n; i++){
        tat[i] = bt[i] + wt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround
Time\n");
    for(i=0; i<n; i++){
        printf("P%d\t%d\t\t%d\t\t%d\n", p[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_wt/n);
    printf("Average Turnaround Time: %.2f\n", (float)total_tat/n);

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit sjf.c
suman@LAPTOP-R96B2LVV:~$ gcc sjf.c -o sjf
suman@LAPTOP-R96B2LVV:~$ ./sjf
Enter number of processes: 4
Enter burst times for each process:
P1: 5
P2: 3
P3: 8
P4: 6

Process Burst Time      Waiting Time    Turnaround Time
P2      3               0               3
P1      5               3               8
P4      6               8               14
P3      8               14              22

Average Waiting Time: 6.25
Average Turnaround Time: 11.75
```

(iii) Priority Scheduling (Non-preemptive):

CODE:

```
#include <stdio.h>

int main() {
    int n, i, j;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int bt[n], priority[n], wt[n], tat[n], p[n], temp;
    int total_wt=0, total_tat=0;

    printf("Enter burst time and priority for each process (lower number = higher priority):\n");
    for(i=0; i<n; i++){
        printf("P%d - Burst Time: ", i+1);
        scanf("%d", &bt[i]);
        printf("Priority: ");
        scanf("%d", &priority[i]);
        p[i] = i+1;
    }

    // Sort by priority
    for(i=0; i<n-1; i++){
        for(j=i+1; j<n; j++){
            if(priority[i] > priority[j]){
                temp = priority[i]; priority[i] = priority[j]; priority[j] = temp;
                temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;
                temp = p[i]; p[i] = p[j]; p[j] = temp;
            }
        }
    }

    // Waiting Time
    wt[0] = 0;
    for(i=1; i<n; i++){
        wt[i] = wt[i-1] + bt[i-1];
    }

    // Turnaround Time
    for(i=0; i<n; i++){
        tat[i] = bt[i] + wt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for(i=0; i<n; i++){
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i], bt[i], priority[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_wt/n);
    printf("Average Turnaround Time: %.2f\n", (float)total_tat/n);

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit priority_scheduling.c
suman@LAPTOP-R96B2LVV:~$ gcc priority_scheduling.c -o priority_scheduling
suman@LAPTOP-R96B2LVV:~$ ./priority_scheduling
Enter number of processes: 4
Enter burst time and priority for each process (lower number = higher priority):
P1 - Burst Time: 5
Priority: 2
P2 - Burst Time: 3
Priority: 1
P3 - Burst Time: 8
Priority: 4
P4 - Burst Time: 6
Priority: 3

Process Burst Time      Priority        Waiting Time    Turnaround Time
P2      3               1               0               3
P1      5               2               3               8
P4      6               3               8               14
P3      8               4               14              22

Average Waiting Time: 6.25
Average Turnaround Time: 11.75
```

(iv) Round Robin Scheduling:

CODE:

```
#include <stdio.h>

int main() {
    int n, i, j, tq;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int bt[n], wt[n], tat[n], rem_bt[n];
    int total_wt=0, total_tat=0, t=0;

    printf("Enter burst times for each process:\n");
    for(i=0; i<n; i++){
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
        rem_bt[i] = bt[i];
    }

    printf("Enter time quantum: ");
    scanf("%d", &tq);

    int done;
    do {
        done = 1;
        for(i=0; i<n; i++){
            if(rem_bt[i] > 0){
                done = 0;
                if(rem_bt[i] > tq){
                    t += tq;
                    rem_bt[i] -= tq;
                } else {
                    t += rem_bt[i];
                    wt[i] = t - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
    } while(!done);

    for(i=0; i<n; i++){
        tat[i] = bt[i] + wt[i];
        total_wt += wt[i];
        total_tat += tat[i];
    }

    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for(i=0; i<n; i++){
        printf("P%d\t%d\t\t%d\t\t%d\n", i+1, bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time: %.2f\n", (float)total_wt/n);
    printf("Average Turnaround Time: %.2f\n", (float)total_tat/n);

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit RR.c
suman@LAPTOP-R96B2LVV:~$ gcc RR.c -o RR
suman@LAPTOP-R96B2LVV:~$ ./RR
Enter number of processes: 4
Enter burst times for each process:
P1: 5
P2: 4
P3: 8
P4: 6
Enter time quantum: 2

Process Burst Time      Waiting Time    Turnaround Time
P1      5               12              17
P2      4               8               12
P3      8               15              23
P4      6               15              21

Average Waiting Time: 12.50
Average Turnaround Time: 18.25
```

ASSIGNMENT-9(SIGNAL)

9.1 Write a system call to implement SIGSTOP, SIGCONT and SIGKILL using Parent and child process.

CODE:

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

int main() {
    pid_t pid = fork();

    if(pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if(pid == 0) { // Child process
        while(1) {
            printf("Child process running (PID: %d)\n", getpid());
            sleep(1);
        }
    } else { // Parent process
        sleep(2);
        printf("Parent: Sending SIGSTOP to child\n");
        kill(pid, SIGSTOP);  // Stop child
        sleep(3);

        printf("Parent: Sending SIGCONT to child\n");
        kill(pid, SIGCONT);  // Continue child
        sleep(3);

        printf("Parent: Sending SIGKILL to child\n");
        kill(pid, SIGKILL);  // Kill child

        wait(NULL); // Wait for child to terminate
        printf("Child process terminated\n");
    }

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit signal1.c
suman@LAPTOP-R96B2LVV:~$ gcc signal1.c -o signal1
suman@LAPTOP-R96B2LVV:~$ ./signal1
Child process running (PID: 6844)
Child process running (PID: 6844)
Parent: Sending SIGSTOP to child
Parent: Sending SIGCONT to child
Child process running (PID: 6844)
Child process running (PID: 6844)
Child process running (PID: 6844)
Parent: Sending SIGKILL to child
Child process terminated
```

9.2 Write a system call to display “welcome message” in the CHILD process every 50 milliseconds and kill the child process after 1 second by PARENT.

CODE:

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

int main() {
    pid_t pid = fork();

    if(pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if(pid == 0) { // Child process
        while(1) {
            printf("Welcome message from child (PID: %d)\n", getpid());
            usleep(50000); // 50 milliseconds
        }
    } else { // Parent process
        sleep(1); // Let child print for 1 second
        printf("Parent: Killing child process\n");
        kill(pid, SIGKILL); // Kill child
        wait(NULL); // Wait for child to terminate
        printf("Child process terminated\n");
    }

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit signal2.c
suman@LAPTOP-R96B2LVV:~$ gcc signal2.c -o signal2
suman@LAPTOP-R96B2LVV:~$ ./signal2
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Welcome message from child (PID: 6864)
Parent: Killing child process
Child process terminated
```

ASSIGNMENT- 10(SEMAPHORE)

10.1Write a C program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
sem_t empty; // counts empty slots
sem_t full;  // counts filled slots
sem_t mutex; // mutual exclusion

void* producer(void* arg) {
  int item, i;
  for(i=0; i<10; i++) {
    item = rand() % 100;
    sem_wait(&empty);
    sem_wait(&mutex);

    buffer[in] = item;
    printf("Produced: %d at buffer[%d]\n", item, in);
    in = (in + 1) % BUFFER_SIZE;

    sem_post(&mutex);
    sem_post(&full);

    sleep(1); // simulate production time
  }
  pthread_exit(NULL);
}
void* consumer(void* arg) {
  int item, i;
  for(i=0; i<10; i++) {
    sem_wait(&full);
    sem_wait(&mutex);

    item = buffer[out];
    printf("Consumed: %d from buffer[%d]\n", item, out);
    out = (out + 1) % BUFFER_SIZE;

    sem_post(&mutex);
    sem_post(&empty);

    sleep(2); // simulate consumption time
  }
  pthread_exit(NULL);
```

```
}
int main() {
    pthread_t prod, cons;

    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit Producer_Consumer.c
suman@LAPTOP-R96B2LVV:~$ gcc Producer_Consumer.c -o Producer_Consumer
suman@LAPTOP-R96B2LVV:~$ ./Producer_Consumer
Produced: 83 at buffer[0]
Consumed: 83 from buffer[0]
Produced: 86 at buffer[1]
Consumed: 86 from buffer[1]
Produced: 77 at buffer[2]
Produced: 15 at buffer[3]
Consumed: 77 from buffer[2]
Produced: 93 at buffer[4]
Produced: 35 at buffer[0]
Consumed: 15 from buffer[3]
Produced: 86 at buffer[1]
Produced: 92 at buffer[2]
Consumed: 93 from buffer[4]
Produced: 49 at buffer[3]
Produced: 21 at buffer[4]
Consumed: 35 from buffer[0]
Consumed: 86 from buffer[1]
Consumed: 92 from buffer[2]
Consumed: 49 from buffer[3]
Consumed: 21 from buffer[4]
```

10.2 Write a program to Implementing Semaphores : Critical Section of n process problems.

CODE:

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

#define N 5 // number of processes

sem_t mutex;

void* process(void* arg) {
    int id = *(int*)arg;
    printf("Process %d wants to enter critical section\n", id);

    sem_wait(&mutex); // enter critical section
    printf("Process %d entering critical section\n", id);
    sleep(2); // simulate critical section
    printf("Process %d leaving critical section\n", id);
    sem_post(&mutex); // leave critical section

    pthread_exit(NULL);
}

int main() {
    pthread_t processes[N];
    int ids[N];
    sem_init(&mutex, 0, 1); // binary semaphore for mutual exclusion

    for(int i=0; i<N; i++) {
        ids[i] = i + 1;
        pthread_create(&processes[i], NULL, process, &ids[i]);
    }

    for(int i=0; i<N; i++)
        pthread_join(processes[i], NULL);

    sem_destroy(&mutex);
    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit critical_section.c
suman@LAPTOP-R96B2LVV:~$ gcc critical_section.c -o critical_section
suman@LAPTOP-R96B2LVV:~$ ./critical_section
Process 1 wants to enter critical section
Process 1 entering critical section
Process 2 wants to enter critical section
Process 3 wants to enter critical section
Process 4 wants to enter critical section
Process 5 wants to enter critical section
Process 1 leaving critical section
Process 2 entering critical section
Process 2 leaving critical section
Process 3 entering critical section
Process 3 leaving critical section
Process 4 entering critical section
Process 4 leaving critical section
Process 5 entering critical section
Process 5 leaving critical section
```

ASSIGNMENT: 11(POSIX Threads)

11.1 Write a program to create a thread and display the sequence numbers from 1 to 5. (viz. pthread_create, pthread_join).

CODE:

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

void* print_numbers(void* arg) {
    for(int i = 1; i <= 5; i++) {
        printf("Thread: %d\n", i);
        sleep(1);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t tid;

    pthread_create(&tid, NULL, print_numbers, NULL);
    pthread_join(tid, NULL);

    printf("Main thread completed\n");
    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit 11.1.c
suman@LAPTOP-R96B2LVV:~$ gcc 11.1.c -o 11.1
suman@LAPTOP-R96B2LVV:~$ ./11.1
Thread: 1
Thread: 2
Thread: 3
Thread: 4
Thread: 5
Main thread completed
```

11.2 Write a program to implement mutex lock for UNIX / LINUX Thread Synchronization

CODE:

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

pthread_mutex_t mutex;

void* critical_section(void* arg) {
    int id = *(int*)arg;

    pthread_mutex_lock(&mutex);
    printf("Thread %d entered critical section\n", id);
    sleep(2);  // simulate critical work
    printf("Thread %d leaving critical section\n", id);
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}

int main() {
    pthread_t t1, t2;
    int id1 = 1, id2 = 2;

    pthread_mutex_init(&mutex, NULL);

    pthread_create(&t1, NULL, critical_section, &id1);
    pthread_create(&t2, NULL, critical_section, &id2);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    pthread_mutex_destroy(&mutex);
    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit 11.2.c
suman@LAPTOP-R96B2LVV:~$ gcc 11.2.c -o 11.2
suman@LAPTOP-R96B2LVV:~$ ./11.2
Thread 1 entered critical section
Thread 1 leaving critical section
Thread 2 entered critical section
Thread 2 leaving critical section
```

ASSIGNMENT- 12(IPC)

12.1 Write C programs to illustrate the following IPC mechanisms: (i) Pipes (ii) FIFOs (iii) Message Queues (iv) Shared Memory

(i) Pipes (Unnamed Pipe):

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

int main() {
    int fd[2];
    char msg[] = "Hello from Parent";
    char buffer[50];

    pipe(fd);          // create pipe
    pid_t pid = fork();

    if(pid == 0) {     // Child process
        close(fd[1]);  // close write end
        read(fd[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(fd[0]);
    } else {           // Parent process
        close(fd[0]);  // close read end
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    }

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit pipes.c
suman@LAPTOP-R96B2LVV:~$ gcc pipes.c -o pipes
suman@LAPTOP-R96B2LVV:~$ ./pipes
Child received: Hello from Parent
```

(ii) **FIFOs (Named Pipes)**

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

int main() {
    char *fifo = "myfifo";
    char msg[] = "Hello using FIFO";
    char buffer[50];

    mkfifo(fifo, 0666);   // create FIFO

    if(fork() == 0) {     // Child: read
        int fd = open(fifo, O_RDONLY);
        read(fd, buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(fd);
    } else {              // Parent: write
        int fd = open(fifo, O_WRONLY);
        write(fd, msg, strlen(msg) + 1);
        close(fd);
    }

    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit fifo.c
suman@LAPTOP-R96B2LVV:~$ gcc fifo.c -o fifo
suman@LAPTOP-R96B2LVV:~$ ./fifo
Child received: Hello using FIFO
```

(iii) **Message Queues:**

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>

struct msg {
    long type;
    char text[50];
};

int main() {
    key_t key = ftok("progfile", 65);
    int msgid = msgget(key, 0666 | IPC_CREAT);

    struct msg message;
    message.type = 1;
    strcpy(message.text, "Hello Message Queue");

    msgsnd(msgid, &message, sizeof(message.text), 0);
    printf("Message sent\n");

    msgrcv(msgid, &message, sizeof(message.text), 1, 0);
    printf("Message received: %s\n", message.text);

    msgctl(msgid, IPC_RMID, NULL);
    return 0;
}
```

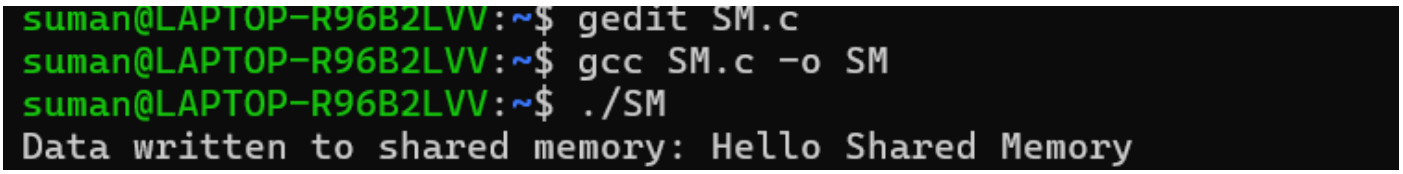
OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit MQ.c
suman@LAPTOP-R96B2LVV:~$ gcc MQ.c -o MQ
suman@LAPTOP-R96B2LVV:~$ ./MQ
Message sent
Message received: Hello Message Queue
```

(iv) **Shared Memory:**

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>

int main() {
    key_t key = ftok("shmfile", 65);
    int shmid = shmget(key, 1024, 0666 | IPC_CREAT);

    char *str = (char*) shmat(shmid, NULL, 0);
    strcpy(str, "Hello Shared Memory");

    printf("Data written to shared memory: %s\n", str);

    shmdt(str);
    shmctl(shmid, IPC_RMID, NULL);
    return 0;
}
```

OUTPUT:



12.2 Write a system call to create a pipe for one-way communication i.e., it creates two descriptors, first one is connected to read from the pipe and other one is connected to write into the pipe.

CODE:

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

int main() {
    int fd[2];
    char write_msg[] = "One way communication using pipe";
    char read_msg[100];

    // Create pipe
    if (pipe(fd) == -1) {
        perror("Pipe failed");
        return 1;
    }

    if (fork() == 0) {
        // Child process (Reader)
        close(fd[1]);              // Close write end
        read(fd[0], read_msg, sizeof(read_msg));
        printf("Child received: %s\n", read_msg);
        close(fd[0]);
    } else {
        // Parent process (Writer)
        close(fd[0]);              // Close read end
        write(fd[1], write_msg, strlen(write_msg) + 1);
        close(fd[1]);
    }

    return 0;
}
```

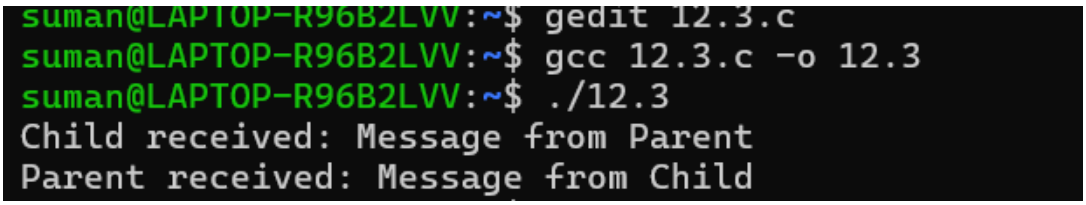
OUTPUT:

12.3 Write a Program to write and read two messages through the pipe using the parent and the child processes.

CODE:

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

int main() {
    int p2c[2], c2p[2];
    char msg1[] = "Message from Parent";
    char msg2[] = "Message from Child";
    char buffer[100];

    // Create two pipes
    pipe(p2c); // Parent to Child
    pipe(c2p); // Child to Parent

    if (fork() == 0) {
        // Child process
        close(p2c[1]);
        close(c2p[0]);

        read(p2c[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);

        write(c2p[1], msg2, strlen(msg2) + 1);

        close(p2c[0]);
        close(c2p[1]);
    } else {
        close(p2c[0]); // close read end of parent-to-child
        close(c2p[1]); // close write end of child-to-parent

        write(p2c[1], msg1, strlen(msg1) + 1);

        read(c2p[0], buffer, sizeof(buffer));
        printf("Parent received: %s\n", buffer);

        close(p2c[1]);
        close(c2p[0]);
    }

    return 0;
}
```

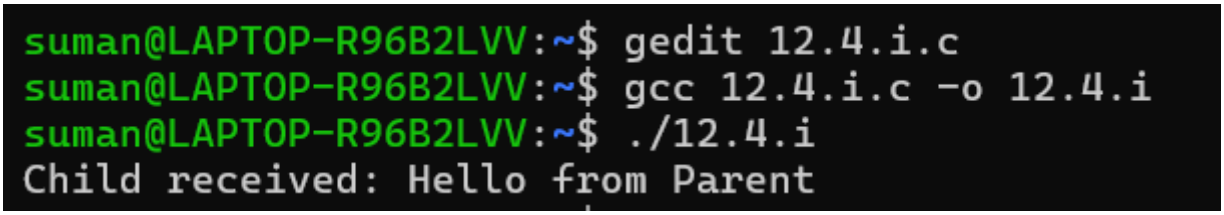
OUTPUT:



12.4 (i) TWO-WAY COMMUNICATION USING PIPES

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

int main() {
    int fd[2];
    char msg[50];

    pipe(fd);

    if (fork() == 0) {
        // Child
        close(fd[1]);
        read(fd[0], msg, sizeof(msg));
        printf("Child received: %s\n", msg);
        close(fd[0]);
    } else {
        // Parent
        close(fd[0]);
        strcpy(msg, "Hello from Parent");
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    }
    return 0;
}
```

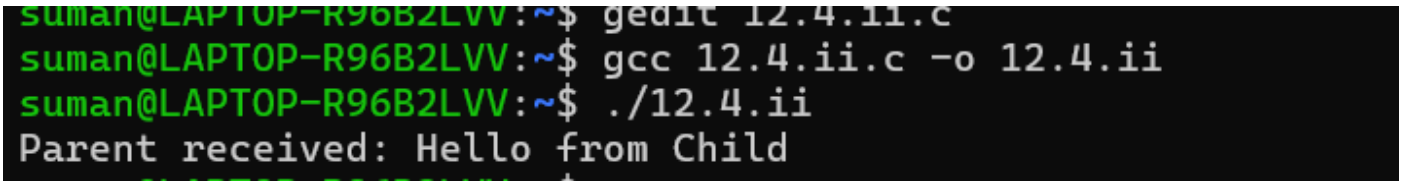
OUTPUT:



12.4 (ii) TWO-WAY COMMUNICATION:

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

int main() {
    int fd[2];
    char msg[50];

    pipe(fd);

    if (fork() == 0) {
        // Child
        close(fd[0]);
        strcpy(msg, "Hello from Child");
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    } else {
        // Parent
        close(fd[1]);
        read(fd[0], msg, sizeof(msg));
        printf("Parent received: %s\n", msg);
        close(fd[0]);
    }
    return 0;
}
```

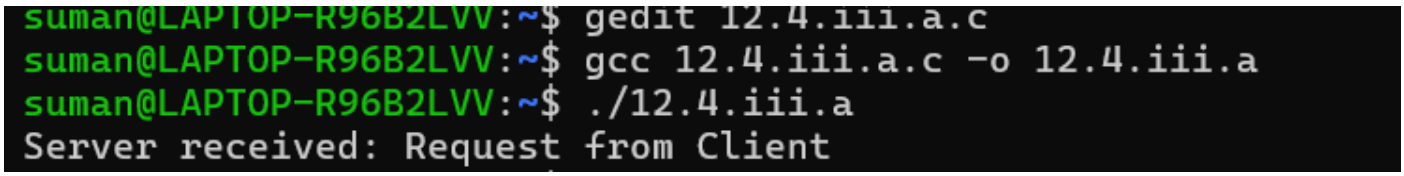
OUTPUT:



12.4 (iii) CLIENT–SERVER PROGRAM
(a) Using ONE PIPE:

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

int main() {
    int fd[2];
    char msg[50];

    pipe(fd);

    if (fork() == 0) {
        // Server
        close(fd[1]);
        read(fd[0], msg, sizeof(msg));
        printf("Server received: %s\n", msg);
        close(fd[0]);
    } else {
        // Client
        close(fd[0]);
        strcpy(msg, "Request from Client");
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    }
    return 0;
}
```

OUTPUT:



(b) CLIENT–SERVER USING TWO PIPES (Full Duplex):

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

int main() {
    int p1[2], p2[2];
    char msg[50];

    pipe(p1); // Client → Server
    pipe(p2); // Server → Client

    if (fork() == 0) {
        // Server
        close(p1[1]);
        read(p1[0], msg, sizeof(msg));
        printf("Server received: %s\n", msg);

        close(p2[0]);
        strcpy(msg, "Reply from Server");
        write(p2[1], msg, strlen(msg) + 1);
    } else {
        // Client
        close(p1[0]);
        strcpy(msg, "Request from Client");
        write(p1[1], msg, strlen(msg) + 1);

        close(p2[1]);
        read(p2[0], msg, sizeof(msg));
        printf("Client received: %s\n", msg);
    }
    return 0;
}
```

OUTPUT:

```
suman@LAPTOP-R96B2LVV:~$ gedit 12.4.iii.b.c
suman@LAPTOP-R96B2LVV:~$ gcc 12.4.iii.b.c -o 12.4.iii.b
suman@LAPTOP-R96B2LVV:~$ ./12.4.iii.b
Server received: Request from Client
Client received: Reply from Server
```