



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering
Assessment - I, JANUARY 2020
B.Tech, Winter-2019-2020

NAME	PRIYAL BHARDWAJ
REG. NO.	18BIT0272
COURSE CODE	ITE2002
COURSE NAME	OPERATING SYSTEMS
SLOT	L-37+L-38
FACULTY	Prof. SUDHA S.

a) Shell Programming

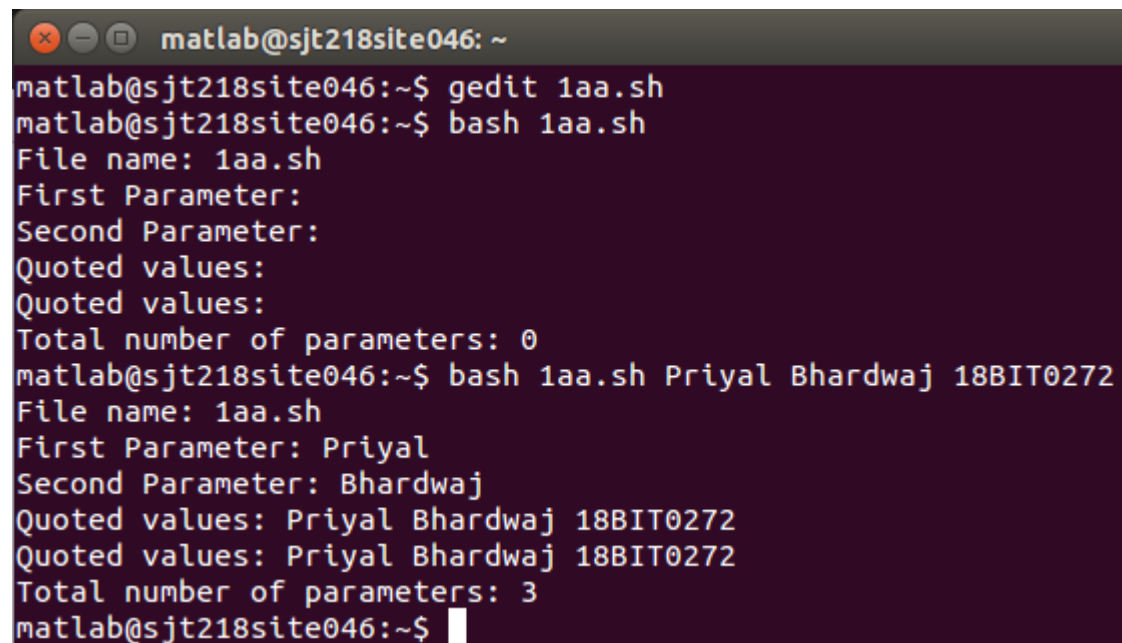
- Handling the command line arguments
- String reversal
- If-Else, Nested If Else, Switch cases in shell

CODE/OUTPUT:

Handling the command line arguments

```
#!/bin/sh
```

```
echo "File name: $0"
echo "First Parameter: $1"
echo "Second Parameter: $2"
echo "Quoted values: $@"
echo "Quoted values: $*"
echo "Total number of parameters: $#"
```

A terminal window titled 'matlab@sjt218site046: ~' showing the execution of a shell script named '1aa.sh'. The script is first run without arguments, then with three arguments: 'Priyal', 'Bhardwaj', and '18BIT0272'. The output shows the script correctly identifying the file name, individual parameters, quoted values, and the total number of parameters in each case.

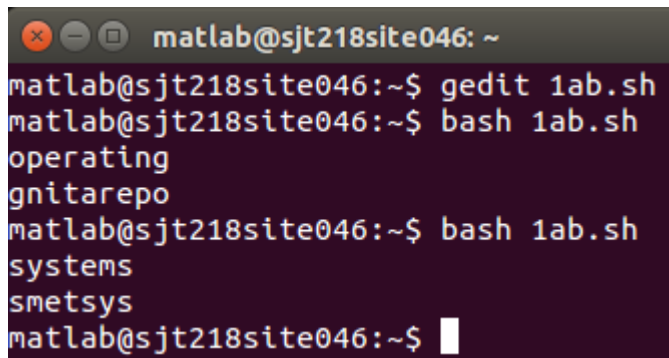
```
matlab@sjt218site046: ~
matlab@sjt218site046:~$ gedit 1aa.sh
matlab@sjt218site046:~$ bash 1aa.sh
File name: 1aa.sh
First Parameter:
Second Parameter:
Quoted values:
Quoted values:
Total number of parameters: 0
matlab@sjt218site046:~$ bash 1aa.sh Priyal Bhardwaj 18BIT0272
File name: 1aa.sh
First Parameter: Priyal
Second Parameter: Bhardwaj
Quoted values: Priyal Bhardwaj 18BIT0272
Quoted values: Priyal Bhardwaj 18BIT0272
Total number of parameters: 3
matlab@sjt218site046:~$
```

String reversal

```
#!/bin/bash
```

```
read str
len=`echo $str | wc -c`
len=`expr $len - 1`
rev=""
while test $len -gt 0
do
rev1=`echo $str | cut -c$len`
```

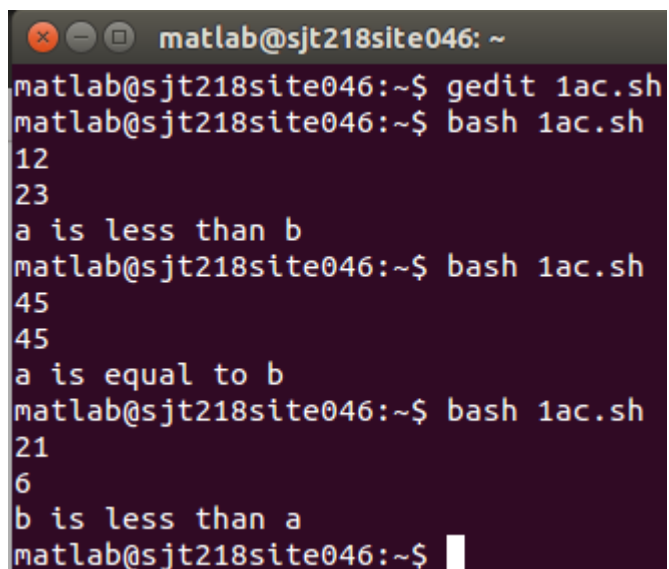
```
rev=$rev$rev1
len=`expr $len - 1`
done
echo $rev
```

A terminal window titled 'matlab@sjt218site046: ~' showing the execution of a script named '1ab.sh'. The script is first edited with 'gedit 1ab.sh' and then run with 'bash 1ab.sh'. The output of the script is 'operating', 'gnitarepo', 'systems', and 'smetsys'.

```
matlab@sjt218site046: ~$ gedit 1ab.sh
matlab@sjt218site046: ~$ bash 1ab.sh
operating
gnitarepo
matlab@sjt218site046: ~$ bash 1ab.sh
systems
smetsys
matlab@sjt218site046: ~$
```

If-else

```
#!/bin/sh
read a
read b
if [ $a == $b ]
then
    echo "a is equal to b"
elif [ $a -lt $b ]
then
    echo "a is less than b"
elif [ $b -lt $a ]
then
    echo "b is less than a"
else
    echo "No condition satisfied"
fi
```

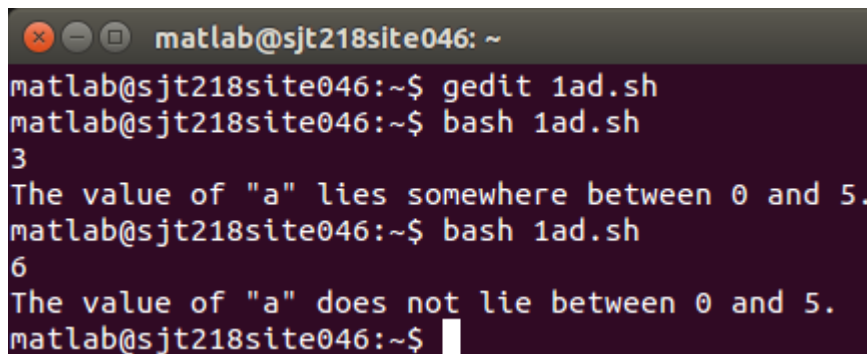
A terminal window titled 'matlab@sjt218site046: ~' showing the execution of a script named '1ac.sh'. The script is first edited with 'gedit 1ac.sh' and then run with 'bash 1ac.sh'. The script prompts for two numbers, 12 and 23, and outputs 'a is less than b'. It is then run again with inputs 45 and 45, outputting 'a is equal to b'. Finally, it is run with inputs 21 and 6, outputting 'b is less than a'.

```
matlab@sjt218site046: ~$ gedit 1ac.sh
matlab@sjt218site046: ~$ bash 1ac.sh
12
23
a is less than b
matlab@sjt218site046: ~$ bash 1ac.sh
45
45
a is equal to b
matlab@sjt218site046: ~$ bash 1ac.sh
21
6
b is less than a
matlab@sjt218site046: ~$
```

Nested

```
#!/bin/sh
read a

if [ "$a" -gt 0 ]
then
    if [ "$a" -lt 5 ]
    then
        echo "The value of \"a\" lies somewhere between 0 and 5."
    else
        echo "The value of \"a\" does not lie between 0 and 5."
    fi
fi
```

A terminal window titled 'matlab@sjt218site046: ~' with a dark background. It shows the execution of a script named '1ad.sh'. The first run takes input '3' and outputs 'The value of "a" lies somewhere between 0 and 5.'. The second run takes input '6' and outputs 'The value of "a" does not lie between 0 and 5.'. The prompt 'matlab@sjt218site046:~\$' is visible at the end of each line.

```
matlab@sjt218site046:~$ gedit 1ad.sh
matlab@sjt218site046:~$ bash 1ad.sh
3
The value of "a" lies somewhere between 0 and 5.
matlab@sjt218site046:~$ bash 1ad.sh
6
The value of "a" does not lie between 0 and 5.
matlab@sjt218site046:~$
```

Switch

```
#!/bin/sh
echo "Enter a number"
read num
case $num in
    [0-9])
        echo "you have entered a single digit number"
        ;;
    [1-9][1-9])
        echo "you have entered a two-digit number"
        ;;
    [1-9][1-9][1-9])
        echo "you have entered a three-digit number"
        ;;
    *)
        echo "your entry does not match any of the conditions"
        ;;
esac
```

```
matlab@sjt218site046: ~
matlab@sjt218site046:~$ gedit 1ae.sh
matlab@sjt218site046:~$ bash 1ae.sh
"Enter a number"
4
"you have entered a single digit number"
matlab@sjt218site046:~$ bash 1ae.sh
"Enter a number"
23
"you have entered a two-digit number"
matlab@sjt218site046:~$ bash 1ae.sh
"Enter a number"
272
"you have entered a three-digit number"
matlab@sjt218site046:~$
```

b) Parent child process creation using fork() and exec() system call

Checking the Process Identifier

Assigning new task to child

Providing the path name and program name to exec()

Synchronizing Parent and child process using wait()

CODE:

1)

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

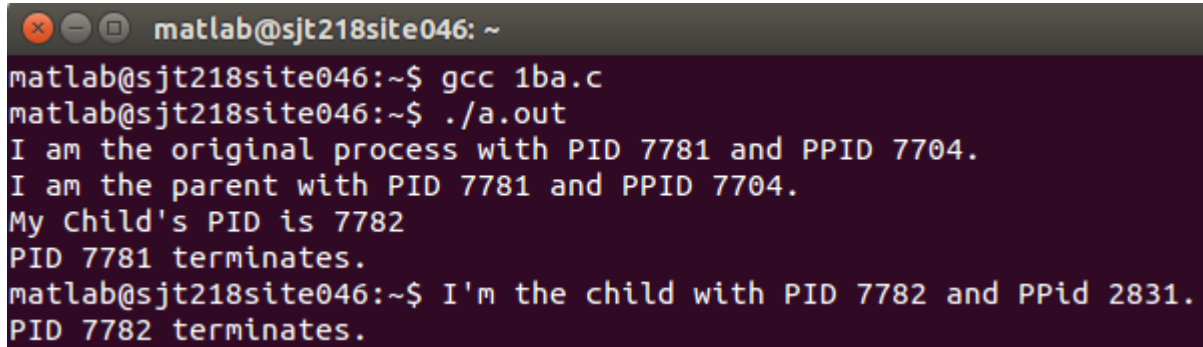
int main(int argc, char const *argv[])
{
    int pid;

    printf("I am the original process with PID %d and PPID %d.\n",getpid(),getppid());
    pid = fork();
    if(pid !=0 )
    {
        printf("I am the parent with PID %d and PPID %d.\n",getpid(),getppid());
        printf("My Child's PID is %d\n",pid);
    }
    else
    {
        sleep(4);
    }
}
```

```

        printf("I'm the child with PID %d and PPid
%d.\n",getpid(),getppid());
    }
    printf("PID %d terminates.\n",getpid());
}

```



```

matlab@sjt218site046:~$ gcc 1ba.c
matlab@sjt218site046:~$ ./a.out
I am the original process with PID 7781 and PPID 7704.
I am the parent with PID 7781 and PPID 7704.
My Child's PID is 7782
PID 7781 terminates.
matlab@sjt218site046:~$ I'm the child with PID 7782 and PPid 2831.
PID 7782 terminates.

```

2)

```

#include<stdio.h>
#include <unistd.h>
#include<string.h>
int main(int argc, char const *argv[])
{
    int a = 12;
    int b = 23;
    int n1 = fork();
    int n2 = fork();

    if(n1>0 && n2>0) //PARENT process IS BEING EXECUTED.
    {
        printf("ADDITION : %d\n",a+b);
    }
    else if(n1==0 && n2>0) //FIRST CHILD IS BEING EXECUTED.
    {
        printf("SUBTRACTION : %d\n",a-b);
    }
    else if(n1>0 && n2==0) //SECOND CHILD IS BEING EXECUTED.
    {
        printf("MULTIPLICATION : %d\n",a*b);
    }
    else if(n1==0 && n2==0) //THIRD CHILD IS BEING EXECUTED.
    {
        printf("DIVISION : %d\n",a/b);
    }
}

```

```

matlab@sjt121site013: ~
matlab@sjt121site013:~$ gcc 1bb.c
matlab@sjt121site013:~$ ./a.out
ADDITION : 35
SUBTRACTION : -11
MULTIPLICATION : 276
matlab@sjt121site013:~$ DIVISION : 0

```

3)

```

#include<stdio.h>
#include<stdlib.h>
#include <unistd.h>
int main(int argc, char const *argv[])
{
    char *args[] = {"/os2",NULL};
    printf("HI I AM THE ORIGINAL PROCESS WHICH IS RUNNING
    NORMALLY.\n");
    execv(args[0],args);
    printf("THIS CODE WONT BE EXECUTED SINCE THE FLOW WAS DIRECTED TO
    THE OTHER PROGRAM.\n");
    return 0;
}

```

```

#include<stdio.h>
#include <unistd.h>
int main(int argc, char const *argv[])
{
    printf("HI I AM THE SECOND PROGRAM THAT HAS TAKEN OVER DUE TO THE
    FUNCTION CALL OF EXEC.\n");
    return 0;
}

```

```

root@LAPTOP-3TJ8TEKK:~# gcc -o os2 os2.c
root@LAPTOP-3TJ8TEKK:~# ./os2 os2.c
HI I AM THE ORIGINAL PROCESS WHICH IS RUNNING NORMALLY.
HI I AM THE SECOND PROGRAM THAT HAS TAKEN OVER DUE TO THE FUNCTION CALL OF EXEC.

```

4)

```

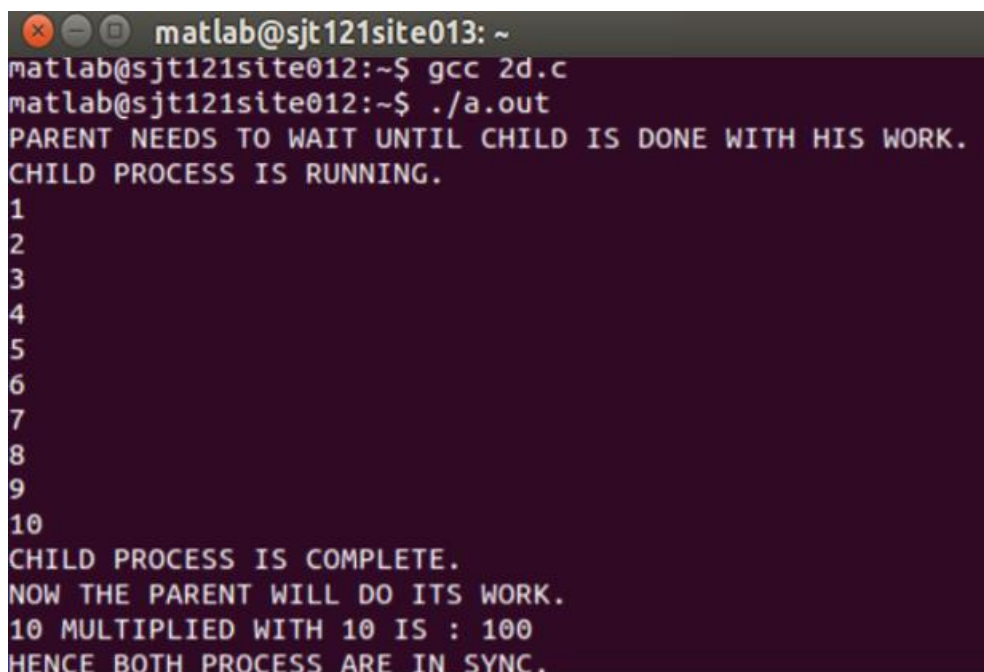
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include <unistd.h>
int main(int argc, char const *argv[])

```

```

{
int a = 0 ;
int d = fork();
int c= 100 ;
int i;
//PERFORM THE OPERATION (a*b)-d
if(d==0)
{
printf("CHILD PROCESS IS RUNNING.\n");
for(i=0 ; i< 10 ; i++)
{
a++;
printf("%d\n",a);
}
printf("CHILD PROCESS IS COMPLETE.\n");
}
else
{
printf("PARENT NEEDS TO WAIT UNTIL CHILD IS DONE WITH HIS WORK.\n");
wait(0);
printf("NOW THE PARENT WILL DO ITS WORK.\n");
printf("10 MULTIPLIED WITH 10 IS : %d\n",c);
printf("HENCE BOTH PROCESS ARE IN SYNC.\n");
}
}

```



```

matlab@sjt121site013: ~
matlab@sjt121site012:~$ gcc 2d.c
matlab@sjt121site012:~$ ./a.out
PARENT NEEDS TO WAIT UNTIL CHILD IS DONE WITH HIS WORK.
CHILD PROCESS IS RUNNING.
1
2
3
4
5
6
7
8
9
10
CHILD PROCESS IS COMPLETE.
NOW THE PARENT WILL DO ITS WORK.
10 MULTIPLIED WITH 10 IS : 100
HENCE BOTH PROCESS ARE IN SYNC.

```

c) Process and Thread Management

Write a program to create a thread and perform the following **(Easy)**

- Create a thread runner function
- Set the thread attributes
- Join the parent and thread
- Wait for the thread to complete

CODE:

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

void *print_message_function( void *ptr );

main()
{
    pthread_t thread1, thread2;
    char *message1 = "Thread 1";
    char *message2 = "Thread 2";
    int  iret1, iret2;

    /* Create independent threads each of which will execute
    function */

    iret1 = pthread_create( &thread1, NULL, print_message_function,
(void*) message1);
    iret2 = pthread_create( &thread2, NULL, print_message_function,
(void*) message2);

    /* Wait till threads are complete before main continues. Unless
we */
    /* wait we run the risk of executing an exit which will
terminate */
    /* the process and all threads before the threads have
completed. */

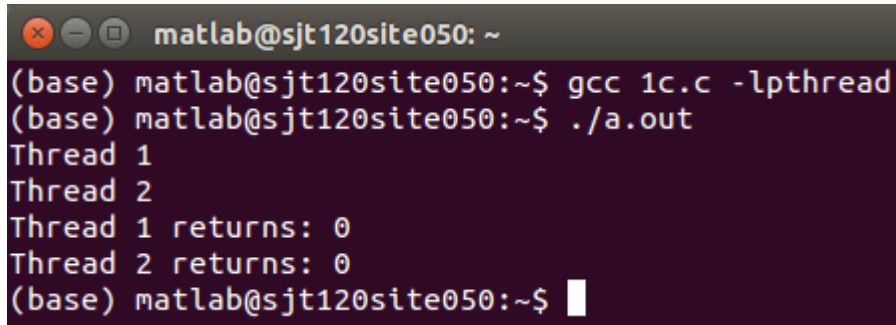
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);

    printf("Thread 1 returns: %d\n",iret1);
    printf("Thread 2 returns: %d\n",iret2);
    exit(0);
}

void *print_message_function( void *ptr )
{
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

```
}
```

OUTPUT:



```
matlab@sjt120site050: ~  
(base) matlab@sjt120site050:~$ gcc 1c.c -lpthread  
(base) matlab@sjt120site050:~$ ./a.out  
Thread 1  
Thread 2  
Thread 1 returns: 0  
Thread 2 returns: 0  
(base) matlab@sjt120site050:~$
```

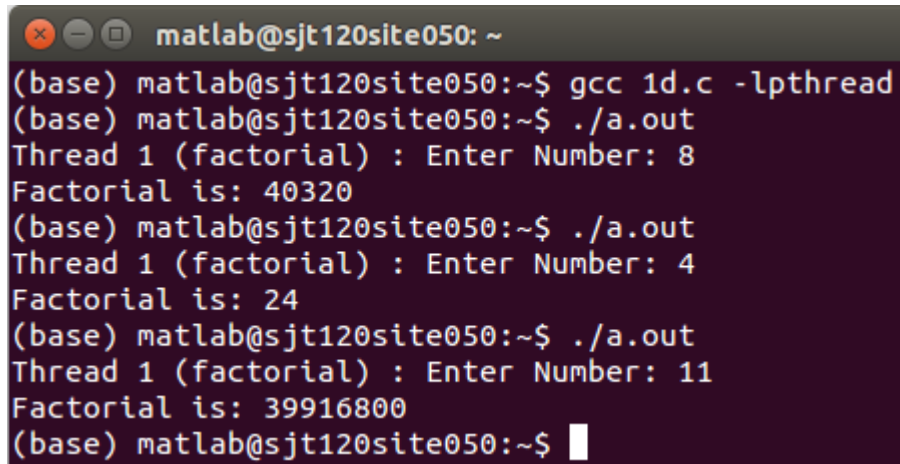
d) Write a program to create a thread to find the factorial of a natural number 'n'. **(Medium)**

CODE:

```
#include<stdio.h>  
#include<pthread.h>  
#include<tgmath.h>  
void *factorial(void *p);  
int fact(int n);  
int main(){  
    pthread_t tid1;  
    pthread_t tid2;  
    pthread_attr_t attr;  
  
    pthread_attr_init(&attr);  
    pthread_create(&tid1,&attr,factorial,NULL);  
    pthread_join(tid1,NULL);  
    pthread_join(tid2,NULL);  
  
}  
int fact(int n){  
    if(n==0 || n==1)  
        return 1;  
    else  
        return n*fact(n-1);  
}  
void *factorial(void *p){  
    int i,num1;  
    printf("Thread 1 (factorial) : ");  
    printf("Enter Number: ");  
    scanf("%d",&num1);  
    printf("Factorial is: %d\n",fact(num1));  
}
```

```
pthread_exit(0);  
  
}
```

OUTPUT:



```
matlab@sjt120site050: ~  
(base) matlab@sjt120site050:~$ gcc 1d.c -lpthread  
(base) matlab@sjt120site050:~$ ./a.out  
Thread 1 (factorial) : Enter Number: 8  
Factorial is: 40320  
(base) matlab@sjt120site050:~$ ./a.out  
Thread 1 (factorial) : Enter Number: 4  
Factorial is: 24  
(base) matlab@sjt120site050:~$ ./a.out  
Thread 1 (factorial) : Enter Number: 11  
Factorial is: 39916800  
(base) matlab@sjt120site050:~$
```

e) Assume that two processes named client and server running in the system. It is required that these two processes should communicate with each other using shared memory concept. The server writes alphabets from a..z to the shared memory .the client should read the alphabets from the shared memory and convert it to A...Z. Write a program to demonstrate the above mentioned scenario. **(Medium)**

CODE:

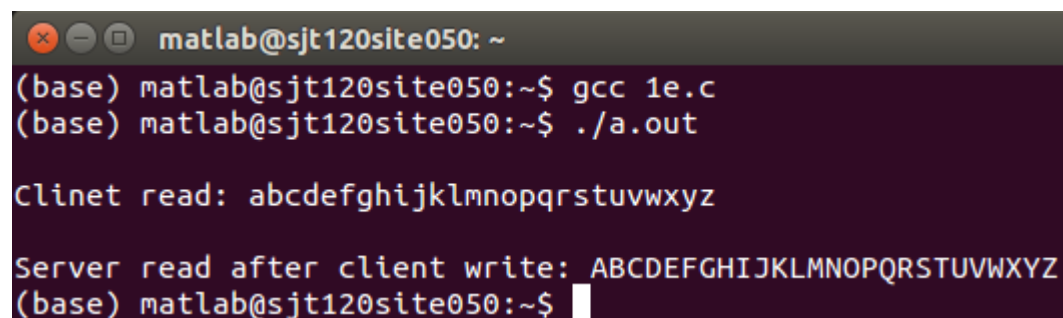
```
#include <stdio.h>  
#include <stdlib.h>  
#include <sys/mman.h>  
#include <unistd.h>  
#include <string.h>  
#include <sys/wait.h>  
#include <sys/types.h>  
#define SIZE 27  
  
int main() {  
    void* shared_mem = mmap(NULL, SIZE, PROT_READ | PROT_WRITE,  
        MAP_ANONYMOUS | MAP_SHARED, -1, 0);  
    int i;  
    char ch;  
    char *server_write = (char *)calloc(SIZE, sizeof(char));  
    for(ch = 'a', i = 0; ch <= 'z'; ++ch){  
        server_write[i++] = ch;  
    }
```

```

}
server_write[i] = '\0';
memcpy(shared_mem, server_write, strlen(server_write));
pid_t pid = fork();
if(pid < 0) {
    printf("\nFork error occurred!");
    exit(-1);
}
else if(pid == 0) {
    printf("\nClient read: %s\n", (char *)shared_mem);
    char *client_write = (char *)malloc(sizeof(char) *
    (strlen(shared_mem)));
    strcpy(client_write, shared_mem);
    for(i = 0; client_write[i] != '\0'; ++i){
        client_write[i] -= 32;
    }
    memcpy(shared_mem, client_write, strlen(client_write));
    exit(0);
}
else {
    wait(&pid);
    printf("\nServer read after client write: %s", (char
    *)shared_mem);
}
return 0;
}

```

OUTPUT:



```

matlab@sjt120site050: ~
(base) matlab@sjt120site050:~$ gcc 1e.c
(base) matlab@sjt120site050:~$ ./a.out

Client read: abcdefghijklmnopqrstuvwxyz

Server read after client write: ABCDEFGHIJKLMNOPQRSTUVWXYZ
(base) matlab@sjt120site050:~$

```

f) The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

$n = n/2$, if n is even $n = 3 \times n + 1$, if n is odd

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if $n = 35$,

the sequence is 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1. Write a C program using the fork () system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the Command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the wait () call to wait for the child process to complete before exiting the program **(High)**.

CODE:

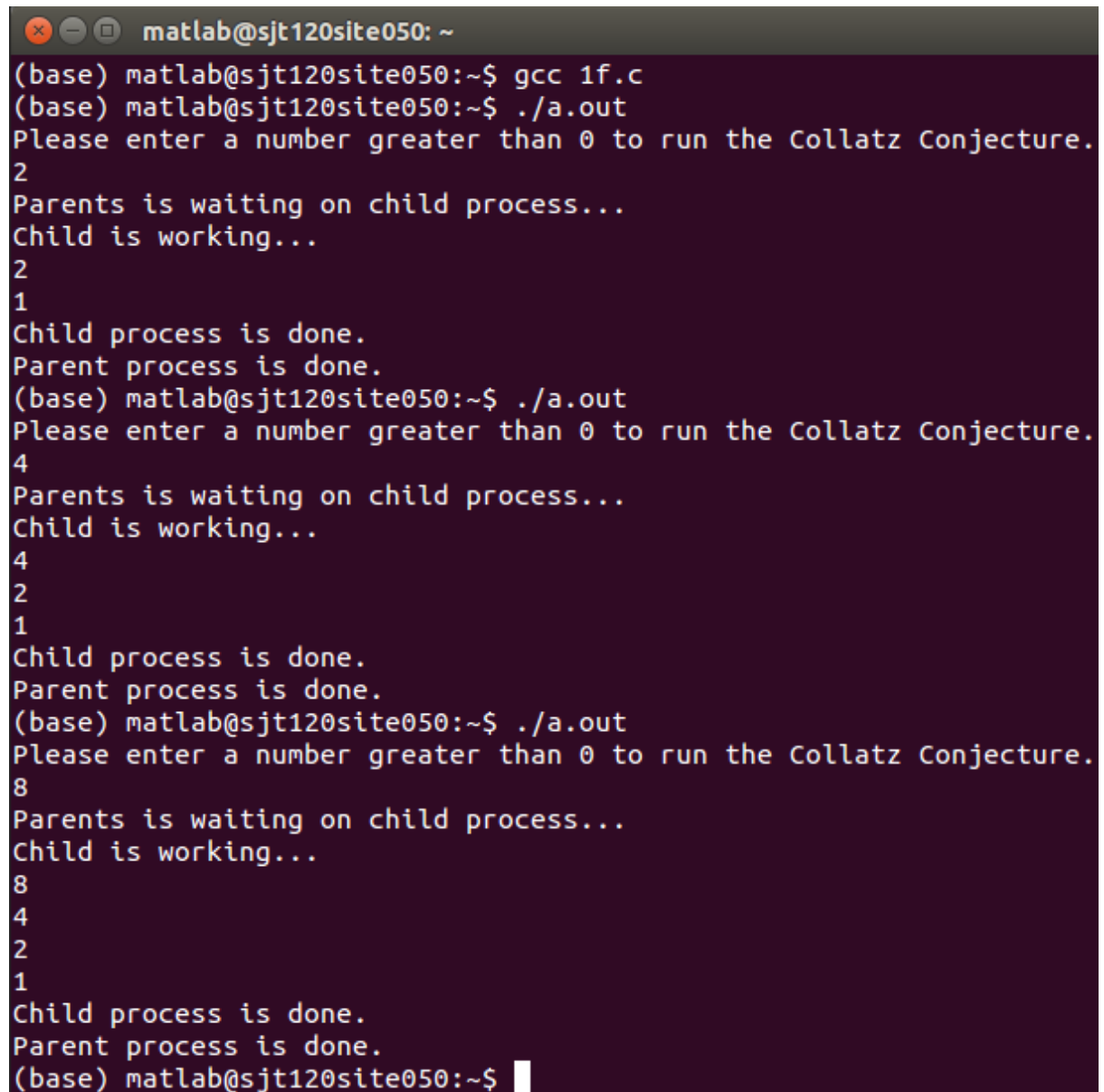
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    int n=0;
    int k=0;
    pid_t pid;
    do
    {
        printf("Please enter a number greater than 0 to run the
        Collatz Conjecture.\n");
        scanf("%d", &k);
    }while (k <= 0);
    pid = fork();
    if (pid == 0)
    {
        printf("Child is working...\n");
        printf("%d\n",k);
        while (k!=1)
        {
            if (k%2 == 0)
            {
                k = k/2;
            }
            else if (k%2 == 1)
            {
                k = 3 * (k) + 1;
            }
            printf("%d\n",k);
        }
        printf("Child process is done.\n");
    }
    else
```

```

{
    printf("Parents is waiting on child process...\n");
    wait();
    printf("Parent process is done.\n");
}
return 0;
}

```

OUTPUT:



```

matlab@sjt120site050: ~
(base) matlab@sjt120site050:~$ gcc 1f.c
(base) matlab@sjt120site050:~$ ./a.out
Please enter a number greater than 0 to run the Collatz Conjecture.
2
Parents is waiting on child process...
Child is working...
2
1
Child process is done.
Parent process is done.
(base) matlab@sjt120site050:~$ ./a.out
Please enter a number greater than 0 to run the Collatz Conjecture.
4
Parents is waiting on child process...
Child is working...
4
2
1
Child process is done.
Parent process is done.
(base) matlab@sjt120site050:~$ ./a.out
Please enter a number greater than 0 to run the Collatz Conjecture.
8
Parents is waiting on child process...
Child is working...
8
4
2
1
Child process is done.
Parent process is done.
(base) matlab@sjt120site050:~$ █

```

g) Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of

numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers 90 81 78 95 79 72 85 , the program will report the average value as 82. The minimum value as 72. The maximum value as 95. The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited. **(High)**

CODE:

```
#include<stdio.h>
#include<pthread.h>
int arr[50],n,i;

void *th()
{
    int sum=0;
    float average;
    printf("enter your number :=");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    for(i=0;i<n;i++)
    {
        sum=sum+arr[i];
    }
    average=sum/n;
    printf("The average value is:%f",average);
}
void *th1()
{
    int temp=arr[0];
    for(i=1;i<n;i++)
    {
        if(temp>arr[i])
        {
            temp=arr[i];
        }
    }
    printf("\nThe Minimum value is:=%d",temp);
}
```

```

}
void *th2()
{
    int temp=arr[0];
    for(i=1;i<n;i++)
    {
        if(temp<arr[i])
        {
            temp=arr[i];
        }
    }
    printf("\nThe Maximum value is:=%d",temp);
}
int main()
{
    int n,i;
    pthread_t t1;
    pthread_t t2;
    pthread_t t3;
    n=pthread_create(&t1,NULL,&th,NULL);
    pthread_join(t1,NULL);
    printf("\n done and my value is %d",n);
    n=pthread_create(&t2,NULL,&th1,NULL);
    pthread_join(t2,NULL);
    printf("\n done and my value is %d",n);
    n=pthread_create(&t3,NULL,&th2,NULL);
    pthread_join(t3,NULL);
    printf("\n done and my value is %d",n);
}

```

OUTPUT:


```
matlab@sjt120site050: ~  
(base) matlab@sjt120site050:~$ gcc 1g.c -lpthread  
(base) matlab@sjt120site050:~$ ./a.out  
Enter Number: 7  
90  
81  
78  
95  
79  
72  
85  
Average value: 82.000000  
done and my value is 0  
Minimum value: 72  
done and my value is 0  
Maximum value: 95  
done and my value is 0(base) matlab@sjt120site050:~$ ./a.out  
Enter Number: 5  
26  
31  
53  
22  
41  
Average value: 34.000000  
done and my value is 0  
Minimum value: 22  
done and my value is 0  
Maximum value: 53  
done and my value is 0(base) matlab@sjt120site050:~$
```