MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

Nazirabad, P.O. - Uchhepota, Near URBANA Complex, Anandapur, Kolkata 700 150

Computer Science & Engineering

LABORATORY NOTE BOOK

MAKAUT EVEN / ODD SEMESTER 2024-25

PAPER NAME: OPERATING SYSTEMS

PAPER CODE: PCC-CS592

STUDENT NAME: AMIR SOHEL SARKAR MASUM

ROLL NO:	14200122052	REGN. NO.:	221420110073
YEAR:	3rd	SEMESTER:	5th
SECTION:	A	SESSION:	2024-25

“LIST OF ASSIGNMENT/EXPERIMENT SUBMISSION DETAILS”

SL. NO.	ASSIGNMENT / EXPERIMENT NAME	DATE OF ASSIGNMENT / EXPERIMENT DONE	DATE OF SUBMISION	CHECKED BY	REMARKS (ANY DEVIATION REGARDING SUBMISSION DATES, CONTENT, FORMAT, ETC)

OBSERVATIONS / COMMENTS ON THE OVERALL PERFORMANCE:

Signature in full with date
Faculty / Technical Assistant

1. Understanding UNIX/LINUX environments and familiarization with different type of commands

Briefly explain the following commands with suitable examples:

(i) pwd (ii) ls (iii) date (iv) cal (v) echo (vi) mkdir (vii) cd (viii) cat

(ix) tty (x) mv (xi)cp (xii) rmdir (xiv)rm (xv) vi editor (xv) chmod

Solution :

I. **pwd -** Print Working Directory

Description: Displays the current working directory's path.

Example: pwd

Output: /home/user/Documents

II. **ls** - List Files and Directories

Description: Lists the files and directories in the current directory.

Example: ls

Output: file1.txt file2.txt directory1

III. **date** - Display the Current Date and Time

Description: Shows the current date and time.

Example: date

Output: Thu Nov 3 14:30:00 UTC 2023

IV. **cal -** Display a Calendar

Description: Shows a calendar for the current month.

Example: cal

Output: A calendar for the current month.

V. **echo -** Display a Message

Description: Prints a message to the terminal.

Example: echo "Hello, World"

Output: Hello, World

VI. **mkdir -** Make a Directory

Description: Creates a new directory.

Example: mkdir new_directory

Creates a directory named **"new_directory."**

VII. **cd -** Change Directory

Description: Changes the current working directory.

Example: cd new_directory

Changes to the **"new_directory"** directory.

VIII. **cat -** Concatenate and Display File Content

Description: Displays the content of a file.

Example: cat file.txt

Displays the content of the **"file.txt"** file.

IX. **tty-**

Description: Display the Terminal Device Name

Example: tty

Output: /dev/pts/0

X. **mv -**

Description: Move or Rename Files and Directories

Example: `mv file.txt new_location/`

Moves **"file.txt"** to the **"new_location"** directory.

XI. **cp-**

Description: Copy Files and Directories

Example: cp file.txt backup/

Copies **"file.txt"** to the **"backup"** directory.

XII. **rmdir -**

Description: Remove Empty Directories

Example: rmdir empty_dir

Removes the **"empty_dir"** directory if it's empty.

XIII. **rm-**

Description: Remove Files or Directories

Example: rm file.txt

Deletes the **"file.txt"** file.

XIV. **`vi` -**

Description: Used as a Text Editor

Example: vi newfile.txt

Opens the **"newfile.txt"** file for editing in the vi text editor.

XV. **chmod –**

Description: Change File Permissions

Example: chmod 644 file.txt

Sets read and write permissions for the owner and read-only permissions for others on **"file.txt."**

2. SHELL SCRIPT

2.1. Write a shell script to display “Hello world” message on the screen

Code:

```
echo “Hello world”
```

Output:

```
Hello world
```

2.2. Write a shell script to show all files having extension .sh

Code:

```
echo "Files with the .sh extension in the current directory:"
for file in *.sh
do
        echo "$file"
done
```

Output:

```
Files with the .sh extension in the current directory:
 Assignment_2_1.sh
```

2.3. Write a shell script to show all files whose names are beginning with letters a,b,c.

Code:

```
echo "Files with names starting with 'a,' 'b,' or 'c' in the current directory:"
for file in [a-c]*
 do
        echo "$file"
done
```

Output:

```
Files with names starting with 'a,' 'b,' or 'c' in the current directory:
ankit.txt
bin.c
calculator.py
```

2.4. Write a shell script to input two numbers and find addition, subtraction,multiplication, division and remainder.

Code:

```
echo "Enter the first number: "
read a
echo "Enter the second number: "
read b
sum=`expr $a + $b`
difference=`expr $a - $b`
product=`expr $a \* $b`
division=`expr $a / $b`
remainder=`expr $a % $b`

echo   "Results   Are   :"
echo "Addition: $sum"
echo "Subtraction: $difference"
echo "Multiplication: $product"
echo "Division: $division"
echo "Remainder: $remainder"
```

Output:

```
Enter the first number:
20
Enter the second number:
10
Results Are :
Addition: 30
Subtraction: 10
Multiplication: 200
Division: 2
Remainder: 0
```

2.5. Write a shell script to perform menu driven calculator operations.

Code:

```
echo "Enter the first number"
read a
echo "Enter the second number"
read b
echo "Enter the operator:"
echo -e "Addition: +\nSubtraction: -\nMultiplication: x\nDivision: /"
read op
case $op in
        +) c=`expr $a + $b`
                echo "Sum of $a and $b is $c";;
        -) c=`expr $a - $b`
                echo "Difference of $a and $b is $c";;
        x) c=`expr $a \* $b`
                echo "Product of $a and $b is $c";;
        /) c=`expr $a / $b`
                echo "Division of $a and $b is $c";;
        *) echo "Invalid Operator"
                exit;;
esac
```

Output:

```
Enter the first number
7
Enter the second number
9
Enter the operator:
Addition: +
Subtraction: -
Multiplication: x
Division: /
+
Sum of 7 and 9 is 16
```

2.6. Write a shell script to input three numbers and find maximum value using command line arguments.

Code:

```
echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
   echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
   echo $num2
else
   echo $num3
fi
```

Output:

```
Enter Num1
1
Enter Num2
34
Enter Num3
2
34
```

2.7. Write a shell script to input marks of three subjects and find grade according to MAKAUT rule.

Code:

```
echo “Enter number of subject 1 : “
read a
echo “Enter number of subject 2 : “
read b
echo “Enter number of subject 3 : “
read b
sum=`expr $a + $b + $c`
sum=`expr $sum / 3`
if [ $sum -ge 90 ]
then
        echo “Grade : O”
elif [ $sum -ge 80 -a $sum -le 89 ]
then
        echo “Grade : E”
elif [ $sum -ge 70 -a $sum -le 79 ]
then
        echo “Grade : A”
elif [ $sum -ge 60 -a $sum -le 69 ]
then
        echo “Grade : B”
elif [ $sum -ge 50 -a $sum -le 59 ]
then
        echo “Grade : C”

elif [ $sum -ge 40 -a $sum -le 49 ]
then
        echo “Grade : D”
else
        echo “Grade : D”
fi
```

Output :

```
Enter number of subject 1 :
85
Enter number of subject 2 :
86
Enter number of subject 3 :
84
Grade : E
```

3. SHELL SCRIPT

3.1. Write a shell script to print 1+2+3+4+5=15 (e.g, n=5)

Code:

```
echo "Enter the value of n: "
read n
sum=0
for ((i = 1; i <= n; i++))
do
   sum=$((sum + i))
done
echo -n "1"
for ((i = 2; i <= n; i++))
do
   echo -n "+$i"
done
echo "=$sum"
```

Output:

```
Enter the value of n:
5
Sum of numbers from 1 to 5 is 15
1+2+3+4+5=15
```

3.2. Write a shell script to input a number and find the factorial of a given integer number

Code:

```
read -p "Enter a number : " num
fact=1
for((i=2;i<=num;i++))
{
  fact=$((fact*i))
}
echo "Factorial of $num is : $fact"
```

Output:

```
Enter a number : 5

Factorial of 5 is : 120
```

3.3. <u>Write a shell script to input a number and find the sum of digit and count the number of digit of a given integer number.</u>

Code:

```
read -p "Enter a number : " num

sum=0

while [ $num -gt 0 ]

do

   mod=`expr $num % 10`

   sum=`expr $sum + $mod`

   num=`expr $num / 10`

done

echo “The sum of the digits : $sum”
```

Output:

```
Enter a number : 786

The sum of the digits : 21
```

3.4. Write a shell script to input a number , find reverse of a number and check whether an input number is Palindrome or NOT.

Code:

```
read -p "Enter the number : " n

num=0

on=$n

while [ $n -gt 0 ]

do

   num=`expr $num \* 10`

   k=`expr $n % 10`

   num=`expr $num + $k`

   n=`expr $n / 10`

done

echo "The reversed number is : $num"

if [ $num -eq $on ]

then

   echo "$on is a palindrome"

else

   echo "$on is not a palindrome"

fi
```

Output:

```
number : 10

The reversed number is : 1

10 is not a palindrome
```

3.5. Write a shell script to input a number and check whether the number is prime nor Not.

Code:

```
read -p "Enter number: " n
d=2
r=1
while [ $d -lt $n -a $r -ne 0 ]
do
    r=`expr $n % $d`
    d=`expr $d + 1`
done

if [ $r -eq 0 ]
then
        echo "$n is not a prime number"
else
        echo "$n is a prime number"
fi
```

Output :

```
Enter number: 12
12 is not a prime number
```

3.6. Write a shell script to input a number and check whether the number is Armstrong or Not.

Code:

```
read -p "Enter a number: " num
original_num=$num
num_of_digits=${#num}
sum=0
while [ $num -gt 0 ]
do
   digit=$((num % 10))
   sum=$((sum + digit ** num_of_digits))
   num=$((num / 10))
done

if [ $sum -eq $original_num ]; then
   echo "$original_num is an Armstrong number."
else
   echo "$original_num is not an Armstrong number."
Fi
```

Output:

```
Enter the number: 111
111 is not an Armstrong number
```

4. SHELL SCRIPT

4.1. Write a shell script to print Fibonacci series.

Code:

```
read -p "Enter the number of elements : " N
a=0
b=1
echo "The Fibonacci series is : "
for (( i=0; i<N; i++ ))
do
        echo -n "$a "
        fn=$((a + b))
        a=$b
        b=$fn
done
```

Output:

```
Enter the number of elements : 5
The Fibonacci series is :
0 1 1 2 3
```

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

Code:

```
num=2
while [ $num -le 100 ]
do
    is_prime=true
    i=2
    while [ $i -lt $num ]
    do
        if [ $(($num%$i)) -eq 0 ]
        then
            is_prime=false
            break
        fi
        i=`expr $i + 1`
    done
    if $is_prime
    then
        echo $num
    fi
    num=`expr $num + 1`
done
```

Output :

```
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
```

4.3. Write a shell script to display all ARMSTRONG numbers from1 to 10,000.

Code:

```
number=1
while [ $number -le 10000 ]
do
  n=$number
  num_of_digits=${#n}
  sum=0

  while [ $n -gt 0 ]
  do
    digit=`expr $n % 10`
    sum=$((sum + digit ** num_of_digits))
    n=$((n / 10))
  done
  if [ $sum -eq $number ]; then
    echo $number
  fi
  number=`expr number + 1`
done
```

Output:

```
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.

Code:

```
echo "Enter the number of elements: "
read n
echo "Enter the elements: "
for ((i = 0; i < n; i++))
do
   read arr[i]
done
for ((i = 0; i < n-1; i++))
do
   for ((j = 0; j < n-i-1; j++))
   do
      if [ ${arr[j]} -gt ${arr[j+1]} ]
      then
         temp=${arr[j]}
         arr[j]=${arr[j+1]}
         arr[j+1]=$temp
      fi
   done
done
echo "Sorted array in ascending order:"
for ((i = 0; i < n; i++)); do
   echo "${arr[i]}"
done
```

Output :

```
Enter the number of elements:
5
Enter the elements:
4
1
5
2
3
Sorted array in ascending order:
1
2
3
4
5
```

5 : FILE ORGANISATION SHELL SCRIPT

5.1. Write a shell script to input a string and check whether the string is palindrome or Not.

Code:

```
echo "Enter a String"
read input
reverse=""
len=${#input}
for (( i=$len-1; i>=0; i-- ))
do
        reverse="$reverse${input:$i:1}"
done
if [ $input == $reverse ]
then
  echo "$input is palindrome"
else
  echo "$input is not palindrome"
fi
```

Output :

```
Enter a String
test
test is not palindrome
```

5.2. Write a shell script to input a user name and check whether the input user is valid or not.

Code:

```
read -p "Enter the username to check: " username
if grep -q "^$username:" /etc/passwd; then
   echo "User '$username' exists."
else
   echo "User '$username' does not exist."
fi
```

Output :

```
Enter the username to check: someuser
User 'someuser' exists.
```

5.3. Write a shell script to input a word and count the number of vowels.

Code:

```
read -p "Enter a word: " word
word="${word,,}"  # Convert the word to lowercase for case-insensitive matching
vowels="aeiou"
count=0
for ((i=0; i<${#word}; i++))
 do
        letter="${word:i:1}"
        if [[ "$vowels" == *"$letter"* ]]
        then
                ((count++))
        fi
done
echo "The word '$word' contains $count vowel(s)."
```

Output :

```
Enter a word: Hello
The word 'hello' contains 2 vowel(s).
```

5.4. Write a shell script to receives two file names as argument and check whether the content of two files are same or not. If they are same then second file should be deleted.

Code:

```
file1=$1

file2=$2

if cmp $file1 $file2

then

        echo "Both are same"

        rm -i $file2

else

        echo "Contents are different"

fi

echo "Job over"
```

Output:

```
$ ./compare_and_delete.sh file1.txt file2.txt

Both files have the same content.

remove file2.txt? y

Job over
```

5.5. Write a shell script to accept a string from the terminal and echo a suitable message if it does not have at least 10 characters.

Code:

```
read -p "Enter a string: " input_string

if [ ${#input_string} -lt 10 ]

then

        echo "The entered string has less than 10 characters."

else

        echo "The entered string has at least 10 characters or more."

Fi
```

Output:

```
Enter a string: hello

The entered string has less than 10 characters.
```

6-8 : PROCESS

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

Code:

```
#include <stdio.h>

#include <unistd.h>


int main() {

        uid_t uid = getuid();

        printf("UID (User ID): %d\n", uid);

        pid_t pid = getpid();

        printf("PID (Process ID): %d\n", pid);

        pid_t ppid = getppid();

        printf("PPID (Parent Process ID): %d\n", ppid);

        return 0;

}
```

Output :

```
UID (User ID): 1000

PID (Process ID): 12345

PPID (Parent Process ID): 6789
```

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

Code:

```
#include   <stdio.h>
#include  <stdlib.h>
#include <unistd.h>
int main() {
        pid_t child_pid = fork();
        if (child_pid < 0) {
                perror("Fork  failed");
                exit(1);
}
if (child_pid == 0) {
        printf("Child process (PID %d) created\n", getpid());
        sleep(2);
        printf("Child process (PID %d) exiting\n", getpid());
}
else {
        printf("Parent process (PID %d) created child process (PID %d)\n", getpid(), child_pid);
        sleep(1);
        printf("Parent process (PID %d) exiting\n", getpid());
  }
  return 0;
}
```

Output :

```
Parent process (PID 12345) created child process (PID 12346)
Parent process (PID 12345) exiting
Child process (PID 12346) created
Child process (PID 12346) exiting
```

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

Code:

```
#include   <stdio.h>

#include  <stdlib.h>

#include <unistd.h>

int main() {

        printf("This is the original program.\n");

        if (execl("/bin/ls", "ls", "-l", NULL) == -1) {

                perror("execl()  failed");

                exit(1);

        }

        printf("This line will not be printed.\n");

        return 0;

}
```

Output :

```
This is the original program.

<list of files and directories displayed by 'ls -l' output>
```

8. i. Write C programs to simulate the following CPU Scheduling algorithms FCFS and SJF

SJF:

Code:

```
#include <stdio.h>

#include <string.h>


struct SJF {

   int exetime;

   char pid[4];

};


int main() {

   struct SJF P[10];

   int n, wtime = 0, tAT, tot_wtime = 0;

   int i, j, temp;

   char t[4];

   float avgwaitingTime;


   printf("Enter how many number of processes: ");

   scanf("%d", &n);


   for (i = 0; i < n; i++) {

      printf("Enter PID and Burst Time: ");

      scanf("%s%d", P[i].pid, &P[i].exetime);

   }


   printf("\nPID\tBurst Time\n");

   for (i = 0; i < n; i++) {

      printf("%s\t%d\n", P[i].pid, P[i].exetime);

   }


   printf("\n\n");


   for (i = 1; i < n; i++) {

      for (j = 0; j < n - i; j++) {
            if (P[j].exetime > P[j + 1].exetime) {
                temp = P[j].exetime;
                P[j].exetime = P[j + 1].exetime;
                P[j + 1].exetime = temp;

                strcpy(t, P[j].pid);
                strcpy(P[j].pid, P[j + 1].pid);
                strcpy(P[j + 1].pid, t);
            }
        }
    }

    printf("\nPID\tBurst Time\n");
    for (i = 0; i < n; i++) {
        printf("%s\t%d\n", P[i].pid, P[i].exetime);
    }

    printf("\n\n");

    printf("PID\tWaiting Time\tBurst(EXE) Time\tTurn Around Time\n");
    for (i = 0; i < n; i++) {
        tAT = P[i].exetime + wtime;
        printf("%s\t%d\t%d\t%d\n", P[i].pid, wtime, P[i].exetime, tAT);
        tot_wtime = tot_wtime + wtime;
        wtime = wtime + P[i].exetime;
    }

    avgwaitingTime = (float)tot_wtime / n;
    printf("Total Waiting Time: %d\nAverage Waiting Time: %.2f\n", tot_wtime, avgwaitingTime);

    return 0;
}
```

Output :

```
Enter how many number of processes: 3

Enter PID and Burst Time for P1: A 5

Enter PID and Burst Time for P2: B 3

Enter PID and Burst Time for P3: C 6

PID     Burst Time

A           5

B           3

C           6


PID     Burst Time

B           3

A           5

C           6


PID     Waiting Time     Burst(EXE) Time     Turn Around Time

B           0                    3                    3

A           3                    5                    8

C           8                    6                    14

Total Waiting Time: 11

Average Waiting Time: 3.67
```

FCFS:

Code:

```
#include <stdio.h>
int main() {
    int n, exetime[10], wtime = 0, tAT, tot_wtime = 0;
    int i;
    float avgwaitingTime;
    printf("Enter how many number of processes: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("Enter exe time for process %d: ", i + 1);
        scanf("%d", &exetime[i]);
    }
    printf("\nPID\tBurst Time\n");
    for (i = 0; i < n; i++) {
        printf("P%d\t%d\n", i + 1, exetime[i]);
    }
    printf("\n\n");
    printf("PID\tWaiting Time\tBurst(EXE) Time\tTurn Around Time\n");
    for (i = 0; i < n; i++) {
        tAT = exetime[i] + wtime; // Turn around time = Burst time + wait time
        printf("P%d\t%d\t%d\t%d\n", i + 1, wtime, exetime[i], tAT);
        tot_wtime = tot_wtime + wtime;
        wtime = wtime + exetime[i];
    }
    avgwaitingTime = (float)tot_wtime / n;
    printf("Total Waiting Time: %d\nAverage Waiting Time: %.2f\n", tot_wtime, avgwaitingTime);
    return 0;
}
```

Output :

```
Enter how many number of processes: 3

Enter exe time for process 1: 5

Enter exe time for process 2: 3

Enter exe time for process 3: 6


PID    Burst Time

P1       5

P2       3

P3       6


PID    Waiting Time    Burst(EXE) Time    Turn Around Time

P1       0                  5                  5

P2       5                  3                  8

P3       8                  6                  14

Total Waiting Time: 13

Average Waiting Time: 4.33
```

8.ii. **Write C programs to simulate the following CPU Scheduling algorithms Priority Scheduling and Round Robin Algorithm**

Priority Scheduling :

Code:

```
#include <stdio.h>

#include <string.h>


struct PriorityScheduling {

    int exetime;

    int priority;

    char pid[6];

};


int main() {

    struct PriorityScheduling P[10];

    int n, wtime = 0, tAT, tot_wtime = 0;

    int i, j, temp;


    float avgwaitingTime;


    printf("Enter how many number of processes: ");

    scanf("%d", &n);


    for (i = 0; i < n; i++) {

        printf("Enter PID, Burst Time, and Priority: ");

        scanf("%s %d %d", P[i].pid, &P[i].exetime, &P[i].priority);

    }


    printf("\nPID\tBurst Time\tPriority\n");

    for (i = 0; i < n; i++) {

        printf("%s\t%d\t%d\n", P[i].pid, P[i].exetime, P[i].priority);

    }


    printf("\n\n");
    for (i = 1; i < n; i++) {
        for (j = 0; j < n - i; j++) {
            if (P[j].priority > P[j + 1].priority) {
                temp = P[j].priority;
                P[j].priority = P[j + 1].priority;
                P[j + 1].priority = temp;

                strcpy(P[j].pid, P[j + 1].pid);
                strcpy(P[j + 1].pid, P[j].pid);

                temp = P[j].exetime;
                P[j].exetime = P[j + 1].exetime;
                P[j + 1].exetime = temp;
            }
        }
    }

    printf("\nPID\tBurst Time\tPriority\n");
    for (i = 0; i < n; i++) {
        printf("%s\t%d\t%d\n", P[i].pid, P[i].exetime, P[i].priority);
    }

    printf("\n\n");

    printf("\nPID\tWaiting Time\tBurst(EXE) Time\tTurn Around Time");
    for (i = 0; i < n; i++) {
        tAT = P[i].exetime + wtime; // Turn around time = Burst time + wait time
        printf("\n%s\t%d\t%d\t%d\n", P[i].pid, wtime, P[i].exetime, tAT);
        tot_wtime = tot_wtime + wtime;
        wtime = wtime + P[i].exetime;
    }

    avgwaitingTime = (float)tot_wtime / n;
    printf("Total Waiting Time: %d\nAverage Waiting Time: %.2f\n", tot_wtime, avgwaitingTime);
    return 0;
}
```

Output:

```
Enter how many number of processes: 3

Enter PID, Burst Time, and Priority for Process 1: P1 5 3

Enter PID, Burst Time, and Priority for Process 2: P2 3 2

Enter PID, Burst Time, and Priority for Process 3: P3 6 1

PID    Burst Time   Priority

P1          5          3

P2          3          2

P3          6          1


PID     Burst Time  Priority

P3          6          1

P2          3          2

P1     5          3


PID     Waiting Time    Burst(EXE) Time     Turn Around Time

P3          0                   6                     6

P2          6                   3                     9

P1          9                   5                     14

Total Waiting Time: 15

Average Waiting Time: 5.00
```

Round Robin Algorithm:

Code:

```
#include <stdio.h>

struct Process {
    int process_id;
    int burst_time;
    int remaining_time;
};

int main() {
    int n, time_quantum;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

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

    struct Process processes[n];

    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Enter burst time for Process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
    }

    int current_time = 0;
    int completed = 0;

    printf("\nExecution Order:\n");

    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                int execute_time = (processes[i].remaining_time < time_quantum) ?
                processes[i].remaining_time : time_quantum;

                processes[i].remaining_time -= execute_time;

                current_time += execute_time;


                printf("Process %d for %d units of time\n", processes[i].process_id, execute_time);


                if (processes[i].remaining_time == 0) {

                    completed++;

                }

            }

        }

    }

    printf("\nAll processes have completed.\n");

    return 0;

}
```

Output:

```
Enter the number of processes: 3

Enter the time quantum: 2

Enter burst time for Process 1: 5

Enter burst time for Process 2: 3

Enter burst time for Process 3: 7


Execution Order:

Process 1 for 2 units of time

Process 2 for 2 units of time

Process 3 for 2 units of time

Process 1 for 2 units of time

Process 2 for 1 units of time

Process 3 for 2 units of time

Process 1 for 1 units of time

Process 3 for 1 units of time

Process 3 for 3 units of time


All processes have completed.
```

09. SIGNAL

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

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <signal.h>
int main(int argc, char *argv[]) {
    int pid = fork();
    if (pid == -1) {
        printf("\nFork creation failure.");
        return 1;
    }
    if (pid == 0) {
        while (1) {
            printf("\nWelcome.");
            usleep(50000);
        }
    } else {
        kill(pid, SIGSTOP);
        int t;
        do {
            printf("\nTime in seconds for execution: ");
            scanf("%d", &t);
            if (t > 0) {
                kill(pid, SIGCONT);
                sleep(t);
                kill(pid, SIGSTOP);
            }
        } while (t > 0);
        kill(pid, SIGKILL);
        wait(NULL);
    }
    return 0;
}
```

Output:

```
Time in seconds for execution: 1

Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.
Welcome.

Time in seconds for execution: 0
```

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

int main() {
    pid_t child_pid;

    // Create a child process
    child_pid = fork();

    if (child_pid == -1) {
        perror("Fork failed");
        return 1;
    }

    if (child_pid == 0) {
        // Child process
        while (1) {
            printf("Welcome from the child process\n");
            usleep(50000); // Sleep for 50 milliseconds
        }
    } else {
        // Parent process
        sleep(1); // Sleep for 1 second
        kill(child_pid, SIGKILL); // Kill the child process
        wait(NULL); // Wait for the child process to exit
    }

    return 0;
}
```

Output :

```
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
Welcome from the child process
```

10. SEMAPHORE

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

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define BUFFER_SIZE 5
sem_t empty, full, mutex;
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
void* producer(void* arg) {
  for(int i = 0; i < 10; i++) {
    sem_wait(&empty);
    sem_wait(&mutex);
    buffer[in] = i;
    printf("Produced: %d\n", i);
    in = (in + 1) % BUFFER_SIZE;
    sem_post(&mutex);
    sem_post(&full);
  }
  return NULL;
}

void* consumer(void* arg) {
  for(int i = 0; i < 10; i++) {
        sem_wait(&full);
        sem_wait(&mutex);
        int item = buffer[out];
        printf("Consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;
        sem_post(&mutex);
        sem_post(&empty);
    }
    return NULL;
}
int main() {
    pthread_t prod_thread, cons_thread;
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);
    sem_init(&mutex, 0, 1);
    pthread_create(&prod_thread, NULL, producer, NULL);
    pthread_create(&cons_thread, NULL, consumer, NULL);
    pthread_join(prod_thread, NULL);
    pthread_join(cons_thread, NULL);
    sem_destroy(&empty);
    sem_destroy(&full);
    sem_destroy(&mutex);
    return 0;
}
```

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

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define N 5
sem_t mutex;
void* process(void* arg) {
    int id = *((int*)arg);
    while (1) {
        sem_wait(&mutex);
        // Critical Section
        printf("Process %d entered critical section\n", id);
        printf("Process %d exiting critical section\n", id);
        sem_post(&mutex);
        // Non-critical section (outside of the critical section)
        sleep(1);
    }
}
int main() {
    pthread_t threads[N];
    int ids[N];
    sem_init(&mutex, 0, 1);
    for (int i = 0; i < N; i++) {
        ids[i] = i + 1;
        pthread_create(&threads[i], NULL, process, &ids[i]);
    }
    for (int i = 0; i < N; i++) {
        pthread_join(threads[i], NULL);
    }
    sem_destroy(&mutex);
    return 0;
}
```

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).

```
#include <stdio.h>
#include <pthread.h>
void* print_sequence(void* arg) {
   for (int i = 1; i <= 5; i++) {
      printf("%d\n", i);
   }
   return NULL;
}
int main() {
   pthread_t thread;
   if (pthread_create(&thread, NULL, print_sequence, NULL)) {
      fprintf(stderr, "Error creating thread\n");
      return 1;
   }
   if (pthread_join(thread, NULL)) {
      fprintf(stderr, "Error joining thread\n");
      return 2;
   }
   return 0;
}
```

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

```
#include <stdio.h>
#include <pthread.h>
pthread_mutex_t mutex;
int shared_variable = 0;
void* increment(void* arg) {
   for (int i = 0; i < 5; i++) {
      pthread_mutex_lock(&mutex);
      shared_variable++;
      printf("Incremented: %d\n", shared_variable);
      pthread_mutex_unlock(&mutex);
   }
   return NULL;
}
void* decrement(void* arg) {
   for (int i = 0; i < 5; i++) {
      pthread_mutex_lock(&mutex);
      shared_variable--;
      printf("Decremented: %d\n", shared_variable);
      pthread_mutex_unlock(&mutex);
   }
   return NULL;
}
int main() {
   pthread_t thread1, thread2;
   pthread_mutex_init(&mutex, NULL);
   pthread_create(&thread1, NULL, increment, NULL);
   pthread_create(&thread2, NULL, decrement, NULL);
   pthread_join(thread1, NULL);
   pthread_join(thread2, NULL);
   pthread_mutex_destroy(&mutex);
   return 0;
}
```

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:

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

```
int main() {
    int fd[2];
    char buffer[100];

    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }
    pid_t pid = fork();
    if (pid == 0) {
        close(fd[1]);
        read(fd[0], buffer, sizeof(buffer));
        printf("Child received: %s", buffer);
        close(fd[0]);
    } else if (pid > 0) {
        close(fd[0]);
        write(fd[1], "Hello from parent!\n", 19);
        close(fd[1]);
    } else {
        perror("fork");
        return 1;
    }
    return 0;
}
```

(ii) FIFOs:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFO_FILE "myfifo"
int main() {
    mkfifo(FIFO_FILE, 0666);
    int fd = open(FIFO_FILE, O_WRONLY);
    if (fd == -1) {
        perror("open");
        return 1;
    }
    write(fd, "Hello from FIFO!\n", 17);
    close(fd);
    return 0;
}
```

(iii) Message Queues:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct msgbuf {
    long mtype;
    char mtext[100];
};
int main() {
    key_t key;
    int msgid;
    key = ftok(".", 'a');
    msgid = msgget(key, IPC_CREAT | 0666);
    struct msgbuf msg;
    msg.mtype = 1;
    sprintf(msg.mtext, "Hello from message queue!");
    msgsnd(msgid, &msg, sizeof(msg), 0);
    return 0;
}
```

(iv) Shared Memory:

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

int main() {
    key_t key;
    int shmid;
    key = ftok(".", 'a');
    shmid = shmget(key, 1024, IPC_CREAT | 0666);
    char *shmaddr = (char*) shmat(shmid, (void*)0, 0);
    sprintf(shmaddr, "Hello from shared memory!");
    shmdt(shmaddr);
    return 0;
}
```

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.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
    int fd[2];
    char buffer[100];
    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }
    pid_t pid = fork();
    if (pid == 0) {
        close(fd[0]);
        char message[] = "Hello from child!";
        write(fd[1], message, strlen(message) + 1);
        close(fd[1]);
    } else if (pid > 0) {
        close(fd[1]);
        read(fd[0], buffer, sizeof(buffer));
        printf("Parent received: %s\n", buffer);
        close(fd[0]);
    } else {
        perror("fork");
        return 1;
    }
    return 0;
}
```

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

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
    int fd[2];
    char buffer[100];
    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }
```

```
   pid_t pid = fork();
   if (pid == 0) {
      close(fd[0]);
      char message1[] = "Message from child to parent";
      write(fd[1], message1, strlen(message1) + 1);
      char message2[] = "Another message from child";
      write(fd[1], message2, strlen(message2) + 1);
      close(fd[1]);
   } else if (pid > 0) {
      close(fd[1]);
      read(fd[0], buffer, sizeof(buffer));
      printf("Parent received: %s\n", buffer);
      read(fd[0], buffer, sizeof(buffer));
      printf("Parent received: %s\n", buffer);
      close(fd[0]);
   } else {
      perror("fork");
      return 1;
   }
   return 0;
}
```

12.4 Write a system call to create a pipe for TWO-way communication i.e.,

(i) Parent process to write a message and child process to read and display on the screen.

(ii) Child process to write a message and parent process to read and display on the screen.

(iii) Write a client-server program using one and two pipes.

(i) Parent writes a message, child reads and displays it:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
   int fd[2];
   char buffer[100];
   if (pipe(fd) == -1) {
      perror("pipe");
      return 1;
   }
   pid_t pid = fork();
   if (pid == 0) {
      close(fd[0]);
      char message[] = "Hello from child!";
      write(fd[1], message, strlen(message) + 1);
      close(fd[1]);
   } else if (pid > 0) {
      close(fd[0]);
      char message[] = "Hello from parent!";
      write(fd[1], message, strlen(message) + 1);
      close(fd[1]);
   } else {
      perror("fork");
      return 1;
   }
   return 0;
}
```

(ii) Child writes a message, parent reads and displays it:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
   int fd[2];
```

```
    char buffer[100];
    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }
    pid_t pid = fork();
    if (pid == 0) {
        close(fd[0]);
        char message[] = "Hello from child!";
        write(fd[1], message, strlen(message) + 1);
        close(fd[1]);
    } else if (pid > 0) {
        close(fd[1]);
        read(fd[0], buffer, sizeof(buffer));
        printf("Parent received: %s\n", buffer);
        close(fd[0]);
    } else {
        perror("fork");
        return 1;
    }
    return 0;
}
```

(iii) Write a client-server program using one and two pipes.

(a) Server program:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
    int fd[2];
    char buffer[100];
    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }
    pid_t pid = fork();
    if (pid == 0) {
        close(fd[1]);
        char message[] = "Hello, server!";
        write(fd[1], message, strlen(message) + 1);
        close(fd[0]);
    } else if (pid > 0) {
        close(fd[0]);
        read(fd[1], buffer, sizeof(buffer));
        printf("Server received: %s\n", buffer);
        strcat(buffer, " Processed: ");
        write(fd[0], buffer, strlen(buffer) + 1);
        close(fd[1]);
    } else {
        perror("fork");
        return 1;
    }
    return 0;
}
```

(b) Client program:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main() {
    int fd[2];
    char buffer[100];
    if (pipe(fd) == -1) {
        perror("pipe");
        return 1;
    }
    pid_t pid = fork();
    if (pid == 0) {
        close(fd[1]);
        char message[] = "Hello, client!";
        write(fd[1], message, strlen(message) + 1);
        close(fd[0]);
    } else if (pid > 0) {
        close(fd[0]);
        read(fd[1], buffer, sizeof(buffer));
        printf("Server sent: %s\n", buffer);
        close(fd[1]);
        read(fd[0], buffer, sizeof(buffer));
        printf("Client received: %s\n", buffer);
        close(fd[0]);
    } else {
        perror("fork");
        return 1;
    }
    return 0;
}
```