

Tribhuvan University
SHANKER DEV CAMPUS
Putalisadak, Kathmandu

A Lab report on
Operating Systems

Submitted To
BIM Department

In partial fulfillment of the requirement for the award of the Degree of
Bachelor of Information Management (semester 8th)

Submitted By
Name: Nibesh Upadhyaya
Roll No: 11952/20
TU Regd. No: 7-2-39-1685-2020

Date: 2082-02-16

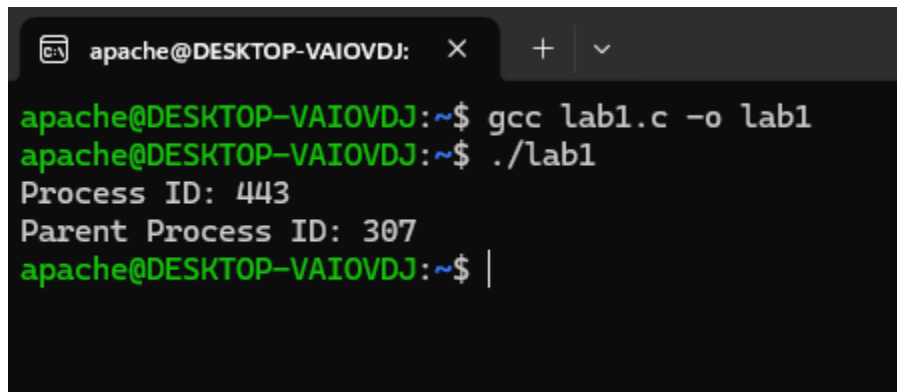
LAB 1: PROCESSES

1.1 Write a program to display process id and parent process id.

Program:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    printf("Process ID: %d\n", getpid() );
    printf("Parent Process ID: %d\n", getppid() );
    return 0;
}
```

Output:

A terminal window titled 'apache@DESKTOP-VAIOVDJ:' with a dark background. The prompt is 'apache@DESKTOP-VAIOVDJ:~\$'. The first command is 'gcc lab1.c -o lab1', followed by the second command './lab1'. The output shows 'Process ID: 443' and 'Parent Process ID: 307'. The prompt returns to 'apache@DESKTOP-VAIOVDJ:~\$' with a cursor at the end.

```
apache@DESKTOP-VAIOVDJ:~$ gcc lab1.c -o lab1
apache@DESKTOP-VAIOVDJ:~$ ./lab1
Process ID: 443
Parent Process ID: 307
apache@DESKTOP-VAIOVDJ:~$ |
```

1.2 Write a program to run linux commands using system call.

Program:

```
#include <stdlib.h>

int main() {

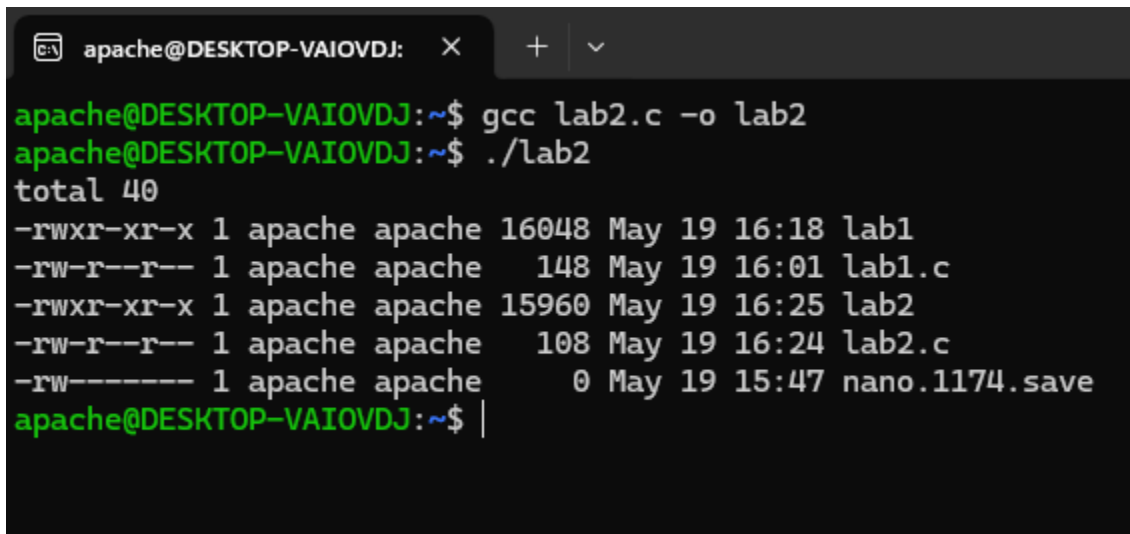
    int r_value;

    r_value = system("ls -l");

    return r_value;

}
```

Output:

A terminal window titled 'apache@DESKTOP-VAIOVDJ:' with a dark background and light green text. The user enters 'gcc lab2.c -o lab2' and then './lab2'. The output shows the result of the 'ls -l' command, listing files in the current directory with their permissions, sizes, dates, and names. The files listed are 'lab1', 'lab1.c', 'lab2', 'lab2.c', and 'nano.1174.save'.

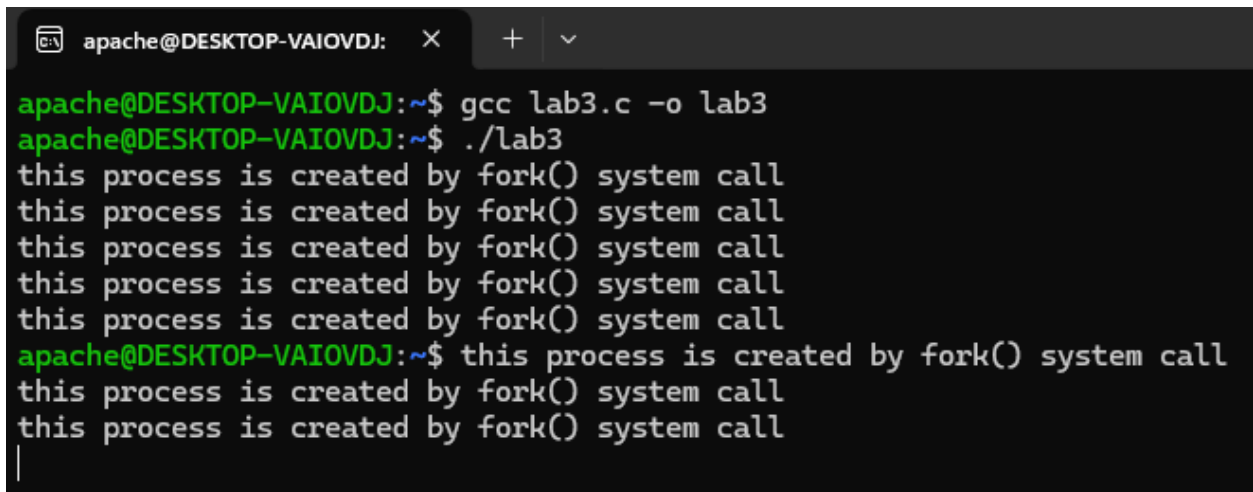
```
apache@DESKTOP-VAIOVDJ:~$ gcc lab2.c -o lab2
apache@DESKTOP-VAIOVDJ:~$ ./lab2
total 40
-rwxr-xr-x 1 apache apache 16048 May 19 16:18 lab1
-rw-r--r-- 1 apache apache  148 May 19 16:01 lab1.c
-rwxr-xr-x 1 apache apache 15960 May 19 16:25 lab2
-rw-r--r-- 1 apache apache  108 May 19 16:24 lab2.c
-rw----- 1 apache apache    0 May 19 15:47 nano.1174.save
apache@DESKTOP-VAIOVDJ:~$ |
```

1.3 Write a program to demonstrate fork().

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(){
    fork();
    fork();
    fork();
    printf("this process is created by fork() system call\n");
    return 0;
}
```

Output:



```
apache@DESKTOP-VAIOVDJ: ~$ gcc lab3.c -o lab3
apache@DESKTOP-VAIOVDJ: ~$ ./lab3
this process is created by fork() system call
this process is created by fork() system call
this process is created by fork() system call
this process is created by fork() system call
this process is created by fork() system call
apache@DESKTOP-VAIOVDJ: ~$ this process is created by fork() system call
this process is created by fork() system call
this process is created by fork() system call
```

1.4 Write a program that creates four processes using fork() system call.

Program:

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

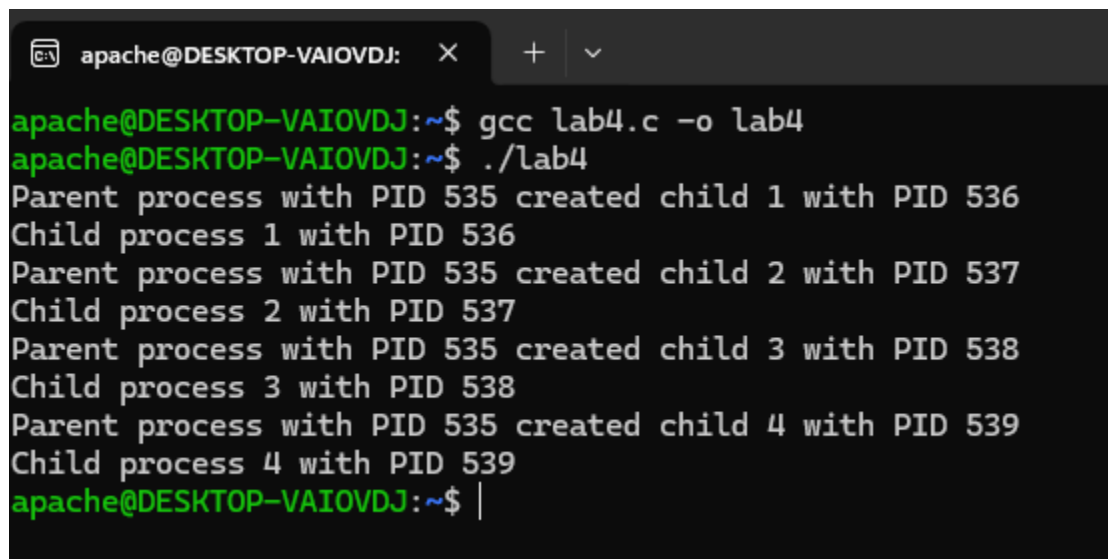
int main() {
    int i;
    pid_t child_pid;

    for (i = 0; i < 4; i++) {
        child_pid = fork();

        if (child_pid < 0) {
            perror("Fork failed");
            exit(EXIT_FAILURE);
        } else if (child_pid == 0) {
            printf("Child process %d with PID %d\n", i + 1, getpid());
            exit(EXIT_SUCCESS);
        } else {
            printf("Parent process with PID %d created child %d with PID %d\n",
                getpid(), i + 1, child_pid);
        }
    }

    return 0;
}
```

Output:



```
apache@DESKTOP-VAIOVDJ: x + v
apache@DESKTOP-VAIOVDJ:~$ gcc lab4.c -o lab4
apache@DESKTOP-VAIOVDJ:~$ ./lab4
Parent process with PID 535 created child 1 with PID 536
Child process 1 with PID 536
Parent process with PID 535 created child 2 with PID 537
Child process 2 with PID 537
Parent process with PID 535 created child 3 with PID 538
Child process 3 with PID 538
Parent process with PID 535 created child 4 with PID 539
Child process 4 with PID 539
apache@DESKTOP-VAIOVDJ:~$ |
```

1.5 List the uses of exec family

The exec family of functions in Unix-like operating systems (including Linux) is used to replace the current process image with a new process image. These functions are commonly used to execute other programs or commands from within a C or C++ program. The primary purpose of the exec functions is to load a new program into the current process's memory space and start its execution. The exec family includes functions like `execv`, `execvp`, `execve`, `execl`, and more, each with slightly different ways of specifying the program to run and the arguments it takes. The choice of which exec function to use depends on your specific requirements and how you want to pass arguments to the new program. Here are some common use cases and reasons for using the exec family of functions:

- **Running External Programs:** One of the most common uses is to run external programs or system commands from within a C/C++ program. You can use exec functions to launch programs like compilers, interpreters, system utilities, or any other executable.
- **Process Replacement:** You can use exec to replace the current process with a different program. This is often done in shell scripting when you want to execute a different program based on certain conditions.
- **Custom Shell Implementation:** Building a custom shell involves using exec functions to execute user-entered commands. This is how shells like Bash work under the hood. They use exec to execute commands entered by the user.
- **Setting Up Environment:** You can use exec to set up a specific environment for the child process. This allows you to modify environment variables, working directories, and other process attributes before launching a new program.
- **Running Shell Scripts:** You can use exec to run shell scripts (e.g., Bash, Python, Perl) from within a C/C++ program. This is useful for automation and scripting tasks.
- **Replacing a Running Daemon:** In daemon programming, you may want to replace the running daemon process with a new version without stopping the service. exec allows you to do this seamlessly.
- **Running Compiled Code Dynamically:** In some cases, you might want to compile and run code dynamically within your program. You can use exec to execute code generated at runtime.
- **Creating Child Processes:** The fork system call is often used in conjunction with exec to create child processes. After forking, the child process can use exec to load a different program, while the parent process continues with its own execution.

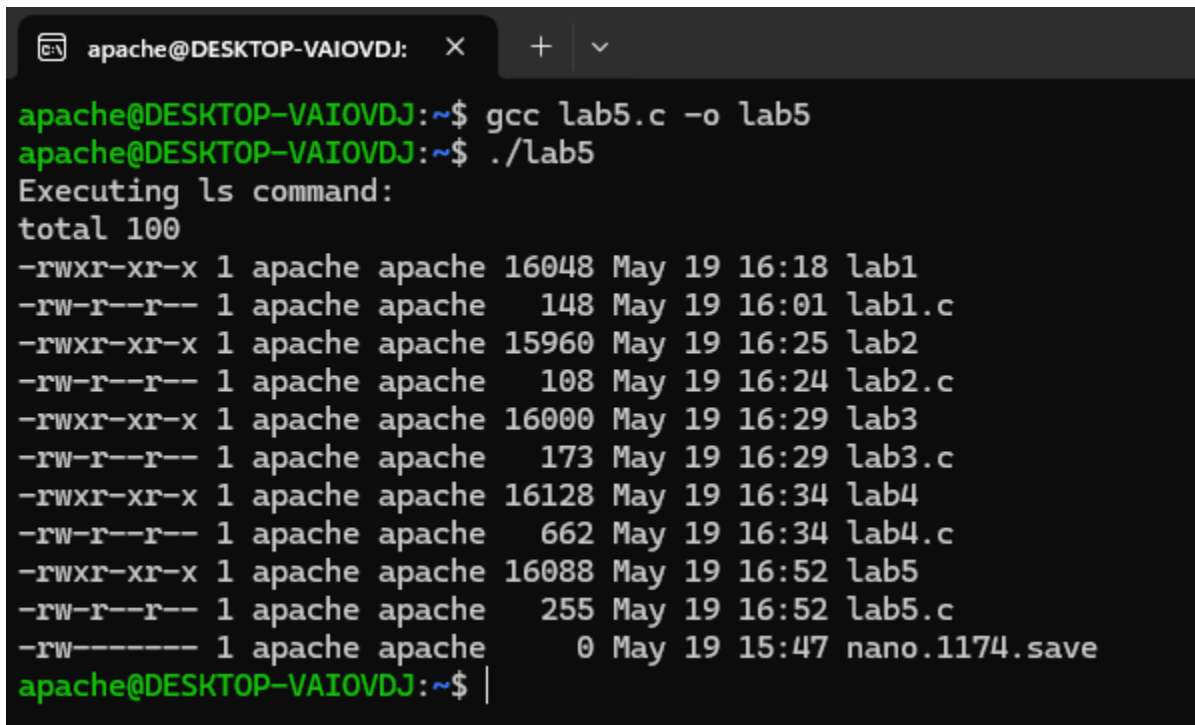
1.6 Write a program to use `execl` to run the `ls` command, which lists files and directories in the current directory.

Program:

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

int main() {
    printf("Executing ls command:\n");
    execl("/bin/ls", "ls", "-l", NULL);
    perror("execl");
    exit(EXIT_FAILURE);
}
```

Output:



```
apache@DESKTOP-VAIOVDJ: ~$ gcc lab5.c -o lab5
apache@DESKTOP-VAIOVDJ: ~$ ./lab5
Executing ls command:
total 100
-rwxr-xr-x 1 apache apache 16048 May 19 16:18 lab1
-rw-r--r-- 1 apache apache 148 May 19 16:01 lab1.c
-rwxr-xr-x 1 apache apache 15960 May 19 16:25 lab2
-rw-r--r-- 1 apache apache 108 May 19 16:24 lab2.c
-rwxr-xr-x 1 apache apache 16000 May 19 16:29 lab3
-rw-r--r-- 1 apache apache 173 May 19 16:29 lab3.c
-rwxr-xr-x 1 apache apache 16128 May 19 16:34 lab4
-rw-r--r-- 1 apache apache 662 May 19 16:34 lab4.c
-rwxr-xr-x 1 apache apache 16088 May 19 16:52 lab5
-rw-r--r-- 1 apache apache 255 May 19 16:52 lab5.c
-rw----- 1 apache apache 0 May 19 15:47 nano.1174.save
apache@DESKTOP-VAIOVDJ: ~$ |
```

1.7: Write a program that uses `execvp` to run a user-specified command.

Program:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#define MAX_ARGS 10

int main() {

    char input[100];

    char *args[MAX_ARGS + 1];

    int i = 0;

    printf("Enter a command: ");

    if (fgets(input, sizeof(input), stdin) == NULL) {

        perror("fgets failed");

        exit(EXIT_FAILURE);

    }

    size_t len = strlen(input);

    if (len > 0 && input[len - 1] == '\n') {

        input[len - 1] = '\0';

    }

    char *token = strtok(input, " ");

    while (token != NULL && i < MAX_ARGS) {

        args[i++] = token;

        token = strtok(NULL, " ");

    }

    args[i] = NULL;
```

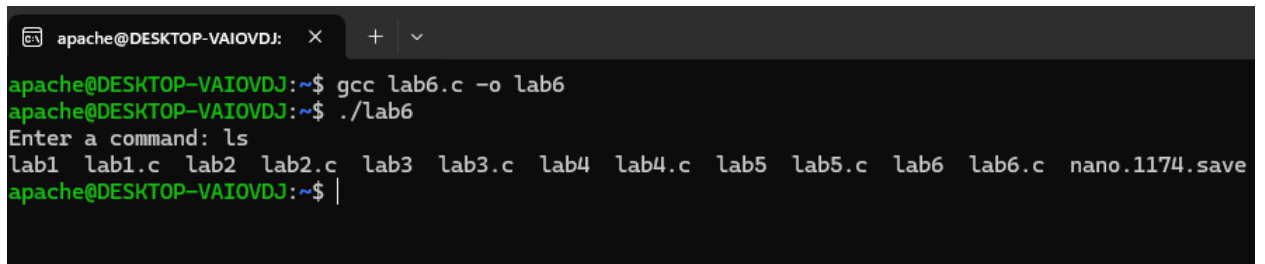


```
    execvp(args[0], args);

    perror("execvp");

    exit(EXIT_FAILURE);
}
```

Output:

A terminal window with a dark background and light green text. The window title is 'apache@DESKTOP-VAIOVDJ:'. The terminal shows the following commands and output:

```
apache@DESKTOP-VAIOVDJ:~$ gcc lab6.c -o lab6
apache@DESKTOP-VAIOVDJ:~$ ./lab6
Enter a command: ls
lab1 lab1.c lab2 lab2.c lab3 lab3.c lab4 lab4.c lab5 lab5.c lab6 lab6.c nano.1174.save
apache@DESKTOP-VAIOVDJ:~$ |
```

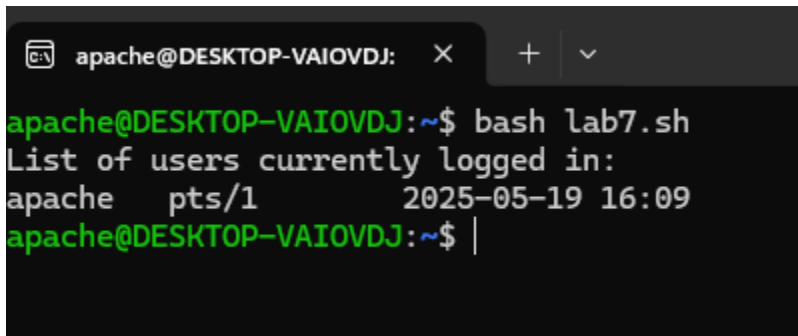
LAB 2: SHELL SCRIPTING

2.1: Write a shell script program to display list of users currently logged in.

Program:

```
#!/bin/bash  
echo "List of users currently logged in:"  
who
```

Output:

A terminal window titled 'apache@DESKTOP-VAIOVDJ:' with standard window controls. The prompt is 'apache@DESKTOP-VAIOVDJ:~\$'. The user enters 'bash lab7.sh'. The script outputs 'List of users currently logged in:' followed by the output of the 'who' command: 'apache pts/1 2025-05-19 16:09'. The prompt returns to 'apache@DESKTOP-VAIOVDJ:~\$' with a cursor.

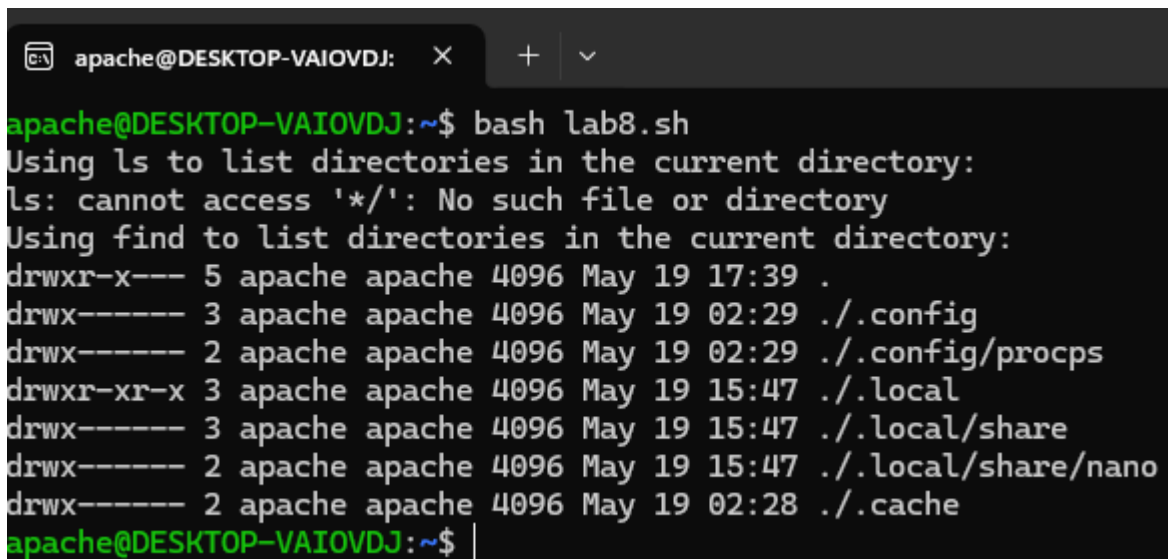
```
apache@DESKTOP-VAIOVDJ:~$ bash lab7.sh  
List of users currently logged in:  
apache pts/1 2025-05-19 16:09  
apache@DESKTOP-VAIOVDJ:~$ |
```

2.2: Write a shell script program to display the long list of directories.

Program:

```
#!/bin/bash
echo "Using ls to list directories in the current directory:"
ls -ld */
echo "Using find to list directories in the current directory:"
find . -type d -exec ls -ld {} \;
```

Output:

A screenshot of a terminal window with a dark background. The window title bar shows 'apache@DESKTOP-VAIOVDJ:'. The terminal content shows a user running 'bash lab8.sh'. The script first prints 'Using ls to list directories in the current directory:' followed by an error message 'ls: cannot access '*/': No such file or directory'. Then it prints 'Using find to list directories in the current directory:' followed by a long listing of directories. The listing shows permissions, file size, owner, group, and modification time for the current directory and several subdirectories like ./.config, ./.config/procps, ./.local, ./.local/share, and ./.cache.

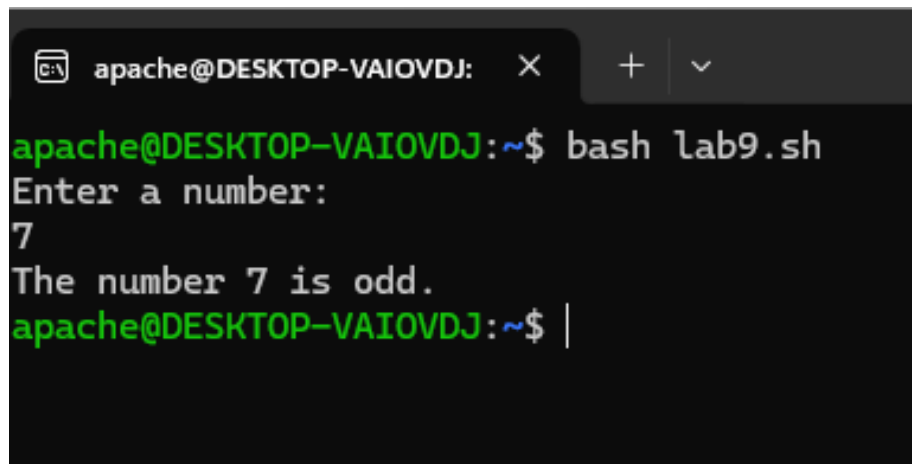
```
apache@DESKTOP-VAIOVDJ:~$ bash lab8.sh
Using ls to list directories in the current directory:
ls: cannot access '*/': No such file or directory
Using find to list directories in the current directory:
drwxr-x--- 5 apache apache 4096 May 19 17:39 .
drwx----- 3 apache apache 4096 May 19 02:29 ./.config
drwx----- 2 apache apache 4096 May 19 02:29 ./.config/procps
drwxr-xr-x 3 apache apache 4096 May 19 15:47 ./.local
drwx----- 3 apache apache 4096 May 19 15:47 ./.local/share
drwx----- 2 apache apache 4096 May 19 15:47 ./.local/share/nano
drwx----- 2 apache apache 4096 May 19 02:28 ./.cache
apache@DESKTOP-VAIOVDJ:~$ |
```

2.3: Write a shell script program to check whether the given number is even or odd.

Program:

```
#!/bin/bash
# Prompt the user to enter a number
echo "Enter a number: "
read number
# Check if the number is even or odd
if [ $((number % 2)) -eq 0 ]; then
    echo "The number $number is even."
else
    echo "The number $number is odd."
fi
```

Output:

A terminal window titled 'apache@DESKTOP-VAIOVDJ:' with a dark background. The prompt is 'apache@DESKTOP-VAIOVDJ:~\$'. The user enters 'bash lab9.sh'. The prompt changes to 'Enter a number:'. The user enters '7'. The output is 'The number 7 is odd.'. The prompt returns to 'apache@DESKTOP-VAIOVDJ:~\$' with a cursor at the end.

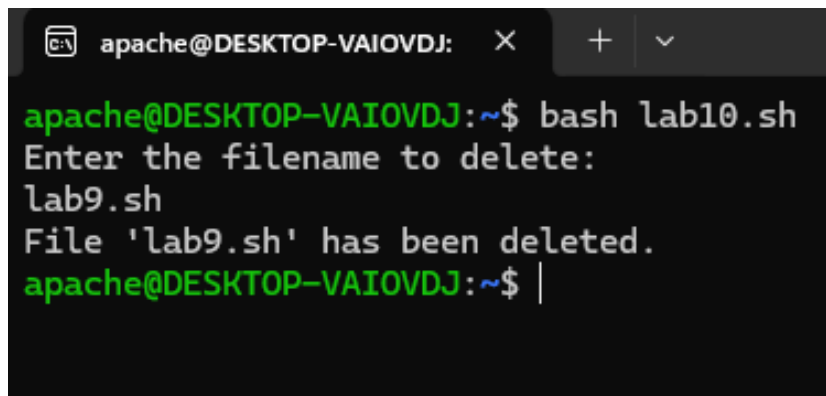
```
apache@DESKTOP-VAIOVDJ:~$ bash lab9.sh
Enter a number:
7
The number 7 is odd.
apache@DESKTOP-VAIOVDJ:~$ |
```

2.4: Write a shell script program to take filename as input and delete the file.

Program:

```
#!/bin/bash
echo "Enter the filename to delete: "
read filename
if [ -f "$filename" ]; then
    rm "$filename"
    echo "File '$filename' has been deleted."
else
    echo "File '$filename' does not exist."
fi
```

Output:

A terminal window titled 'apache@DESKTOP-VAIOVDJ:' with standard window controls. The prompt is 'apache@DESKTOP-VAIOVDJ:~\$'. The user enters 'bash lab10.sh'. The script prompts 'Enter the filename to delete:' and the user enters 'lab9.sh'. The script outputs 'File 'lab9.sh' has been deleted.' and returns to the prompt 'apache@DESKTOP-VAIOVDJ:~\$ |'.

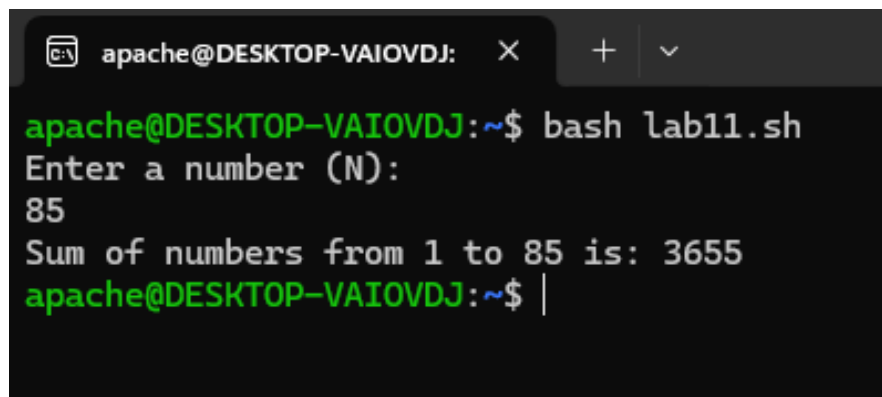
```
apache@DESKTOP-VAIOVDJ:~$ bash lab10.sh
Enter the filename to delete:
lab9.sh
File 'lab9.sh' has been deleted.
apache@DESKTOP-VAIOVDJ:~$ |
```

2.5: Write a shell script to add numbers from 1 to N.

Program:

```
#!/bin/bash
echo "Enter a number (N): "
read N
sum=0
for (( i=1; i<=N; i++ )); do
    sum=$((sum + i))
done
echo "Sum of numbers from 1 to $N is: $sum"
```

Output:

A terminal window titled 'apache@DESKTOP-VAIOVDJ:' with standard window controls. The prompt is 'apache@DESKTOP-VAIOVDJ:~\$'. The user enters 'bash lab11.sh'. The script prompts 'Enter a number (N):' and the user enters '85'. The script outputs 'Sum of numbers from 1 to 85 is: 3655'. The prompt returns to 'apache@DESKTOP-VAIOVDJ:~\$' with a cursor.

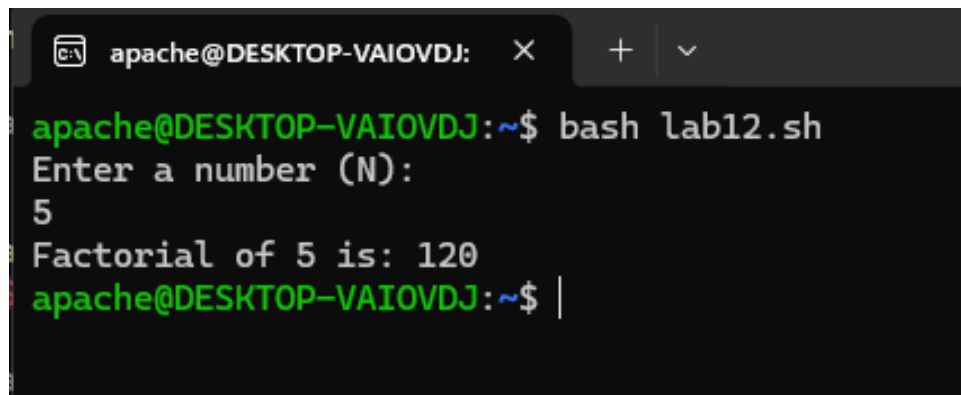
```
apache@DESKTOP-VAIOVDJ:~$ bash lab11.sh
Enter a number (N):
85
Sum of numbers from 1 to 85 is: 3655
apache@DESKTOP-VAIOVDJ:~$ |
```

2.6: Write a shell script to find the factorial of N.

Program:

```
#!/bin/bash
echo "Enter a number (N): "
read N
factorial=1
if [ $N -lt 0 ]; then
    echo "Factorial is not defined for negative numbers."
elif [ $N -eq 0 ]; then
    factorial=1
else
    for ((i = 1; i <= N; i++)); do
        factorial=$((factorial * i))
    done
fi
echo "Factorial of $N is: $factorial"
```

Output:

A terminal window titled 'apache@DESKTOP-VAIOVDJ:' with standard window controls. The prompt is 'apache@DESKTOP-VAIOVDJ:~\$'. The user enters 'bash lab12.sh'. The script outputs 'Enter a number (N):', followed by the user input '5'. The script then outputs 'Factorial of 5 is: 120'. The prompt returns to 'apache@DESKTOP-VAIOVDJ:~\$' with a cursor.

```
apache@DESKTOP-VAIOVDJ:~$ bash lab12.sh
Enter a number (N):
5
Factorial of 5 is: 120
apache@DESKTOP-VAIOVDJ:~$ |
```

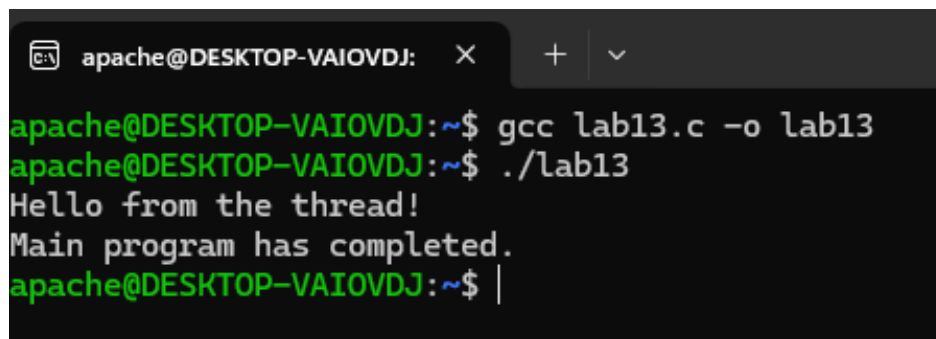
LAB 3: THREADS

3.1 Write a program for thread creation.

Program:

```
#include <stdio.h>
#include <pthread.h>
void *threadFunction(void *arg) {
    printf("Hello from the thread!\n");
    return NULL;
}
int main() {
    pthread_t thread_id;
    int thread_create_result;
    thread_create_result = pthread_create(&thread_id, NULL, threadFunction, NULL);
    if (thread_create_result != 0) {
        printf("Thread creation failed.\n");
        return 1;
    }
    pthread_join(thread_id, NULL);
    printf("Main program has completed.\n");
    return 0;
}
```

Output:

A terminal window with a dark background and light green text. The window title is 'apache@DESKTOP-VAIOVDJ:'. The terminal shows the following commands and output:

```
apache@DESKTOP-VAIOVDJ:~$ gcc lab13.c -o lab13
apache@DESKTOP-VAIOVDJ:~$ ./lab13
Hello from the thread!
Main program has completed.
apache@DESKTOP-VAIOVDJ:~$ |
```


3.2 Write a program using threads that prints sum of numbers up to given positive number n.

Program:

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

#define MAX_THREADS 4

struct ThreadParams {
    int start;
    int end;
};

int sum = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *calculateSum(void *arg) {
    struct ThreadParams *params = (struct ThreadParams *)arg;
    int localSum = 0;
    for (int i = params->start; i <= params->end; i++) {
        localSum += i;
    }

    pthread_mutex_lock(&mutex);
    sum += localSum;
    pthread_mutex_unlock(&mutex);

    pthread_exit(NULL);
}

int main() {
    int n;
    printf("Enter a positive number (n): ");
    scanf("%d", &n);

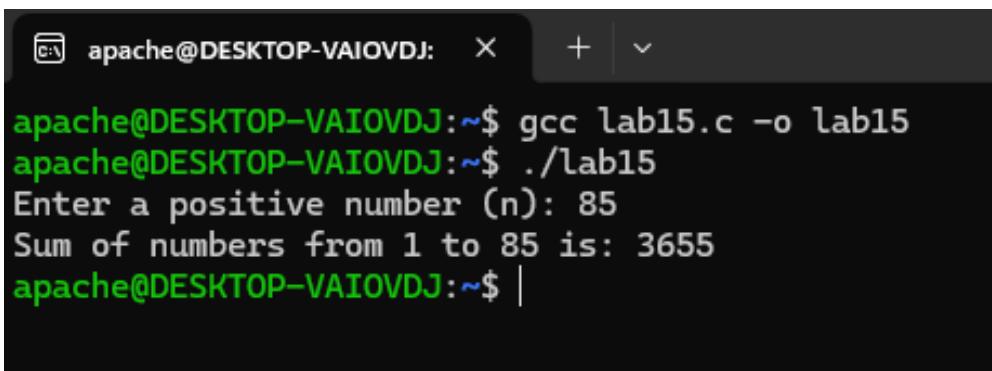
    if (n <= 0) {
```

```

        printf("Please enter a positive number.\n");
        return 1;
    }
    pthread_t threads[MAX_THREADS];
    struct ThreadParams params[MAX_THREADS];
    int chunkSize = n / MAX_THREADS;
    int remaining = n % MAX_THREADS;
    int start = 1;
    for (int i = 0; i < MAX_THREADS; i++) {
        params[i].start = start;
        params[i].end = start + chunkSize - 1;
        if (i == MAX_THREADS - 1) {
            params[i].end += remaining; // Add remaining numbers to the last thread
        }
        pthread_create(&threads[i], NULL, calculateSum, (void *)&params[i]);
        start = params[i].end + 1;
    }
    for (int i = 0; i < MAX_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Sum of numbers from 1 to %d is: %d\n", n, sum);
    pthread_mutex_destroy(&mutex);
    return 0;
}

```

Output:



```

apache@DESKTOP-VAIOVDJ: ~$ gcc lab15.c -o lab15
apache@DESKTOP-VAIOVDJ: ~$ ./lab15
Enter a positive number (n): 85
Sum of numbers from 1 to 85 is: 3655
apache@DESKTOP-VAIOVDJ: ~$ |

```

3.3: Write a program that has two threads, one reads a word from keyboard and another checks for valid word (you can use your own word list (at least 10) to check for validity).

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#define MAX_CLUBS 10
#define MAX_CLUB_NAME_LENGTH 30
const char *topClubs[MAX_CLUBS] = {
    "FC Barcelona", "Real Madrid", "Manchester United", "Liverpool",
    "Bayern Munich", "AC Milan", "Juventus",
    "Paris Saint-Germain", "Chelsea", "Borussia Dortmund"
};
int isTopClub(const char *club) {
    for (int i = 0; i < MAX_CLUBS; i++) {
        if (strcmp(club, topClubs[i]) == 0) {
            return 1;
        }
    }
    return 0;
}
void *readClub(void *arg) {
    char club[MAX_CLUB_NAME_LENGTH];
    while (1) {
        printf("Enter a football club (type 'exit' to terminate): ");
        fgets(club, MAX_CLUB_NAME_LENGTH, stdin);

        // Remove newline
        int len = strlen(club);
        if (len > 0 && club[len - 1] == '\n') {
            club[len - 1] = '\0';
        }
        if (strcmp(club, "exit") == 0) {
            break;
        }
    }
}
```

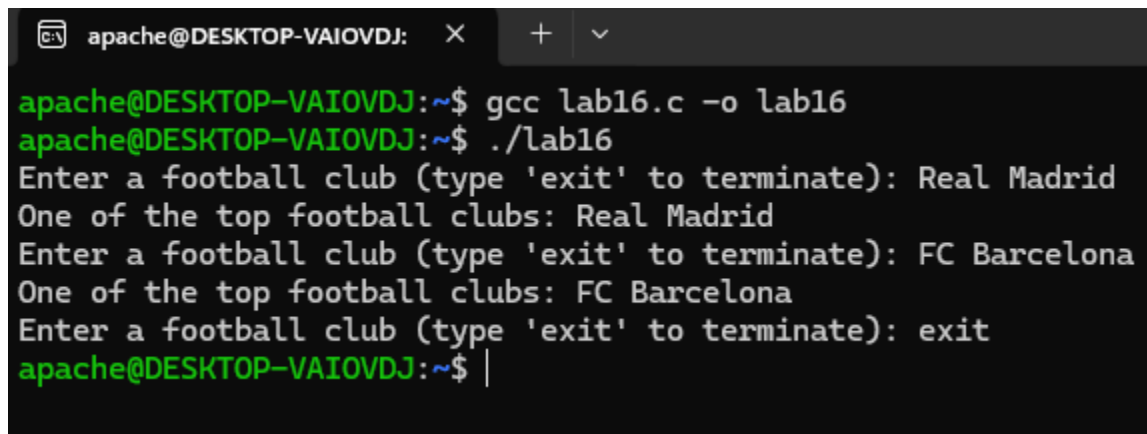
```

    }
    if (isTopClub(club)) {
        printf("One of the top football clubs: %s\n", club);
    } else {
        printf("Not in the list of top football clubs: %s\n", club);
    }
}
return NULL;
}

int main() {
    pthread_t thread_id;
    pthread_create(&thread_id, NULL, readClub, NULL);
    pthread_join(thread_id, NULL);
    return 0;
}

```

Output:



```

apache@DESKTOP-VAIOVDJ: ~$ gcc lab16.c -o lab16
apache@DESKTOP-VAIOVDJ: ~$ ./lab16
Enter a football club (type 'exit' to terminate): Real Madrid
One of the top football clubs: Real Madrid
Enter a football club (type 'exit' to terminate): FC Barcelona
One of the top football clubs: FC Barcelona
Enter a football club (type 'exit' to terminate): exit
apache@DESKTOP-VAIOVDJ: ~$ |

```

LAB 4: INTERPROCESS COMMUNICATION

4.1: Write a program to demonstrate the solution (strict alternation) for critical region problem.

Program:

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

#define NUM_ITERATIONS 20
int turn = 1; // 1 = thread1's turn, 0 = thread2's turn
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *thread1f(void *arg) {
    for (int i = 0; i < NUM_ITERATIONS; i++) {
        pthread_mutex_lock(&mutex);
        while (turn != 1) {
            pthread_cond_wait(&cond, &mutex);
        }
        fputc('b', stderr);
        turn = 0;
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

void *thread2f(void *arg) {
    for (int i = 0; i < NUM_ITERATIONS; i++) {
        pthread_mutex_lock(&mutex);
        while (turn != 0) {
            pthread_cond_wait(&cond, &mutex);
        }
        fputc('a', stderr);
        turn = 1;
        pthread_cond_signal(&cond);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Output:

[illegible]

4.2 Modify the above program in 4.1 to demonstrate lock variables solution and comment the deficiencies of this solution.

Program:

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

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *thread1f(void *arg);
void *thread2f(void *arg);

int main() {
    pthread_t thid1, thid2;

    pthread_create(&thid1, NULL, thread1f, NULL);
    pthread_create(&thid2, NULL, thread2f, NULL);

    pthread_join(thid1, NULL);
    pthread_join(thid2, NULL);

    pthread_mutex_destroy(&mutex);
    return 0;
}

void *thread1f(void *arg) {
    for (int i = 0; i < 20; i++) {
        pthread_mutex_lock(&mutex);
        fputc('b', stderr);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

void *thread2f(void *arg) {
    for (int i = 0; i < 20; i++) {
        pthread_mutex_lock(&mutex);
        fputc('a', stderr);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

Output:

```
apache@DESKTOP-VAIOVDJ: ~$ gcc lab18.c -o lab18
apache@DESKTOP-VAIOVDJ: ~$ ./lab18
bbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaapache@DESKTOP-VAIOVDJ: ~$ |
```

Deficiencies:

- **Efficiency:** Mutex locks provide a more efficient solution compared to busy waiting. In the previous version, the threads were constantly checking the value of turn, which could lead to excessive CPU usage. With mutex locks, a thread is put to sleep when it cannot acquire the lock, and it's awakened when the lock becomes available.
- **Resource Management:** Mutex locks ensure that only one thread can access the critical section (printing 'a' or 'b' in this case) at a time. This prevents race conditions and ensures orderly execution.
- **Synchronization:** Mutex locks provide a way for threads to synchronize their actions and avoid conflicts. In the previous version, there was no synchronization mechanism, which could lead to unpredictable behavior.
- **Reduced CPU Usage:** Mutex locks significantly reduce CPU usage because threads do not need to continuously check for the availability of the resource. Instead, they block and wait for the lock to become available.

4.3: Write the problems in the solution of 4.1.

Problems:

The provided code implements strict alternation for the critical region problem using a shared variable turn to control which thread should execute. However, this code has a few problems:

- **Busy-Waiting:** The code uses busy-waiting (the while loop checking turn) which is inefficient and wastes CPU cycles.
- **Lack of Mutual Exclusion:** While it enforces strict alternation, it doesn't provide mutual exclusion, meaning both threads can access the critical section simultaneously.
- **Deadlock Risk:** There is a risk of deadlock if one of the threads gets blocked or delayed for some reason.
- **No Prioritization:** It doesn't prioritize which thread should execute first; it just enforces strict alternation.

LAB 5: SEMAPHORES

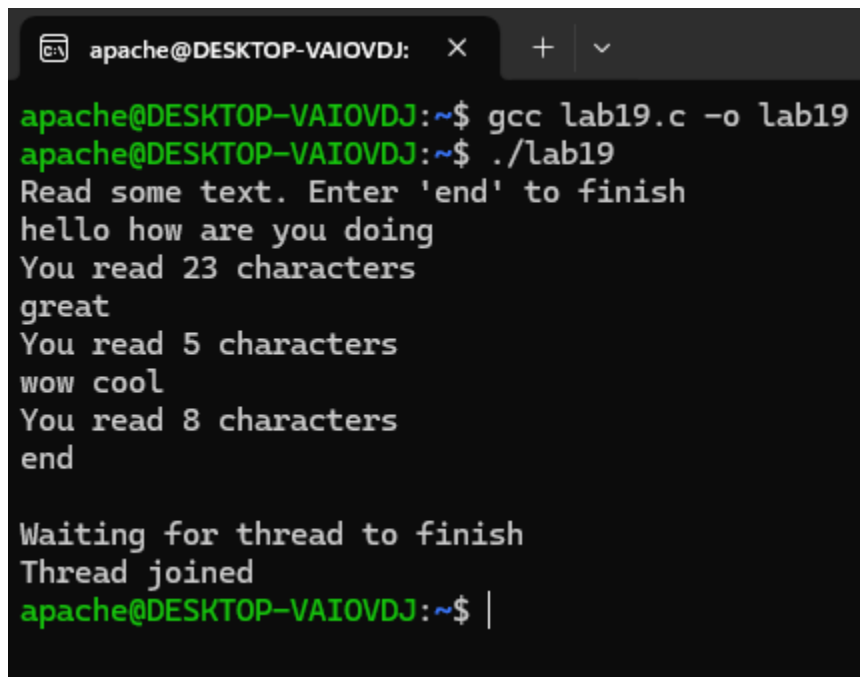
5.1: Write a program to demonstrate the use of semaphore.

Program:

```
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#define WORK_SIZE 1024
char work_area[WORK_SIZE];
sem_t bin_sem;
void *threadf(void *arg);
int main() {
    pthread_t thid;
    sem_init(&bin_sem, 0, 0);
    pthread_create(&thid, NULL, threadf, NULL);
    printf("Read some text. Enter 'end' to finish\n");
    while (1) {
        fgets(work_area, WORK_SIZE, stdin);
        sem_post(&bin_sem);
        if (strncmp("end", work_area, 3) == 0)
            break;
    }
    printf("\nWaiting for thread to finish\n");
    pthread_join(thid, NULL);
    printf("Thread joined\n");
    sem_destroy(&bin_sem);
    exit(EXIT_SUCCESS);
}
void *threadf(void *arg) {
    while (1) {
        sem_wait(&bin_sem);
```

```
        if (strcmp("end", work_area, 3) == 0)
            break;
        printf("You read %ld characters\n", strlen(work_area) - 1); // -1 to exclude newline
    }
    return NULL;
}
```

Output:



```
apache@DESKTOP-VAIOVDJ: ~$ gcc lab19.c -o lab19
apache@DESKTOP-VAIOVDJ: ~$ ./lab19
Read some text. Enter 'end' to finish
hello how are you doing
You read 23 characters
great
You read 5 characters
wow cool
You read 8 characters
end

Waiting for thread to finish
Thread joined
apache@DESKTOP-VAIOVDJ: ~$ |
```

5.2 Write the program 5.1 using mutexes.

Program:

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

#define WORK_SIZE 1024
char work_area[WORK_SIZE];
pthread_mutex_t mutex;
pthread_cond_t cond;
int data_available = 0;
int done = 0;
void *workerThread(void *arg);
int main() {
    pthread_t workerThreadId;
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&cond, NULL);
    pthread_create(&workerThreadId, NULL, workerThread, NULL);
    printf("Read some text. Enter 'end' to finish\n");
    while (1) {
        fgets(work_area, WORK_SIZE, stdin);
        pthread_mutex_lock(&mutex);
        data_available = 1;
        pthread_cond_signal(&cond);
        if (strncmp("end", work_area, 3) == 0) {
            done = 1; // Signal worker to exit
        }
        pthread_mutex_unlock(&mutex);
        if (done)
            break;
    }

    printf("\nWaiting for the worker thread to finish\n");
```

```

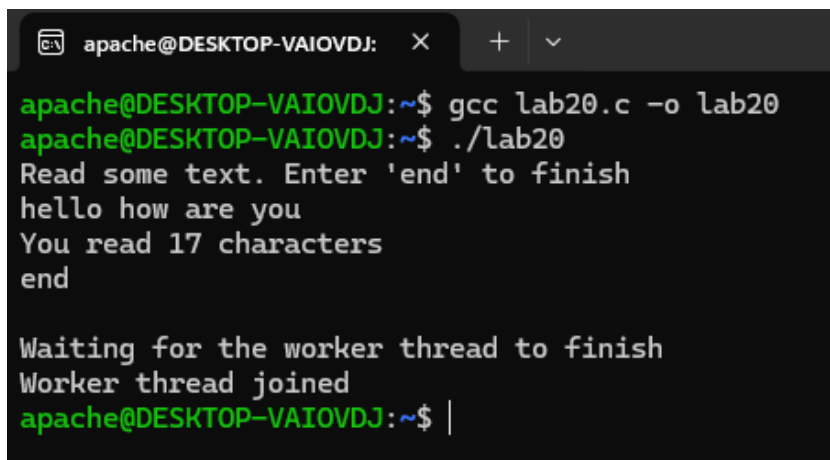
pthread_join(workerThreadId, NULL);
printf("Worker thread joined\n");
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&cond);

exit(EXIT_SUCCESS);
}

void *workerThread(void *arg) {
    while (1) {
        pthread_mutex_lock(&mutex);
        while (!data_available && !done) {
            pthread_cond_wait(&cond, &mutex);
        }
        if (done) {
            pthread_mutex_unlock(&mutex);
            break;
        }
        printf("You read %zu characters\n", strlen(work_area) - 1);
        data_available = 0;
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}
}

```

Output:



```

apache@DESKTOP-VAIOVDJ: ~$ gcc lab20.c -o lab20
apache@DESKTOP-VAIOVDJ: ~$ ./lab20
Read some text. Enter 'end' to finish
hello how are you
You read 17 characters
end

Waiting for the worker thread to finish
Worker thread joined
apache@DESKTOP-VAIOVDJ: ~$ |

```

5.3: Write a program to implement producer-consumer problem.

Program:

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

int mutex = 1;
int full = 0;
int empty = 10;
int x = 0;

void producer() {
    --mutex;
    ++full;
    --empty;
    x++;
    printf("\nProducer produces item %d", x);
    ++mutex;
}

void consumer() {
    --mutex;
    --full;
    ++empty;
    printf("\nConsumer consumes item %d", x);
    x--;
    ++mutex;
}

int main() {
    int choice;
    while (1) {
        printf("\n\n--- MENU ---\n");
        printf("1. Produce\n");
        printf("2. Consume\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
switch (choice) {
    case 1:
        if ((mutex == 1) && (empty != 0)) {
            producer();
        } else {
            printf("Buffer is full!");
        }
        break;
    case 2:
        if ((mutex == 1) && (full != 0)) {
            consumer();
        } else {
            printf("Buffer is empty!");
        }
        break;
    case 3:
        exit(0);
    default:
        printf("Invalid choice!");
}
return 0;
}
```

```
apache@DESKTOP-VAIOVDJ: ~$ gcc lab21.c -o lab21
apache@DESKTOP-VAIOVDJ: ~$ ./lab21

--- MENU ---
1. Produce
2. Consume
3. Exit
Enter your choice: 1

Producer produces item 1

--- MENU ---
1. Produce
2. Consume
3. Exit
Enter your choice: 2

Consumer consumes item 1

--- MENU ---
1. Produce
2. Consume
3. Exit
Enter your choice: 3
apache@DESKTOP-VAIOVDJ: ~$ |
```


5.4 Write a program to implement the counting semaphores using more than three threads.

Program:

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

#define MAX_COUNT 5
int count = 0;
pthread_mutex_t mutex;
pthread_cond_t condition;

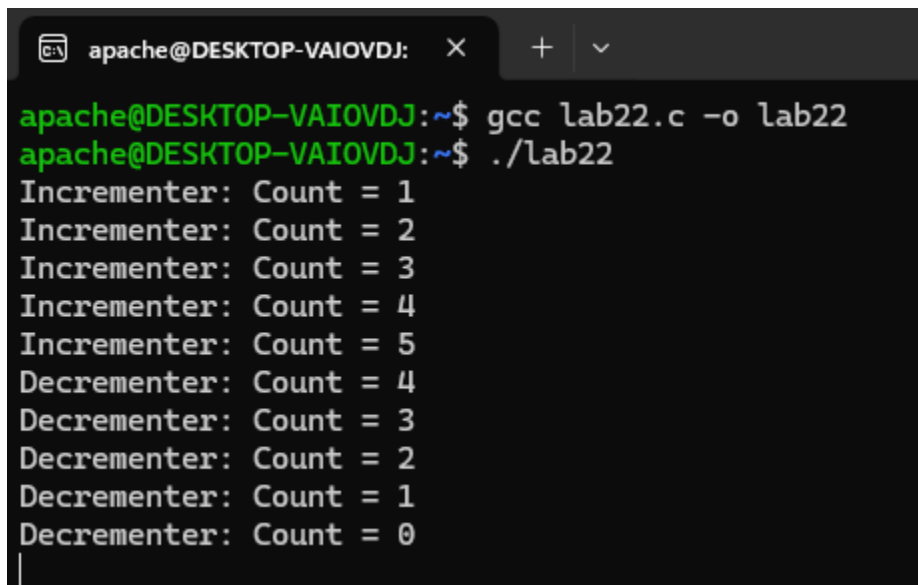
void *incrementer(void *arg) {
    for (int i = 0; i < MAX_COUNT; i++) {
        pthread_mutex_lock(&mutex);
        count++;
        printf("Incrementer: Count = %d\n", count);
        pthread_cond_signal(&condition);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

void *decrementer(void *arg) {
    for (int i = 0; i < MAX_COUNT; i++) {
        pthread_mutex_lock(&mutex);
        while (count == 0) {
            pthread_cond_wait(&condition, &mutex);
        }
        count--;
        printf("Decrementer: Count = %d\n", count);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

int main() {
    pthread_t incrementerThread, decrementerThread1, decrementerThread2;
```

```
pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&condition, NULL);
pthread_create(&incrementerThread, NULL, incrementer, NULL);
pthread_create(&decrementerThread1, NULL, decrementer, NULL);
pthread_create(&decrementerThread2, NULL, decrementer, NULL);
pthread_join(incrementerThread, NULL);
pthread_join(decrementerThread1, NULL);
pthread_join(decrementerThread2, NULL);
pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&condition);
return 0;
}
```

Output:



```
apache@DESKTOP-VAIOVDJ: ~$ gcc lab22.c -o lab22
apache@DESKTOP-VAIOVDJ: ~$ ./lab22
Incrementer: Count = 1
Incrementer: Count = 2
Incrementer: Count = 3
Incrementer: Count = 4
Incrementer: Count = 5
Decrementer: Count = 4
Decrementer: Count = 3
Decrementer: Count = 2
Decrementer: Count = 1
Decrementer: Count = 0
```