.31 . Check the following limits:
No. of clock ticks, Max. no. of child processes, Max. path length, Max. no. of characters in a file name, Max. no. of open files/ process

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

int main() {
    // Check and print the number of clock ticks
    printf("No. of clock ticks: %ld\n", sysconf(_SC_CLK_TCK));

    // Check and print the max number of child processes
    printf("Max. no. of child processes: %ld\n", sysconf(_SC_CHILD_MAX));

    // Check and print the max path length
    printf("Max. path length: %ld\n", pathconf("/", _PC_PATH_MAX));

    // Check and print the max number of characters in a file name
    printf("Max. no. of characters in a file name: %ld\n", pathconf("/", _PC_NAME_MAX));

    // Check and print the max number of open files per process
    printf("Max. no. of open files/process: %ld\n", sysconf(_SC_OPEN_MAX));

    return 0;
}
```

Output :

```
No. of clock ticks: 100
Max. no. of child processes: 14379
Max. path length: 4096
Max. no. of characters in a file name: 255
Max. no. of open files/process: 1024
```

2. a. Copy of a file using system calls.
   b. Output the contents of its Environment list

```
#include <iostream>
  #include <fstream>
  #include <cstdlib>
  #include <cstring>
  #include <sys/types.h>
  #include <sys/stat.h>
  #include <fcntl.h>
  #include <unistd.h>


int main() {
    const char* sourceFile = "source.txt";
    const char* destinationFile = "destination.txt";

    int source_fd = open(sourceFile, O_RDONLY);
    if (source_fd == -1) {
```

```
        perror("Error opening source file");
        return 1;
    }

    int dest_fd = open(destinationFile, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    if (dest_fd == -1) {
        perror("Error opening destination file");
        close(source_fd);
        return 1;
    }

    char buffer[4096];
    ssize_t bytes_read;
```

O_WRONLY: This flag indicates that the file should be opened in write-only mode.

O_CREAT: If the file does not exist, this flag ensures that it will be created.

S_IRUSR: This sets the file's read permission for the owner.

S_IWUSR: This sets the file's write permission for the owner.

```
    while ((bytes_read = read(source_fd, buffer, sizeof(buffer))) > 0) {
        ssize_t bytes_written = write(dest_fd, buffer, bytes_read);
        if (bytes_written != bytes_read) {
            perror("Error writing to destination file");
            close(source_fd);
            close(dest_fd);
            return 1;
        }
    }

    // Close file descriptors
    close(source_fd);
    close(dest_fd);

    std::cout << "File copied successfully!" << std::endl;

    return 0;
}
```

```
File copied successfully!
```

b. Output the contents of the Environment list:

```
#include <iostream>

extern char** environ;

int main() {
    char** env = environ;

    while (*env != nullptr) {
        std::cout << *env << std::endl;
        env++;
    }

    return 0;
}
```

```
SHELL=/bin/bash
WSL2_GUI_APPS_ENABLED=1
WSL_DISTRO_NAME=Ubuntu-20.04
NAME=LAPTOP-8OM5GDDE
PWD=/home/abhi/newdirectoryubun
LOGNAME=abhi
MOTD_SHOWN=update-motd
HOME=/home/abhi
LANG=C.UTF-8
WSL_INTEROP=/run/WSL/9_interop
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:s
37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=
;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:
tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:
cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;
:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=0
35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt
1;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=0
35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=
;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
WAYLAND_DISPLAY=wayland-0
LESSCLOSE=/usr/bin/lesspipe %s %s
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=abhi
DISPLAY=:0
SHLVL=1
XDG_RUNTIME_DIR=/mnt/wslg/runtime-dir
WSLENV=
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/usr/lib/wsl/lib:/mnt/c/Program Files/Java/jdk1.8.0_361/bin:
nt/c/Program Files/PuTTY/:/mnt/c/kubectl:/mnt/c/ProgramData/chocolatey/bin:/mnt/c/HashiCorp/Vagrant/bin:/mnt/c/ProgramData/chocolatey/lib/maven/apache-mav
-3.9.1/bin:/mnt/c/Program Files/Amazon/AWSCLIV2/:/mnt/c/Windows/System32/WindowsPowerShell/v1.0:/mnt/c/Program Files/CodeBlocks/MinGW/bin:/mnt/c/Users/abh
aj/AppData/Roaming/npm:/mnt/c/Users/abhiraj/AppData/Local/Programs/Python/Python311/Scripts/:/mnt/c/Users/abhiraj/AppData/Local/Programs/Python/Python311/
mnt/c/Users/abhiraj/AppData/Local/Programs/Python/Launcher/:/mnt/c/Program Files/Java/jre1.8.0_51/bin:/mnt/c/Users/abhiraj/AppData/Local/Programs/Hyper/re
urces/bin:/mnt/c/WINDOWS/system32:/mnt/c/Users/abhiraj/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Program Files/nodejs:/mnt/c/Program Files/Oracl
VirtualBox:/mnt/c/Program Files/Git/bin:/mnt/c/Program Files/Git/cmd:/mnt/d/mingw/clang64/bin:/mnt/d/mingw/mingw64/bin:/mnt/c/Users/abhiraj/AppData/Local/
ograms/Python/Python311/Scripts:/snap/bin
HOSTTYPE=x86_64
```

3. a. Emulate the UNIX ln command

   b. Create a child from parent process using fork() and counter counts till 5 in both processes and displays.

a.

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

int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " source_file target_file" << std::endl;
        return 1;
    }

    const char *source_file = argv[1];
    const char *target_file = argv[2];

    if (link(source_file, target_file) == 0) {
        std::cout << "Hard link created: " << target_file << " -> " << source_file << std::endl;
        return 0;
    } else {
        perror("Error creating hard link");
        return 2;
    }
}
```

```
abhi@LAPTOP-8OM5GDDE:~/newdirectoryubun$ g++ prog3a.cpp
abhi@LAPTOP-8OM5GDDE:~/newdirectoryubun$ ./a.out source.txt newlinkk
Hard link created: newlinkk -> source.txt
```

b.

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

int main() {
    pid_t child_pid;
```

```
    // Fork a child process
    child_pid = fork();

    if (child_pid == -1) {
        std::cerr << "Fork failed." << std::endl;
        return 1;
    }

    for (int i = 1; i <= 5; i++) {
        if (child_pid == 0) {
            // Child process
            std::cout << "Child Count: " << i << std::endl;
        } else {
            // Parent process
            std::cout << "Parent Count: " << i << std::endl;
        }
        sleep(1); // Sleep for 1 second
    }

    return 0;
}
```

```
Parent Count: 1
Child Count: 1
Child Count: 2
Parent Count: 2
Parent Count: 3
Child Count: 3
Parent Count: 4
Child Count: 4
Parent Count: 5
Child Count: 5
```

4 . Illustrate two processes communicating using shared memory

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <sys/wait.h>


// Define the shared memory key
#define SHM_KEY 1234
// Define the size of the shared memory segment
#define SHM_SIZE 1024

int main() {
    // Create a key for the shared memory segment
    key_t key = ftok(".", SHM_KEY);
    if (key == -1) {
        perror("ftok"); // Print an error message if ftok fails
```

key_t: This is a data type used for storing IPC (Interprocess Communication) keys.

ftok(".", SHM_KEY): The ftok() function generates a unique key based on:

"." — This specifies the current directory (it could also be a specific file path).

SHM_KEY — A predefined identifier (constant) to differentiate shared memory segments.

Why Use ftok()?
It ensures that processes access the same shared memory segment using a consistent key.

It converts a file path (or directory) and a project ID into a system-wide unique key.

This key is later used in functions like shmget() to create or access shared memory.

```
        exit(1);
    }

    // Create (or get) a shared memory segment
    int shmid = shmget(key, SHM_SIZE, IPC_CREAT | 0666);
    if (shmid == -1) {
        perror("shmget"); // Print an error message if shmget fails
        exit(1);
    }

    // Attach the shared memory segment to the process's address space
    char *shm_ptr = (char *)shmat(shmid, NULL, 0);
```

shmat() is a system call that attaches the shared memory segment to the process's address space.

```
    if (shm_ptr == (char *)(-1)) {
        perror("shmat"); // Print an error message if shmat fails
        exit(1);
    }

    // Parent process writes a message to shared memory
    std::string message = "Hello, shared memory!";
    std::strcpy(shm_ptr, message.c_str());

    // Fork a child process
    pid_t child_pid = fork();

    if (child_pid == -1) {
        perror("fork"); // Print an error message if fork fails
        exit(1);
    }

    if (child_pid == 0) {
        // Child process reads from shared memory and prints
        std::cout << "Child process reads: " << shm_ptr << std::endl;

        // Detach the shared memory segment from the child process
        if (shmdt(shm_ptr) == -1) {
            perror("shmdt"); // Print an error message if shmdt fails
            exit(1);
        }
    } else {
        // Parent process waits for the child to finish
        wait(NULL);

        // Detach the shared memory segment from the parent process
        if (shmdt(shm_ptr) == -1) {
            perror("shmdt"); // Print an error message if shmdt fails
            exit(1);
        }

        // Remove the shared memory segment
        if (shmctl(shmid, IPC_RMID, NULL) == -1) {
```

shmctl() is a system call used to control shared memory segments.

```
            perror("shmctl"); // Print an error message if shmctl fails
            exit(1);
        }
    }

    return 0;
}
```

5. Demonstrate producer and consumer problem using semaphores

A semaphore is a synchronization tool used in operating systems to manage access to shared resources. It helps prevent race conditions and ensures that multiple processes or threads do not interfere with each other when accessing shared memory or I/O devices.

Types of Semaphores:
Binary Semaphore (Mutex):

Can have only two values: 0 (locked) and 1 (unlocked).

Used for mutual exclusion, ensuring only one process accesses a resource at a time.

Counting Semaphore:

Allows multiple processes to access a resource up to a defined limit.

Useful when managing a pool of resources, like database connections or thread pools.

Operations on Semaphores:
Wait (P operation): Decreases the semaphore value. If it's 0, the process waits until another process releases the resource.

Signal (V operation): Increases the semaphore value, allowing waiting processes to proceed.

Example Usage:
Semaphores are commonly used in producer-consumer problems, where:

A producer adds items to a buffer.

A consumer removes items from the buffer.

Semaphores ensure that the buffer is neither overfilled nor emptied incorrectly.

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <vector>

#define MAX_BUFFER_SIZE 5
#define NUM_PRODUCERS 2
#define NUM_CONSUMERS 2

std::vector<int> buffer; // Shared buffer
sem_t mutex;            // Semaphore for mutual exclusion
sem_t empty;            // Semaphore for tracking empty slots in the buffer
sem_t full;           // Semaphore for tracking filled slots in the buffer

void* producer(void* arg) {
   int item = *((int*)arg);
   while (true) {
      sleep(1);

      sem_wait(&empty); // Wait for an empty slot in the buffer
      sem_wait(&mutex); // Enter critical section

      buffer.push_back(item); // Produce an item and add it to the buffer
      std::cout << "Produced: " << item << ", Buffer size: " << buffer.size() << std::endl;

      sem_post(&mutex); // Exit critical section
      sem_post(&full);  // Signal that a slot in the buffer is filled
   }
   return NULL;
}

// Consumer function
void* consumer(void* arg) {
   while (true) {
      sleep(1); // Simulate time to consume an item

      sem_wait(&full);  // Wait for a filled slot in the buffer
      sem_wait(&mutex); // Enter critical section

      int item = buffer.back(); // Consume an item from the buffer
      buffer.pop_back();
      std::cout << "Consumed: " << item << ", Buffer size: " << buffer.size() << std::endl;

      sem_post(&mutex); // Exit critical section
      sem_post(&empty); // Signal that a slot in the buffer is empty
   }
   return NULL;}

int main() {
   // Initialize semaphores
   sem_init(&mutex, 0, 1);        // Mutex semaphore
   sem_init(&empty, 0, MAX_BUFFER_SIZE); // Empty semaphore (buffer slots available)
   sem_init(&full, 0, 0);        // Full semaphore (buffer slots filled)

   // Create producer and consumer threads
   pthread_t producer_threads[NUM_PRODUCERS];
```

```cpp
    pthread_t consumer_threads[NUM_CONSUMERS];

    for (int i = 0; i < NUM_PRODUCERS; ++i) {
        int* item = new int(i);
        pthread_create(&producer_threads[i], NULL, producer, (void*)item);
    }

    for (int i = 0; i < NUM_CONSUMERS; ++i) {
        pthread_create(&consumer_threads[i], NULL, consumer, NULL);
    }

    // Join threads
    for (int i = 0; i < NUM_PRODUCERS; ++i) {
        pthread_join(producer_threads[i], NULL);
    }

    for (int i = 0; i < NUM_CONSUMERS; ++i) {
        pthread_join(consumer_threads[i], NULL);
    }

    // Destroy semaphores
    sem_destroy(&mutex);
    sem_destroy(&empty);
    sem_destroy(&full);

    return 0;
}
```

```
abhi@LAPTOP-8OM5GDDE:~/newdirectoryubun$ g++ -pthread prog5.cpp
abhi@LAPTOP-8OM5GDDE:~/newdirectoryubun$ ./a.out
Produced: 1, Buffer size: 1
Produced: 0, Buffer size: 2
Consumed: 0, Buffer size: 1
Consumed: 1, Buffer size: 0
Produced: 1, Buffer size: 1
Consumed: 1, Buffer size: 0
Produced: 0, Buffer size: 1
Consumed: 0, Buffer size: 0
Produced: 1, Buffer size: 1
Consumed: 1, Buffer size: 0
Produced: 0, Buffer size: 1
Consumed: 0, Buffer size: 0
Produced: 1, Buffer size: 1
Consumed: 1, Buffer size: 0
Produced: 0, Buffer size: 1
Consumed: 0, Buffer size: 0
Produced: 1, Buffer size: 1
Consumed: 1, Buffer size: 0
Produced: 0, Buffer size: 1
Consumed: 0, Buffer size: 0
Produced: 1, Buffer size: 1
Consumed: 1, Buffer size: 0
Produced: 0, Buffer size: 1
```

6 . Demonstrate round robin scheduling algorithm and calculates average waiting time and average turnaround time
//dont know weather the answer is right or wrong

```
// 6.ROUND ROBIN SCHEDULING

#include <iostream>
using namespace std;

int main() {
    int i, limit, total = 0, x, counter = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];
    float average_wait_time, average_turnaround_time;

    cout << "Enter Total Number of Processes: ";
    cin >> limit;
    x = limit;

    for (i = 0; i < limit; i++) {
        cout << "\nEnter Details of Process[" << i + 1 << "]\n";
        cout << "Arrival Time: ";
        cin >> arrival_time[i];
        cout << "Burst Time: ";
        cin >> burst_time[i];
        temp[i] = burst_time[i];
    }

    cout << "\nEnter Time Quantum: ";
    cin >> time_quantum;

    cout << "\nProcess ID\tBurst Time\tTurnaround Time\tWaiting Time\n";
    for (total = 0, i = 0; x != 0;) {
        if (temp[i] <= time_quantum && temp[i] > 0) {
            total += temp[i];
            temp[i] = 0;
            counter = 1;
        } else if (temp[i] > 0) {
            temp[i] -= time_quantum;
            total += time_quantum;
        }

        if (temp[i] == 0 && counter == 1) {
            x--;
            cout << "\nProcess[" << i + 1 << "]\t\t" << burst_time[i] << "\t\t" << total - arrival_time[i] << "\t\t\t" << total -
arrival_time[i] - burst_time[i];
            wait_time += total - arrival_time[i] - burst_time[i];
            turnaround_time += total - arrival_time[i];
            counter = 0;
        }

        if (i == limit - 1)
            i = 0;
        else if (arrival_time[i + 1] <= total)
            i++;
        else
            total++;
    }

    average_wait_time = wait_time * 1.0 / limit;
    average_turnaround_time = turnaround_time * 1.0 / limit;

    cout << "\n\nAverage Waiting Time: " << average_wait_time;
```

```
    cout << "\nAvg Turnaround Time: " << average_turnaround_time << endl;

    return 0;
}
```

Output :

```
Enter Total Number of Processes: 3
Enter Details of Process[1]
Arrival Time: 0
Burst Time: 2
Enter Details of Process[2]
Arrival Time: 1
Burst Time: 3
Enter Details of Process[3]
Arrival Time: 2
Burst Time: 4
Enter Time Quantum: 2
Process ID  Burst Time  Turnaround Time Waiting Time

Process[1]      2       2           0
Process[2]      3       6           3
Process[3]      4       7           3

Average Waiting Time: 2
Avg Turnaround Time: 5
```

7. Implement priority-based scheduling algorithm and calculates average waiting time and average turnaround time

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Process {
    int processID;
    int burstTime;
    int priority;
    int waitingTime;
    int turnaroundTime;
};

bool comparePriority(const Process &a, const Process &b) {
    return a.priority < b.priority;
}

int main() {
    int numProcesses;
    cout << "Enter the number of processes: ";
    cin >> numProcesses;

    vector<Process> processes(numProcesses);

    for (int i = 0; i < numProcesses; i++) {
```

```
        processes[i].processID = i + 1;
        cout << "Enter burst time for process " << i + 1 << ": ";
        cin >> processes[i].burstTime;
        cout << "Enter priority for process " << i + 1 << ": ";
        cin >> processes[i].priority;
    }

    sort(processes.begin(), processes.end(), comparePriority);

    processes[0].waitingTime = 0;
    processes[0].turnaroundTime = processes[0].burstTime;

    for (int i = 1; i < numProcesses; i++) {
        processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;
        processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;
    }

    double totalWaitingTime = 0;
    double totalTurnaroundTime = 0;

    for (const Process &p : processes) {
        totalWaitingTime += p.waitingTime;
        totalTurnaroundTime += p.turnaroundTime;
    }

    double averageWaitingTime = totalWaitingTime / numProcesses;
    double averageTurnaroundTime = totalTurnaroundTime / numProcesses;

    cout << "Process\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n";
    for (const Process &p : processes) {
        cout << p.processID << "\t\t" << p.burstTime << "\t\t" << p.priority << "\t\t" << p.waitingTime << "\t\t" <<
p.turnaroundTime << endl;
    }

    cout << "\nAverage Waiting Time: " << averageWaitingTime << endl;
    cout << "Average Turnaround Time: " << averageTurnaroundTime << endl;

    return 0;
}
```

```
Enter the number of processes: 2
Enter burst time for process 1: 1
Enter priority for process 1: 1
Enter burst time for process 2: 2
Enter priority for process 2: 2
Process Burst Time      Priority        Waiting Time    Turnaround Time
1               1               1               0               1
2               2               2               1               3

Average Waiting Time: 0.5
Average Turnaround Time: 2
```

8. Act as sender to send data in message queues and receiver that reads data from message queue.

```
Sender.cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

using namespace std;

// Define a structure for the message data
struct Message {
    long mtype;
    char mtext[100];
};

int main() {
    key_t key;
    int msgid;
    Message message;

    // Step 1: Create a key for the message queue
    key = ftok("/tmp", '1');
    if (key == -1) {
        perror("ftok");
        exit(1);
    }

    // Step 2: Create or open the message queue
    msgid = msgget(key, 0666 | IPC_CREAT);
    if (msgid == -1) {
        perror("msgget");
        exit(1);
    }

    // Sender: Send data to the message queue
    message.mtype = 1; // Message type (you can use different types for different purposes)
    strcpy(message.mtext, "Hello, this is a message from the sender!");

    // Step 3: Send the message to the queue
    if (msgsnd(msgid, &message, sizeof(message.mtext), 0) == -1) {
        perror("msgsnd");
        exit(1);
    }

    cout << "Data sent to message queue." << endl;

    return 0;
}
```

Output :

```
Data sent to message queue.
```

Receiver.cpp

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

using namespace std;

// Define a structure for the message data
```

```
struct Message {
    long mtype;
    char mtext[100];
};

int main() {
    key_t key;
    int msgid;
    Message message;

    // Step 1: Create a key for the message queue (use the same key as in the sender)
    key = ftok("/tmp", '1');
    if (key == -1) {
        perror("ftok");
        exit(1);
    }

    // Step 2: Create or open the message queue
    msgid = msgget(key, 0666 | IPC_CREAT);
    if (msgid == -1) {
        perror("msgget");
        exit(1);
    }

    // Receiver: Read data from the message queue
    // Step 3: Receive a message from the queue with message type 1
    if (msgrcv(msgid, &message, sizeof(message.mtext), 1, 0) == -1) {
        perror("msgrcv");
        exit(1);
    }

    cout << "Data received gmessage queue: " << message.mtext << endl;

    return 0;
}
```

Output :

```
Data received from message queue: Hello, this is a message from the sender!
```

9. Where a parent writes a message to pipe and child reads message from pipe

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

int main() {
    int pipe_fd[2]; // File descriptors for the pipe

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

    pid_t child_pid = fork(); // Fork a child process

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

```
    if (child_pid > 0) { // Parent process
        close(pipe_fd[0]); // Close the read end in the parent

        std::string message = "Hello from parent!";

        // Write the message to the pipe
        if (write(pipe_fd[1], message.c_str(), message.length()) == -1) {
            perror("Write to pipe failed");
            return 1;
        }

        close(pipe_fd[1]); // Close the write end in the parent
    } else { // Child process
        close(pipe_fd[1]); // Close the write end in the child

        char buffer[50];
        ssize_t bytes_read;

        // Read the message from the pipe
        bytes_read = read(pipe_fd[0], buffer, sizeof(buffer));

        if (bytes_read == -1) {
            perror("Read from pipe failed");
            return 1;
        }

        buffer[bytes_read] = '\0'; // Null-terminate the string

        std::cout << "Child process received message: " << buffer << std::endl;

        close(pipe_fd[0]); // Close the read end in the child
    }

    return 0;
}
```

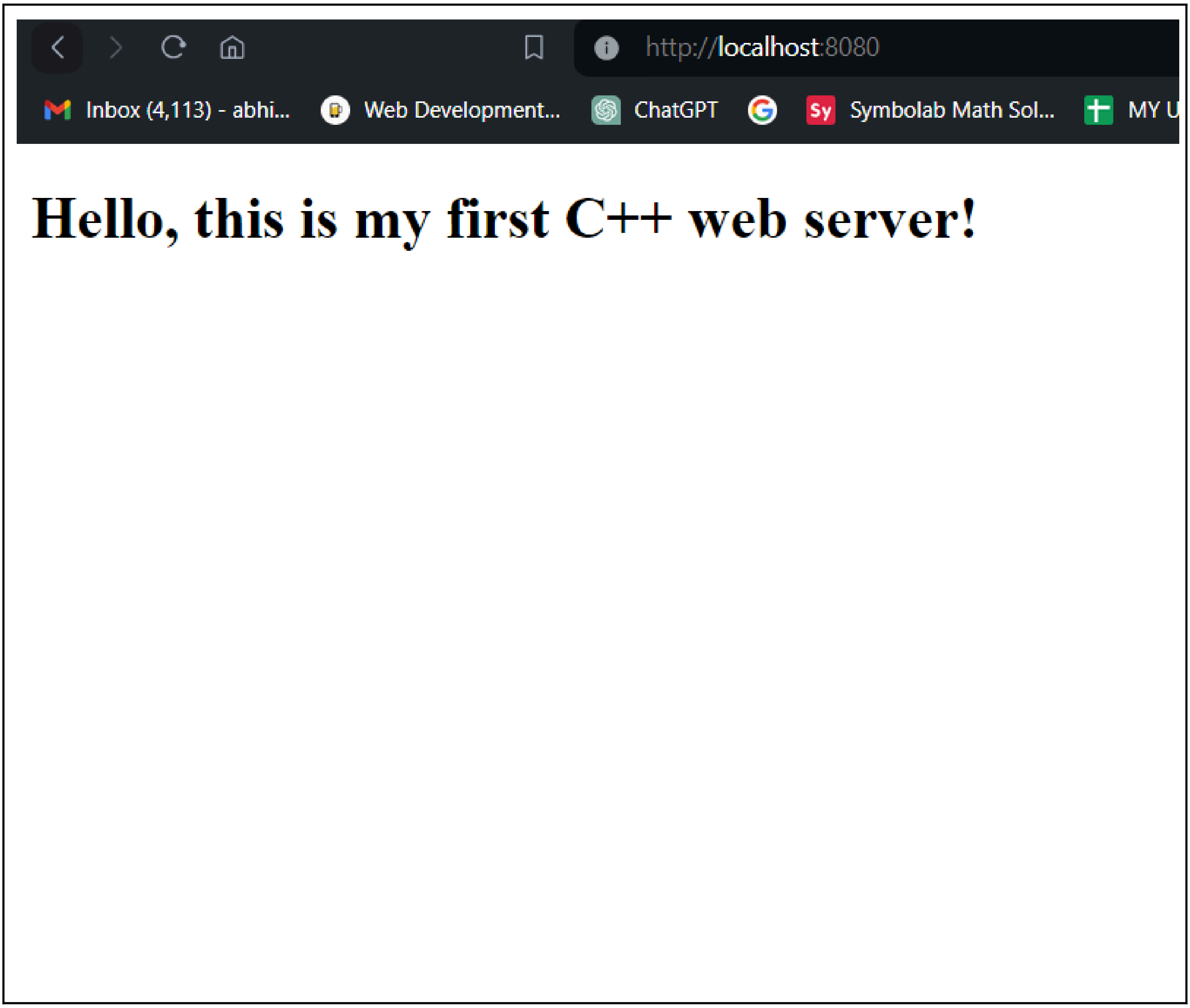
```
Child process received message: Hello from parent!
```

10. Demonstrate setting up a simple web server and host website on your own Linux computer

```
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>

const int PORT = 8080;

void handle_request(int client_socket) {
    const char* response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE
html><html><head><title>My C++ Web Server</title></head><body><h1>Hello, this is my first C++ web
server!</h1></body></html>";
    send(client_socket, response, strlen(response), 0);
    close(client_socket);
}

int main() {
    int server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
```

```
        std::cerr << "Error creating server socket" << std::endl;
        return -1;
    }

    sockaddr_in server_address{};
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(PORT);

    if (bind(server_socket, (struct sockaddr*)&server_address, sizeof(server_address)) == -1) {
        std::cerr << "Error binding to port " << PORT << std::endl;
        close(server_socket);
        return -1;
    }

    if (listen(server_socket, 10) == -1) {
        std::cerr << "Error listening on port " << PORT << std::endl;
        close(server_socket);
        return -1;
    }

    std::cout << "Server is listening on port " << PORT << std::endl;

    while (true) {
        sockaddr_in client_address{};
        socklen_t client_address_len = sizeof(client_address);

        int client_socket = accept(server_socket, (struct sockaddr*)&client_address, &client_address_len);
        if (client_socket == -1) {
            std::cerr << "Error accepting connection" << std::endl;
            continue;
        }

        handle_request(client_socket);
    }

    close(server_socket);

    return 0;
}
```

```
abhi@LAPTOP-8OM5GDDE:~/programs$ g++ prog10.cpp
abhi@LAPTOP-8OM5GDDE:~/programs$ ./a.out
Server listening on port 8080...
```



11. a. Create two threads using pthread, where both thread counts until 100 and joins later.
b. Create two threads using pthreads. Here, main thread creates 5 other threads for 5 times and each new thread print “Hello World” message with its thread number

a.

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

// Function that will be executed by each thread
void* countTo100(void* arg) {
    int item = *((int*)arg);

    for (int i = 1; i <= 100; ++i) {
        std::cout << "Thread " << item << ": Count " << i << std::endl;
    }

    pthread_exit(NULL);
}

int main() {
```

```
    const int numThreads = 2;
    pthread_t threads[numThreads];

    // Loop to create threads
    for (int i = 0; i < numThreads; ++i) {
  int* item = new int(i);
        int threadCreateStatus = pthread_create(&threads[i], NULL, countTo100, (void*)item);

        if (threadCreateStatus) {
            std::cerr << "Error creating thread: " << threadCreateStatus << std::endl;
            return -1;
        }
    }

    // Wait for both threads to finish
    for (int i = 0; i < numThreads; ++i) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

```
abhi@LAPTOP-8OM5GDDE:~/programs$ g++ prog11a.cpp -pthread
abhi@LAPTOP-8OM5GDDE:~/programs$ ./a.out
Thread 2: Count 1
Thread 2: Count 2
Thread 2: Count 3
Thread 2: Count 4
Thread 2: Count 5
Thread 2: Count 6
Thread 2: Count 7
Thread 2: Count 8
Thread 2: Count 9
Thread 2: Count 10
Thread 2: Count 11
```

b.

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

// Function that will be executed by each thread
void* printHello(void* threadNumber) {
    int* num = static_cast<int*>(threadNumber);
    std::cout << "Hello World from Thread " << *num << std::endl;
    pthread_exit(NULL);
}

int main() {
    // Number of threads to create
    const int numThreads = 5;

    // Loop to create threads
    for (int i = 1; i <= numThreads; ++i) {
        pthread_t thread;
```

```
        // Create a thread and pass the thread number as an argument
        int threadNumber = i;
        int threadCreateStatus = pthread_create(&thread, NULL, printHello, &threadNumber);

        if (threadCreateStatus) {
            std::cerr << "Error creating thread: " << threadCreateStatus << std::endl;
            return -1;
        }

        // Wait for the thread to finish
        pthread_join(thread, NULL);
    }

    return 0;
}
```

```
abhi@LAPTOP-8OM5GDDE:~/programs$ g++ prog11b.cpp -pthread
abhi@LAPTOP-8OM5GDDE:~/programs$ ./a.out
Hello World from Thread 1
Hello World from Thread 2
Hello World from Thread 3
Hello World from Thread 4
Hello World from Thread 5
```

12. Using Socket APIs establish communication between remote and local processes

```
//server.cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    // Step 1: Create a socket
    int serverSocket = socket(AF_INET, SOCK_STREAM, 0);

    // Check for errors
    if (serverSocket == -1) {
        std::cerr << "Error creating socket." << std::endl;
        return -1;
    }

    // Step 2: Bind the socket to an IP address and port
    sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(8080); // Port 8080

    // Bind the socket
    if (bind(serverSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress)) == -1) {
        std::cerr << "Error binding socket." << std::endl;
        close(serverSocket);
        return -1;
    }

    // Step 3: Listen for incoming connections
    if (listen(serverSocket, 5) == -1) {
        std::cerr << "Error listening for connections." << std::endl;
        close(serverSocket);
```

```
        return -1;
    }

    std::cout << "Server listening on port 8080..." << std::endl;

    // Step 4: Accept a connection
    sockaddr_in clientAddress;
    socklen_t clientAddrSize = sizeof(clientAddress);
    int clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddress, &clientAddrSize);

    // Check for errors
    if (clientSocket == -1) {
        std::cerr << "Error accepting connection." << std::endl;
        close(serverSocket);
        return -1;
    }

    std::cout << "Connection accepted. Waiting for data..." << std::endl;

    // Step 5: Receive data from the client
    char buffer[1024];
    ssize_t bytesRead = recv(clientSocket, buffer, sizeof(buffer), 0);

    // Check for errors
    if (bytesRead == -1) {
        std::cerr << "Error receiving data." << std::endl;
        close(serverSocket);
        close(clientSocket);
        return -1;
    }

    // Step 6: Print the received data
    std::cout << "Received data from client: " << buffer << std::endl;

    // Step 7: Close the sockets
    close(serverSocket);
    close(clientSocket);

    return 0;
}
```

```
abhi@LAPTOP-8OM5GDDE:~/programs$ g++ prog12server.cpp
abhi@LAPTOP-8OM5GDDE:~/programs$ ./a.out
Server listening on port 8080...
```

```
//client.cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    // Step 1: Create a socket
    int clientSocket = socket(AF_INET, SOCK_STREAM, 0);

    // Check for errors
    if (clientSocket == -1) {
        std::cerr << "Error creating socket." << std::endl;
        return -1;
    }
```

```
    // Step 2: Set up the server address and port
    sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(8080); // Port 8080

    // Convert IP address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serverAddress.sin_addr) <= 0) {
        std::cerr << "Invalid address/Address not supported." << std::endl;
        close(clientSocket);
        return -1;
    }

    // Step 3: Connect to the server
    if (connect(clientSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress)) == -1) {
        std::cerr << "Connection failed." << std::endl;
        close(clientSocket);
        return -1;
    }

    std::cout << "Connected to the server. Sending data..." << std::endl;

    // Step 4: Send data to the server
    const char* message = "Hello from the client!";
    if (send(clientSocket, message, strlen(message), 0) == -1) {
        std::cerr << "Error sending data." << std::endl;
        close(clientSocket);
        return -1;
    }

    // Step 5: Close the socket
    close(clientSocket);

    return 0;
}
```

```
abhi@LAPTOP-8OM5GDDE:~/programs$ g++ prog12client.cpp
abhi@LAPTOP-8OM5GDDE:~/programs$ ./a.out
Connected to the server. Sending data...
```

//server console

```
abhi@LAPTOP-8OM5GDDE:~/programs$ g++ prog12server.cpp
abhi@LAPTOP-8OM5GDDE:~/programs$ ./a.out
Server listening on port 8080...
Connection accepted. Waiting for data...
Received data from client: Hello from the client!
```