7. C++ program for implementing a priority-based scheduling algorithm and calculating average waiting time and average turnaround time in a Unix environment.

cpp

```
#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;
}
```

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

Reciever:

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mesg_buffer {
 long mesg_type;
 char mesg_text[100];
} message;
int main()
{
 key_t key;
 int msgid;

 key = ftok("progfile", 65);

 msgid = msgget(key, 0666 | IPC_CREAT);

 msgrcv(msgid, &message, sizeof(message), 1, 0);
 printf("Data Received is : %s \n", message.mesg_text);

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

Writer:

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10

struct mesg_buffer {
 long mesg_type;
 char mesg_text[100];
} message;
int main() {
 key_t key;
 int msgid;

 key = ftok("progfile", 65);

 msgid = msgget(key, 0666 | IPC_CREAT);
 message.mesg_type = 1;
 printf("Write Data : ");
 fgets(message.mesg_text,MAX,stdin);

 msgsnd(msgid, &message, sizeof(message), 0);

 printf("Data send is : %s \n", message.mesg_text);
 return 0;
}
```

9.Refer pic.

10.Skip:)

11.Using pthread_t..

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

void* printHello(void* threadID) {
 int* id = static_cast<int*>(threadID);
 std::cout << "Hello from thread: " << *id  << std::endl;
 pthread_exit(NULL);
}


int main(){
    const int numThreads = 5;

 for (int i = 1; i <= numThreads; i++) {
      pthread_t thread;
      pthread_create(&thread, NULL, printHello, &i);
      pthread_join(thread, NULL);
 }

 return 0;
}
```

12…..... C++ using socket APIs to establish communication between a remote and a local process.

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

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

    // 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);
    bind(serverSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress));

    // Listen for incoming connections
    listen(serverSocket, 5);
```

```
    // Accept a connection
    int clientSocket = accept(serverSocket, nullptr, nullptr);

    // Receive data from the client
    char buffer[1024];
    recv(clientSocket, buffer, sizeof(buffer), 0);

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

    // Close the sockets
    close(clientSocket);
    close(serverSocket);

    return 0;
}



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

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

    // Set up the server address and port
    sockaddr_in serverAddress;
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_port = htons(8080);
    inet_pton(AF_INET, "127.0.0.1", &serverAddress.sin_addr);

    // Connect to the server
    connect(clientSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress));

    // Send data to the server
    const char* message = "Hello from the client!";
    send(clientSocket, message, strlen(message), 0);

    // Close the socket
    close(clientSocket);

    return 0;
}
```