TP : SOCKET

1. General instructions

The purpose of this TP is to put into practice the main concepts (processes, threads, synchronization between processes / threads; inter-process communication; and memory allocation) seen in the class.

This TP is divided into two parts:

A. Tutorial about Socket

B. Problem to be implemented using the concepts seen in class about socket. **The problems must be implemented using the C language.**

2. Homework Expectations

Technical Report

You should write a technical report containing the answers of the tutorial part

Provide a detailed report on each programming assignment

1. Explain how your code fulfills the required functionality
2. Describe interesting parts of your implementation
3. Answer questions in the tutorial part
4. You can include capture screenshots to show what your code does on test input data

Teamwork

General discussion is acceptable, but your solutions must be your own work. Any occurrence of dishonesty, a zero grade for the assignment for all students involved

Homework is due to: **to Be announced**

- Delays due to circumstances beyond your control must be requested well in advance. Otherwise, for late submission, 20% penalty per day (zero point after 5 days)

Handwritten submission is to be avoided

- Write in word, LaTex or other editors

Projet submission requirements:

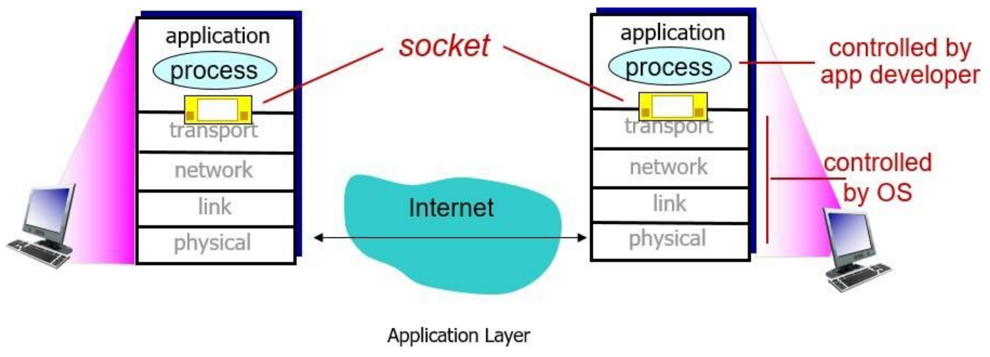
1. ZIP file named <last name1>-<last name2>.zip (last name of students in your team)
   a. E.g., sirot-bouchon.zip
2. All c and h files **must start with a commented section** with your name and a brief description of the purpose or functions
3. The ZIP file must contain a README file providing detailed instructions on compilation and execution of your code

Submit the ZIP at Campus Moodle or TEAM

Tutorial A

Inter-process communication: Socket

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.



- Two different types of sockets :
  - Stream (TCP)
  - Datagram (UDP)

Stages for server (API ou System call for the Server sice)

Socket creation:

int sockfd = socket(domain, type, protocol)

sockfd: socket descriptor, an integer (like a file-handle)

domain: AF_INET (IPv4 protocol) or AF_INET6 (IPv6 protocol)

type (communication type): TCP or UDP

protocol: Protocol value for Internet Protocol(IP), which is 0.

Bind:

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure).

Listen:

int listen(int sockfd, int backlog);

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection.

Accept:

int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

It extracts the first connection request on the queue of pending connections for the listening socket.

Stages for Client

Socket connection: Exactly same as that of server's socket creation

Connect:

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

The program below exchanges one hello message between server and client to demonstrate the client/server model.

1. Using the gedit editor, write the following program.
2. Compile and run this program
3. Explain how this program works

```
// Server side C program to demonstrate Socket

programming #include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 8080

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

{

        int server_fd, new_socket, valread;

        struct sockaddr_in address;

        int opt = 1;

        int addrlen = sizeof(address);

        char buffer[1024] = {0};
char *hello = "Hello from server";

// Creating socket file descriptor
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
        perror("socket failed");
        exit(EXIT_FAILURE);
}

address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( PORT );

// Bind : attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address))<0)
{
        perror("bind failed");
        exit(EXIT_FAILURE);
}
// Listen
if (listen(server_fd, 3) < 0)
{
        perror("listen");
        exit(EXIT_FAILURE);
}
// Accept
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen))<0)
{
        perror("accept");
        exit(EXIT_FAILURE);
}
// Read Message from Client
valread = read( new_socket , buffer, 1024);
```

```
        printf("%s\n",buffer );
        // Send Message to Client
        send(new_socket , hello , strlen(hello) , 0 );
        printf("Hello message sent\n");
        return 0;
}

// Client side C/C++ program to demonstrate Socket programming
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
{
        int sock = 0, valread;
        struct sockaddr_in serv_addr;
        char *hello = "Hello from client";
        char buffer[1024] = {0};
        // Creating socket file descriptor
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                printf("\n Socket creation error \n");
                return -1;
        }

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(PORT);

      // Convert IPv4 and IPv6 addresses from text to binary form
            if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
            {
            printf("\nInvalid address/ Address not supported \n"); return -1;

            }


// Connect

        if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
        {
                printf("\nConnection Failed \n");
                return -1;
        }
        // Send Message to Server
        send(sock , hello , strlen(hello) , 0 );
        printf("Hello message sent\n");
        // Read Message from Server
        valread = read( sock , buffer, 1024);
        printf("%s\n",buffer );
        return 0;
}
```

Compilation:

```
gcc client.c -o client
gcc server.c -o server
```

Part B: Problem

Problem 1) By using Socket, create a calculation server inspired by HP calculators with reverse Polish notation (we will limit ourselves to the support of the 4 basic operations, +, -, *, /). This type of calculator uses a stack to perform the calculations. The client connects to the server and then sends to it commands :

I. a number;

II. or an operator (+, -, *, /).

Your program should works as follows:

1. the client sends the command terminated by a newline character;
2. the server processes the command:
   a. If it is a number: the number is stacked,

   b. If it is an operator: The operands of the operator are unstacked, and then the result of the calculation (application of the operator to the operands) is stacked.
3. Finally, the result is returned to the client.

When the client is closed, the server can listen again the sock to accept another client. It is assumed for the implementation that basic stack operations are available in a library.