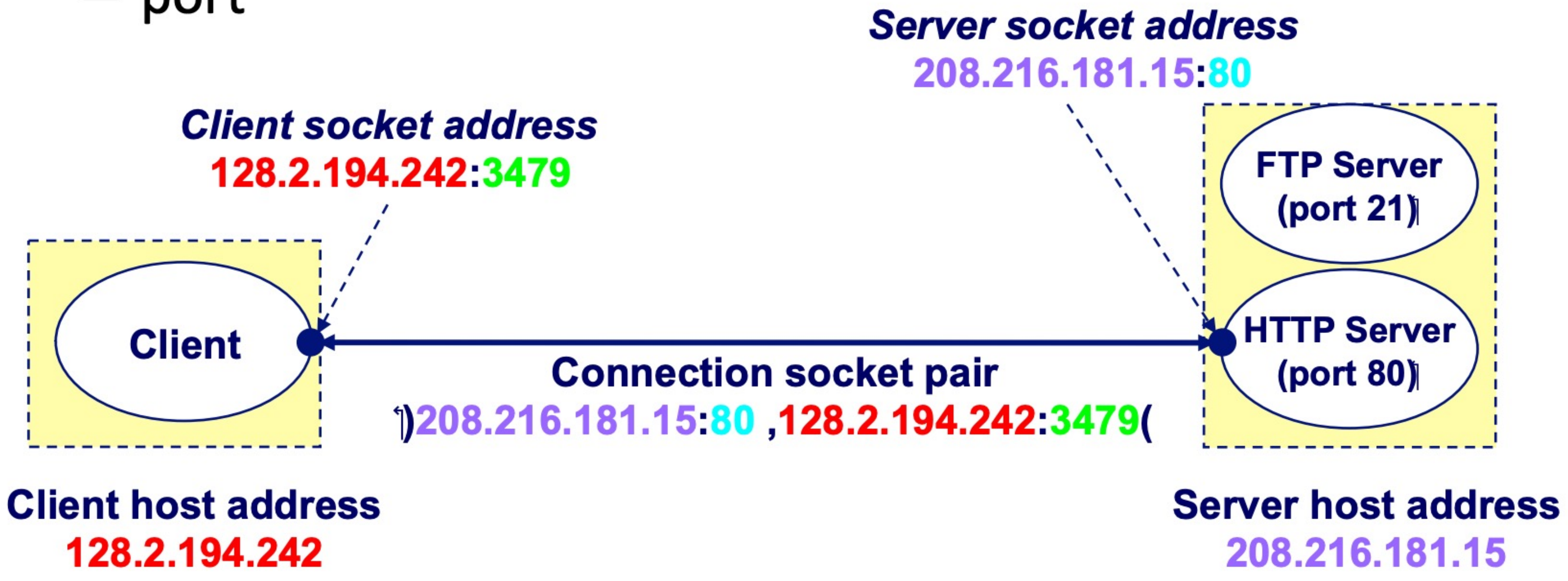
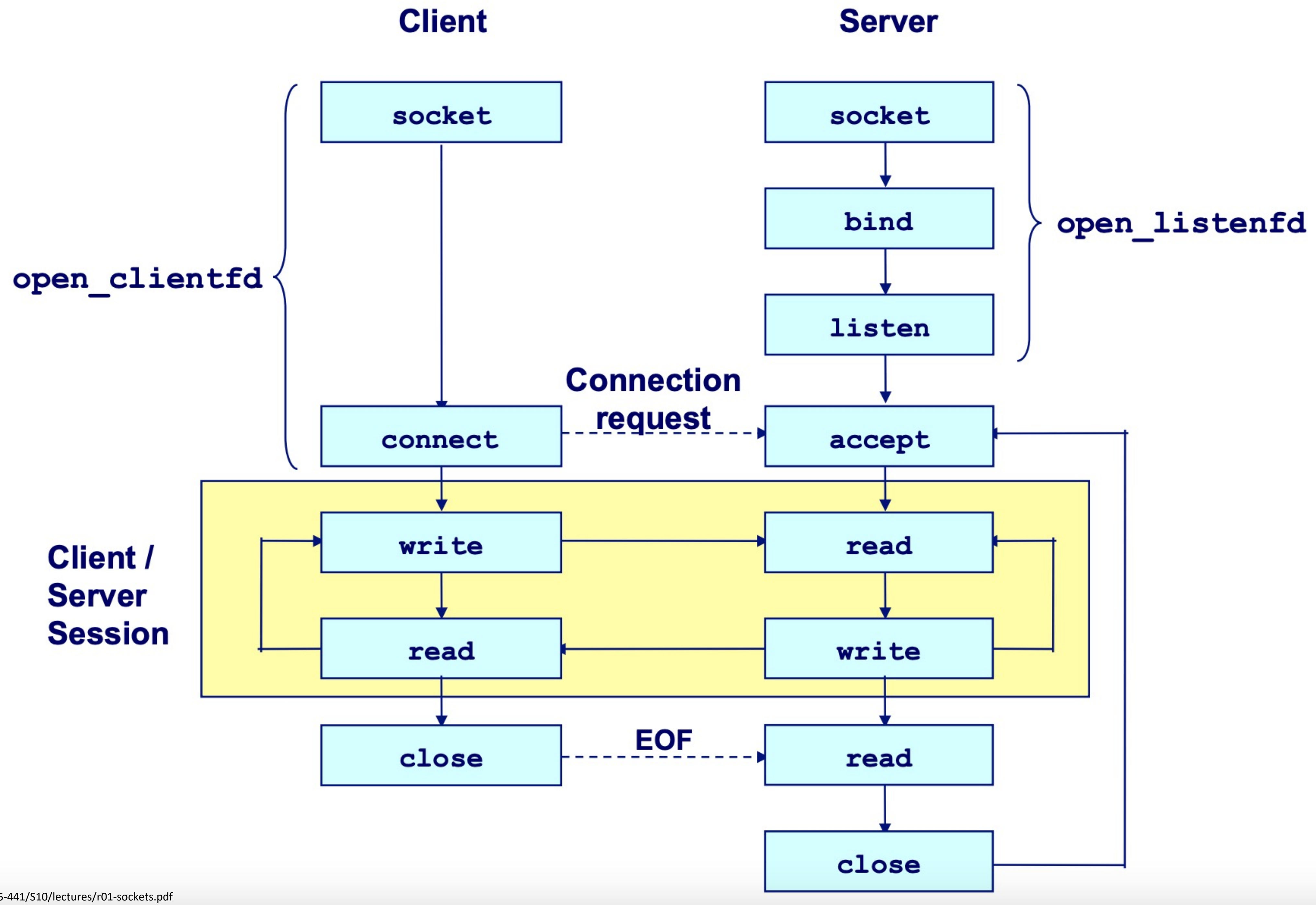
Identify the Destination

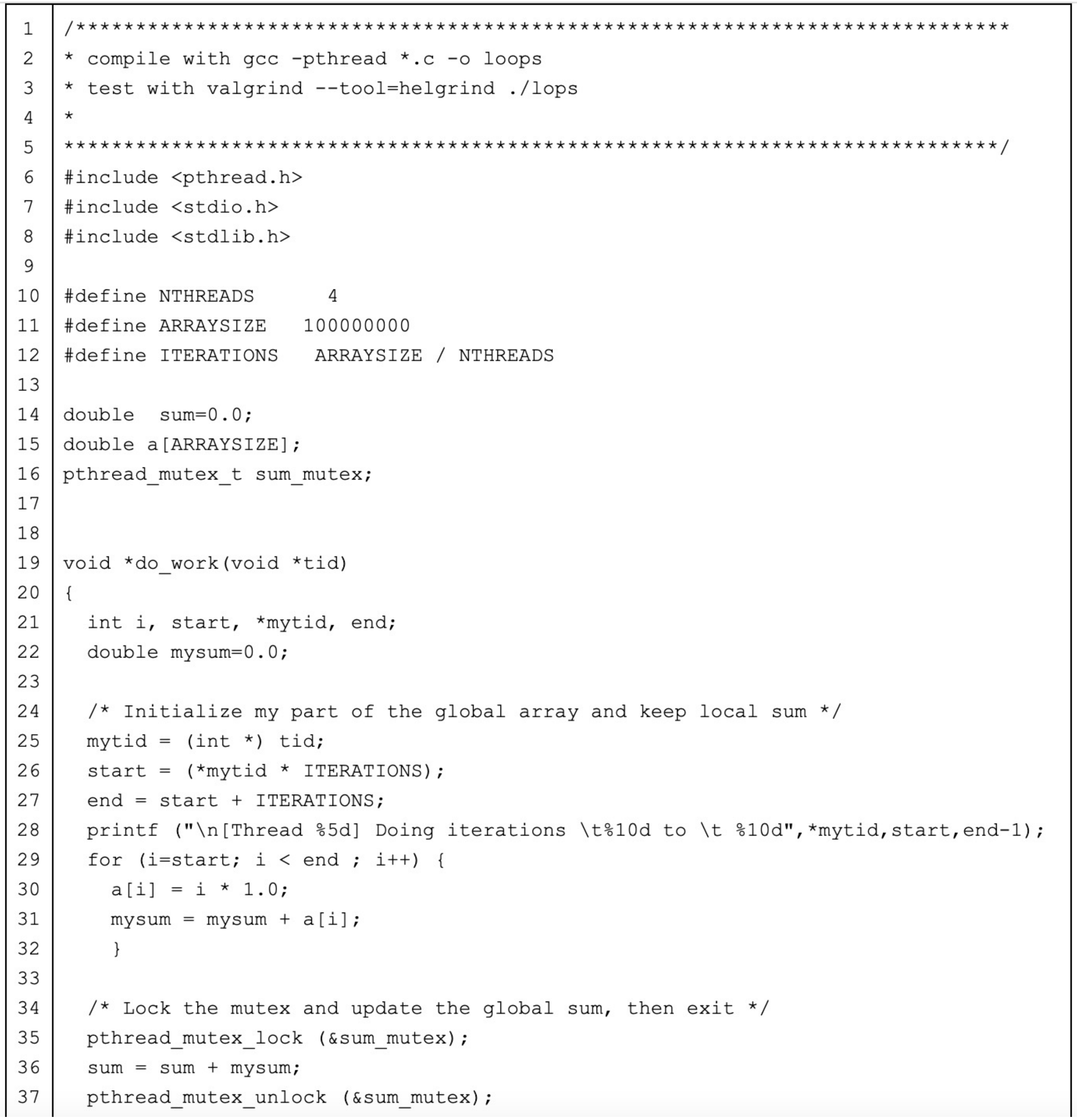
- Addressing
  - IP address
  - hostname (resolve to IP address via DNS)
- Multiplexing
  - port




Client
Server
socket
open_clientfd
connect
Connection request
socket
bind
listen
open_listenfd
accept
Client / Server Session
write
read
read
write
close
EOF
read
close

- See the example “Concurrent array processing (with threads and mutex)” from http://www.it.uc3m.es/pbasanta/asng/course_notes/c_threads_activities_multi_thread_loop_mutex_en.html

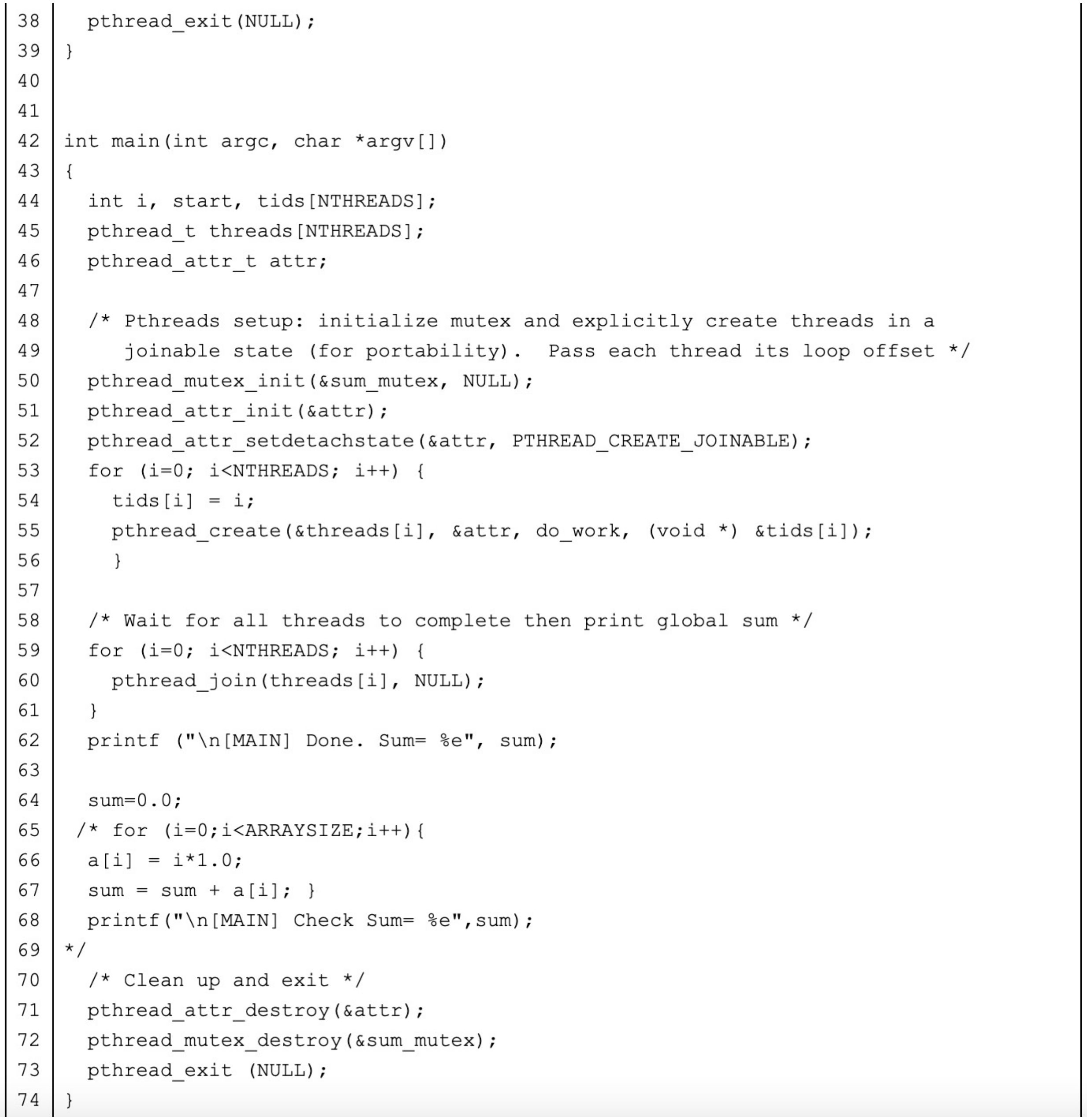
Part 1



```
/******************************************************************************
* compile with gcc -pthread *.c -o loops
* test with valgrind --tool=helgrind ./lops
*
******************************************************************************/
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NTHREADS      4
#define ARRAYSIZE   100000000
#define ITERATIONS   ARRAYSIZE / NTHREADS

double  sum=0.0;
double a[ARRAYSIZE];
pthread_mutex_t sum_mutex;


void *do_work(void *tid)
{
  int i, start, *mytid, end;
  double mysum=0.0;

  /* Initialize my part of the global array and keep local sum */
  mytid = (int *) tid;
  start = (*mytid * ITERATIONS);
  end = start + ITERATIONS;
  printf ("\n[Thread %5d] Doing iterations \t%10d to \t %10d",*mytid,start,end-1);
  for (i=start; i < end ; i++) {
    a[i] = i * 1.0;
    mysum = mysum + a[i];
    }

  /* Lock the mutex and update the global sum, then exit */
  pthread_mutex_lock (&sum_mutex);
  sum = sum + mysum;
  pthread_mutex_unlock (&sum_mutex);
```

- See the example “Concurrent array processing (with threads and mutex)” from http://www.it.uc3m.es/pbasanta/asng/course_notes/c_threads_activities_multi_thread_loop_mutex_en.html

Part 2

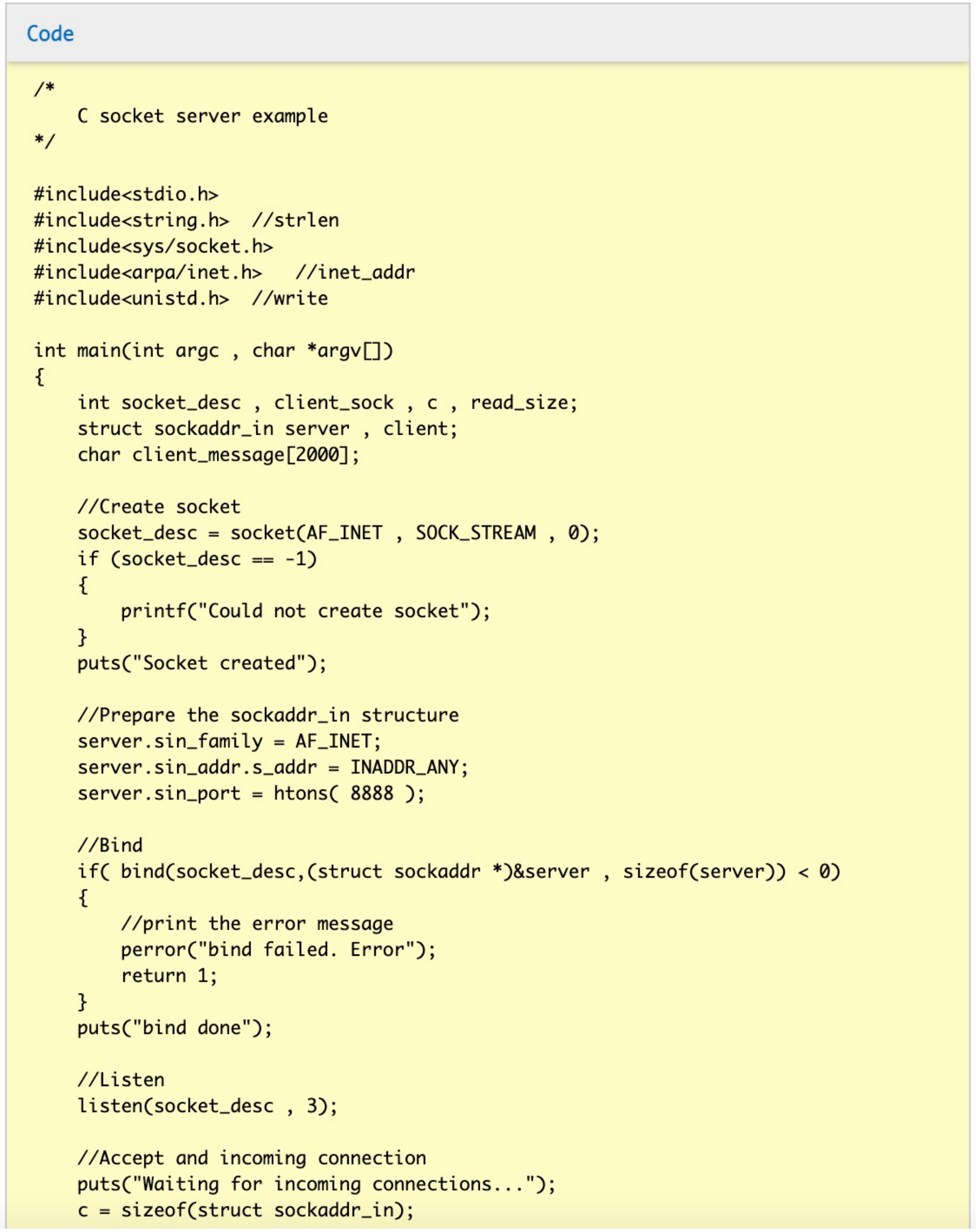


```c
  pthread_exit(NULL);
}


int main(int argc, char *argv[])
{
  int i, start, tids[NTHREADS];
  pthread_t threads[NTHREADS];
  pthread_attr_t attr;

  /* Pthreads setup: initialize mutex and explicitly create threads in a
     joinable state (for portability).  Pass each thread its loop offset */
  pthread_mutex_init(&sum_mutex, NULL);
  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
  for (i=0; i<NTHREADS; i++) {
    tids[i] = i;
    pthread_create(&threads[i], &attr, do_work, (void *) &tids[i]);
    }

  /* Wait for all threads to complete then print global sum */
  for (i=0; i<NTHREADS; i++) {
    pthread_join(threads[i], NULL);
  }
  printf ("\n[MAIN] Done. Sum= %e", sum);

  sum=0.0;
 /* for (i=0;i<ARRAYSIZE;i++){
  a[i] = i*1.0;
  sum = sum + a[i]; }
  printf("\n[MAIN] Check Sum= %e",sum);
*/
  /* Clean up and exit */
  pthread_attr_destroy(&attr);
  pthread_mutex_destroy(&sum_mutex);
  pthread_exit (NULL);
}
```

- See the example “How to Code a Server and Client in C with Sockets” from https://www.binarytides.com/server-client-example-c-sockets-linux/

Server

Part 1



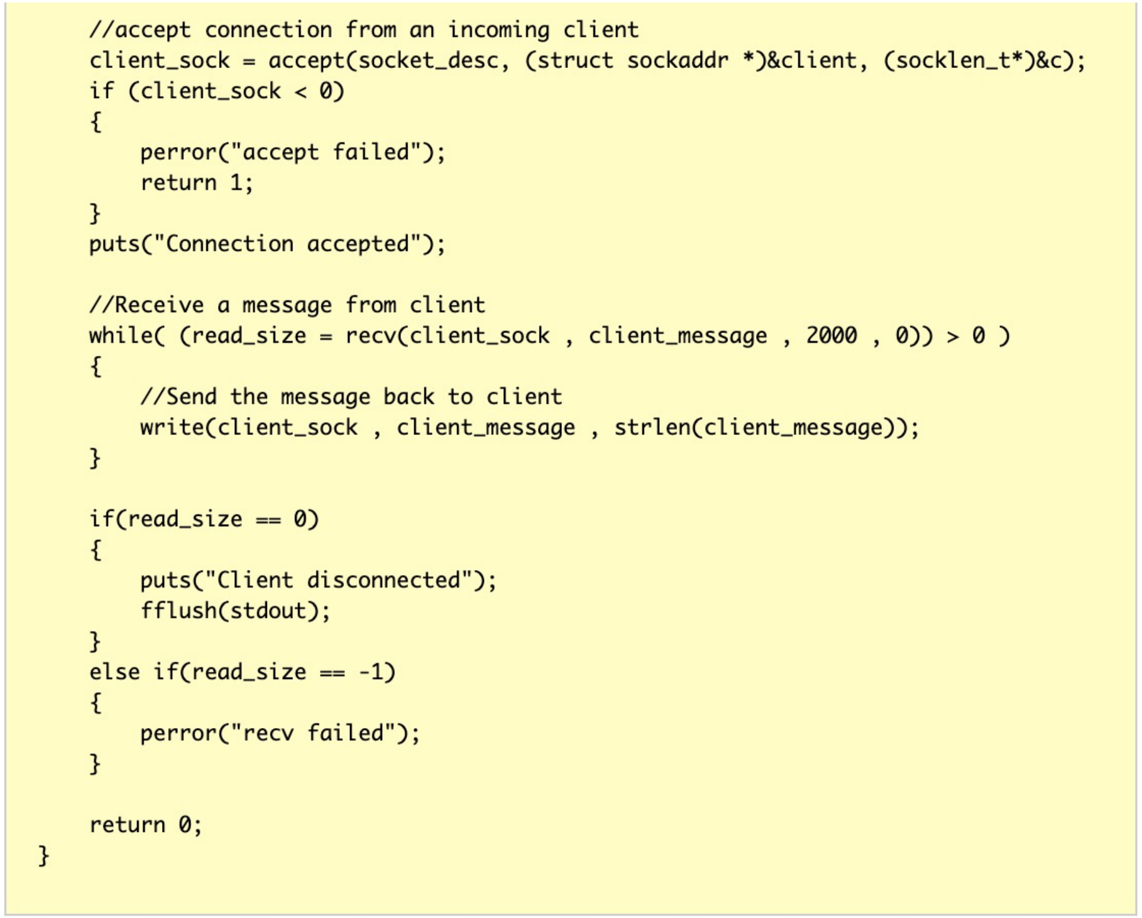
Code

```
/*
    C socket server example
*/

#include<stdio.h>
#include<string.h>  //strlen
#include<sys/socket.h>
#include<arpa/inet.h>   //inet_addr
#include<unistd.h>  //write

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        //print the error message
        perror("bind failed. Error");
        return 1;
    }
    puts("bind done");

    //Listen
    listen(socket_desc , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);
```

- See the example “How to Code a Server and Client in C with Sockets” from https://www.binarytides.com/server-client-example-c-sockets-linux/

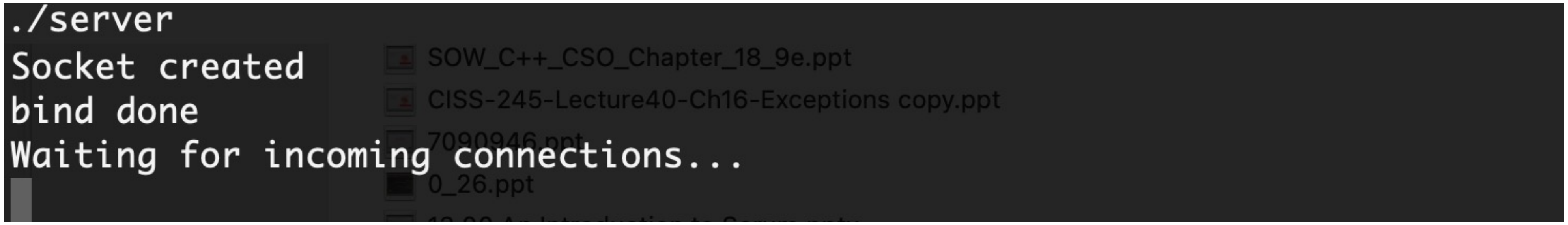
Server

Part 2



```
    //accept connection from an incoming client
    client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
    if (client_sock < 0)
    {
        perror("accept failed");
        return 1;
    }
    puts("Connection accepted");

    //Receive a message from client
    while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
    {
        //Send the message back to client
        write(client_sock , client_message , strlen(client_message));
    }

    if(read_size == 0)
    {
        puts("Client disconnected");
        fflush(stdout);
    }
    else if(read_size == -1)
    {
        perror("recv failed");
    }

    return 0;
}
```

Run it after compiling

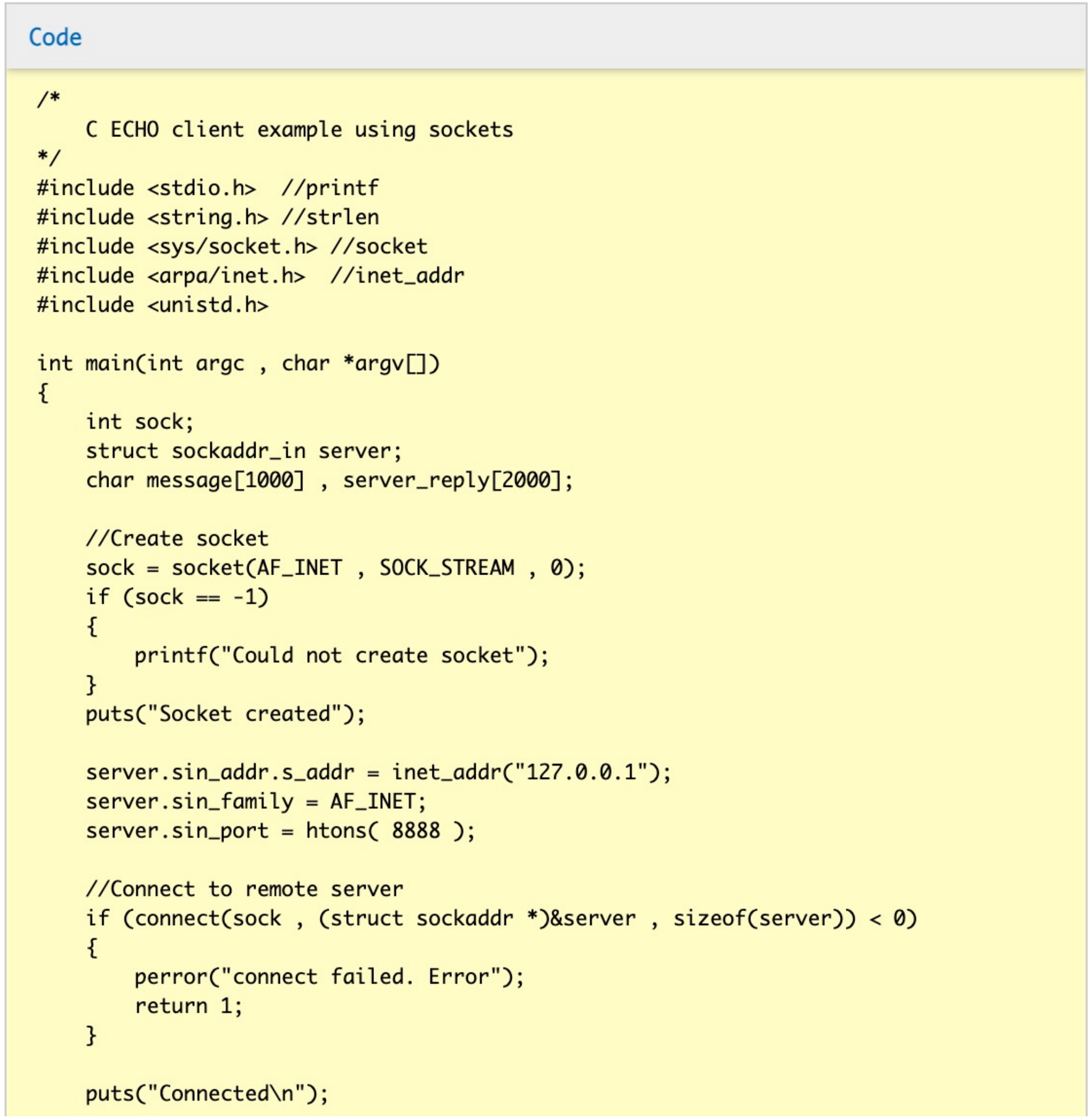


```
./server
Socket created
bind done
Waiting for incoming connections...
```

- See the example “How to Code a Server and Client in C with Sockets” from https://www.binarytides.com/server-client-example-c-sockets-linux/

Client

Part 1



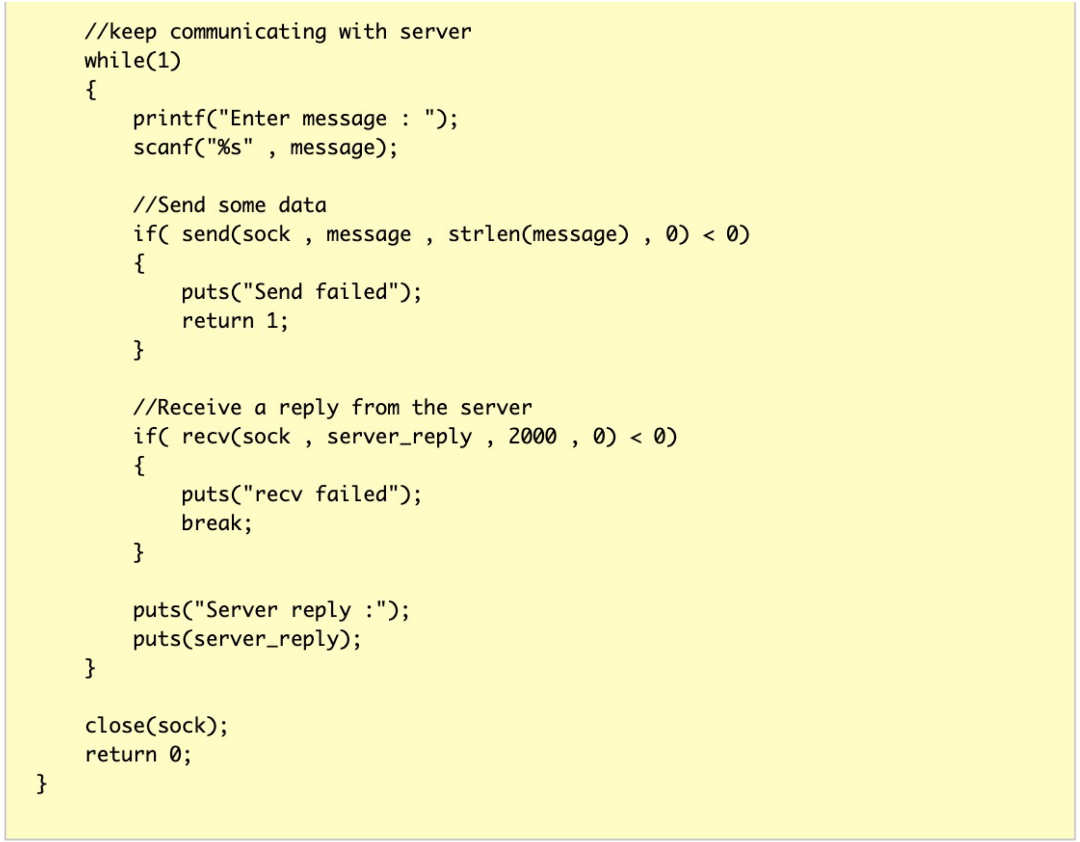
Code

```
/*
    C ECHO client example using sockets
*/
#include <stdio.h>  //printf
#include <string.h> //strlen
#include <sys/socket.h> //socket
#include <arpa/inet.h>  //inet_addr
#include <unistd.h>

int main(int argc , char *argv[])
{
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];

    //Create socket
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    server.sin_addr.s_addr = inet_addr("127.0.0.1");
    server.sin_family = AF_INET;
    server.sin_port = htons( 8888 );

    //Connect to remote server
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("connect failed. Error");
        return 1;
    }

    puts("Connected\n");
```

- See the example “How to Code a Server and Client in C with Sockets” from
https://www.binarytides.com/server-client-example-c-sockets-linux/

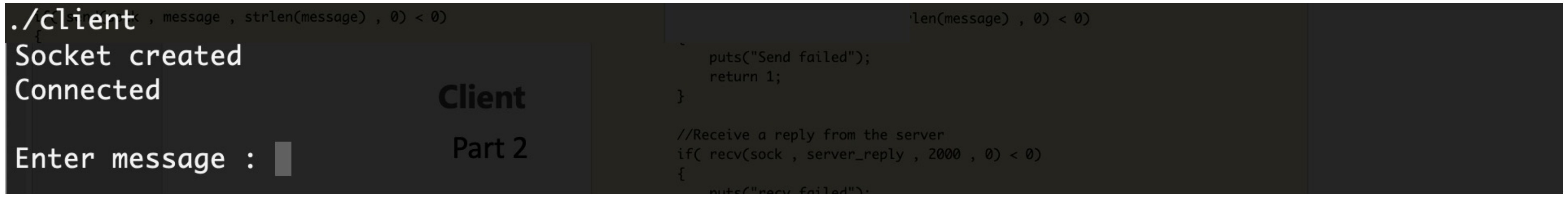
Client

Part 2



```c
    //keep communicating with server
    while(1)
    {
        printf("Enter message : ");
        scanf("%s" , message);

        //Send some data
        if( send(sock , message , strlen(message) , 0) < 0)
        {
            puts("Send failed");
            return 1;
        }

        //Receive a reply from the server
        if( recv(sock , server_reply , 2000 , 0) < 0)
        {
            puts("recv failed");
            break;
        }

        puts("Server reply :");
        puts(server_reply);
    }

    close(sock);
    return 0;
}
```

Run it after compiling



```
./client
 Socket created
 Connected

 Enter message : 
```

The web page contains an example of a server handling multiple connections