4. Correcting a run-time error in the server from the MT C/S practice

Alberto Miguel Diez

a. Explain the problem with your own words.

The problem is in the following piece of code that is in a while loop in such a way that it always accepts connections:

```
delegateSocket = accept(welcomeSocket, (struct sockaddr *) &clientAddress, &addressLength);

printClient(clientAddress);

pthread_t threadForClient;

if (pthread_create(&threadForClient, (pthread_attr_t *) NULL, (void *(*)(void *)) clientServerProtocol, &delegateSocket) != 0)
    perror("Thread creation error\n");
else
    printf("New worker thread created\n");
```

When a TCP connection is accepted, an integer representing the identifier of that established connection is stored in the delegateSocket variable. Subsequently, the pthread will be created to which the function in which this new thread is going to be executed and the memory address of the delegated socket are passed as a parameter. Next, the thread will execute the clientServerProtocol function where it will save in the integer delegateSocket the pointer to that memory address (int delegateSocket = * ds;). The problem is that if a new connection arises, the value of delegateSocket will be overwritten and may lead to errors in the execution of the previous thread.

b. Create an environment for reproducing the problem and (c.) Document the results that you have obtained.

I made section b and c together.

To reproduce the experiment I have modified the following lines in the clientServerProtocol.c file. I have added a three second sleep to give me time to execute several client processes that allow me to replicate the error associated with the delegateSocket pointer. Also, I have added two lines that show the integer value of the delegated socket before and after sleep.

```
void * clientServerProtocol(int *ds) {

    printf("New per-client worker thread started\n");
    printf("delegateSocket id before sleep: %d\n",*ds);
    sleep(3);
    printf("delegateSocket id after sleep: %d\n",*ds);

    int delegateSocket = *ds;

    char *request, *response;
```

To replicate the experiment I do *make* to compile all the files.

To carry out the experiment I have opened three consoles, two of them will act as clients (executing the client program) and the last one will be the console where the server is. I start the server that will listen for requests on port 60001. In one of the other consoles I start the client program with the command *./clientSolution 127.0.0.1 60001*

The program will run and on the server screen I got that the connection had been established. Quickly, in the last console that I have left to use, I start the same client program and it appears on the server that the connection has been established.

Next I will explain the error of this program. For this, below I show a capture of the server execution:

```
alberto@alberto-Lenovo:~/Alberto/DS/DSPro/Exercise4/src$ ./serverSolution 60001
server starting on port (60001)
Server loop restarted

Client IP 127.0.0.1     Client port 33677
New worker thread created
Server loop restarted

New per-client worker thread started
delegateSocket id before sleep: 4
Client IP 127.0.0.1     Client port 43443
New worker thread created
Server loop restarted

New per-client worker thread started
delegateSocket id before sleep: 5
delegateSocket id after sleep: 5
Received 25 bytes from client
Check each of the 4 bytes from first integer in net byte order:
byte[0] = 12 ; byte[1] = 34 ; byte[2] = 56 ; byte[3] = 78
Read word-32 in network byte order: 0x78563412
Read integer translated into machine byte order: 0x12345678
Computed result in machine byte order: 0x2468acf0
Calculated response (Without the result sent back) = Integer multiplied by 2
27 bytes were sent back to the clientAddress
Per-client worker thread exiting gracefully
Server thread will remain running

delegateSocket id after sleep: 5
Error upon recv() on a delegateSocket. Worker thread exiting.
```

We can see that a connection is established with the IP 127.0.0.1 and port 33677, immediately after another connection is established with the IP 127.0.0.1 and port 43443. The first connection with port 33677 has a value in the delegate socket of 4. The second connection has a value of 5. After the first connection has finished its sleep, its delegateSocket value will be 5 because the last execution of the client has overwritten this value. Therefore, the first connection will reply to the delegateSocket with a value of 5 that corresponds to that of the second client. The second connection comes out of sleep with delegateSocket = 5 but nevertheless gives us an error because the client has already been answered by the server and cannot receive the data.

Keep in mind that the first client has never been answered because it's delegateSocket value was overwritten and the server could not respond due to the error. Here is a screenshot of the two client programs that I have run:
Client1:

```
alberto@alberto-Lenovo:~/Alberto/DS/DSPro/Exercise4/src$ ./clientSolution 127.0.0.1 60001
Return value of connect() = 0
Sending a request of type 'Multiply integer by 2' with integer 0x12345678 concatenated in Net Byte
 Order
```

Client2:

```
alberto@alberto-Lenovo:~/Alberto/DS/DSPro/Exercise4/src$ ./clientSolution 127.0.0.1 60001
Return value of connect() = 0
Sending a request of type 'Multiply integer by 2' with integer 0x12345678 concatenated in Net Byte
 Order
Length of response = 27
Bytes in response:
Received response is [equal] to: 'Integer multiplied by 2'
Received word-32 still in Network byte order:
[0] = 0x24; [1] = 0x68; [2] = 0xac; [3] = 0xf0
Received result = 0x2468acf0
```

d. Devise a solution to this problem, implement it and demonstrate that it works

I have added the following lines:

int copyDelegateSocket = malloc(sizeof(int));*

**copyDelegateSocket = delegateSocket;*

Making these changes, space is reserved for each delegated socket that is created and is not overwritten when a new client connects. Keep in mind that the space that is reserved is freed in the clientServerProtocol function.

```
delegateSocket = accept(welcomeSocket, (struct sockaddr *) &clientAddress, &addressLength);

printClient(clientAddress);

pthread_t threadForClient;

int* copyDelegateSocket = malloc(sizeof(int));
*copyDelegateSocket = delegateSocket;

if (pthread_create(&threadForClient, (pthread_attr_t *) NULL, (void *(*)(void *)) clientServerProtocol, copyDelegateSocket) != 0)
    perror("Thread creation error\n");
else
    printf("New worker thread created\n");
```

I have replicated the experiment carried out in the previous section to check the code and demonstrate that it works correctly. Next, I show screenshot of the experiment:

Client1 and Client2 (both have the same output):

```
alberto@alberto-Lenovo:~/Alberto/DS/DSPro/Exercise4/src/programSolved$ ./clientSolution 127.0.0.1 60001
Return value of connect() = 0
Sending a request of type 'Multiply integer by 2' with integer 0x12345678 concatenated in Net Byte Order
Length of response = 27
Bytes in response:
Received response is [equal] to: 'Integer multiplied by 2'
Received word-32 still in Network byte order:
[0] = 0x24; [1] = 0x68; [2] = 0xac; [3] = 0xf0
Received result = 0x2468acf0
```

Server:

```
alberto@alberto-Lenovo:~/Alberto/DS/DSPro/Exercise4/src/programSolved$ ./serverSolution 60001
server starting on port (60001)
Server loop restarted

Client IP 127.0.0.1     Client port 54647
New worker thread created
Server loop restarted

New per-client worker thread started
delegateSocket id before sleep: 4
Client IP 127.0.0.1     Client port 58317
New worker thread created
Server loop restarted

New per-client worker thread started
delegateSocket id before sleep: 5
delegateSocket id after sleep: 4
Received 25 bytes from client
Check each of the 4 bytes from first integer in net byte order:
byte[0] = 12 ; byte[1] = 34 ; byte[2] = 56 ; byte[3] = 78
Read word-32 in network byte order: 0x78563412
Read integer translated into machine byte order: 0x12345678
Computed result in machine byte order: 0x2468acf0
Calculated response (Without the result sent back) = Integer multiplied by 2
27 bytes were sent back to the clientAddress
Per-client worker thread exiting gracefully
Server thread will remain running

delegateSocket id after sleep: 5
Received 25 bytes from client
Check each of the 4 bytes from first integer in net byte order:
byte[0] = 12 ; byte[1] = 34 ; byte[2] = 56 ; byte[3] = 78
Read word-32 in network byte order: 0x78563412
Read integer translated into machine byte order: 0x12345678
Computed result in machine byte order: 0x2468acf0
Calculated response (Without the result sent back) = Integer multiplied by 2
27 bytes were sent back to the clientAddress
Per-client worker thread exiting gracefully
Server thread will remain running
```

In this case, it can be verified that the two clients that have connected consecutively to the server have answered correctly. If you look at the server's output, you can see that the value of the socket delegated in each of the connections with the client is what it has to be, 4 and 5 respectively. Therefore, the value has not been overwritten in the connection of client 2. Also, the two clients have repliedP perfectly. It can be concluded that the program error has been resolved.