Distributes Systems

Kilian Calefice (796461)

20.04.2024

1. Parameter Passing

a) In the lecture, we discussed the parameter passing approaches Call-By-Value and Call-By-Reference. Here, we also explore Call-By-Copy/Restore. With Call-By-Value, a copy of the provided parameter value is given to the callee to operate on. In contrast, with Call-By-Reference, a reference to the variable provided as a parameter is given to the callee to operate on. Again, in Call-By-Copy/Restore, parameter passing is handled by sending a copy of the referenced variable to the callee, and on return replacing the caller's copy with that modified by the callee.
With RPC, all three parameter passing approaches may be used. What will the output of the procedure `MAIN` be (i. e., the value of the array *a* on line 11) if `ROTATE` is called with an RPC protocol using...

- ...Call-By-Value
- ...Call-By-Reference
- ...Call-By-Copy/Restore

```
 1 procedure ROTATE(var x,y: array[6], var n: integer, var i:integer)
 2   y[i] <- x[(i + n) mod 6];
 3   if i < 5 then
 4     ROTATE(x, y, n, i+1)
 5   end if
 6 end procedure
 7
 8 procedure MAIN
 9   var a: array[6];
10   a <- [A, B, C, D, E, F];
11   ROTATE(a, a, 3, 0);
12   print a;
13 end procedure
```

- Call-By-Value: [A, B, C, D, E, F]
- Call-By-Reference: [D, E, F, D, E, F]
- Call-By-Copy/Restore: [D, E, F, A, B, C]

2. Socket-Programming, Chat - Client-Server

Implement a chat system, which uses UDP (User Datagram Protocol) in C:

a) First the server is communicating with only the client.

- The server (`chat_server.c`) should wait on a port given by the parameter for requests.

- The client (`chat_client.c`) sends a message to the server. The server's hostname, portnumber and chatname will be given a parameters.

- The chat server prints the message, the origin, and the timestamp on `stdout`.

- The chat server sends an acknowledgment to the chat client.

Example:

- The server is called with: `chat_server portnumber`

- The client call looks like: `chat_client host portnumber chatname`

- Your implementation should first be tested on your `localhost`. The solution for this assignment is working between `tiree` and `coll`. Please use the portnumbers between 20001 and 20020.

3. Chord System

a) Node 30 needs to resolve the key $k = 16$. Give the sequence of nodes that are visited by this request, until the node storing k is reached. Also, give the complete finger table of each node storing k is reached. Also, give the complete finger table of each node visited by this request.

We start at node 30 and the finger table looks the following:

1	3
2	3
3	3
4	7
5	16

As we can see the node 16 is already in the finger table and since that node is our destination we can immediately jump to that node. The finger table of node 16 would be:

1	21
2	21
3	21
4	26
5	3

b) Now, assume that node 19 has just joined the network. What is the finger table of node 19 after the join procedure has completed? What other finger tables in the network will actually change (you do not have to compute the changed finger tables)?

Finger table of node 19:

1	21
2	21
3	26
4	26
5	3

The finger table of node 3, 11 and 16 have to be updated to provide consistency.

c) Assume that two nodes join the network at exactly the same time. In which case can this lead to inconsistencies in the Chord ring? Name these inconsistencies and briefly illustrate a small example of the problem.

Let's assume that the nodes 0 and 1 are joining the ring. Since they joined the finger table of 30 needs to be updated. When the node 1 joins the column 1 and 2 should be updated to 1 of the node 30. Also when the node 0 joins the column 1 and 2 should be updated to 0 of the node 30. Since they join at the same time and are both requiring an update on the finger table for different values we might run into a race condition.

Let's say node 0 joins reads the finger table entries of node 30 and realizes that the column 1 and 2 needs to be updated to 0. Now node 1 does the same and also realizes that the column 1 and 2 needs to be updated but to the value 1. They both have performed their read and now want to write the same columns. Node 0 writes first and since node 1 has already performed their read it does not check again which state the columns shall assume and just performes it's write on column 1 and 2 based on the information it gathered from the read it had performed. The finger table now looks like this:

1	1
2	1
3	26
4	26
5	3

The finger table has now an inconsistent state and when trying to retrieve data from the key 0 when entering at node 30 it will fail since the finger table says that the data of key 0 is on node 1 when node 0 actually exists and holds the data.

d) In the lecture, part of the an algorithm for repairing inconsistencies in the Chord ring was presented, which is also shown below. Complete this algorithm, i. e., fill in the missing section between lines 7 and 8.

```
 1 Run the following on each node p:
 2 while true do
 3     SLEEP(n seconds);
 4     q <- SUCC(p + 1).PRED;
 5     if q = p then
 6         continue;
 7     else
 8         i <- ring_size ** 1/2;
 9         a <- q - p;
10         if a < 0 then
11             a <- a * -1;
12         while i > 0 do
13             if a >= i ** 2 then
14                 ft_p[i] = q;
15             else
16                 break;
17             i <- i - 1;
18     end if
19 end while
```

Assuming that `ring_size` is the size of the whole ring and also a power of 2. If q does not equal p we know we have to repair the finger table of p. Assuming `ft_p` is the finger table of p. We start at the highest power and count the iterator down and check if the distance from p to q is lower then i squared and if so we have to update the column i.

e) Assume that an arbitrary single node in the Chord ring can fail (Crash-Stop, i. e., the node remains stopped). Given the function `x.isNodeActive(y)`, that will report whether node `y` is active provided that a direct link between `x` and `y` existed in the Chord ring, describe an algorithm that detects such a failure and subsequently repairs the Chord ring.