גילוי תשובות

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3</td>
<td>Q4</td>
</tr>
<tr>
<td>Q5</td>
<td>Q6</td>
</tr>
</tbody>
</table>

TOTAL

שאלה 1

(30 נקודות)

מעקי A (6 נקודות)

1.1.

a set to 1

c set to 0

b set to 1

c set to 1

1.2.

b set to 1

c set to 0

2.

a set to 1

c set to 0

b set to 1

c set to 1

b set to 1

c set to 0

a set to 1

c set to 1

a set to 1

b set to 1

deadlock

יש עוד... בקושי, המסך:

a set to 1

c set to 0

b set to 1

c set to 1

b set to 1

c set to 0

a set to 1

c set to 1

a set to 1

b set to 1

deadlock
a set to 1
b set to 1
deadlock

a set to 1
b set to 1
c set to 0
c set to 1

a set to 1
b set to 1
c set to 0
c set to 1

T1
wire1.setValue(1);
System.out.println(_id + " set to " + value);
_value = value;
_toGate.onChange();
_inwire1.getValue();

T2
wire2.setValue(1);
System.out.println(_id + " set to " + value);
_value = value;

T1
_inwire2.getValue();

T2
_toGate.onChange();
null

boolean synchronized public test()
{
    synchronized(_inWIRE1)
    synchronized(_inWIRE2)
    synchronized(_outWIRE)
    return (_outWIRE.getValue() == _outWIRE.getValue() & _outWIRE.getValue())
}

וגרדה האוטוכירובטים ממוקדות על מספר סתימות 0 ו1 וה千伏ולה באומל שולח.

null

null

public void synchronized setValue(int value, Object key) {
    if (_sourceKey == key) {
        System.out.println("set to " + value);
        _value = value;
        _toGate.onChange();
    }
}

אומר שדא אובייקט רציף לכל גישה ממוקדות המזרחיים במעולקה

המודל כולל (דא מזרחי全长 ובוש אז מהבלד)

class BasicAndGate implements Gate
{
    protected Wire _inwire1;
    protected Wire _inwire2;
    protected Wire _outwire;
}
private Object _outwireKey;

BasicAndGate() {} 

public synchronized void connect(Wire inwire1, Wire inwire2, Wire outwire) {
    _inwire1 = inwire1;
    _inwire2 = inwire2;
    _outwire = outwire;
    _outWireKey = new Object();
    _outWire.setValue(_inwire1.getValue() & _inwire2.getValue(), _outWireKey);
    onChange();
}

public synchronized void onChange() {
    if (_outWireKey != null) {
        _outWire.setValue(_inwire1.getValue() & _inwire2.getValue(), _outWireKey);
    }
}

public void setValue(int value) {
    synchronized(this) {
        System.out.println(_id + " set to " + value);
        _value = value;
    }
    _toGate.onChange();
}

:BasicAndGate() Sheldon setOnce

public synchronized void onChange() {
    synchronized(_inwire1)
    synchronized(_inwire2)
    _outWire.setValue(_inwire1.getValue() & _inwire2.getValue());
}

:wire setValue() Sheldon _toGate

:Selection = _outwire get _outwireKey

:Unselect _outwire

:wire setValue() Sheldon _toGate

:Selection = _outwire get _outwireKey

:Unselect _outwire
- שדרוג לוח אתה מחקר. לחרת בבערות 2 בשתי ביהדות ויתקן את זה עם תצורה.

3. מאואר עתדה/ הדרונות חבק. בחוד לוחת את חומרים וזרם ואת הדרונות המוחלט שישבלי הדרונות.

4. להלך בנית/ הדרונות בלועות ידועה בוחן לא מוכנה מבית.

5. בוחר את שמות מגדיר את התוכנית וapiKey่าน על 10 בתיוクトструкמד


dבצל פקודת תכנית למלכודת של מגדיר שיתulia ידועה זו/זאו/זא מודל/בר. לוחר השבטים/icie

סקוג א (6 בחודות)

0
1
鲕ית לקחה לא וחד מקורי

ובכונה: לא מגדיר אתประสית השעט. הוא מגדיר את ב時の המחוז להקשר集市ית טכנית. ב-3-ע"ש scope a3 פורק רתבע א3 a2 ו2 רתבע א3 - מגדיר פרסום של הבוחר догוד: ב-3-ע"ש scope a2 מגדיר h destructor והפיבוד של a2 לקחה זו/זא

סקוג ב (6 בחודות)

וע Lýמן לא אסורה ומשמעה ובו התכונה A:

class A {
    public:
    ...
    .
    .
    A& operator=(const A& other) {
        if (this != &other) {
            delete _p;
            _p = new int(*((other._p);
        }
        return *this;
}
The server is a TCP server. The application protocol includes a single request:

- GetAddressOfPlayer(in string MovieName, out string AddressOfPlayer)
  errors: No such movie currently playing

There are two possible answers based on the assumptions:

**Assumptions 1:**
- There is a "small" number of players (for example - less than 100).
- The list of players is static (it is not updated frequently).
- The server does not verify anything about the player before sending back an address to the client. (for example, do not verify that the player is indeed playing, or do not verify access rights or payment for the client).
- The work of the server is then "very short": lookup the movie name in a hashtable and send back the address.

In this case, a single threaded sequential server is appropriate. (loop: accept connection, lookup address, send back reply)

Note: if there can be very many client requests (millions) - the best scalability solution would be to run several instances of the same single threaded server on different machines and to add a router solution in front of them (load balancing).

**Assumptions 2** If any of the following condition holds:
- There can be a large number of players.
- The set of players is updated frequently (new players are added or removed frequently).
- The server verifies that players are operating correctly in the background.
- The server verifies access rights for the client in a database or payment conditions.

Then the handling of the client request can be long and heavy. In this case a reactor design is more appropriate (thread pool, selector, acceptor, work queue).

Note: it is never correct in this scenario to use a thread-per-connection server because:
  i. We do not control how many clients will connect
  ii. In any case, the work required for a single connection is "short" (even under assumption 2) compared to the cost of creating a thread. (In the application protocol, we only reply to a single message - there is no conversation).

---

**Key points from the description of the protocol:**
- one player plays to more than one receiver
- one player sends the same content to all receivers
- since the player plays regardless of whether receivers are connected, we are working in "streaming" mode - that is, the receiver plays the chunks of data as it receives it. This is in contrast to "file exchange" mode - where the receiver would download a file then play it once it has arrived.

**Key points about video playing:**
- video files are large (many megabytes) - even for highly compressed formats.
- video is made up of several units (chunks) which contain one or several frames (pictures) and the corresponding audio data.
- in streaming mode, we want to be able to play data without keeping a large buffer on the receiver: this means we must start showing video frames as soon as one or a few video units have arrived.
The most appropriate protocol in these settings is: UDP in multicasting:
- Multicasting allows the player to send video data only once and have all receivers receive it with little overhead on the network.
- UDP is appropriate because:
  - There is no multicast TCP option
  - There is no need for connection handling
  - There is no need for acknowledgments being sent from the receiver to the player (one way communication)
  - Packets should be sent by the player before any ack is received (pipeline)
  - The receiver can deal with some loss of data and loss of order of the packets (assuming packets are sent with redundant information
  - checksum plus possibility to resend part of the data in several packets - and and with packet numbering).

With the new request, streaming is not possible. If taken literally this means the player must now have:
- Server model: a thread per connection model would be appropriate - one thread per receiver connected.
  Note that a reactor would not improve much the situation.
  Note also that naturally we must impose a limitation on the number of clients accepted - it is better to refuse new connections than to crash.
- Connection model: UDP unicast. Not that TCP is still not appropriate for most of the reasons listed above.

Would you accept to implement this modification:
The rational answer is to propose a price formula (not to say "no").
The price of the request is that each connection now takes the same resources on disk access, CPU and network bandwidth as a single movie in the previous model for the duration of the movie.
So the price of a single movie in the new model must cover the price we would obtain in the previous model. If we can charge this or more - then it is worth it. Alternative solution: propose a way to queue requests for a movie for a while (maximum delay acceptable to a customer), then multicast to all the clients who have sent their requests within this time window. If there are many requests for the same movie that arrive around the same time period - we would save a lot.
public static void main(String[] args) throws Exception {
    DatagramSocket socket = new DatagramSocket(4445);
    DatagramSocket mcSocket = new DatagramSocket();
    InetSocketAddress mcAddress = InetSocketAddress.getByName("124.1.1.1");
    while (true) {
        byte[] buf = new byte[256];
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
        String message = new String(packet.getData());
        System.out.println(message);

        // send the packet to the multicast group
        mcSocket.send(new DatagramPacket(packet.getData(), packet.getData().length,
                                          mcAddress, 3000));
    }
}
### Cows

<table>
<thead>
<tr>
<th>ID</th>
<th>NickName</th>
<th>BornDate</th>
</tr>
</thead>
</table>

Primary Key: ID  
Unique Index: NickName

### Production

<table>
<thead>
<tr>
<th>ID</th>
<th>DayInYear</th>
<th>Year</th>
<th>MilkQuantity</th>
</tr>
</thead>
</table>

Primary Key: ID + DayInYear + Year  
Foreign Key: ID

### Weight

<table>
<thead>
<tr>
<th>ID</th>
<th>DayInYear</th>
<th>Year</th>
<th>Weight</th>
</tr>
</thead>
</table>

Primary Key: ID + DayInYear + Year  
Foreign Key: ID

---

### Cows

<table>
<thead>
<tr>
<th>ID</th>
<th>NickName</th>
<th>BornDate</th>
</tr>
</thead>
</table>

Primary Key: ID  
Unique Index: NickName

### Production

<table>
<thead>
<tr>
<th>ID</th>
<th>DayInYear</th>
<th>Year</th>
<th>MilkQuantity</th>
</tr>
</thead>
</table>

Primary Key: ID + DayInYear + Year  
Foreign Key: ID
<table>
<thead>
<tr>
<th>ID</th>
<th>DayInYear</th>
<th>Year</th>
<th>Data</th>
</tr>
</thead>
</table>

Primary Key: ID + DayInYear + Year

Foreign Key: ID

**སོགས**:
- ID – ལྷ་སྟེགས་ཀྱི་དོན་དང་
- DayInYear – སྱིན་དོན(365-1)
- Year – སུད་
- Data – འཕྲོམ་གྱི་བོལ(དཔེར་ཕྱོགས།)
- བོམ་(དཔེར་ཀུན་ལོ།) འཛོམ་བཞི་