גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (5 נקודות)

LiveLock עלול להתרחש, בהסתברות נמוכה, אם בעקבות deadlock של השחקנים, שולף ה Dealer, שוב ושוב, קלף שאינו תואם לאף שחקן.
לדוגמא, נניח כי על השולחן מונח הקלף 5 ולשני השחקנים הקלף 1. בעקבות החבק מגריל ה Dealer קלף חדש בתחום [5-10] השחקנים מתעוררים ונכנסים שוב להמתנה, ושוב מגריל ה Dealer קלף חדש בתחום [5-10], וכן הלאה.

סעיף ב (10 נקודות)

נוסיף במחלקה CardTable מתודה מסונכרנת המחזירה את גודל ערימת הקלפים. זאת כדי לאפשר זיהוי של אי שינוי בטבלה:

```
class CardTable {
...
  public synchronized size() { return _cards.size(); }
...
}
```

נגדיר במחלקה Dealer מונה הסוכם את מספר הפעמים בו טופל ברציפות החבק (ללא שינויים בטבלה).
המונה מאותחל ל-0 לאחר שינוי בטבלה, וגדל בכל פעם שמבוצע reset() ללא שינוי בגודל הטבלה.
במידה והמונה הגיע ל 10, נפסיק את פעילות השחקנים על ידי ביצוע interrupt.

```
class Dealer implements Runnable{
  private CardTable _table;
  private PlayerDeadlockController _deadlockController;
  private Thread _tPlayer1,_tPlayer2;
  Dealer(CardTable table, PlayerDeadlockController deadlockController, Thread tPlayer1,Thread tPlayer2) {
    _table = table;
    _deadlockController = deadlockController;
    _tPlayer1 = tPlayer1; _tPlayer2 = tPlayer2;
  }
  public void run() {
    int resetCounter=0;
    while (!_deadlockController .isEndOfTheGame()) {
      try {
        _deadlockController.waitForDeadlock();
```

```
        catch(InterruptedException e) {
          return;
        }
        if (_table.size() == 1) // no card was added {
          if (resetCounter==10) {
            tPlayer1.interrupt();
            tPlayer2.interrupt();
            return;
          } else
            resetCounter ++;
        } else  // cards were added
          resetCounter = 1;
        _table.reset();
      } // while
    }
  }
```

]

כמו כן נעדכן את **Player** כך שתכלול את אפשרות ה **interrupt** בתנאי הלולאה, מלבד תפישת ה **InterruptedException**

```
class Player implements Runnable {
  …
  public void run() {
    …
    while (!_cards.isEmpty() && !Thread.currentThread().isInterrupted()) {
     …
  }
  …
}
```

[

סעיף ג (5 נקודות)

אם בזמן ביצוע ההדפסה על ידי הת'רד הראשי, מתחולל חבק בין שני השחקנים, וה Dealer מנקה את השולחן ומניח קלף חדש, תיתכן הדפסה שאינה מקיימת את האינריאנטה.
לדוגמא: לוח המשחק מכיל את הקלפים [1,2,3] הת'רד הראשי הדפיס 1, בוצע reset על ידי ה Dealer בעקבות חבק וכעת הלוח מכיל את הקלפים [5,6,7], הת'רד הראשי ממשיך בהדפסה ומדפיס 6.

סעיף ד (10 נקודות)

נוסיף למחלקה CardTable שדה המציין את הגרסה של הטבלה.
שינוי הטבלה יגדיל את הגרסה.

במהלך ההדפסה נבדוק, תחת סנכרון, לפני הדפסת כל איבר האם השתנתה הגרסה, אם כן נזרוק Exception.

```
class CardTable {
  protected int _version;
  …
  CardTable(PlayerDeadlockController deadlockController) {
    …
    _version=0;
  }
  …
  public synchronized void addCard(Integer card) ) throws WrongCardValueException, InterruptedException {
    …
    _version++;
  }
  public synchronized void reset() {
    …
    _version++;
  }
  protected synchronized int getVesion() { return _version; }
  public synchronized void print cards() throws ConcurrentModificationException  {
      int version = getVersion();
       for (Integer card : _cards)
         if (version == getVersion())
           System.out.println(card);
         else
           throw new ConcurrentModificationException("Cards table was changed during printing");
  }
}
```

שאלה 2 (30 נקודות)

סעיף א (3 נקודות)

לאחר הקריאה לאופרטור ההשמה של b2 גם האובייקט b2 וגם b1 יצביעו לאותה כתובת על ה heap. ביציאה מה- scope של יקרא ה dtor פעמיים וינסה למחוק קטע בזיכרון שהוא כבר מחוק.

סעיף ב (5 נקודות)

אופרטור ההשמה חייב לבדוק תחילה אפשרות של השמה עצמית. יש להוסיף בשורה הראשונה:

```
if ( this == &other) return *this;
```

סעיף ג (2 נקודות)

```
B(const B& other) { copy(other); }
```

הערה: תמיד עדיף להשתמש בפונקציה קיימת. טעויות נפוצות היו קריאה ל clean או החזרת ערך this.

סעיף ד (10 נקודות)

מחסנית:

החל מכתובת	ערך	פרוש
5024	5	הערך של i
5020	5004	הערך של b
5016	8700	כתובת חזרה ל Q4
5012	5004	הערך של pb1
5008	2000	הערך של m_pData
5004	1000	מצביע ל vtable
5000	9000	כתובת חזרה ל main

ערימה:

החל מכתובת	ערך	פרוש
2000	1	הערך של *m_pData
1004	8500	כתובת של B::getData()
1000	8550	כתובת של B::~B()

סעיף ה (10 נקודות)

מחסנית:

החל מכתובת	ערך	פרוש
5020	5004	ה- this של d
5016	8800	כתובת חזרה ל Q5
5012	0	הערך של D ::m_pData
5008	2000	הערך של B::m_pData
5004	1000	מצביע ל vtable
5000	9000	כתובת חזרה ל main

שימו לב שבקריאה לשיטה הנמצאת במחלקה נוסף פרמטר ה- this המאפשר לפונקציה (השמורה פעם אחת בזיכרון ומשמשת את על האובייקטים) לפעול על המשתנים של האובייקט הספציפי.

ערימה:

פרוש	ערך	החל מכתובת
הערך של `*m_pData`	1	2000
הכתובת של `D::getData()`	8500	1004
כתובת של `D::~D()`	8550	1000

D::~D() קיים גם אם לא נכתב מפורשות. הוא נחשב virtual כי B::~B() מוגדר virtual ולכן הוא מופיע ב vtable. למען הדיוק גם הכתובת של B::~B() מופיע כי כל ה dtors של שרשרת הירושה יקראו בהריסת d.
אם B::~B() לא היה מוגדר virtual אז אף אחד מהם לא היה מופיע ב vtable והקריאה להם הייתה מתורגמת לכתובת בעת הקומפילציה.

שאלה 3 (30 נקודות)

סעיף א (8 נקודות)

PRELIMINARY NOTES:
We are asked to give your opinion on a proposal to change the system and evaluate the proposal along 4 dimensions.

Our answer must therefore include 4 paragraphs (one for each dimension) and each paragraph must be a comparison between the existing architecture (Reactor) and the proposed architecture (MultiServer).

Our answer must also be specific – it must take into account the specific data of the case we are evaluating, including number of concurrent connections, expected duration of the connections, and expected rate of requests.

Background definitions:
These definitions were not required as part of the answer.
It is important to try and define formally the concepts we are evaluating before we can compare the two architectures.
Fairness: it is more difficult to define fairness than to detect "unfairness". What does it mean that clients are served fairly? By the negation, that there is no client that is given better service than the others. A case of "unfairness" would be that some clients are served fast and others very slowly.
Those curious about formal definitions can look at the following Wikipedia article: http://en.wikipedia.org/wiki/Fairness_measure for formal measures of fairness in protocols.
Future Expansion: expansion means we expect more requests, more users connected concurrently – but the requests will remain of the same expected duration (the mode of voting is not expected to change).
CPU Utilization: *utilization* is not the same concept as *usage*. To measure usage, we just look at which rate the CPU is used for a given task. If 2 methods solve the same problem, and method M1 uses X1% of CPU and method M2 uses X2% > X1% - then we would say that M1 is more *efficient* in CPU. *Utilization* is a different concept. It refers to a relation between *capacity* and *usage*. Suppose we have one hotel room to rent. We have requests to rent the room for 2 months. We could rent the house in January, leave it empty in February and rent it again in March. In this example, the

utilization of our resource (the room to be rented) would be 2/3 (the resource is used at 2/3 of its capacity and is left un-used 1/3 of the time). We could instead rent the room 2 months and reach 100% utilization over a shorter period of time. Higher utilization means we exploit our existing resources to the maximum of their capacity instead of leaving them unused and taking more time to perform the same service.
Amount of required resources: we must define which resources can vary between the 2 architectures. Resources we have discussed in the course are: CPU time, RAM, network ports, threads and network bandwidth. A resource can be allocated to one task for a certain time period; during this time, the resource is not available for any other tasks. There is a finite amount of resources of each type.

Background specific data on the case
What do we know about the problem – and how these parameters influence our answer?

- There are 10 voting stations. This means we expect up to 10 concurrent clients to our voting server. There cannot be more than 10 concurrent connections. There could be less – if there is one station where no one is voting at a given point in time.
- Voting consists of obtaining data about the candidates (getParties), then sending the voter decision (vote). These operations are expected to be short.
- In the voting system, each client connects to the server for each request, sends a single request, then disconnects. (This appears in the code of the stub and skel.)
- On the server side, we expect the vote operation to be synchronized (if many clients send simultaneously a vote request, they will have to be sequentially served on the server side). getParties does not need to be synchronized, and concurrent calls can be performed in parallel on the server side.

Expected answers:

The multiserver has one thread listening to the server socket for connections in a blocking manner. Once the thread performs the "accept" on the server socket, a new thread is created. This takes some time (thread creation is a heavy operation). After the thread is created, it reads (in a blocking manner) the request from the client. There can be up to 10 threads reading from 10 connections at the same time in 10 threads on the server side + 1 thread waiting for an accept.

In contrast, the reactor has 1 thread listening to the selector for any event (accept or read) and 10 threads waiting for actions to be posted on the executor task queue. (Note that the executor has 10 threads – poolsize).

Fair Service to customers:
There is not difference in delay in servicing a connection (accept) between the 2 architectures – because the thread pool of the reactor has as many threads as the maximum number of concurrent connections.
There is an advantage in the delay it takes to start reading the request sent by the client in the reactor – because there is no delay between the accept and the read operations while there is a delay (duration of the time it takes to create a thread). But in both cases all the clients are served in the same manner.
Therefore, there is no difference in fairness between the 2 architectures.
The fairness of the server depends on the fairness of the thread scheduler used by the runtime environment.

Future Expansion:
If the number of concurrent connections grows, there could potentially be many threads running at once in the multiserver architecture. If this grows to more than a few hundreds concurrently connected clients, this could lead to a collapse of the server. The reactor is immune to such overload – the worst that could happen is that clients would have to wait a long time, or suffer from timeouts.
The reactor is more capable of scaling up than the multiserver.

CPU Utilization:
[We accepted answers that understood "efficiency" instead of "utilization" as explained above].
The reactor architecture is more efficient – because for each connection, we do not pay the heavy price of creating a new thread. This means that overall, the reactor server will perform less CPU/OS operations than the multiserver.
In terms of utilization, the pattern of CPU usage of the reactor is much more "smooth" than the one of the multiserver. This is because, for a short client connection (connect, send request, process request with a short computation, send small answer), the reactor uses the CPU without waiting for blocking operations. In contrast, the multiserver threads block on blocking operations (accept, create thread, read, write). Therefore, the pattern of CPU usage of each thread is "bursty" – heavy usage, then long periods of waiting without doing anything.
The reactor CPU utilization is therefore better than that of the multiserver.

Amount of required resources:
At any given time, in all the clients are connected, the reactor and the multiserver will use a similar amount of resources (11 threads, the RAM associated to them in the form of stacks, the same number of connected sockets, the same amount of bytes passing through the network).
The reactor is better than the multiserver in the way the resources are allocated/freed – because the reactor allocates the threads once (when the process starts running) and that's it. The multiserver will repeatedly allocate/free the threads.
On the other hand, if there are less than 10 concurrently connected clients, the multiserver will use less resources than the reactor with 10 threads in the thread pool.

סעיף ב (4 נקודות)

```
public class Voting_Skel {
  Voting_Skel(Voting voting) throws Exception {
    int port = 1984;
    new Thread(new MultipleClientProtocolServer(port, new VotingProtocol(voting))).start();
  }
}
```

סעיף ג (6 נקודות)

Preliminary Notes:
We are asked 3 questions:
- Why the scientists proposed to use UDP
- Whether UDP saves in communication costs (less bits exchanged)
- Whether UDP saves in RAM usage on the client and server side

We are told that in the dictature:
- The vote of the citizens has no influence on the results of the election
- The fact that a citizen voted is of importance (what he voted is not important).

A good answer must be specific – it cannot be just a list of the differences between UDP and TCP. It must explain which differences are relevant to the scientists' decision.

Expected answer:
UDP is a non-reliable protocol – it does not ensure arrival of packets in the same order they were sent and it does not ensure all packets reach their destination.
TCP in contrast is reliable – and ensures no-reordering and no-loss of packets.

The scientists recommended UDP because they expected that reliability of the vote is not necessary in a dictatorship. Their reasoning is wrong: UDP does ensure no-change (it uses a checksum mechanism on the datagrams) but it does not ensure no-loss. The dictatorship **is** interested in registering who voted. UDP does not provide this guarantee.

The scientists are interested in reducing the cost of the protocol in communication (number of bits and roundtrips exchanged). **This is achieved by using UDP** – because UDP does not use any session control mechanism, which is used in TCP. Session control is achieved by sending packets between client and server to establish the TCP connection, and acknowledgement packets (using algorithms such as GoBackN). TCP can also resend packets if it has doubts they have been received.

The scientists are interested in reducing the cost of the implementation in RAM usage on the client and server side. **This is achieved by using UDP** – because UDP does not use any session-oriented re-ordering of packets. TCP, in order to maintain proper ordering of packets must maintain, both on the sender and on the receiver side, a buffer of received packets (if GoBackN is used, up to N packets). No such buffers are required with UDP – a datagram is sent, and that's it – no buffers needed.

סעיף ד (6 נקודות)

```
public class Voting_Stub implements Voting {
  …
  public void vote(String party, long userId) throws RemoteException {
    try {
      String msg = "VOTE " + party + "#" + userId + "\n";
      byte[] buf = msg.getBytes("UTF-8");  // or (new Encoder()).toBytes(msg);
      DatagramPacket packet = new DatagramPacket(buf, buf.length, _skelAddress, _skelPort);
      _datagramSocket.send(packet);
    } catch (Exception e) {
      throw new RemoteException(e.toString());
```

```
    }
}
```

סעיף ה (6 נקודות)

```
public void accept() throws IOException {
  // Get a new channel for the connection request
  SocketChannel sChannel = _ssChannel.accept();
   // If serverSocketChannel is non-blocking, sChannel may be null
  if (sChannel != null) {
    sChannel.configureBlocking(false);
    SelectionKey key = sChannel.register(_data.getSelector(), 0);
    ConnectionHandler handler = ConnectionHandler.create(sChannel, _data, key);
    handler.switchToReadOnlyMode(); // set the handler to read only mode
    InetSocketAddress  clientAddress = new InetSocketAddress(
                                    sChannel.socket().getRemoteSocketAddress(),sChannel.socket().getPort());
    DatagramChannel c = DatagramChannel.open();
    c.connect(clientAddress);
    SelectionKey udpKey = c.register(_data.getSelector(), OP_READ,handler);
}
```

שאלה 4 (10 נקודות)

סעיף א (5 נקודות)

```
CREATE TABLE Voters(
   ID Int PRIMARY KEY,
   KalpiId Int FOREIGN KEY REFERENCES Kalpi.ID,
   HasVoted bool,
   VotedFor varchar(100) FOREIGN KEY REFERENCES Parties.Name)
```

סעיף ב (5 נקודות)

```
SELECT ID
FROM Voters
WHERE HasVoted = true and VotedFor <> 'The Worms'
```