אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

תאריך הבחינה: 25.2.2007
שם המורה: ד"ר מיכאל אלחדד
ניר צחר
מני אדלר
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשס"ז
סמסטר: א'
מועד: ב'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

במערכות שונות המריצות מספר ת'רדים במקביל באותו תהליך, נדרש במקרים מסוימים להגדיר ת'רד אחד, כלשהוא, כ'מנהיג'. במקרים רבים נעשית הבחירה תוך כדי הריצה של הת'רדים (עקב נפילתו של המנהיג הקודם וכיוצא בזה).
ניתן להגדיר אלגוריתם פשוט בו המנהיג הנבחר יהיה תמיד הת'רד שמספרו 0, אולם יתכן כי לא כל הת'רדים אכן רצים, ואיננו רוצים לבחור כמנהיג ת'רד שלא קיים. התוכנית הבאה מממשת בחירה של מנהיג יחיד לקבוצת ת'רדים בעזרת זיכרון המשותף לכל הת'רדים.
עבור N ת'רדים בתהליך, הזיכרון המשותף הינו מערך של N מספרים, כך שכל ת'רד מיוצג על ידי תא אחד במערך (בדוגמא שלנו התא i במערך מייצג את הת'רד שמספרו i). ת'רד יכול לקרוא את ערכם של כל תאי המערך המשותף, אך הוא יכול לשנות רק את הערך של התא במערך המייצג אותו. כל תא i במערך יכול להיות באחד משלושה מצבים: לא מאותחל (-2), מאותחל (-1), או מכיל את מספרו של המנהיג שנבחר על ידי ת'רד i (מספר בתחום [0,N-1]). תא i שאינו מאותחל במערך מסמן כי ת'רד i אינו עובד עדיין, בעוד שתא i מאותחל מציין כי הת'רד i החל כבר לעבוד אך טרם בחר את מנהיגו.
להלן הגדרתה של המחלקה SharedMemory:

```
class SharedMemory {
  public static final int UNSET = -2;
  public static final int INITIALIZED = -1;
  private final int[] _arr;

  public SharedMemory(int size) {
     _arr = new int[size];
     for (int i=0; i < size; i++)
       set(i, UNSET);
  }

  public synchronized int size() { return _arr.length; }

  public synchronized int get(int index) throws IllegalArgumentException {
    if (!checkIndex(index))
      throw new IllegalArgumentException ();
    return _arr[index];
  }
```

```
  public void synchronized  set(int index, int val) throws IlegalArgumentException {
    if (!checkIndex(index) || !checkValue(val))
      throw new IlegalArgumentException ();
    _arr[index] = val;
  }
  public boolean synchronized checkIndex(int index) {
  ...
  }
  public boolean synchronized checkValue(int val) {
  ...
  }
}
```

א. - השלימו את המתודות **checkIndex** ו **checkValue**.

- הגדירו תכונה נשמרת (invariant) עבור המחלקה **SharedMemory**
- כתבו מתודה בוליאנית **testInv()** הבודקת את קיומה של תכונה זו.

(10 נקודות)

להלן הגדרתה של המחלקה Processor המממשת במתודת ה run() אלגוריתם לבחירת מנהיג, על בסיס הזיכרון המשותף. מטרת האלגוריתם להביא את כל התאים למצב בו הם מכילים אותו ערך שאינו שלילי, כלומר את אותו המנהיג לכל הת'רדים. כל ת'רד מאתחל תחילה את התא שלו. לאחר מכן מתבצע תהליך בחירת המנהיג של הת'רד. המנהיג ההתחלתי הינו הת'רד עצמו. מכאן ואילך עובר הת'רד על התאים במערך המשותף. אם הוא נתקל בתא מאותחל בעל אינדקס גדול מאינדקס המנהיג הנוכחי הוא מגדיר אותו כמנהיג. אם הוא נתקל בתא המכיל מנהיג נבחר של ת'רד אחר הוא בוחר בו סופית כמנהיג (ומסתיים תהליך הבחירה מבחינתו). בסיום תהליך הבחירה של כל ת'רדים מושם המנהיג שנבחר על ידי הת'רד לתא שלו.

```
class Processor implements Runnable {
  private final int _id;
  private final SharedMemory _memory;

  public Processor(int id, SharedMemory memory) throws IlegalArgumentException {
    if (!memory.checkIndex(id))
      throw new IlegalArgumentException ();
    _memory = memory;
    _id = id;
  }

  public void run() {
    _memory.set(_id,SharedMemory.INITIALIZED);
    int leader = _id;
    for (int i=0; i < _memory.size(); i++)) {
      if (_memory.get(i) == SharedMemory.INITIALIZED &&  leader < i)
        leader = i;
      else if (_memory.get(i) >= 0) {
        leader = _memory.get(i);
        break;
      }
    }
    _memory.set(_id, leader);
    System.out.println(_id + "'s leader is " + leader);
```

```
}}
```

ב. ציינו את הפלט של התוכניות הבאות (אם יש כמה פלטים אפשריים ציינו את כולם)? (10 נקודות)

```
public static void main(String [] args) throws IllegalArgumentException {
   SharedMemory memory = new SharedMemory(3);
   new Processor(0, memory)).run();
   new Processor(1, memory)).run();
   new Processor(2, memory);
}
```

```
public static void main(String [] args) throws IllegalArgumentException {
   SharedMemory memory = new SharedMemory(3);
   new Processor(2, memory);
   new Processor(1, memory)).run();
   new Processor(0, memory)).run();
```

```
public static void main(String [] args) throws IllegalArgumentException {
   SharedMemory memory = new SharedMemory(3);
   new Thread(new Processor(0, memory)).start();
   new Thread(new Processor(1, memory)).start();
   new Thread(new Processor(2, memory));
}
```

ג. הוסיפו פעולת סנכרון אחת במתודת ה **run()** של המחלקה **Processor** כך שהאלגוריתם לבחירת מנהיג יהיה נכון גם בהרצה מקבילית – כלומר ייבחר בכל מקרה מנהיג מוסכם אחד כל שהוא. (4 נקודות)

ד. הפתרון בסעיף ג לוקה בחסר שכן הוא מבצע נעילה של אובייקט לפרק זמן ממושך יחסית. ממשו במתודת ה **run()** של המחלקה **Processor** אלגוריתם אחר לבחירת מנהיג שאינו מבוסס על **SharedMemory**.
<u>רמז</u>: במקום ה **SharedMemory** השתמשו במחלקה **CompareNSwap** המובאת להלן. (6 נקודות)

```
class CompareNSwap {
  private int _val;
  public CompareNSwap () { _val = -1; }
  public int synchronized CompareNSwap (int cmp, int swp) {
    if (_val == cmp)
      _val = swp;
    return _val;
  }
}
```

שאלה 2 (16 נקודות)

נתונה מחלקה Point המייצגת קואורדינאטה [x,y] של נקודה במרחב דו ממדי, אשר ניתנים לה בבונה (בסדר x,y)

עיינו בארבעת קטעי הקוד הבאים (שימו לב לשפת התכנות) וציינו את הפלט של כל אחד מהם.

1. Java

```
class Util
{
  public static void F(Point p, Point q) {
    p.x = 42;
    p = q;
  }
}
…
Point a = new Point(10,20);
Point b = new Point(30,40);
Util.F(a,b);
System.out.println(a.x + " " + b.x);
```

2. C++

```
class Util
{
  public:
    static void F(Point p, Point q) {
      p.x = 42;
      p = q;
    }
};
…
Point a(10,20);
Point b(30,40);
Util.F(a,b);
std::cout << a.x << " " << b.x << std::endln;
```

3. C++

```
class Util
{
  public:
    static void F(Point* p, Point* q) {
      p->x = 42;
      p = q;
    }
};
…
Point a(10,20);
Point b(30,40);
Util.F(&a,&b);
std::cout << a.x << " " << b.x << std::endln;
```

4. C++

```
class Util
{
  public:
    static void F(Point& p, Point& q) {
      p.x = 42;
      p = q;
    }
};
…
Point a(10,20);
Point b(30,40);
Util.F(a,b);
std::cout << a.x << " " << b.x << std::endln;
```

שאלה 3 (14 נקודות)

נתונות הגדרתן של המחלקות A ו B

```
class A {
public:
  A()   { std::cout << "A  ";     _p1 = new int(1); }
  ~A() { std::cout << "~A  ";   delete _p1; }
  int* _p1;
};

class B : public A {
public:
  B() {  std::cout << "B  ";      _p2 = new int(2); }
  ~B() { std::cout << "~B  ";   delete _p2; }

  int* _p2;
};
```

א. מה מדפיס קטע הקוד הבא? (4 נקודות)

```
{
   B b1;
}
```

ב. ציינו בעיות בניהול זיכרון בקטע הקוד הבא (4 נקודות)

```
{
  B b1;
```

```
  {
    B b2;
    b1 = b2;
  }
  std::cout  <<  (*b1._p1)  << "," << (*b1._p2);
}
```

ג. תקנו את הגדרת המחלקות כך שימנעו הבעיות אותם ציינתם בסעיף הקודם (6 נקודות)

שאלה 4 (30 נקודות)

שרת סרטים הינו תהליך המשדר סרטים, סרט אחד בכל נקודת זמן, על פי תוכנית שבועית. השרת ממומש כתהליך בעל ת'רד אחד מעל ספריית קבצי divX, כאשר כל קובץ מכיל סרט. התהליך משדר את הסרט על ידי שליחת הקובץ בחלקים לקבוצה שכתובתה ניתן כפרמטר בשורת הפקודה, ל port 2007. להלן הקוד של שרת זה.

השרת משתמש במתודה סטטית **getNextMovie()** של המחלקה **Scheduler** (אופן מימושה לא רלבנטי לשאלה זו), מתודה זו מחזירה את קובץ ה divX הבא לשידור.

השרת שולח את הקובץ בחלקים (לכל היותר 1K בכל פעם) לכתובת ברשת של הקבוצה, דרך DatagramChannel. המחלקה DatagramChannel הינה הרחבה של DatagramSocket כך שניתן לקרוא ולכתוב ב non blocking וכן ניתן לרשום אותו ב Selector.

```
class SimpleMoviesServer
{
  public static void main(String[] args) {
  // usage: SimpleMoviesServer <ip address of a group> <port of group>
    try {
      InetSocketAddrees group =
          new InetSocketAddress(InetAddress.getByName(args[0]),Integer.parseInt(args[1]));
      DatagramChannel out = DatagramChannel.open();
      byte[] buf = new byte[1024];
      while (true) {
        File file = Scheduler.getNextMovie();
        FileInputStream in = new FileInputStream(file);
        while (in.read(buf) != -1)
          out.send(buf,group);
      }
    } catch (Exception e) {
    }
  }
}
```

לקוח המצטרף כמנוי בשרת, מריץ תכנית **client**, הקוראת מ **MulticastSocket** את המידע המשודר מהשרת ומעביר אותו לאובייקט מטיפוס **Player**, המנגן במקביל את הבתים של הסרט המועברים אליו על ידי המתודה **play**. המתודה **keepWatching** בודקת האם המשתמש עדיין מעוניין לצפות בסרט (אופן מימושה אינו רלבנטי לשאלה).

```
class Player extends Thread {
```

```
  public void synchronized play(byte[] buf) {
    ...
  }
}
class SimpleMoviesClient
{
  public static void main(String[] args) {
    // usage: SimpleMoviesClient <ip address of a group> <port of group>
    try {
      Player player = new Player();
      player.start();
      InetAddress groupPort = Integer.parseInt(args[1]);
      MulticastSocket socket = new MulticastSocket (groupPort);
      InetAddress groupAddress = InetAddress.getByName(args[0]);
      socket.  //@1: TODO
      DatagramPacket packet = new DatagramPacket(new byte[1024], 1024);
      while (keepWatching()) {
        socket.receive(packet);
        player.play(packet.getData());
      }
      socket.  //@2: TODO
    } catch (Exception e) {
    }
  }
  private static boolean keepWatching() {
    // return true if the user still want to watch movies
  }
}
```

א. השלימו את שתי השורות המצוינת ב TODO :@ (4 נקודות)

במימוש זה, שרת הסרטים הוא למעשה סוג של טלוויזיה ברשת: לקוח המתחבר יצפה בסרט היחיד המשודר כעת, וגם זאת רק מהמקום אליו הגיע הסרט כאשר הוא התחבר.
בישיבת ההנהלה האחרונה נדונה הדרישה לשדרג את המערכת כך שלקוח יוכל להתחבר לשרת, לבקש סרט ספציפי ממאגר הסרטים, ולצפות בו מתחילתו ועד סופו. העברת הסרט תבוסס על UDP מטעמי חיסכון בפעולות תקשורת, ולאור העובדה כי החברה משווקת גם נגן סרטים משוכלל המתקן שגיאות באמצעות אלגוריתמים מתקדמים לעיבוד תמונה.
בישיבה הוחלט לממש שרת שכזה בתבנית הדומה ל Reactor.

ב. הגדירו <u>במילים</u> את פרוטוקול ההודעות בין הלקוח והשרת עבור שרת הסרטים המשודרג (4 נקודות)

ג. עדכנו את תוכנית הלקוח כך שיתחבר לשרת ב TCP, ויבקש (על פי הפרוטוקול שהגדרתם בסעיף הקודם) לצפות בסרט "The Cabinet of Doctor Caligari". מכאן ואילך, על הלקוח לקבל את נתוני הסרט בחלקים ל DatagramSocket ב port 2007 ולצפות בסרט אקספרסיוניסטי זה להנאתו (הקרנת הסרט מתבססת על הקריאה ל player כפי שזה נעשה בקוד הקודם של **SimpleMoviesClient**). (6 נקודות)

```
class Player extends Thread {
  public void synchronized play(byte[] buf) {
```

```
    ...
  }
}

class SimpleMoviesClient
{
  public static void main(String[] args) {
    //usage: SimpleMoviesClient <ip address of server> <port of server>
     try {
       Player player = new Player();
       player.start();
       InetAddress host = InetAddress.getByName("localhost");
       Socket outsocket = new Socket(InetAddress.getByName(args[0]),
                                     Integer.parseInt(args[1]));
       DatagramSocket insocket = new DatagramSocket(2007);
       //@: TODO
     } catch (Exception e) {
     }
  }
}
```

ד. בתבנית ה Reactor שנלמדה בכיתה הוקדש ת'רד אחד בלבד לניהול התקשורת עם הלקוחות השונים, ומאגר של ת'רדים לטיפול בבקשות על פי הפרוטוקול, בניתוק מוחלט מהלקוח. הסבירו בקצרה מדוע החלטה דומה עבור השרת שלנו אינה טובה (4 נקודות)

ה. להלן מימוש של שרת סרטים המטפל בקבוצה גדולה של לקוחות על פי תבנית ה Reactor. לקוח יכול להתחבר ב TCP ל **ServerSocket** של השרת (בעזרת **Socket** כפי שנעשה בסעיף ג') ולהזמין סרט על פי הפרוטוקול של סעיף ב. בהגיע בקשה שכזו צריך השרת להעביר את קובץ הסרט ללקוח ב UDP. כדי לממש זאת מוגדר **DatagramChannel** להעברת המידע ללקוח, והוא נרשם ב Selector מעל אירוע OP_WRITE עם אובייקט מטיפוס **ConnectionHandler** המטפל באירוע זה במתודת הwrite. מתודה זו מוסיפה משימה מטיפוס **DataSenderTask** ל **Executor**. **DataSenderTask** מממש מתודת **run()** על ידי כתיבת חלק מהסרט ל **DatagramChannel** אליו הוא צמוד.

המחלקות **Reactor** ו **ConnectionAcceptor** הן בדיוק אותן מחלקות שנלמדו בכיתה ואין לשנותן (מתודת ה **run()** של **ה Reactor** מובאת כאן לשם נוחות). עליכם להשלים את הקוד החסר של **ConnectionHandler** ו **DataSenderTask** (ארבעת המקומות המסומנים ב @). (12 נקודות)

```
public class Reactor extends Thread {
...
  public void run() {
    try {
      ExecutorService executor = Executors.newFixedThreadPool(_poolSize);
      Selector selector = Selector.open();
      _data = new ReactorData(executor, selector, _protocol);
      ServerSocketChannel ssChannel = ServerSocketChannel.open();
      ssChannel.configureBlocking(false);
      ssChannel.socket().bind(new InetSocketAddress(_port));
```

```
        ConnectionAcceptor connectionAcceptor =
            new ConnectionAcceptor(ssChannel, _data);
        ssChannel.register(selector, SelectionKey.OP_ACCEPT, connectionAcceptor);
        while (_shouldRun) {
          selector.select();
          Iterator it = selector.selectedKeys().iterator();
          while (it.hasNext()) {
            SelectionKey selKey = (SelectionKey)it.next();
            it.remove();
            if (selKey.isValid() && selKey.isAcceptable()) {
              ConnectionAcceptor acceptor = (ConnectionAcceptor)selKey.attachment();
              acceptor.accept();
            }
            if (selKey.isValid() && selKey.isReadable()) {
              ConnectionHandler handler = (ConnectionHandler)selKey.attachment();
              handler.read();
             }
             if (selKey.isValid() && selKey.isWritable()) {
               ConnectionHandler handler = (ConnectionHandler)selKey.attachment();
               handler.write();
             }
           }
        }
      } catch (Exceptiion e) {
        …
      }
…
}
```

```
public class ConnectionHandler {
...
  public void read() throws IOException {
    SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
    ByteBuffer buf = ByteBuffer.allocate(BUFFER_SIZE);
    while (true) {
      buf.clear();
      int numBytesRead = _sChannel.read(buf);
      if (numBytesRead > 0) {
        buf.flip();
        String str = new String(buf.array(), 0, numBytesRead);
        _incomingData = _incomingData + str;
      }
      if (numBytesRead < BUFFER_SIZE)
        break;
    }
    while (true) {
      int pos = _incomingData.indexOf(MESSAGE_END);
      if (pos==-1)
```

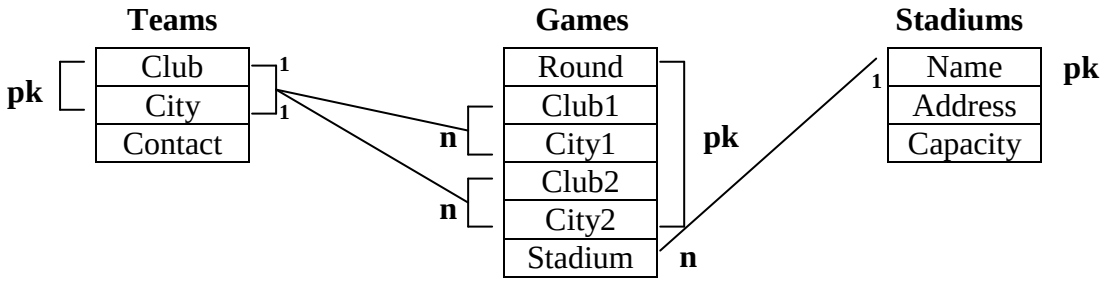
```
            break;
        String message = _incomingData.substring(0, pos);
        _incomingData =
            (pos==_incomingData.length()-1 ? "" : _incomingData.substring(pos+1));
        // @1: TODO: Do something with the message
      }
    }
  }

  public synchronized void write() throws IOException, ClosedChannelException{
    // @2: TODO: Define the parameters for the new DataSender
    _data.getExecutor().execute(new DataSender(...);
  }
...
}
```

```
class DataSenderTask implements Runnable {
  // @3 TODO: Define the fields of the class and the constructor
   public DataSenderTask(...) {
   }
   public void run()  {
     // @4 TODO: send movie data to the client through DatagramChannel
   }
}
```

שאלה 5 (10 נקודות)

להלן מודל הנתונים שנבחר במועד א' לאחסון נתוני המשחקים בליגת העל. כזכור, משחק מוגדר על ידי שתי קבוצות, מגרש, ומספר המחזור. נתוני קבוצה כוללים את שם המועדון, שם היישוב בו הוא פועל, ואת שם איש הקשר של המועדון. נתוני מגרש כוללים את שמו, כתובתו, והקיבולת שלו (מספר צופים מקסימאלי).



א. כתבו שאילתת SQL המחזירה את שמות אנשי הקשר של כל הקבוצות, ואת שמות האצטדיונים בהן שיחקו הקבוצות שלהם (5 נקודות)

ב. כעת נדרש להרחיב את המודל כך שיכלול גם את תוצאת המשחק. תוצאת המשחק מורכבת משני מספרים, האחד מציין את מספר השערים שהובקעו על הקבוצה האחת והשני מציין את מספר השערים שהורכבו על ידי הקבוצה השנייה. כמו כן צריך המודל לכלול את סך הנקודות שהשיגה כל קבוצה בכל משחקיה. עדכנו את מודל הנתונים כך שיכלול נתונים אלו. (5 נקודות)