Tentamen
Nätverksprogrammering
Lösningsförslag

2014–06–02, 8.00–13.00

Del 1

1. a) TCP (Transmission Control Protocol) och UDP (User Datagram Protocol).
   b) Protokollen i transportlagret erbjuder funktioner för att överföra generella data, t.ex. i form av bytes, utan att göra någon tolkning av betydelsen av den data som överförs.

2. a) Sant
   b) Falskt
   c) Falskt
   d) Sant
   e) Falskt
   f) Falskt
   g) Sant
   h) Sant

3. - Låt servern vara flertrådad och låt en tråd sköta uppkopplingar. När en uppkoppling sker skapas en ny tråd som tar hand om den nya uppkopplade klienten. När klienten kopplar ned avslutas tråden. Den första tråden återgår till att vänta på nya uppkopplingar så fort den nya klienttråden skapats. Fördelen är att flera klienter kan hanteras samtidigt. En nackdel vid väldigt hög trafik till servern kan vara att det kan vara resurskrävande att hela tiden starta nya trådar för varje ny uppkoppling.
   - Låt servern bestå av en tråd som tar emot nya uppkopplingar och lägger in dem i en kö. Så fort information om den nya uppkopplingen lagts in i kön återgår tråden till att vänta på nya uppkopplingar. Uppkopplingarna hanteras av pool bestående av ett fixt antal trådar som tar hand om de uppköade uppkopplingarna i tur och ordning. Fördelen med denna lösning är att flera uppkopplingar kan hanteras parallellt, men vi slipper kostnaden för att hela tiden skapa nya trådar. Storleken på trådpoolen måste dock bestämmas så att bra prestanda erhålls.

4.
```
public class Account {
    /* Saldo. Anges i ören för att undvika avrundningsfel. */
    private long balance;

    public synchronized long getBalance() {
        return balance;
    }

    public synchronized void withdraw(long amount) {
        balance = balance-amount;
    }

    public synchronized void deposit(long amount) {
        balance = balance+amount;
    }
}

% Insättning
account.deposit(sum);

% Uttag
account.withdraw(sum);
```

5. a) Det är inte säkert att read kommer att läsa in 100 bytes. Det beror på hur mycket data som har hunnit anlända via nätverket.

b)
```
int bytesToRead = 100;
int bytesRead = 0;
while(bytesRead<bytesToRead) {
    int result = input.read(buffer,bytesRead,bytesToRead-bytesRead);
    if (result==-1 break;
    bytesRead += result;
}
```

6.
```
public class WaitingThread extends Thread {
    private boolean finished = false;

    public void synchronized releaseThread() {
        finished = true;
        notify();
    }

    private synchronized void awaitRelease() {
        while (!finished) {
            try { wait(); } catch(InterruptedException e) { }
        }
    }

    public void run() {
        ...
        /* Wait for computation to finish. */
        awaitRelease();

        /* The external thread has now finished its computation */
        ...
    }
}
```

7. a) Ursprungligen använde man objekt som motsvarade strömmar (klasserna `InputStrem/OutputStream`) för att läsa och skriva data. I det nya paketet arbetar man i stället med buffertar som man fyller med data innan man begär att innehållet skickas. För att ta emot data begär man att data ska läsas in till en buffert som man sedan kan hämta data ifrån.

   b) I java.nio infördes en mekanism som gör det möjligt för en tråd att vänta på att det ska kunna gå att utföra I/O på någon av flera angivna strömmar. Det gör det möjligt att hantera flera öppna förbindelser samtidigt utan att använda sig av flera trådar.

   c) Ett objekt av klassen `Selector` används för att registrera vilka förbindelser (och händelser på dessa) som man vill bevaka. Sedan anropar man metoden `select` i Selector-objektet. Tråden blockeras då ända tills det går att utföra I/O på någon av de registrerade trådarna utan att I/O-anropet blockeras. Tråden får efter select-anropet information om på vilka strömmar det går att utföra I/O och kan beta av dessa i tur och ordning. Därefter kan den utföra ett nytt select-anrop, osv.

Del 2

1. Nätverksbaserat bokningssystem

BookingClient.java

```
import java.net.*;
import java.io.*;

public class BookingClient {

    private static final PORT_NO = 7878;

    public static void main(String [] args) {
        try {
            Socket sock = new Socket(args[0],PORT_NO);
            DataInputStream di = new DataInputStream(sock.getInputStream());
            DataOutputStream do = new DataOutputStream(sock.getOutputStream());

            do.writeUTF(args[1]); // Write command
            do.writeUTF(args[2]); // Write resource name

            // Send additional arguments. No further data to be sent for create.
            if (args[1].equals("book") {
                do.writeUTF(args[3]); // Write date
                do.writeUTF(args[4]); // Write start time
                do.writeUTF(args[5]); // Write end time
                do.writeUTF(args[6]); // Write description
            } else if (args[1].equals("delete") {
                do.writeUTF(args[3]); // Write date
                do.writeUTF(args[4]); // Write start time
            } else if (args[1].equals("show") {
                do.writeUTF(args[3]); // Write date
            } else if (args[1].equals("watch") {
                do.writeUTF(args[3]); // Write date
            }
            do.flush();

            // Receive response
            try {
                while (true) {
                    String s = di.readUTF();
                    system.out.println(s);
                }
            } catch(EOFException | IOException | UTFDataFormatException ex) { }

        } catch(IOException e} {
            System.out.println("I/O Error - terminating");
            System.exit(1);
        }
    }

}
```

BookingServer.java

```
import java.net.*;
import java.io.*;

class SyncBM {
    private BookingManager bm = new BookingManager();

    public synchronized boolean create(String name) {
        return bm.create(name);
    }

    public synchronized boolean book(String resource, String date, String start_time,
                        String end_time, String description) {
        notifyAll()
        return bm.book(resource,date,start_time,end_time,description);
    }

    public synchronized boolean delete(String resource, String start_date,
                                       String end_date) {
        notifyAll()
        return bm.delete(resource,start_date,end_date);
    }

    public synchronized String[] show(String resource, String date) {
        return bm.show(resource,date);
    }

    public synchronized String[] awaitChangedSchedule(String resource, String date,
                                                      String[] schedule) {
        String[] new_schedule = bm.show(resource,date);
        while(Arrays.equals(schedule,new_schedule)) {
            try { wait(); } catch(InterruptedException e) { }
            new_schedule = bm.show(resource,date);
        }
        return new_schedule = bm.show(resource,date);
    }
}

class ClientHandler extends Thread {
    private Socket socket;
    private SyncBM bm;

    public ClientHandler(Socket socket, SyncBM bm) {
        this.socket = socket;
        this.bm = bm;
    }

    public void run() {
        try {
            DataInputStream di = new DataInputStream(socket.getInputStream());
            DataOutputStream do = new DataOutputStream(socket.getOutputStream());

            String command = di.readUTF();
            String resource = di.readUTF();
```

```
            if (command.equals("create")) {
                if (bm.create(resource)) {
                    do.writeUTF("New resource created successfully.");
                } else {
                    do.writeUTF("Resource creation failed.");
                }
            } else if (command.equals("book")) {
                String date = di.readUTF();
                String start = di.readUTF();
                String end = di.readUTF();
                String descr = di.readUTF();
                if (bm.book(resource,date,start,end,descr)) {
                    do.writeUTF("Booking succeeded.");
                } else {
                    do.writeUTF("Booking failed.");
                }
            } else if (command.equals("delete")) {
                String date = di.readUTF();
                String start = di.readUTF();
                if (bm.delete(resource,date,start)) {
                    do.writeUTF("Deletion succeeded.");
                } else {
                    do.writeUTF("Deletion failed.");
                }
            } else if (command.equals("show")) {
                String date = di.readUTF();
                String[] schedule = bm.show(resource,date);
                for(int i=0;i<schedule.length;i++) {
                    do.writeUTF(schedule[i]);
                }
            } else if (command.equals("show")) {
                String date = di.readUTF();
                String[] schedule = bm.show(resource,date);
                while (true) {
                    for(int i=0;i<schedule.length;i++) {
                        do.writeUTF(schedule[i]);
                    }
                    schedule = bm.awaitChangedSchedule(resource,date,schedule);
                }
            }
            s.close();
        } catch(EOFException | IOException | UTFDataFormatException ex) { }
    }
}

public class BookingServer {
    private static final PORT_NO = 7878;

    public static void main(String[] args) {
        SyncBM bm = new SyncBM();
        try {
            ServerSocket ss = new ServerSocket(PORT_NO);
            while(true) {
                Socket s = ss.accept();
                new ClientHandler(s,bm).start();
            }
        } catch(IOException e} {
            System.out.println("I/O Error - terminating");
            System.exit(1);
        }
    }
}
```

2. Adressöversättning

```
public class URLMapper {

    /** Translates the symbolic internet addresses in name_list into
        IP number form (as strings). Returns a list with the
        translated addresses. */
    public static List<String> mapper(List<String> name_list)
                          throws InterruptedException, ExecutionException {
        List<MapURLTask> tasks = new ArrayList<>();
        for (int j = 0; j < 3; j++) {
            List<String> sublist = new ArrayList<>();
            for (int i = 0; i < urllist.size(); i++) {
                if (i % 3 == j) {
                    sublist.add(urllist.get(i));
                }
            }
            tasks.add(new MapURLTask(sublist));
        }

        ExecutorService service = Executors.newFixedThreadPool(3);

        Future<List<String>> future1 = service.submit(tasks.get(0));
        Future<List<String>> future2 = service.submit(tasks.get(1));
        Future<List<String>> future3 = service.submit(tasks.get(2));
        List<String> i1 = future1.get();
        List<String> i2 = future2.get();
        List<String> i3 = future3.get();
        service.shutdown();
        i1.addAll(i2);
        i1.addAll(i3);
        return i1;
    }
}

class MapURLTask implements Callable<List<String>> {
    private List<String> urllist;

    MapURLTask(List<String> urllist) {
        this.urllist = urllist;
    }

    public List<String> call() {
        List<String> iplist = new ArrayList<>();
        for (String url : urllist) {
            try {
                InetAddress ia = InetAddress.getByName(url);
                iplist.add(ia.getHostAddress());
            } catch (UnknownHostException e) {}
        }
        return iplist;
    }
}
```