BSS Praticum Week 1 Exercises

> Andre Nanninga
> 25-april-2014

Chapter 1

> *1.1 Explain the difference between protection and security. (1.7)*

From the book "Operating System Concepts with Java 8th edition" comes the following quote:

> Protection is strictly an *internal* problem: How do we proviode controlled access to programs and data stored in the computer system? Security, on the other hand, requires not only an adequate protection system but also consideration of the *external* environment within which the system operates. A protection system is ineffective if user authentication is compromised or a program is run by an unauthorized user
>
> -- Silberschatz, Galvin, Gagne

Protection is basicly making sure that data stored or the programs running on a system can only be access via secured ways. This is to protect the data or programs from being accessed by users or processes who do not have the proper privileges to access said data or programs.

Security on the other hand takes in consideration the external factors with which the system has to deal. This means for example that it tries to ensure that user authentication is effective.

Chapter 2

> *2.1 What is the main function of the command interpreter? (2.3)*

The primary objective of a command interpreter is to get and execute a command given by the user. An example of a command is `mkdir Documents` which would be interpreted as the command interpreter as the method `mkdir` with the parameter `Document`. The method `mkdir` would then in turn create a directory called `Document`.

> *2.2 Describe three general methods for passing parameters to the operating system (2.13) system calls (2.8)*

The simplest way to pass parameters to the operating system is by using registers. To use registers you would directly write your parameter in a register. There are however limited amount of registers and thus the possibility can occur that not enough registers are available for the amount of variables

Another method is to place the variables in a block and then pass the block position in as a parameter in a register.

A third method is to use the stack. A program can *push* parameters in the stack and the operation system can subsequently *pop* the parameters for use.

Chapter 3

> *3.1 What is the difference between a process and a program? (3.1)*

A process is simply said a program in execution.

> *3.2 What are the five process states that defines the current activity of a process (3.2)*

From the book "Operating System Concepts with Java 8th edition" comes the following quote:

> - **New**. The process is being created

> - **Running**. Instructions are being executed
> - **Waiting**. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
> - **Ready**. The process is waiting to be assigned to a processor.
> - **Terminated**. The process has finished execution.
>
> -- Silberschatz, Galvin, Gagne
>
> *3.3 What is the difference between a process and a thread? (3.3)*

A process consists per default of a single thread in which the instructions are executed. A process can contain multiple threads in an effort to execute multiple tasks simultaneously.

> *3.4 What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches? (3.10)*

One model is Shared-Memory Systems. This model allows two (or more) processes to access the same block(s) of memory. A strength of the Shared-Memory System is the lack of overhead. While there is some overhead to setup the shared memory when it is in place the process can simply write and read as normally without any overhead. A weakness is that to processes themself must ensure that they won't write in the same block at the same time.

The other model to use is a Message-Passing system. This model involves an object which controls the sending and receiving of data between processes. This involves some overhead but is much safer in terms of data preservation.

> *3.5 Describe the actions taken by a kernel to context-switch between processes. (3.17)*

When a context-switch occurs the context of the old process is saved in the PCB (the Process Control Block). The context of another process is loaded from this same PCB and the new process is set to run.

This process is pure overhead as the system is not doing anything useful during a context-switch

Programming execersise

Server.java

```java
import java.io.IOException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;

/**
 * Created by Andre Nanninga on 25-4-14.
 */
public class Server {

    private int port = 6052;
    private ServerSocket socket;
    private ArrayList<Connection> connections;
    private Boolean isDebug = true;

    public Server() {
        connections = new ArrayList<Connection>();

        try {
            createServerSocket();
            listenToServerSocket();
        }
        catch (IOException e) {
            System.err.println("Critical error: " + e.getMessage());
            System.exit(0);
        }
    }

    /**
     * Create a server socket
     *
     * @throws IOException
     */
    public void createServerSocket() throws IOException {
        debug("server/createServerSocket", "Creating serverSocket with port: " + port);
        socket = new ServerSocket(port);
    }

    /**
     * Set the server to listen to clients
     *
     * @throws IOException
     */
    public void listenToServerSocket() throws IOException {
        while(true) {
            Socket client = socket.accept();
            debug("server/listenToServerSocket", "accepting client: " + client.getInetAddress());

            createConnection(client);
        }
    }

    /**
     * create a new connection
     *
     * @param client The socket of the client of the connection
     * @throws IOException
     */
    private void createConnection(Socket client) throws IOException {
        Connection connection = new Connection(this, client);
        new Thread(connection).start();
        connections.add(connection);
    }

    /**
     * Close the connection and remove it from the connections pool
     *
     * @param connection The connection to close and remove
     * @throws IOException
     */
    private void closeConnection(Connection connection) throws IOException {
        connection.close();
        connections.remove(connection);
    }

    /**
     * Get the ip of a hostname
     *
     * @param hostname The hostname
     * @return The resolved ip of the hostname
     */
    public String getIpByHostname(String hostname) {
        try {
            InetAddress hostAddress = InetAddress.getByName(hostname);
            return hostAddress.getHostAddress();
        }
        catch (UnknownHostException e) {
            debug("Server/getIpByHostname", "could not resolve: " + hostname);
            return "Unable to resolve host " + hostname;
        }
    }

    /**
     * Print a message to the console when isDebug is set to true
     *
     * @param location Location of the code where the message was debugged
     * @param message Message of the debug
     */
    public void debug(String location, String message) {
        if(isDebug) {
            System.out.println("[" + location + "] - " + message);
        }
    }

    public static void main(String[] args) {
        new Server();
    }
}
```

Connection.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

/**
 * Created by Andre Nanninga on 25-4-14.
 */
public class Connection implements Runnable {

    private Socket socket;
    private Server server;

    private BufferedReader reader;
    private PrintWriter writer;

    private Boolean isRunning = true;

    /**
     * A connection of a client
     *
     * @param server The server parent
     * @param socket The socket of the client
     * @throws IOException
     */
    public Connection(Server server, Socket socket) throws IOException {
        this.server = server;
        this.socket = socket;

        writer = new PrintWriter(socket.getOutputStream(), true);
        reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }

    /**
     * Close the connection
     *
     * @throws IOException
     */
    public void close() throws IOException {
        isRunning = false;
        socket.close();
    }

    @Override
    /**
     * Reads and writes messages from the client
     */
    public void run() {
        String line = "";
        while(isRunning) {

            try {
                line = reader.readLine();
            }
            catch (IOException e) {
                server.debug("Connection/Run", "Error while reading line: " + e.getMessage());
                try {
                    close();
                }
                catch (IOException e1) { }
            }

            if(line != null) {
                server.debug("Connection/Run", "Received: " + line);
                String ip = server.getIpByHostname(line);

                writer.println(ip);
            }
        }
    }
}
```

Client.java

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

/**
 * Created by Andre Nanninga on 25-4-14.
 */
public class Client {

    private String serverHost = "localhost";
    private int serverPort = 6052;

    private Socket socket;
    private BufferedReader reader;
    private PrintWriter writer;

    private Boolean isDebug = true;

    /**
     * The host to resolve the ip of
     *
     * @param host
     */
    public Client(String host) {
        String message = host;
        Boolean listening = true;

        try {
            socket = new Socket(serverHost, serverPort);
            writer = new PrintWriter(socket.getOutputStream(), true);
            reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            debug("Client/Client", "sending message: " + message);
            writer.println(message);

            String line;
            while(listening) {
                if((line = reader.readLine()) != null) {
                    debug("Client/Client", "received message: " + line);
                    listening = false;
                }
            }
            socket.close();
        }
        catch (IOException e) {
            System.err.println("Critical error: " + e.getMessage());
            System.exit(0);
        }
    }

    /**
     * Print a message to the console when isDebug is set to true
     *
     * @param location Location of the code where the message was debugged
     * @param message Message of the debug
     */
    public void debug(String location, String message) {
        if(isDebug) {
            System.out.println("[" + location + "] - " + message);
        }
    }

    public static void main(String[] args) {
        if(args.length == 1) {
            new Client(args[0]);
        }
        else {
            System.err.println("Expected host as first argument");
            System.exit(0);
        }
    }
}
```