Practicum Week 1 Exercises

Chapter 1

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

> *Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system. This mechanism must provide a means for specifying the controls to be imposed, together with a means of enforcement. We distinguish between protection and security, which is a measure of confidence that the integrity of a system and its data will be preserved.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 637*

> *Protection is strictly an internal problem: How do we provide controlled access to programs and data stored in a 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. Computer resources must be guarded against unauthorized access, malicious destruction or alteration, and accidental introduction of inconsistency. These resources include information stored in the system (both data and code), as well as the CPU, memory, disks, tapes, and networking that are the com- puter.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 667*

Protection is strictly an internal problem, while security also takes external problems into account. In the computer world protection mainly refers to controlling the access of programs, processes, or users to the resources defined by a computer system. While security is the more broad concept of protecting a system.

Chapter 2

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

> *There are several ways for users to interface with the operating system. One provides a command-line interface, or command interpreter, that allows users to directly enter commands to be performed by the operating system.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 52*

To allow users to directly enter commands to be performed by the operating system.

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

> *Three general methods are used to pass parameters to the operating system. The simplest approach is to pass the parameters in registers. In some cases, however, there may be more parameters than registers. In these cases, the parameters are generally stored in a block, or table, in memory, and the address of the block is passed as a parameter in a register. This is the approach taken by Linux and Solaris. Parameters also can be placed, or pushed, onto the stack by the program and popped off the stack by the operating system. Some operating systems prefer the block or stack method because those approaches do not limit the number or length of parameters being passed.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 58*

1. Registers
   The simplest and fastest approach is to pass variables through registers. In this way the parameter values is near to the CPU, so it can be accessed fast.
2. Memory block or table
   When there are more parameters than there registers, the parameters are usually placed in memory.
3. Stack
   Parameters can also be placed on the stack. They can then be *popped* of by the operating system.

Chapter 3

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

> *Early computer systems allowed only one program to be executed at a time. This program had complete control of the system and had access to all the system's resources. In contrast, current-day computer systems allow multiple programs to be loaded into memory and executed concurrently. This evolution required firmer control and more compartmentalization of the various programs, and these needs resulted in the notion of a process, which is a program in execution. A process is the unit of work in a modern time-sharing system.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 103*

A process is a program in execution.

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

> *As a process executes, it changes state. The state of a process is defined in part by the current activity of that process. Each process may be in one of the following states:*
>
> - **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.
>
> *These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems, however. Certain operating systems also delineate process states more finely. It is important to realize that only one process can be running on any processor at any instant. Many processes may be ready and waiting, however.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 105*

The five states are: new, running, waiting, ready, and terminated.

3.3 What is the difference between a process and a thread? (3.3)

> *The process model implies that a process is a program that performs a single thread of execution. For example, when a process is running a word-processing program, a single thread of instructions is being executed. This single thread of control allows the process to perform only one task at a time. The user cannot simultaneously type in characters and run the spell checker within the same process, for example. Many modern operating systems have extended the process concept to allow a process to have multiple threads of execution and thus to perform more than one task at a time. On a system that supports threads, the PCB is expanded to include information for each thread.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 107*

A process is the main thread of execution and can start multiple threads for parallel execution. This allows a graphical user interface to respond to clicks even when a background thread is busy doing some other work.

Also a process has a copy of the entire memory of the parent process, a thread shares this memory with the parent process.

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

> *Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data and information. There are two fundamental models of interprocess communication: (1) shared memory and (2) message passing. In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region. In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 120*

The two models for interprocess communication are shared memory and message passing. Shared memory allows processes to share a region of memory, which the process can both access and thus communicate via. When using message passing as information sharing technique, the kernel can be used to send direct messages (with pieces of data) from one process to another.

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

> *Each process is represented in the operating system by a process control block (PCB)—also called a task control block. A PCB is shown in Figure 3.3. It contains many pieces of information associated with a specific process, including these:*
>
> - **Process state.** The state may be new, ready, running, waiting, halted, and so on.
> - **Program counter.** The counter indicates the address of the next instruction to be executed for this process.
> - **CPU registers.** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
> - **CPU-scheduling information.** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
> - **Memory-management information.** This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.
> - **Accounting information.** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
> - **I/O status information.** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 106*

> *Switching the CPU to another process requires performing a state save of the current process and a state restore of a different process. This task is known as a context switch. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run. Context-switch time is pure overhead, because the system does no useful work while switching. Context-switching speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers). Typical speeds are a few milliseconds.*
>
> *— Silberschatz, Galvin, Gagne, 2010, p. 112*

The kernel first saves the state of the running process, this state is saved in a Process Control Block (PCB). This stops the process. Finally it (reloads or creates the state of the other process and starts executing it. Loading this state (or PCB) overwrites the state of the other proces, but since that one was stored it can be restored later (when the next context switch occurs).

Programming Exercise

The Server

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

public class Server
{
	public static void main( String[] args ) {
		try {
			// Setup server socket
			int port = 6052;

			System.out.println( "Listening on port " + port + ".." );

			ServerSocket sock = new ServerSocket( port );

			// Accept socket connections
			while( true ) {
				Socket client = sock.accept();

				// Handle incoming connection in seperate thread
				Thread thread = new Thread( new Server.Handler( client ) );
				thread.start();
			}
		}
		catch( IOException ioe ) {
			System.err.println( ioe );
		}
	}

	public static class Handler implements Runnable {
		private Socket sock;

		public Handler( Socket sock ) {
			this.sock = sock;
		}

		public void run() {
			try {
				// Get input/output streams
				PrintWriter pout  = new PrintWriter( sock.getOutputStream(), true );
				InputStream in    = sock.getInputStream();
				BufferedReader bin = new BufferedReader( new InputStreamReader( in ) );

				// Read lines (data) from socket
				String line;

				while( ( line = bin.readLine() ) != null ) {
					try {
						// Lookup given address
						System.out.println( "~~ Looking up " + line );

						InetAddress hostAddress = InetAddress.getByName( line );
						String address          = hostAddress.getHostAddress();

						// Output found IP address
						System.out.println( "Resolved " + line + " to " + address );
						pout.println( address );
					}
					catch( UnknownHostException uhe ) {
						// IP address not found
						System.err.println( "Error resolving host " + line );
						pout.println( "Unable to resolve host " + line );
					}
				}

				sock.close();
			}
			catch( IOException ioe ) {
				System.err.println( ioe );
			}
		}
	}
}
```

The Client

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

public class Client
{
	public static void main( String[] args ) {
		try {
			// Get input/output streams
			Socket sock       = new Socket( "127.0.0.1", 6052 );
			PrintWriter pout  = new PrintWriter( sock.getOutputStream(), true );
			InputStream in    = sock.getInputStream();
			BufferedReader bin = new BufferedReader( new InputStreamReader( in ) );

			// Use given addresses (or default if none given)
			String[] addresses = ( args.length > 0 ? args : new String[] { "google.nl" } );

			// Process addresses
			for( int i = 0; i < addresses.length; ++i ) {
				// Write to socket
				pout.println( addresses[i] );

				// Output result from server
				System.out.println( bin.readLine() );
			}

			sock.close();
		}
		catch( IOException ioe ) {
			System.err.println( ioe );
		}
	}
}
```

Usage

The Server and Client programs can be run by executing the following in a command-line tool:

1. `javac Server.java && java Server;`
2. In another tab: `javac Client.java && java Client [<host>, ...]` (optional list of hosts, by default 'google.nl').

Tested with:

```
java version "1.6.0_65"
Java(TM) SE Runtime Environment (build 1.6.0_65-b14-462-11M4609)
Java HotSpot(TM) 64-Bit Server VM (build 20.65-b04-462, mixed mode)
```

By Maurits van Mastrigt (398497), 25th of april 2014