Segurança e Confiabilidade

2016/2017

Introdução

Segurança

- Página da cadeira (https://moodle.ciencias.ulisboa.pt/)
  - Avisos
  - Previsão das aulas
  - Bibliografia
  - Material de apoio
  - Horário de dúvidas
  - Regras de avaliação
  - Grupos de discussão
  - ....
- Inscrição em grupos
  - Grupos de dois/três alunos da mesma turma prática
  - Inscrição na página da disciplia

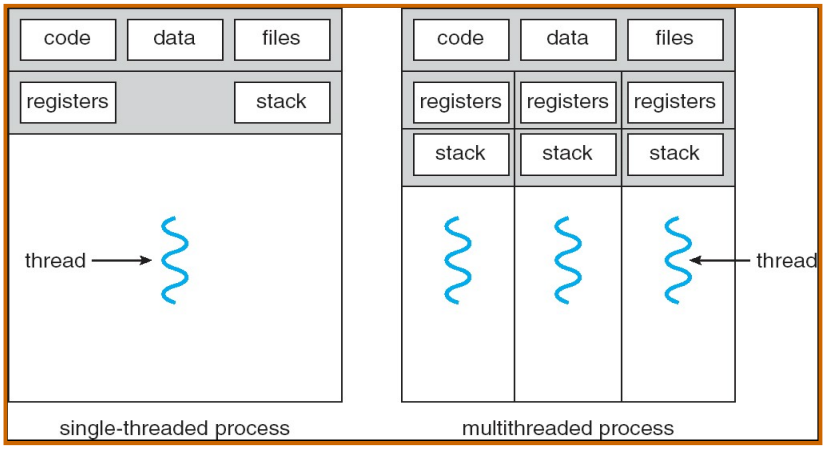
Sumário

- Tópicos úteis para a realização do projeto
  - *Threads*
  - *Sockets*
  - *Streams*
  - *Ficheiros*

- Porque usamos java
  - Problemas de segurança do C

Thread (fio de execução): conceito

- Threads vs Processos



Threads: concretização em Java

http://docs.oracle.com/javase/tutorial/essential/concurrency

- Criação de Threads
  - Criar uma subclasse da classe `Thread` ou
  - Implementar a Interface `Runnable`
- Em ambos os casos:
  - Implementar o método `run`
- Iniciar a execução de uma Thread
  - O método `start` cria os recursos do sistema necessários à execução da Thread (por exemplo, memória), escalona a Thread e invoca o método run.
  - O método `run` nunca é invocado directamente pelo programador
- Terminar a execução de threads
  - O método `stop` da class Thread está "deprecated".
  - A thread deve implementar uma forma segura de terminar (existem exemplos em **http://download.oracle.com/javase/tutorial/essential/concurrency/threads.html**)
- Sincronização de Thread – Cuidado !!
  - ver **http://download.oracle.com/javase/tutorial/essential/concurrency/threads.html**

Threads: exemplos

```
class ThreadsExample extends Thread {
        private String msg = null;

        ThreadsExample(String m) {
           msg = m;
        }

        public void run(){
           for (int i=0; i < msg.length(); i++) {
              System.out.println(msg.charAt(i));
              try {
                   sleep((int)(Math.random()*100));
              } catch (InterruptedException e) {
                   System.err.println(e);
              }
           }
           System.out.println();
        }
}

public class CallThreadsExample {
   public static void main(String[] args) {
        for (int i=0; i < args.length; i++){
           ThreadsExample newThread =
                        new ThreadsExample(args[i]);
           newThread.start();
           System.out.println("fim no main");
        }
   }
}
```

```
class ThreadsExample implements Runnable {
        private String msg = null;

        ThreadsExample(String m) {
           msg = m;
        }

        public void run(){
           for (int i=0; i < msg.length(); i++) {
              System.out.println(msg.charAt(i));
              try {
                   Thread.sleep((int)(Math.random()*100));
              } catch (InterruptedException e) {
                   System.err.println(e);
              }
           }
           System.out.println();
        }
}

public class CallThreadsExample {
   public static void main(String[] args) {
        for (int i=0; i < args.length; i++){
           ThreadsExample newThread =
                   new ThreadsExample(args[i]);
           new Thread(newThread).start();
           System.out.println("fim no main");
        }
   }
}
```


Threads: classes *Timer* e *TimerTask*

- Úteis para a concretização de tarefas periódicas ou para escalonar tarefas futuras de forma mais simples que usando threads
- A classe `java.util.TimerTask` representa uma tarefa
  - Implementa `Runnable` e tem um método abstrato `run`, logo é muito parecida com a `Thread`.
  - Existe um método `cancel()` que serve para cancelar a execução da tarefa, se escalonada.
- A classe `java.util.Timer` permite o escalonamento de tarefas periódicas ou não através dos métodos:
  - `schedule(TimerTask task, Date time)`: define que *task* deve ser executada uma única vez em *time*.
  - `schedule(TimerTask task, long delay, long period)`: define que *task* deve ser executada após *delay* ms e repetida a cada period ms após seu término.
  - `scheduleAtFixedRate(TimerTask task, long delay, long period)`: define que *task* deve ser iniciada a cada *period* ms após *delay* ms.

Sockets (TCP)

http://docs.oracle.com/javase/tutorial/networking/

- Definição:
  - *A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program.*
- Operações a concretizar num cliente
  - Open a socket.
    ```
    Socket echoSocket = new Socket("taranis.di.fc.ul.pt", 7);
    ```
  - Open an (object) input stream and (object) output stream to the socket.
    ```
    ObjectInputStream in = new ObjectInputStream(echoSocket.getInputStream());
    ObjectOutputStream out = new ObjectOutputStream(echoSocket.getOutputStream());
    ```
  - Write to and read from the stream according to the protocol.
    ```
    out.writeObject(userInput);
    String fromServer = (String) in.readObject());
    ```
  - Close the streams.
    ```
    out.close();
    in.close();
    ```
  - Close the socket.
    ```
    echoSocket.close();
    ```


Sockets (TCP)

- Operações a concretizar num servidor
  - Create a socket to listen on a specific port
    ```
    Server serverSocket = new ServerSocket(4444);
    ```
  - Accepting a connection from a client
    ```
    Socket clientSocket = serverSocket.accept();
    ```
  - Open an input stream and output stream to the socket.
    ```
    ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());
    ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
    ```
  - Read from and write to the stream according to the protocol.
    ```
    String fromClient = (String) in.readObject());
    out.writeObject(answer);
    ```
  - Close the streams.
    ```
    out.close();
    in.close();
    ```
  - Close all sockets.
    ```
    clientSocket.close();
    serverSocket.close();
    ```

Para atender vários clientes:
while (true) {
  accept a connection ;
  create a thread to deal with the client ;
  (ou usa uma threadpool)
}

Streams

http://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html

- ByteStreams
  - Programs use *byte streams* to perform input and output of 8-bit bytes.
  - byte stream classes are descended from InputStream and OutputStream.
  - Exemplo: file I/O byte streams, FileInputStream and FileOutputStream
- CharacterStreams
  - automatically translates the Unicode internal format to and from the local character set
  - Exemplo: file I/O: FileReader and FileWriter
- BufferedStreams
  - does io in units that are a line
  - Vantagens – ver **http://docs.oracle.com/javase/tutorial/essential/io/buffers.html**
  - Classes to wrap unbuffered streams:
    Buffered **byte** streams: BufferedInputStream and BufferedOutputStream
    Buffered **character** streams: BufferedReader and BufferedWriter

Streams – cont.

- DataInputStream, DataOutputStream
  - Transferência apenas de dados primitivos num formato independente do hardware (cp. "formato da rede" nas aulas de sistemas distribuídos)

```
DataOutputStream out = new DataOutputStream(clSocket.getOutputStream());
out.writeInt(17);
out.writeFloat(3.1415);
out.writeByte('x');

DataInputStream in = new DataInputStream(clSocket.getInputStream());
int x = in.readInt();
float f = in.readFloat();
byte b = in.readByte();
```

- ObjectInputStream, ObjectOutputStream
  - Transferência de dados complexos (Objects) num formato complexo (inclui tipo do objecto e todos os seus atributos)

```
ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
String userInput = ...;
out.writeObject(userInput);

ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());
String fromClient = (String) in.readObject());
```


Ficheiros

- Provavelmente já conhecem de disciplinas anteriores… as classes fundamentais:
- `File`: representam um nome dentro do sistema de ficheiros, servindo portanto para representar ficheiros, diretorias, links, etc.
  - Método `list()` permite obter a lista de ficheiros numa diretoria
  - Método `lastModified()` permite saber quando o ficheiro foi modificado pela última vez
- `FileInputStream`: input stream (binário) básico para leitura de dados em ficheiros. Assim como nos sockets, pode ser composto com outros (e.g., ObjectInputStream para ler objetos serializados em ficheiros).
- `FileOutputStream`: output stream (binário) básico para escrita de dados em ficheiros. Também pode ser composto.

Porque usamos Java?

Problemas de Segurança em C

Importância de verificar limites de arrays

- C e C++ não fazem essa verificação automaticamente
- Quando o programador não a faz, o programa pode ficar vulnerável a ataques por *buffer overflow:* o top 1 das vulnerabilidades na Internet!

- Exemplo:
  - Code-Red
  - explorou vulnerabilidade buffer overflow no Microsoft Internet Information Server (IIS)

On July 19, 2001, more than 359,000 computers connected to the Internet were infected with the Code-Red (CRv2) worm in less than 14 hours. The cost of this epidemic, including subsequent strains of Code-Red, is estimated to be in excess of $2.6 billion.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//Function performs a copy into variable var and returns.
int copy(char* input) {
        char var[20];
        strcpy (var, input);
        return 0;
}

// Function 'hacked' prints out a string to the console, is not called anywhere
int hacked(void) {
        printf("Can you see me now ?\n");
        exit(0);
}

int main(int argc, char* argv[]){
        // commandline arguments are provided to the executable
        if(argc < 2) {
                printf("Usage: %s <string>\r\n", argv[0]);
                exit(1);
        }
        //prints the address of function hacked onto the console.
        printf("Address of function: 0x%08x\n", (unsigned int) hacked);
        //passes argument 1 to the function copy.
        copy(argv[1]);
        return 0;
}
```


O problema em C: O que é que este código faz?

[compilar]
gcc sample.c -o sample

[executar]
./sample a
(imprime endereço da função. ex: 0x0040106f)

./sample AAAABBBBCCCCDDDDEEEEFFFF
*** stack smashing detected ***: ./sample terminated

[compilar]
gcc -fno-stack-protector sample.c -o sample

[executar]
./sample $'AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH\xcc\x84\x04\x08'
(executa função hacked)
Can you see me now ?

O problema em C: O que é que este código faz? (2)

- Do man da bash:

  "**Words of the form $'string' are treated specially. The word expands to string, with backslash escaped characters replaced as specified by the ANSI C standard. Backslash escape sequences, if present, are decoded as follows:**

  ...

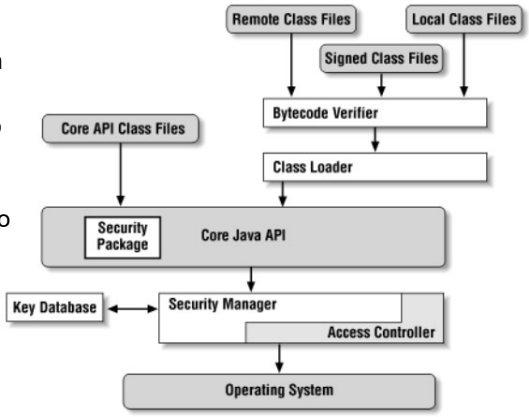
  \xHH the eight-bit character whose value is the hexadecimal value HH (one or two hex digits)"

- Assembly gerado pelo gcc na nemo.alunos:

```
copy:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    8(%ebp), %eax
        movl    %eax, 4(%esp)
        leal    -20(%ebp), %eax
        movl    %eax, (%esp)
        call    strcpy
        movl    $0, %eax
        leave
        ret
```


Segurança no Java: Arquitectura

- ***Verificador de bytecodes*** – garante que os ficheiros class obedecem a um conjunto de regras
- ***Carregadores de classes*** – carregam classes Java
- ***Controlador de acesso*** – faz controlo de acesso aos recursos do sistema
- ***Gestor de segurança*** – principal interface entre a API e o sistema (usa o controlador de acesso)
- ***Package de segurança*** – algoritmos de segurança (cripto, assinaturas,...) e mecanismos para adicionar outros
- ***Bases de dados de chaves*** *(keystores)* – armazenam chaves criptográficas (geralmente são ficheiros)



Exercício

- 1. Considere o servidor <myServer>. Fazer o respetivo cliente.
- 2. Alterar o cliente e o servidor de modo a, após a autenticação, enviar um ficheiro do cliente para o servidor.
  Sugestões:
  a) usar os mesmos ObjectOutputStream e ObjectInputStream
  b) usar os métodos write(byte[] buf, int off, int len) e read(byte[] buf, int off, int len)
  c) enviar previamente a dimensão do ficheiro
- 3. Executar o exemplo do programa em C dos acetatos da aula <hacked.c>.
  bash-4.2$ ./a.out $'aaaabbbbccccddddeeeeffffgggghhhh**\x73\x84\x04\x08**'
  Address of function: **0x08048473**
  Can you see me now ?

