Pale INF422 – Composants d’un système informatique
INF422 Final Exam – Components of a Computing System

2011–2012

- L’examen dure 3 heures.
  The exam lasts 3 hours.
- Tous documents autorisés.
  All documents are authorized.
- Il est impératif de commenter les programmes et de justifier vos réponses.
  Programs must be commented and every answer must be justified.

English text

1 killall

The goal of this exercise is to implement as a Shell script a (simplified) version of the `killall` command.

Our simplified version of `killall` takes a signal number as an option (between 1 and 64, preceded by a dash `-`), and the name of a program as a mandatory argument. The signal is sent to all processes executing a program with this name.

Witnout the optional signal, number 15 is used by default; this is also the behavior of the `kill` command.

As a reminder, the `/proc` directory is associated with the pseudo-file-systeme of the same name, which exposes an internal data base of the Linux kernel through directories and files. In particular, each process appears as a sub-directory of `/proc` in which multiple files describe the characteristics of the process. Thus, `/proc/42/cmd` contains the command line which started the execution of process 42. For example, if process 42 executes `ls -l`, the file `/proc/42/cmd` will contain the string ≪`ls -l`≫.

Question 1.1

Write a command to list all the sub-directories of `/proc` whose name is a process number.

Question 1.2

What does `#!/bin/sh` mean as the first line of a shell script?

Question 1.3

Write a shell script called `name_to_pid` taking a unique argument, and printing the number of each process whose name matches the argument.

Question 1.4

Write a shell script to test if the first argument is an optional signal number (it starts with a dash and contains only digits), then defines a variable called `sig` to this number if present, and to 15 otherwise.

Question 1.5

Write the shell script implementing our simplified `killall` using the previous script `name_to_pid`.

You may use the `shift` function of the shell: `shift` n renumbers the arguments: argument $n+1$ becomes argument 1, argument $n+2$ becomes argument 2, etc.

Question 1.6

Le fichier `/proc/42/status` contient un certain nombre de lignes de texte décrivant l’état du processus 42. One of these has the format ≪`Uid:  1000`≫ if process 42 belongs to the user whose UID (user identification number) is 1000. Modify the `killall` script to check that the process belongs to the user executing the script, before sending it the signall. You may use the `id -u` command which prints the UID of the user.

Question 1.7

We wish to add a `-e` option to the `killall` script to indicate that the program name should be interpreted as a regular expression, with the syntax and semantics of the regular expressions used by `grep`, `sed`, etc.

2 Instant messaging robot

We build upon the networking protocol introduced at TD3.

A server process awaits connexions on port 7777, on a machine whose DNS is `chat.polytechnique.fr`. The clients connect on this port opening a socket according to the TCP protocol (as described in the course in Java and with the `nc` command). As soon as the connection is established, the client sentds the message ≪`LOGIN nickname`≫ to the server, which broadcasts to all connected clients ≪`Welcome nickname`≫. The client is then able to post, sending the message text prefixed by ≪`SEND`≫.

Question 2.1

One of the clients executes the following shell script, connected to the server via the `nc` command.

```
#!/bin/sh

echo LOGIN DummyBot
while true; do
  read s
  if [ "`echo $s | grep -v Bot`" ]; then
    text="What did you say?"
    echo SEND $text
  fi
done
```

What is this script doing? Explain what it does line-after-line.

Question 2.2

We now wish to implement this "robot" client as an extension of the Android application designed at TD3. We add un button labeled ≪Bot≫ to toggle the application into the "robot" mode, with the behavior of the previous script. When the user manually send a message with the interface of TD3, the client resumes its normal operation (sending messages manually). Describe without writing actual Java code the required modifications to the graphical interface and the associated management of the graphical interface's events (mouse clicks, threads, etc.).

Question 2.3

Assuming the abovementionned changes to the graphical interface have taken place, implement the Java equivalent of the script above.

You are strongly advised to use the `Net` class introduced at TD3.

Question 2.4

We use a Java class `AI` (Artificial Intelligence) with a static method `answer()`. It takes a string argument and returns a string "response" to it.

Modify the previous program (no need to copy everything) to test if the text received by the client ends with an interrogation mark (`?`), and if so, call the `answer()` method on this "question", and post the returned string.

Question 2.5

The `answer()` method involves an expensive computation, which may take a few seconds to return an "intelligent" response. Without writing Java code, explain how to let this method execute in the background while leaving the client react to a possible click by the user to return to the manual mode, while continuing to display the messages broadbast by the server, and while posting the message ≪`Thinking...`≫ as a response as long as the `answer()` did not complete.

Question 2.6

Modify the program (no need to copy everything) to implement the mechanism described in the previous question.

3 Encryption of a file or a file system

We wish to set up a filter to encrypt a file or a file system, such that the operations of this filter are fully transparent to the existing programs and their input/output. The first questions consider the encryption of a single file, while the last two deal with a complete file system.

Question 3.1

Explain (no code needed) how to create a block special device file called `/dev/cipher` which will be used to encrypt/decrypt automatically any data written/read on the file.

Question 3.2

List two mechanisms provided by the kernel of an operating system allowing a dedicated (virtual) device driver `/dev/cipher` to implement the specific read/write accesses on this special file.

Question 3.3

An interface provided by the kernel allows to configure this driver, implemented as a Linux module. This interface allows for example to parameterize the encryption key and the name of the file to crypt. What is this interface? How do you propose to exploit it in this context (without writing code for now)?

Question 3.4

The proposed cipher algorithm is parameterized by an 8-bit key. Each byte of data written to the file `/dev/cipher` is reversed bitwise (bit 0 becomes bit 7, bit 1 becomes bit 6, etc.) then the 8 bit key is added modulo 256.

Write a Java method taking an array of bytes as an argument and a key, and returning an array of bytes of the same size encrypted with this cipher algorithm (ridiculously weak).

Question 3.5

Rather than adding the key modulo 256, which bijective opération would allow the decryption operation to be perfectly identical to the encryption (exactly the same Java method)?

Question 3.6

Which modifications should be brought to this system to encrypt and decrypt transparently an entire file system, such as `/dev/sdb1` for a USB stick?

Question 3.7

We wish to access via the `/media/usb` directory to the encrypted data, with 42 as a key. Which commands the administrator (`root`) should use for this?