Exercise 10 – Process Control

Objective

The objective of this session is to explore a number of process control issues, including the use of `system()`, `exec()`, and `signal()`.

Reference Material

This practical session is based entirely on material in the Process Control chapter. The session is located in the following directory:

	Microsoft Windows	*Linux*
Directory:	**c:\qacadv\process**	**~/qacadv/process**
Solution directory:	**c:\qacadv\process\Solution**	**~/qacadv/process/Solution**

Overview

In this session, the first question deals with the `system()` function. Question two uses system() again, but this time with an environment variable to control the child process. The last question deals with signal handling using `signal()`.

Practical Outline

1. **On Microsoft Windows** open **system.sln, on Linux** change your current working directory.

   Using the empty file called **system.c**, write a program that clears the screen then displays a list of files in the current directory. You will need to call `system()` twice.

   On Microsoft Windows use the "cls" command to clear the screen, and "dir" to list files.

   On Linux use the "clear" command to clear the screen, and "ls -l" to list files.

   A sample solution can be found in the **Solution** sub-directory, called **system.c**.

2. **On Microsoft Windows** open **processes.sln**, you will notice that there are two projects, **parent** and **child**. **On Linux** stay in the same working directory.

   This simple application should act as follows:

   > The **parent** calls the child program twice passing several names on the command line. Before each call it sets an environment variable, LC_COLLATE, to indicate the locale collation sequence. The first value of LC_COLLATE is to be “C” (the default on many systems), and the second should be “French”.
   >
   > The **child** sorts the passed names using the collation sequence passed in the environment variable, if there is one.

   Your job is to set and get the environment variable in the parent and child, and to invoke the child program from the parent. In the **parent**, the command to execute (using **system()**) is already set-up in string **szCmd**. Be sure to check the return value from **system()** each time. In the **child**, pick up the value of the environment variable, assigning it to the string **szEnvVar**. Setting the locale, and sorting, has been coded for you. If the environment variable does not exist return ERR_NO_ENV_VAR, and if it has no value return ERR_EMPTY_ENV_VAR. These are set in **child.h**.

   When you run the child you should get two different sort orders.

   On Microsoft Windows: if you use *Control+F5* to execute the program, Visual Studio displays the message "Press any key to continue" when the program terminates. This is usually helpful, but it's getting in the way of our own messages here. Run the program using just *F5* (i.e. without the *Control* key); this suppresses the unwanted message from Visual Studio Note also that the font on the console window does not display non-American characters correctly.

   On Linux you may have to alter the character coding for the terminal session to display the output correctly. Set it to “Western” 8859-1.

   A sample solution can be found in the **Solution** sub-directory.

3. This is the final question in the session, and deals with Control-C signal handling.

   On Microsoft Windows open **signal.sln, on Linux** stay in the same working directory.

   Take a look at the code in **signal.c**. The program sits in an infinite loop, displaying the same message time and again.

   Build and run the program. What happens if you try Control-C? By default, this causes the program to terminate immediately, without giving your program the chance to respond in a graceful manner.

Modify the program so that the user has to type Control-C 3 times before it has any effect, whereupon the program should display the message "I've been interrupted" and terminate gracefully.

Hints: use `signal()` to install a handler for `SIGINT`. In this handler, keep count of how many Control-C have taken place. If the count is 3, set some sort of flag that can be used to break the infinite loop in `main()`.

A simple solution can be found in the **Solution** sub-directory called **signal.c**.