

Practical Session 1: Intro to PS3 Development

Overview

This practical session introduces programming the “Cell Broadband Engine” processor in the PlayStation 3® console. The main purpose of this practical is to get you familiar with the development environment you will be using for further practical sessions and your coursework. As such, the programs you will write will be very basic and won’t exploit the PS3’s hardware capabilities.

If you are unfamiliar with using unix/linux, it is recommended that you download the “unix commands” file from the course materials wiki page.

Development Environment

The simplest way to edit files when using the PlayStation 3® is to use your favourite editor on the PC, e.g. Notepad++ (recommended) or Visual Studio. If you use Visual Studio, you do not need to create workspaces and projects, since none of the compiling & linking will be done by it – it is only used as a text editor. All compiling etc. will be done by tools running on the PS3. The PlayStation 3® consoles are not physically located in the labs, but instead, they are accessible over the network via SSH using the applications WinSCP and PuTTY, which *should be* installed in E113 and E116. Portable versions of WinSCP, PuTTY, and Notepad++ can be downloaded from <http://portableapps.com/> to be installed on a pen drive or your home folder if not present on the lab computers.

Start the WinSCP application from the start menu, and you will be presented with the following dialog:

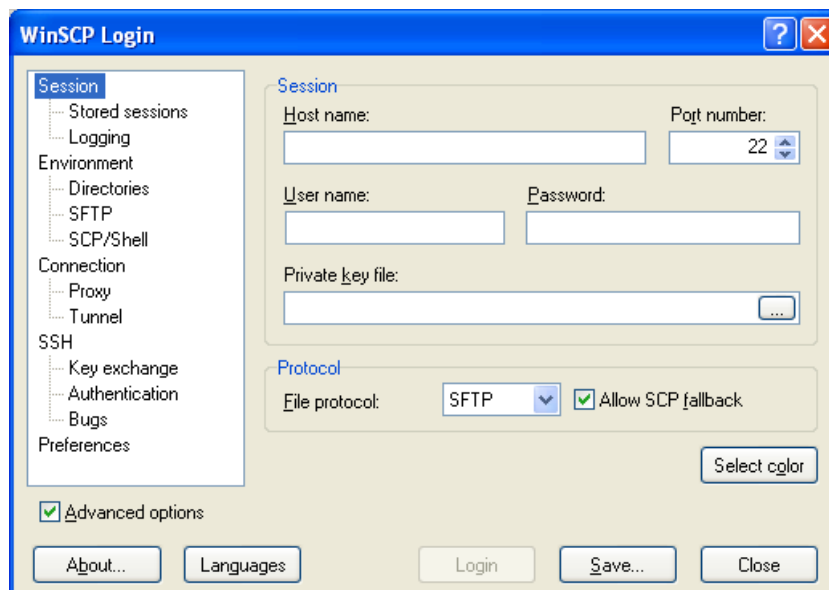


Figure 1: WinSCP login dialog

Enter the host name that has been provided to you and use your banner ID (with a lower case 'b') for the user name. You can also enter your password at this stage, but if you leave the field blank, you will be prompted for your password as the connection is made. You can save the login details in a profile, but it is not recommended that you include your password when saving; otherwise your account could be accessed by other users of the workstation that you are using.

After a successful login, you will be presented with a screen similar to the one shown in Figure 2. The panel on the right shows the contents of your home folder on the remote host. You can drag and drop files to and from this window to copy files to and from the PlayStation

Please note that backups of the home directories on the PlayStation 3 are not currently being kept, and therefore, you should copy all important files to your H: drive or another safe storage location. Default password is '123456' which should be changed on each PS3 console using the 'passwd' command (see below).

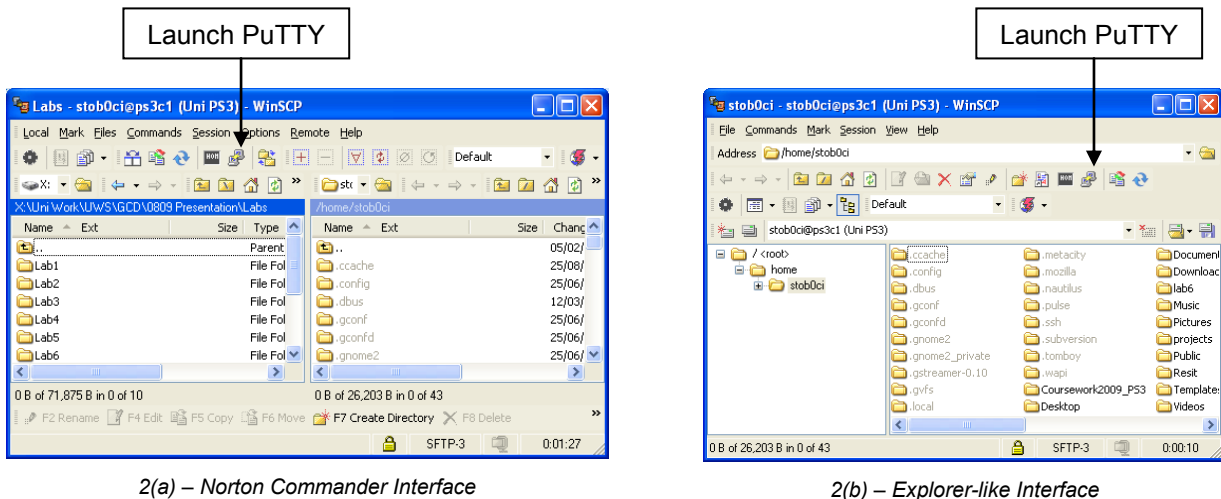


Figure 2: WinSCP connected to a remote host

You will need to use a command line for various things including building and running programs. To start a SSH session in PuTTY, click on the icon indicated in the screen shot above, and this will start the PuTTY application, which will automatically connect to the host that WinSCP is connected to. At this point you can change your password for the PS3 you are currently connected to by typing the command 'passwd' and following the prompts.

It is possible to edit remote files using a text editor and have them automatically uploaded to the remote host whenever they are changed. For this purpose, I recommend that you use the editor "Notepad++," which works better than Visual Studio when editing remote files. You will need to configure WinSCP to use Notepad++ as the default editor. In the "View" or "Options" menu, select "Preferences" and then click "Editors" in the preferences dialog. You can configure WinSCP to use Notepad++ as the editor for all files by

clicking on the existing file association, and editing it (See Figure 3.) The executable for Notepad++ is located at `C:\Program Files\Notepad++\notepad++.exe`

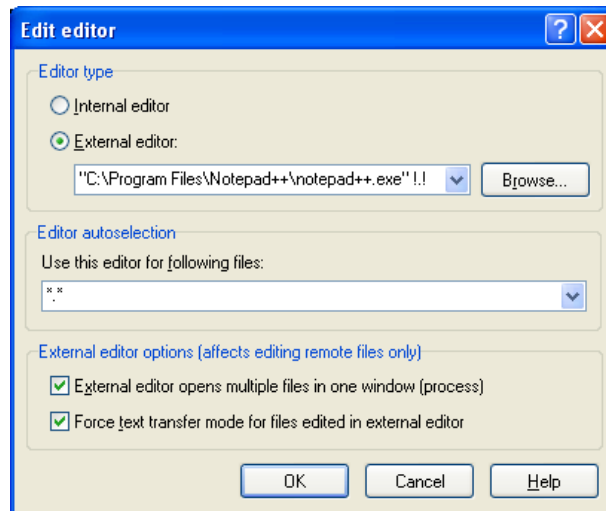


Figure 3: Setting up the WinSCP to use Notepad++

Building Programs

The Cell SDK includes some Makefiles that simplify compiling and linking of executables for the Cell. To begin with, we will work with a very trivial “Hello World” program, which will introduce you to the building environment.

Download the file `lab1.tar` from the course materials wiki and place it in your home directory (on the PS3). Then at the PuTTY prompt, enter the following command:

```
tar xvf lab1.tar
```

This will extract the contents of the tar file into a new directory called “lab1”. Inspect the code in file `lab1.cpp` and you will see that it is a familiar “Hello World” program. The Makefile is a little bit more interesting, but has two lines that are of particular interest. Firstly, the following line specifies the name of the executable file to be created:

```
PROGRAM_ppu := lab1_prog
```

In more complicated Makefiles, you may specify several executable files to be created, but one will suffice for now. A little further down, another line specifies the location of the program:

```
INSTALL_DIR = ~/lab1
```

This tells the Cell SDKs Makefiles where your project is located (the tilde “~” means the root of your home directory in unix / linux). The rest of the Makefile’s contents will include more complicated Makefiles provided by the Cell SDK. This part is standard, and is used in most projects that use the Cell SDK.

To build the program, change to the “lab1” folder at the command prompt, and enter the command “make”. If the program builds successfully, you will have an executable file named *lab1_prog*. Execute the program in the standard unix/linux way (*./lab1_prog*) and verify that the program works. Note that the source file *lab1.cpp* was not explicitly specified in the Makefile; instead, the Cell SDK’s Makefile will compile any source files that it finds in the specified folder.

Homework

The following work should be completed by next week’s practical session, and it will be assumed that you have done so. While you may attempt these exercises during this practical session, vector intrinsics and SIMD instructions will be covered during the lecture, so you may wish to wait.

Using PPE Intrinsics

Using the “hello world” program as a starting point, you will now extend the application to do something more interesting. You can, if you wish, take a separate copy of the previous exercise in a separate folder (e.g. *lab1b*), and use that as a starting point. If you do, be sure to update the Makefile to refer to the new location.

The Cell SDK provides C language intrinsics for the PPE and SPE cores alike. These are extensions to the C/C++ language, which allow programmers to take advantage of the features of the processor without having to resort to using assembly language. The SDK uses compilers that recognize these extensions. When building PPE programs, you may notice that the compiler used is not one of the ones commonly used in linux development (e.g. *cc*, *gcc*, *g++* etc.) but “*ppu32-g++*”. We will start with a simple example of using the additional features by adding two four-dimensional vectors together using the SIMD features of the PPE. In order to use the PPE intrinsics, you must *#include* the file “*altivec.h*”

Next, add the following union declaration to your code:

Lab1b.cpp

```
typedef union
{
    vector float vec;
    float f[4];
    struct
    {
        float x;
        float y;
        float z;
        float w;
    } e;
} vectorUnion;
```

Note the *vector* keyword. This is one of the language extensions provided by the Cell SDK, and it means that the member “vec” is a vector of four 32-bit floating point numbers. The “f” member of the union is useful since it allows us to set the individual components of the vector by treating it as an array of four floats. The following code creates three instances of the above union, and adds two of the vectors together, storing the result in the third.

Lab1b.cpp

```
vectorUnion vecX;
vecX.vec = (vector float) { 1.0,2.0,3.0,1.0 };
vectorUnion vecY;
vecY.vec = (vector float) { 1.5,2.5,3.5,1.0 };
vectorUnion result;
result.vec = vec_add(vecX.vec,vecY.vec); // add the vectors together
// display the result
printf("%.2f,%.2f,%.2f,%.2f\n",result.f[0], result.f[1], result.f[2], result.f[3]);
```

Save and compile your program, and verify that it adds the two vectors together correctly.

The “Cell Broadband Engine Programming Handbook” provides details of all language intrinsics and functions provided by the SDK. This file is available on the course materials wiki page (version 1.1). Download this document, and read section 2.6 (pages 60-62.) Then, refer to section 4 of Appendix A (pages 746-749), which provides details of various vector data types and associated intrinsic vector functions for SIMD operations. Then attempt to implement the following (the operations may not be straight-forward to do):

1. Multiply two vectors together, each containing four 32-bit signed integers
2. Multiply two vectors together, each containing sixteen 8-bit unsigned chars
3. Find the average of (midpoint) of two vectors containing four 32-bit floats
4. Find the maximum of two vectors containing four 32-bit floats

Chapter 8 of the recommended course book should also come in handy here.

Exercises: Optimising Vector-Matrix Operations

Now that you know how to perform SIMD operations, write some code that will transform a 4-dimensional vector by a 4x4 matrix using SIMD instructions.

Download the *vecmat.zip* file from the course materials wiki page. These files contain vector and matrix classes, written in standard C++. Re-write the class operations, so that they take advantage of the SIMD features of the Cell's PPE.