

Programming Project 1: Introduction to the SPANK Tools

Due Date: Monday, Oct. 3, 2005, start of class

Duration: One Week

Overview and Goal

In this course we will be creating and modifying parts of an operating system kernel. We'll be using the SPANK software tools, which were written for this task by Harry Porter. The goals of this project are to make sure that you can use the SPANK tools and to help you gain familiarity with them.

Step 1: Print the Documentation

There are a number of documents describing the SPANK tools. You may obtain the documents either by purchasing them from Clean Copy located on Broadway, or by going to the SPANK homepage:

`http://www.cs.pdx.edu/~harry/spank/index.html`

From there you can access **pdf** versions. Print out the following documents:

- An Overview of the SPANK System** (7 pages)
- An Overview of the SPANK Computer Hardware** (8 pages)
- The SPANK Architecture** (67 pages)
- Example SPANK Assembly Program** (7 pages)
- SPANK Instruction Set** (3 pages)
- The SPANK Emulator** (42 pages)
- An Overview of the Spankish Programming Language** (66 pages)
- Context-Free Grammar of Spankish** (7 pages)
- SPANK Tools: Help Information** (11 pages)
- The Format of SPANK Object and Executable Files** (13 pages)

Step 2: Read the Overview Document

Read the first document (“An Overview of the SPANK System”) before proceeding further.

Step 3: Choose Your Host Platform

The SPANK tools will run on either Apple Macintosh (OS X) or on the CS department's Sun computers (known as "sirius"). You may use either and you should decide now which will be most convenient for you. The tools have not been ported to Little Endian architectures (e.g., Intel-based PCs), so you cannot use a PC.

If you have access to another Sun computer, you may compile the tools on that machine and use it.

You will develop your operating system code on a "host" computer—either a Mac or a Sun—and you will be running the SPANK tools on that host computer.

The source code for all the SPANK tools is available, but you should not need to look at it. Nevertheless, it is available for anyone who is interested.

The SPANK Tools

Here are the programs that constitute the SPANK tool set.

| | |
|------------------------|--|
| <u>comp</u> | The Spankish compiler |
| <u>asm</u> | The SPANK assembler |
| <u>lddd</u> | The SPANK linker |
| <u>spank</u> | The SPANK machine emulator (the virtual machine and debugger) |
| <u>diskUtil</u> | A utility to manipulate the simulated SPANK "DISK" file |
| <u>dumpObj</u> | A utility to print SPANK .o and a.out files |
| <u>hexdump</u> | A utility to print any file in hex |
| <u>check</u> | A utility to run through a file looking for problem ASCII characters |
| <u>endian</u> | A utility to determine if this machine is Big or Little Endian |

These tools are listed more-or-less in the order they would be used. You will probably not need to use the last three tools and you may ignore them. (If interested, read the comments at the beginning of the source code files.)

Organization of the Material I am Making Available

I am providing material in the following directories on **cs.pdx.edu**:

```

~walpole/
  public_html/
    class/
      cs333/
        fall2005/
          spank/
            p1/
              ...files related to project 1...
            p2/
              ...files related to project 2...
          ...etc...
        spank/
          SpankDoc/
            ...files containing the documents mentioned above...
          SpankBin/
            Mac/
              ...executables for the Mac host platform...
            Sun/
              ...executables for the Sun host platform...
          SpankSrc/
            ...source code for the SPANK tools...

```

You may access this material through either the class webpage or the SPANK Home page. You should also be able to “ftp” directly to it, or if you are on the CS department’s Sun system, you can just access it from UNIX.

Keep in mind that when you use a web browser (such as Apple’s Safari), you leave out the “public_html” part of the path. For example:

<http://www.cs.pdx.edu/~walpole/class/cs333/fall2005/spank/SpankDoc/>

If I forget to set the access protection bits correctly for any class-related files, please let me know.

Step 4A: For Macintosh Users

If you are using the CS department’s Sun system for your assignments, skip to the section titled “For Sun/SPARC Users”.

Step 4A-1: Create a directory to put the SPANK tools into. For example, you may wish to create a directory called **SpankTools** in your home directory:

/Users/YourUserName/SpankTools

Then copy all the files from

www.cs.pdx.edu/~walpole//class/cs333/fall2005/spank/SpankBin/Mac

to your **SpankTools** directory. These are obviously binary files, not text files.

(You can use an application called “Fetch” (www.fetchsoftworks.com) to do “ftp” file transfers.)

Step 4A-2: Set the protection bits on these programs to include “executable”, with a command such as:

```
chmod ugo+rx SpankTools/*
```

Step 4A-3: Add the **SpankTools** directory to your search path. I am not exactly sure how OS X is setup initially. In my case, the file **.login** in my home directory contains this line:

```
source .aliases
```

The file **.aliases** contains a line like this:

```
setenv PATH .:${HOME}/bin:${PATH}:/usr/local/bin
```

For you, the exact details may differ. (Perhaps your **.aliases** file does not even set the **PATH** variable.) Between each colon (:) is a directory specification. What we want to do is add the SPANK tools directory in front of everything else. So add the following line after the last place **PATH** is set.

```
setenv PATH ${HOME}/SpankTools:${PATH}
```

The shell builds an internal hash table that speeds up the location of programs whenever you type a command. After changing your **PATH**, you’ll need to restart your shell so that it uses the new **PATH** when builds this hash table. (Quit the “Terminal” application and then start “Terminal” back up.)

Step 4B: For Sun/SPARC Users

If you are using the CS Department’s Sun system (Sirius), then all you need to do is modify your **PATH** variable. Alter your **.aliases** file. You may already have a line that looks something like this:

```
setenv PATH ${PATH}:${HOME}/bin
```

The exact details for you may differ. (Perhaps your **.aliases** file does not even set the **PATH** variable.) Between each colon (:) is a directory specification. What we want to do is add the SPANK tools directory in front of everything else. So add the following line after the last place **PATH** is set.

```
setenv PATH ~walpole/public_html/class/cs333/fall2005/spank/SpankBin/Sun:${PATH}
```

The reason for adding it in front is that there is a UNIX command “comp” to compose and send an e-mail. This conflicts with the SPANK “comp” command; you’ll want your shell to find the SPANK command, not the email command, whenever you type “comp”.

The shell builds an internal hash table that speeds up the location of programs whenever you type a command. After changing your PATH, you’ll need to restart your shell so that it uses the new PATH when builds this hash table. You can log out and log back in. (In C-Shell you can use the command “source .aliases” instead.)

Step 5: Verify the Tools Work

At the UNIX prompt, type the command.

```
comp
```

You should see the following:

```
***** ERROR: Missing package name on command line
```

```
***** 1 error detected! *****
```

If you see this, good. If you see anything else, then something is wrong.

Step 6: Set up a Directory for Project 1

Create a directory in which to place all files concerned with this class. I recommend the following name:

```
cs333
```

Create a directory in which to place the files concerned with project 1. I recommend the following name:

```
cs333/p1
```

Copy all files from:

```
~walpole/public_html/class/cs333/fall2005/spank/p1
```

to your **cs333/p1** directory. My files can also be accessed as:

```
http://www.cs.pdx.edu/~walpole/class/cs333/fall2005/spank/p1
```

The SPANK Assembly Language

In this course you will not have to write any assembly language. However, we will be using some interesting routines which can only be written in assembly. I will write these and provide them to you, but you will need to be able to read them.

Take a look at **Echo.s** and **Hello.s** if you want to see what SPANK assembly code looks like.

Step 7: Assemble, Link, and Execute the “Hello” Program

The **p1** directory contains an assembly language program called “Hello.s”. First invoke the assembler (the tool called “asm”) to assemble the program. Type:

```
asm Hello.s
```

This should produce no errors and should create a file called **Hello.o**.

The **Hello.s** program is completely stand-alone. In other words, it does not need any library functions and does not rely on any operating system. Nevertheless, it must be linked to produce an executable (“a.out” file). The linking is done with the tool called “ld”. (In UNIX, the linker is called “ld”.)

```
lddd Hello.o -o Hello
```

Normally the executable is called **a.out**, but the “-o Hello” option will name the executable **Hello**.

Finally, execute this program, using the SPANK virtual machine. (Sometimes I refer to it as the SPANK “emulator.”) Type:

```
spank -g Hello
```

The “-g” option is the “auto-go” option and it means begin execution immediately. You should see:

```
Beginning execution...
Hello, world!

**** A 'debug' instruction was encountered ****
Done! The next instruction to execute will be:
000080: A1FFFFB8      jmp      0xFFFFB8      ! targetAddr = main

Entering machine-level debugger...
=====
=====
===== The SPANK Machine Emulator =====
=====
===== Copyright 2001, Harry H. Porter III =====
=====
=====
```

```
Enter a command at the prompt. Type 'quit' to exit or 'help' for
info about commands.
>
```

At the prompt, quit and exit by typing “q” (short for “quit”). You should see this:

```
> q
Number of Disk Reads      = 0
Number of Disk Writes     = 0
Instructions Executed     = 1705
Time Spent Sleeping       = 0
Total Elapsed Time       = 1705
```

This program terminates by executing the **debug** machine instruction. This instruction will cause the emulator to stop executing instructions and will throw the emulator into command mode. In command mode, you can enter commands, such as **quit**. The emulator displays the character “>” as a prompt.

After the debug instruction, the **Hello** program branches back to the beginning. Therefore, if you resume execution (with the **go** command), it will result in another printout of “Hello, world!”.

Step 8: Run the “Echo” Program

Type in the following commands:

```
asm Echo.s
lddd Echo.o -o Echo
spank Echo
```

On the last line, we have left out the auto-go “-g” option. Now, the SPANK emulator will not automatically begin executing; instead it will enter command mode. When it prompts, type the “g” command (short for “go”) to begin execution.

Next type some text. Each time the ENTER/RETURN key is pressed, you should see the output echoed. For example:

```
> g
Beginning execution...
abcd
abcd
this is a test
this is a test
q
q
**** A 'debug' instruction was encountered ****
Done! The next instruction to execute will be:
      cont:
0000A4: A1FFFFAC   jmp   0xFFFFAC   ! targetAddr = loop
>
```

(For clarity, the material entered on the input is underlined.)

This program watches for the “q” character and stops when it is typed. If you resume execution with the **go** command, this program will continue echoing whatever you type.

The **Echo** program is also a stand-alone program, relying on no library functions and no operating system.

The Spankish Programming Language

In this course, you will write code in the “Spankish” programming language. Begin studying the document titled “An Overview of the Spankish Programming Language”.

Step 9: Compile and Execute a Spankish Program called “HelloWorld”

Type the following commands:

```
comp -unsafe System
asm System.s
comp HelloWorld
asm HelloWorld.s
asm Runtime.s
ldd Runtime.o System.o HelloWorld.o -o HelloWorld
```

There should be no error messages.

Take a look at the files **HelloWorld.h** and **HelloWorld.c**. These contain the program code.

The **HelloWorld** program makes use of some other code, which is contained in the files **System.h** and **System.c**. These must be compiled with the “-unsafe” option. Try leaving this out; you’ll get 17 compiler error messages, such as:

```
System.h:39: ***** ERROR at PTR: Using 'ptr to void' is unsafe; you must
                compile with the 'unsafe' option if you wish to do this
```

Using the UNIX compiler convention, this means that the compiler detected an error on line 39 of file **System.h**.

Spankish programs are often linked with routines coded in assembly language. Right now, all the assembly code we need is included in a file called **Runtime.s**. Basically, the assembly code takes care of:

Starting up the program

Dealing with runtime errors, by printing a message and aborting
 Printing output (There is no mechanism for input at this stage... This system really needs an OS!)

Now execute this program. Type:

spank -g HelloWorld

You should see the “Hello, world...” message. What happens if you type “g” at the prompt, to resume instruction execution?

The “makefile”

The **p1** directory contains a file called **makefile**, which is used with the UNIX **make** command. Whenever a file in the **p1** directory is changed, you can type “make” to re-compile, re-assemble, and re-link as necessary to rebuild the executables.

Notice that the command

comp HelloWorld

will be executed whenever the file **System.h** is changed. In Spankish, files ending in “.h” are called “header files” and files ending in “.c” are called “code files.” Each package (such as **HelloWorld**) will have both a header file and a code file. The **HelloWorld** package uses the **System** package. Whenever the header file of a package that **HelloWorld** uses is changed, **HelloWorld** must be recompiled. However, if the code file for **System** is changed, you do not need to recompile **HelloWorld**. You only need to re-link (i.e., you only need to invoke **lddd** to produce the executable).

Consult the Spankish documentation for more info about the separate compilation of packages.

Step 10: Modify the HelloWorld Program

Modify the **HelloWorld.c** program by un-commenting the line

--foo (10)

In Spankish, comments are “--” through end-of-line. Simply remove the hyphens and recompile as necessary, using “make”.

The **foo** function calls **bar**. **Bar** does the following things:

- Increment its argument
- Print the value
- Execute a “debug” statement
- Recursively call itself

When you run this program it will print a value and then halt. The keyword **debug** is a statement that will cause the emulator to halt execution. In later projects, you will probably want to place **debug** in programs you write when you are debugging, so you can stop execution and look at variables.

If you type the **go** command, the emulator will resume execution. It will print another value and halt again. Type **go** several times, causing **bar** to call itself recursively several times. Then try the **st** command (**st** is short for “stack”). This will print out the execution stack. Try the **fr** command (short for “frame”). You should see the values of the local variables in some activation of **bar**.

Try the **up** and **down** commands. These move around in the activation stack. You can look at different activations of **bar** with the **fr** command.

Step 11: Try Some of the Emulator Commands

Try the following commands to the emulator.

- quit (q)
- help (h)
- go (g)
- step (s)
- t
- reset
- info (i)
- stack (st)
- frame (fr)
- up
- down

Abbreviations are shown in parentheses.

The “step” command will execute a single machine-language instruction at a time. You can use it to walk through the execution of an assembly language program, line-by-line.

The “t” command will execute a single high-level Spanish language statement at a time. Try typing “t” several times to walk through the execution of the **HelloWorld** program. See what gets printed each time you enter the “t” command.

The **i** command (short for **info**) prints out the entire state of the (virtual) SPANK CPU. You can see the contents of all the CPU registers. There are other commands for displaying and modifying the registers.

The **h** command (short for **help**) lists all the emulator commands. Take a look at what **help** prints.

The **reset** command re-reads the executable file and fully resets the CPU. This command is useful during debugging. Whenever you wish to re-execute a program (without recompiling anything), you

could always **quit** the emulator and then start it back up. The **reset** command does the same thing but is faster.

Make sure you get familiar with each of the commands listed above; you will be using them later. Feel free to experiment with other commands, too.

The “DISK” File

The Spankish virtual machine (the emulator tool, called “spank”) simulates a virtual disk. The virtual disk is implemented using a file on the host machine and this file is called “DISK”. The programs in project 1 do not use the disk, so this file is not necessary. However, if the file is missing, the emulator will print a warning. I have included a file called “DISK” to prevent this warning. For more information, try the “format” command in the emulator.

What to Hand In

Complete all the above steps.

To verify that you did all this, create a transcript of a terminal session showing what happened. In particular, please include the output associated with the following steps in what you hand in.

Step 7
Step 8
Step 9
Step 11

I do not need to see the other steps.

Hand in a hardcopy print-out showing what happened. If you do not know about creating a script file, check out the UNIX **script** command by typing

man script

In LARGE BLOCK LETTERS, write your full name, as it appear in the PSU registration system. In parentheses, list your nickname, if any.

Regarding The Due Date and Time

Please bring your project submission to class. It will be due at the beginning of class.

Working Together

I encourage students to discuss the material in this class. Feel free to discuss the programming projects amongst yourselves.

However, **this is not a group project**. *You must create the code on your own.*

It is okay to look at someone else's code for the purposes of helping that person or for comparing your different approaches, but each student must do the coding alone. You must type in every line of code you submit. You must **not** copy another student's code. Do not look at someone else's code, then turn around and type that code in. You must create your code on your own.

It will be considered cheating to copy code.

Grading for this Project

This project will be graded pass/fail.

Questions/Comments via Email

Questions/comments may be directed to me or the class TA; the best approach is through email. If you send questions via email, please include "cs333" in the subject line so I don't accidentally identify your email as junk!