CSC 3210 Computer Organization and programming

Lab 1

Generating assembly code from a higher level language

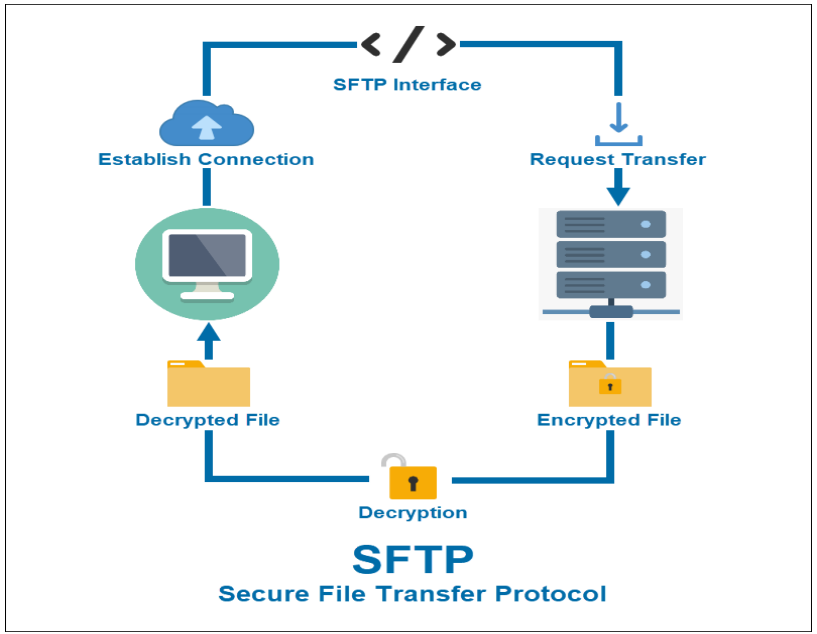
Lab work 1 instructions

This is the first Lab, a weekly homework for you to gain experience with class material, in this lab, we are going to:

- logged into a remote server
- produced assembly language code from a .c file
- changed the assembly language file to produce different results
- verified the changes in the file with the "diff" command
- compiled an assembly language file to an executable
- ran the executable program

1. What is sftp?

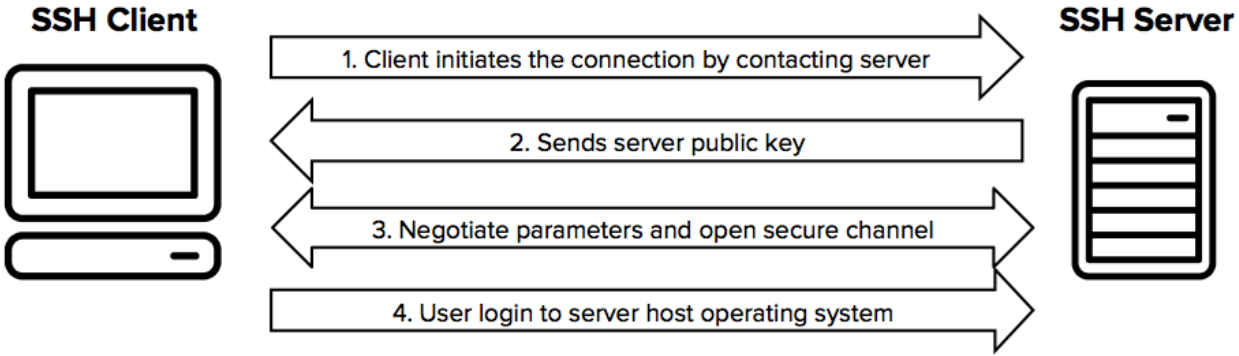
Secure File Transfer Protocol (SFTP) is a network protocol for securely accessing, transferring and managing large files and sensitive data.



2. What is SSH?

The Secure Shell (SSH) protocol is a method for securely sending commands to a computer over an unsecured network.

SSH uses cryptography to authenticate and encrypt connections between devices.



3. Snowball

The Snowball server is an instructional server for the GSU Computer Science department. Here is the link, which you can use as reference to connect the snowball server.

https://cscit.cs.gsu.edu/sp/guide/snowball

Assignment for lab 1

You will create a log file, and demonstrate your ability to complete the lab. The log file is created with the "script" command, and any input or output after that will be stored in the log file, until you give it the "exit" command. Remove any control characters that might be generated from arrow keys, using control-(some key), backspace, etc. In other words, what you turn in should look like what the text on your screen looks like. **Upload the log file to iCollege, making sure to use a .txt extension. Do not upload screen-shots.**

Hint: You can call the log file "lab1.log", as in the example below. You can use the "cat" command on your log file, and copy and paste its output into a text editor. This should get rid of the control characters. Make sure to save the file as ASCII text. You can use the filename "lab1.txt" for the version of the log file that you turn in. Do not upload files to iCollege without an extension, or with an extension of .log.

Step 1

Log in to your account on SNOWBALL.

You will probably need to use the command "**ssh** *your_account*@snowball.cs.gsu.edu".

Notice that the "ls" command returned nothing. It's a new semester, and any of the old files on this server are gone.

```
chunlan@MacBookPro csc3210 % ssh cgao8@snowball.cs.gsu.edu
cgao8@snowball.cs.gsu.edu's password:
Creating home directory for cgao8.
Last login: Mon Dec  9 19:31:57 2024 from 104-6-3-64.lightspeed.tukrga.sbcglobal.net
   +
   |   GSU Computer Science
   |   Instructional Server
   |   SNOWBALL.cs.gsu.edu
   +
[cgao8@gsuad.gsu.edu@snowball ~]$ ls
[cgao8@gsuad.gsu.edu@snowball ~]$
```

Step2

Start your log with script lab1.log

```
[cgao8@gsuad.gsu.edu@snowball ~]$ script lab1.log
Script started, file is lab1.log
```

Step 3

Next, get the "hello.c" file. You can download it from here, or you could even type it yourself.

This is a standard, first program for the C language, which you will learn in csc3320. The "vi" editor is a standard Unix text editor, and is a good utility to know about. If you opt to copy and paste (or type) the "hello.c" program, use the vi. Copy and paste code (from here)in the hello.c. After you close it, use the cat to check the file you created.

```
   +
[cgao8@gsuad.gsu.edu@snowball ~]$ ls
[cgao8@gsuad.gsu.edu@snowball ~]$ vi hello.c
[cgao8@gsuad.gsu.edu@snowball ~]$ ls
hello.c
[cgao8@gsuad.gsu.edu@snowball ~]$ cat hello.c
/*
 *    hello world
 *    */

#include <stdio.h>

int main (int argc, char *argv[]) {

   printf("hello world.\n");

   return 0;
```

Step 4

In the next step, we compile the "hello.c" program. "gcc" is the GNU C compiler, and "-o hello" specifies that we want the output to be a new file called "hello". Be careful with this step; if you were to type "-o hello.c", it would over-write the C file, and you would have to start over. After compiling, we get a list of files with the "ls -l" command. The "-l" option says that we want a long list, i.e. showing details.

```
[cgao8@gsuad.gsu.edu@snowball ~]$ gcc hello.c -o hello
[cgao8@gsuad.gsu.edu@snowball ~]$ ls -l
total 16
-rwxrwxr-x. 1 cgao8@gsuad.gsu.edu cgao8@gsuad.gsu.edu 8360 Jan  7 10:46 hello
-rw-rw-r--. 1 cgao8@gsuad.gsu.edu cgao8@gsuad.gsu.edu  135 Jan  7 10:25 hello.c
-rw-rw-r--. 1 cgao8@gsuad.gsu.edu cgao8@gsuad.gsu.edu    126 Jan  7 10:40 lab1.log
```

Step 5

The file "hello" is executable, and so we run it in the next command. The "./" specifies where to find the file, and "." is short-hand for the current directory.

[cgao8@gsuad.gsu.edu@snowball ~]$ ./hello
hello world.

The program ran as expected, and gave us the line of output. Normally, like in csc3320, you might stop here: so far, we have obtained a C program, showed its contents, compiled it, and ran it. But we're going to go a level deeper.

Step 6

Next, we will compile it again. Notice the "-S" (and yes, that is a capital S).This generates a new file, "hello.s". We then "cat" its contents, like before.

```
[cgao8@gsuad.gsu.edu@snowball ~]$ gcc hello.c -S
[cgao8@gsuad.gsu.edu@snowball ~]$ ls
hello  hello.c  hello.s  lab1.log
[cgao8@gsuad.gsu.edu@snowball ~]$ cat hello.s
        .file   "hello.c"
        .section        .rodata
.LC0:
        .string "hello world."
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
        .section        .note.GNU-stack,"",@progbits
```

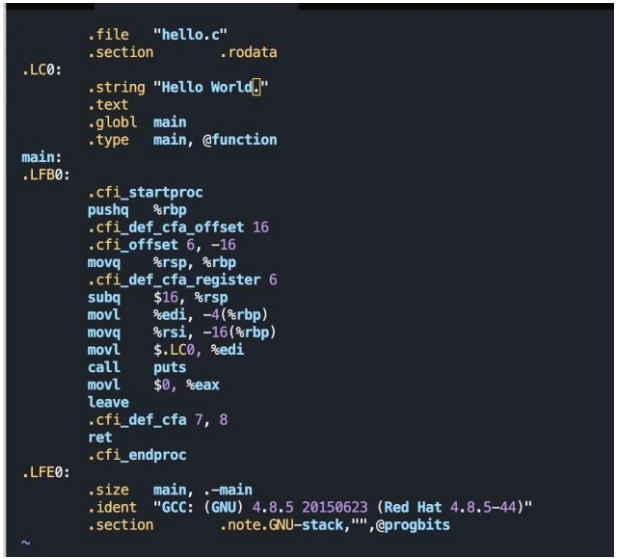
At this point, we now have **an assembly-language program**. We will get into what the program means. It may look like random text now, but by the end of the semester, you'll be able to explain it to others.

Step 7

Next, make a copy of the hello.s, and call the copy "hello_original.s". It's a good idea to keep copies of files that you change.

after making the copy, edit the "hello.s" file to change the "hello world." text to "Hello World!"

```
[cgao8@gsuad.gsu.edu@snowball ~]$ cp hello.s hello_original.s
[cgao8@gsuad.gsu.edu@snowball ~]$ vi hello.s
```



```
        .file   "hello.c"
        .section        .rodata
.LC0:
        .string "Hello World!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        movl    $.LC0, %edi
        call    puts
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-44)"
        .section        .note.GNU-stack,"",@progbits
~
```

Step 8

Next, we will use "diff" to show us the difference between the original and new "hello.s" files

[cgao8@gsuad.gsu.edu@snowball ~]$ **diff hello_original.s hello.s**

4c4

< .string "hello world."

> .string "Hello World."

As you can see, only the string should be changed.

Step 9

Next, we use "gcc" to create an object file called "hello.o". An object file is created before linking produces the executable file. Notice the "-c" argument that is passed to "gcc". Also notice how the second "ls" output shows the new file "hello.o".

Then we use the object file to create an executable file. Again, we use "gcc" for this. Notice the "-o" argument that is passed to "gcc". The output file that this command creates is called "hello2".

Then Now we run the executable program.

cgao8@gsuad.gsu.edu@snowball ~]$ gcc -c hello.s

[cgao8@gsuad.gsu.edu@snowball ~]$ ls

hello hello.c hello.o hello_original.s hello.s lab1.log

[cgao8@gsuad.gsu.edu@snowball ~]$ gcc hello.o -o hello2

[cgao8@gsuad.gsu.edu@snowball ~]$ ls

hello hello2 hello.c hello.o hello_original.s hello.s lab1.log

[cgao8@gsuad.gsu.edu@snowball ~]$ ./hello2

Hello World.

Step 10

The "-v" option for "gcc" means "verbose". The next command produces the same results as "gcc -c hello.s" does, but it generates a lot of additional information. One thing worth pointing out is that the target is "x86_64-redhat-linux", a 64 bit Intel x86 processor. We are not likely to need the "-v" option normally, but if there is a problem, this may reveal what the problem is.

```
[cgao8@gsuad.gsu.edu@snowball ~]$ gcc -v -c hello.s
Using built-in specs.
COLLECT_GCC=gcc
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bu
nable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunw
r-hash-style=gnu --enable-languages=c,c++,objc,obj-c++,java,fortran,ada,go,lto --enable-plugin --enable
50702/obj-x86_64-redhat-linux/isl-install --with-cloog=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86
ic --with-arch_32=x86-64 --build=x86_64-redhat-linux
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-44) (GCC)
COLLECT_GCC_OPTIONS='-v' '-c' '-mtune=generic' '-march=x86-64'
 as -v --64 -o hello.o hello.s
GNU assembler version 2.27 (x86_64-redhat-linux) using BFD version version 2.27-44.base.el7_9.1
COMPILER_PATH=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/:/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/:/
sr/lib/gcc/x86_64-redhat-linux/
LIBRARY_PATH=/usr/lib/gcc/x86_64-redhat-linux/4.8.5/:/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../..
5/../../../:/lib/:/usr/lib/
COLLECT_GCC_OPTIONS='-v' '-c' '-mtune=generic' '-march=x86-64'
```

Step11

Next, run the original "hello" program. Then run the "hello2" program that we created from an altered assembly language listing.

```
[cgao8@gsuad.gsu.edu@snowball ~]$ ./hello

hello world.

[cgao8@gsuad.gsu.edu@snowball ~]$ ./hello2

Hello World!
```

Step 12

Finally, we exit from the script command. This informs the "script" utility that we are done.

You might want to next use "cat" on the "lab1.log" file, copy the output, and paste it into a text file called "lab1.txt". This should get rid of any extraneous (control) characters, though you should look it over to make sure.

[cgao8@gsuad.gsu.edu@snowball ~]$ exit

exit

Script done, file is lab1.log

Then we exit (log out) from the server. You can use "sftp" to get your log file from the server.

[cgao8@gsuad.gsu.edu@snowball ~]$ exit

logout

Connection to snowball.cs.gsu.edu closed.

chunlan@MacBookPro csc3210 %

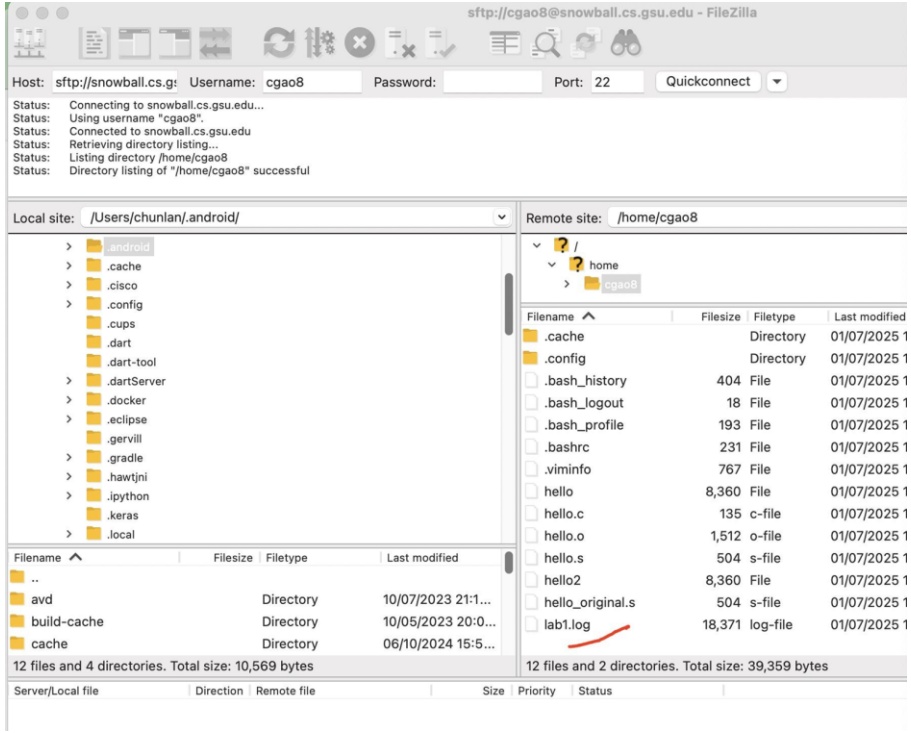
If you use sftp to transfer file, here is an example

How to connect snowball:

https://cscit.cs.gsu.edu/sp/guide/snowball

How to use fileZilla:

https://wiki.filezilla-project.org/FileZilla_Client_Tutorial_(en)



Submit your lab1.log file on icollege, with your name, email and session number!