

Operating Systems (COE628)

Lab 2

Due the Week of January 29, 2024

Objectives

- Learn how to use command line arguments in a C program.
- Learn how to give the same program different names and make it behave differently according to the name it is invoked by.
- Run multiple processes simultaneously and coordinate them using an atomic command.

Notes

- The lab can be done on any computer (Windows, Linux, Mac OS X, etc.)
- You also need a Unix shell and a C compiler. Linux and Macs already come with these. For Windows, you also need cygwin.

Getting Started: Download the templates

- Download the lab 2 Netbeans templates zip file and save the zip file in your coe628 directory.
- Unzip lab2.zip. This creates a lab2 directory and two sub-directories: lab2a and lab2b. Each of these sub-directories contains the template files for a Netbeans project.

Part A: Using command line arguments and exit codes

Some theory for Part A:

1. argc and argv

- C programs start at *int main (int argc, char * argv[])*
- When, for example, command called "foo" is invoked as *foo bar zoo*, then the command line consists of 3 words: "foo", "bar" and "zoo".
- Main is passed the parameter argc (argument count) with the number of words in the command line (in this case 3).
- The parameter argv (arg values) is an array of the command line words. In this example, argv [0] would be the string "foo" and argv[1] and argv[2] would be "bar" and "zoo".
- Note that argc cannot be less than 1; there is always a command name as the first word in the command.
- The exact same file can have different names. In particular, the same executable file can have 2 or more names and its behaviour can depend on the name under which it was invoked.

2. Exit codes

- "main" is declared to return an *int*. The return value is called the exit code.
- If the program works, the exit code should be 0 (zero). If there is a problem, a non-zero exit code should be used (a small integer). Different errors should have different non-zero exit codes.
- An exit code of zero is interpreted by the shell (command line interpreter) as true. Any non-zero value is false.
- Commands can be "joined" with the logical AND operator "&&".
- For example, the command line *foo && bar* will execute the "foo" command. If it is "true" (i.e. has a 0 exit code), it will then execute the "bar" command. (If "foo" is *false*, "bar" will not be executed.
- You can also determine the exit code of the last command executed with *echo \$?*

```

mkdir junk
echo $?
0
mkdir junk
mkdir: cannot create directory `junk': File
exists
echo $?
1
mkdir junk 2> /dev/null
echo $?
1
rmdir junk
echo $?
0
mkdir junk
echo $?
0
^

```

Giving the same file different names with

- If you have a file called "foo", you can give it another name with the ln (link) command: ln foo goo (this is called a "hard link"). Alternatively, you can use a symbolic link: *ln -s foo go*

Requirements for Part A

1. Start Netbeans and open the project at "coe628/lab2/lab2a", It should compile and run.
(NOTE: if you are running Windows or MAC OS X (or another version of Linux))
2. Modify the body of the main function so that it has the following behaviour: It prints a string in the form "*greeting person*". The *greeting* is *Hello* (by default) or *Bye* if the program is invoked with a command that ends in the string *bye*.
3. If there is exactly one argument to the command main should return 0 (zero) as the exit code. Otherwise, it should return 1 if there are no arguments and 2 if there is more than one argument.
4. Once compiled, the executable is placed in the project's *Debug* directory and is called *lab2a*.
5. Make links (or "aliases") of that command using the following shell commands:


```
ln lab2a hello
ln lab2a goodbye
```
6. A typical interactive session is shown below where user input is **this font** and the output is in *italic*.

```

hello
Hello UNKNOWN
bye Alice
Bye Alice
hello bob
Hello bob
bye Cathy Ng
Bye Cathy
hello dave smith && bye al
Hello dave
hello "dave smith" && bye al
Hello dave smith
Bye al

```

Part B: Multiprocessing and synchronization

Consider the following program.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#define N_REPS 50
#define DEFAULT_SLOWDOWN 10000
int main(int argc, char * argv[]) {
    int i;
    int slow_down = DEFAULT_SLOWDOWN;

    if (argc == 1) {
        fprintf(stderr, "Usage: %s string [delay]\n", argv[0]);
        return 1;
    }
    if (argc >= 3) {
        slow_down = atoi(argv[2]);
    }

    for (i = 0; i < N_REPS; i++) {
        char * cp = argv[1];

        while (*cp) {
            printf("%c",*cp++);
            fflush (stdout) ;
            usleep (random()% slow_down) ;
        }
        usleep (5000) ;
    }
    return EXIT_SUCCESS;
}
```

- Suppose the compiled executable is called lab2b. The invoking lab2b abcd will result in the output: abcdabcedabce etc.
- (Note: even if you are not familiar with all the coding conventions, you should examine the code sufficiently to convince yourself that it does do something like this.)
- If you now run two "lab2b" processes concurrently with, for example, the command: lab2b abcd & foo WXYZ, you will see output something like:
aWxbYcZWdXaYZWxbYcZWdXaYZbWcXdYZabWcXdYaZbWcXdYaZbWcXdYaZbW
cdXaYbZcWXdYaZWbXcYZdWaXYbZcWXdYaZWbXcYZdWaXYbZcWXdYaZWbXYc
ZdWxaYbZWcXdYaZWbXcYZdWaXYZWXYZWXYZWXYZWXY
- One process prints lower case letters; the other uppercase letters. But they are all intermixed. We would like the lower and upper case letters not to be jumbled together.

- For example, we would like the output to be something like:
WXYZabcdWXYZabcdWXYZabcdWXYZabcdWXYZabcdWXYZabcd
- To achieve this, only one process at a time should be able to perform the "while (*cp)" loop.
- Your goal in Part B is to modify the code to achieve this.
- To achieve this, you have to identify a "critical section" that only one process at a time should be allowed to execute.
- Use while (system("mkdir junk") != 0) ; and system("rmdir junk") ; to achieve this.

Submit your lab

You must submit the project on a Departmental computer, thus you need to do the following:

- cd coe628
- zip -r lab2.zip lab2
- submit coe628 lab2 lab2.zip