CSCI.4210 Operating Systems
Fall, 2008
Programming Assignment 1

You must work independently on this project. Do not show your code to other students and do not look at other students' code. Do not put your code in a public directory or otherwise make it public. However, you may get all the help you need from the instructors or TAs.

The Unix command `ls` and the Windows command `dir` display information about files in the current directory. For this assignment, you will write two versions of this, one using Unix system calls and one using the WIN32 APIs. Your programs will not take any arguments. For each file, it should display the file name, the size, and the time of last modification. However, unlike `ls` and `dir`, it should go into each subdirectory (subfolder in Windows terminology) and display the contents, including each subdirectory of a subdirectory and so on. The contents of each subdirectory should be indented four spaces from that of its parent.

Here is a sample output.

```
helloworld.c  986  Aug 20 2008 16:33
makefile      230  Aug 21 2008 08:42
personal
    resume.doc  2345  Apr 8 2004 10:21
    oldresume.doc 2330 Apr 6 2004 13:01
courses
    os
        fall08
            syl.doc  56832  Aug 26 2008 14:57
            proj1.doc 89677  Aug 28 2008 11:05
        fall04
            syl.doc  54588  Aug 30 2004 12:43
            proj1.doc 64988  Sept 1 2004 14:56
            proj2.doc 101444 Sept 25 2004 09:00
    netprog
public.html
    index.html 2310  Aug 26 2008 11:11
```

In this example, the current directory has only two regular files, `helloworld.c` and `makefile`. However, it has three subdirectories, called `personal`, `courses`, and `public.html`. The subdirectory `courses` has two subdirectories, `os` and `netprog`. The `os` directory has two subdirectories called `fall08` and `fall04`, each of which contains several files. The `netprog` directory is empty.

Note that you don't display the size or the time of last modification of a subdirectory.

Hint: Your program will probably need to be recursive. Write a helper function that takes a directory name as one of its arguments and the target file name as another, and whenever you find an entry which is a directory, call this helper function, passing in the complete path name of the directory which you found. You might also want to pass in the number of character to indent as an argument.

Alert: In both Unix and Windows, every directory has two files called . and .. The file called . refers to this directory and the file called .. refers to the parent of this directory. You should explicitly ignore both of these files. Otherwise, you will get into infinite recursion.

You need to submit two versions of this, one that runs on Unix and one that runs on Windows.

Unix version

There are two system calls that you need to open and read a directory

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir(const char *dirname);
struct dirent *readdir(DIR *dirp);
```

The type DIR, which is defined in the header `<dirent.h>`, represents a directory stream, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files and subdirectories. The opendir system call opens a directory and returns a pointer to a struct DIR. Once a directory has been opened, you can read the directory with the readdir system call. Each time you call this, it returns another entry in the directory pointed to by its argument. When there are no more entries, it returns NULL.

The `struct dirent` is defined in `dirent.h`. There is only one field in this structure that you need, `char *d_name` which is the name of the entry. Note that this is just the name relative to its parent, not the full pathname.

Once you have the name of an entry, you can use the `stat` system call to get more information about it.

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *path, struct stat *sb)
```

The stat function takes two arguments, the first is a path name, (either relative to the current directory or absolute) and a pointer to a `struct stat` This structure contains information about the file, and this information is filled into the fields of its second argument. There are four fields of struct stat that you will need. The data types `mode_t`, `time_t`, `uid_t`, and `off_t` are all unsigned ints.

`mode_t st_mode` This indicates the file type. (regular file, directory, or other choices that you probably won't find)

`off_t st_size` the size of the file in bytes.

`uid_t st_uid` The user id of the owner of the file.

`time_t st_mtime` The time that the file was last modified.

The stat system call returns 0 if successful, and -1 on failure.

There are macros to test the file type, these take `sb->st_mode` as an argument .
`S_ISDIR(sb->st_mode)` returns true if the entry is a directory, false otherwise.
`S_ISREG(sb->st_mode)` returns true if the entry is a regular file, false otherwise.

Alert: You must allocate memory for the `struct stat` before passing it in as an argument.

Time on a Unix system is measured in the number of seconds since Jan 1, 1970. You can convert this value to a more useful time with the function `char *ctime(const time_t *clock)` which takes a pointer to a `time_t` as an argument, and returns a string of length 26 which represents the time of its argument in human readable form.

```
Fri Sep 13 00:00:00 1986\n\0
```

You can use the man pages to learn more about all of these function.

Win32 Version

There are two WIN32 APIs that you need to traverse a directory.

```
#include <windows.h>
HANDLE FindFirstFile(LPCTSTR lpFilename, LPWIN32_FIND_DATA lpFindFileData)
```

This call takes a filename as its first argument. This file name can include wildcards (the asterisk can be used to represent zero or more characters). If there is at least one such file, it returns a search handle, and the information about the first file found is stored in the second argument. The second argument is a pointer to a structure of type `WIN32_FIND_DATA`. This structure contains a number of fields which have information about a file. There are four that you will need

`DWORD nFileSizeLow` which contains the size of the file in bytes
`CHAR cFileName[MAX_PATH]` a string containing the name of the file
`DWORD FileAttributes` a set of flags about the attributes of the file. You can check to see if the file is a directory by checking the `FILE_ATTRIBUTE_DIRECTORY` flag. Hint: Perform a bitwise `and` operation (`&`) on the attributes and `FILE_ATTRIBUTE_DIRECTORY`, and if the value is non-zero, the file is a directory.
`FILETIME ftLastWriteTime` the time that the file was last modified.

The data type FILETIME is a 64 bit value representing the number of 100 nanosecond intervals since January 1, 1601 (I'm not making that up). Unfortunately this time is in Coordinated Universal Time, not local time. To convert this to local time, use the function.

```
BOOL FileTimeToLocalFileTime(CONST FILETIME *filetime, FILETIME *localtime);
```

Once you have done this, you can convert local time to a useable value with the function.

`BOOL FileTimeToSystemTime(CONST FILETIME *localtime, SYSTEMTIME *systime)`

The struct SYSTEMTIME has the following fields, all of type WORD (unsigned short int).

`wYear, wMonth, wDayOfWeek, wDay, wHour, wMinute, wSecond, and wMilliseconds`

You can use these values to display the time that the file was last modified.

To see if there is more than one file which satisfies the condition, use the API

`BOOL FindNextFile(HANDLE hFindFile, LPWIN32_FIND_DATA lpFindFileData)`

The first argument is the handle returned by FindFirstFile. The second argument has been discussed above. If there is another file which meets the criteria, this function returns true, and the values of the file are stored in the second argument. If there are no more files, it returns false and the values of its second argument are indeterminate.

This program is worth 8% of your course grade. Each of the two programs will be graded independently and will count 4% each.

Grading standards:

Compiles without errors or warnings (use -Wall with gcc): 40%
Prints the correct answer on the current directory 20%
Prints the correct answer for all subdirectories as well 20%
Handles all error conditions gracefully 10%
Is well commented and well designed 10%

Note: You will lose points if you do not check the return values of all system calls to confirm that they executed successfully.

If your program does not have a comment at the top listing your human name and your email and basic information about the program, there will be an automatic 10 point penalty.

The program is due at 11:59PM on Tuesday, Sept. 16. Check the web site for information on how to submit the programs. There will be lateness penalty of ten points per day.