Programming Project: du lite

Introduction

For this assignment you will write a program that implements some of the functions of the Unix **du** command. In doing so, you will have a chance to work with the Unix directory structure, recursion, and the stat() and lstat() system calls.

Explanation of du

There is never enough disk space. You buy a new 50 gigabyte hard disk, and next thing you know, you are running low. You wonder where you parked all that stuff. Somewhere in some directory is a lot of movie files or cdrom images, or who knows what?

The **du** utility tells how much disk space is being used in each directory in a directory tree. It lists all the directories below a specified point and tells how many disk blocks are used by the files and directories in and below that point.

Here is an example:

```
$ cd ~smith/hmwks
$ du aac
421     aac/testdir/old
1219    aac/testdir
11      aac/extra
1570    aac
```

The Assignment

Write a version of **du** (called dulite) that accepts zero or one argument and accepts the **-k** option.

If it is called with no arguments, it processes the current directory. If it receives a filename, it reports the number of 512-byte blocks used by that file. If the argument names a directory, dulite processes that directory and its children. The output should match the output of the 'real' du. The **-k** option causes du to report the number of 1024-byte blocks used.

Getting Started

1. Read the manual page on the du command. Try it out in your own directory or various system directories. For example, you might try:

```
$ du /usr/include
```

You can use the commands `ls -s` and `du -a` to see the size of each thing in disk blocks. It will help you see where du gets its numbers.

2. Look at the code for ls1.c and ls2.c in the text. These programs loop through the contents of a directory printing info about each file or directory. du, instead, has to find the total number of blocks used by all the things in a directory.

3. When du encounters a subdirectory, it has to recursively process it. How does du distinguish a regular file from a subdirectory? Look at the header file for stat.h and/or read the code for stat2.c in the text. Note

that du prints the path to each subdirectory. How does that work?

4. Write a function called `disk_usage( char *path )` This function will open the directory, read it entry by entry, add up the space used by each item in that directory. In addition if any entry in the directory is a directory, then disk_usage() will have to create a new string to hold the new path and pass that string to itself. Use malloc() to create the string: a fixed size buffer is liable to run out when you least expect it.

Testing Your Program

You can make up your own test system data, but you must test it sometime with the official test script. This shell script will exercise your program on a preset directory tree. To run it, type:

```
$ ~COURSE/hmwks/dulite/test.dulite
```

Error Handling

All sorts of errors can occur when traversing a directory tree. You may be denied permission to a directory. You may be unable to stat a file. You may encounter a very deep directory structure with a long path name. How does the real du handle those? Does it quit or report an error and proceed? Do what it does.

What to Hand in

Hand in (1) a copy of the source to your program, (2) a sample of its output from the test.dulite run mentioned above, and any other amusing examples you might want to include.

Extra Credit

A file may have more than one link. The correct action is to count the disk usage only once, not for each link. For an extra ten points, write your program so it handles hard links correctly.