# CMPSC 473, Project 4

cse.psu.edu/~deh25/cmpsc473/assignments/PR4.html

**CMPSC 473, Project 4**

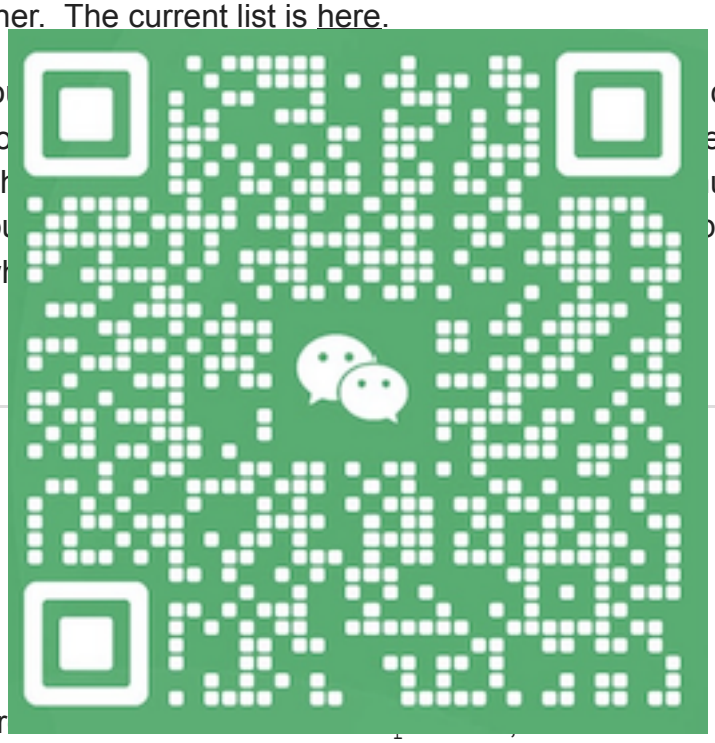
Posted April 9, 2014. Due April 24, 2014, on ANGEL. 60 points.

This project is primarily a design-and-implement exercise, which requires you to understand a disk partition and a file system.

On the whole, planning would be a more productive use of your time than programming, since programming without a plan is going to be difficult. If you don't see how to proceed after two hours of thinking, then ask the instructor or one of the TAs for a suggestion.

Four people can work together on this project. Send email to dheller@cse.psu.edu if you are looking for a partner. The current list is here.

> This project sho class or
> elsewhere. If yo e name of the
> other person, wh uch of the work
> was done by you lain `fork()`, but
> not to explain wh



---

Starter kit

- Makefile
- pr4.c
- pr4.in

The program `pr4.c` r For grading, we will supply a different input file. You will need to finish writing the `do_*` functions, and perhaps add some more.

Note that the input to `pr4.c` is a sequence of text lines, representing commands to be executed immediately, in the style of a command shell. Don't modify the program to read all the input before executing any of the commands.

It would be a good idea to review the bitmap exercise from Homework 1 (solution here).

---

The command `root` indicates to construct a file system within a block of memory. Forty

megabytes should be sufficient; use a call to `malloc()` or `mmap()`, instead of declaring an array. The initial state of the file system is empty except for the top-level directory.

At any time, you have a current working directory; use the command `chdir` to move up one level (`chdir ..`) or down one level (`chdir dirname`). The `dirname` must appear in the current working directory. Moving up one level from the top-level directory leaves you in the same place.

The command `mkdir` (`mkdir dirname`) creates a subdirectory in the current working directory. There is no size limit associated with a directory; if you need to add something to a directory that is already full, make it larger.

The command `mkfil` (`mkfil filename filesize`) creates a file in the current working directory. You should actually allocate `filesize` bytes in your file system, but the file contents will not actually need to be read or written.

The commands `mvdir` (`mvdir oldname newname`) and `mvfil` (`mvfil oldname newname`) rename dir

The commands `rmdi` `ame`) remove directories and files. I works recursively. The `dirname` or `file` ry. It must not be possible to remove the ." or "`rmdir ..`" must be rejected.

The command `szfil` e size of an existing file. The new size to 0 does not remove the file.

The command `print` ." .

---

We have left a lot of th be sure to explain why you did it that way, instead of some other reasonable way. "It was easier to program this way" is a valid explanation, but then you should describe some other good idea that you decided not to implement.

Think of the initial block of memory as if it was a disk partition, then build a file system using that disk-like organization.

1. Allocate space within your process using mmap(); this is to simulate a disk partition. You will need to pick a partition size; 40 MB is probably sufficient, but you might need something larger.

- You will need to open a file with open() before calling mmap(), if you want to save information between runs.

- You could use malloc() instead of open() and mmap(), but then there is no actual file, and you can't save information from one run to the next. This is enough to get started.

- Choose a "disk block" size to use with your "disk partition". Design and implement a mechanism to record which blocks are free, and which are allocated to files, directories, or any other purpose that you require. This "free block table" must be stored in the "disk partition" itself. You will need a descriptor block that stores information about the free block table and perhaps some additional information about the blocks and partition; this also goes in the "disk partition". It only makes sense for the descriptor block to be the first block in the partition. You will need to decide how to communicate the block size to any user of the partition. This much is independent of the file system design.
  See OSC 9e Sec. 10.5 for some ideas, but you don't need to do anything complicated.

- Design a data structure to act as a directory, and another data structure to act as a file control block.
- Construct a root di covered by reading the partition's descript
- Construct a collect the free block table appropriately. T need to keep track of their names a ories.
- Write a function tha line for every file and directory, starting ry this command on a real file system, b
- Write a function tha the free block table, or at least prints artition.
- Write the main pro s you wrote (`pr4.c` with `pr4.in` n, add some files and subdirectories, ar verify that you implemented everythi
Compile the program ssible to read the program. You want to get a grade, and if it won't compile and run, or if no one can make sense of it, the grade won't be very good.

We didn't say what design you should use for the file system, except to imply that it is hierarchical. At least, implement a flat file system (root directory, with files and no subdirectories).

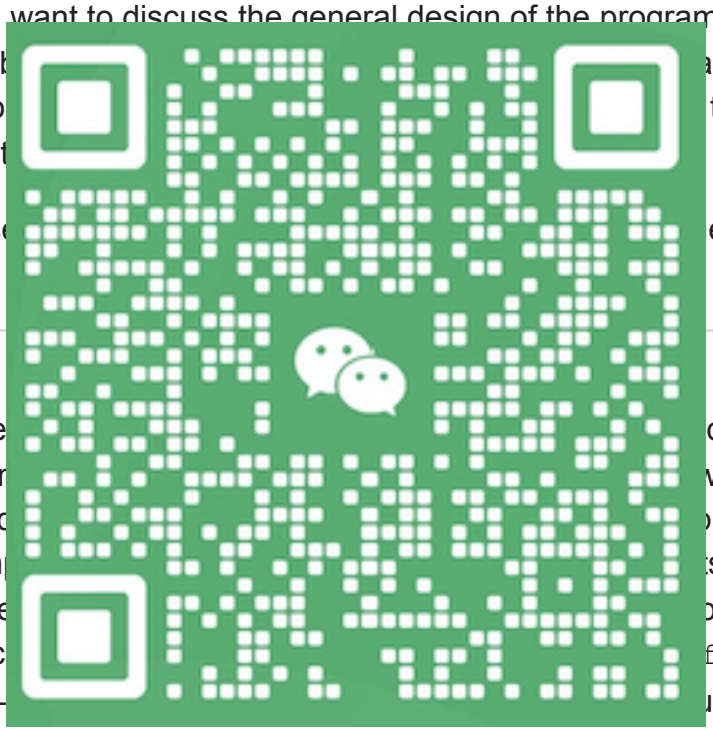
We didn't say exactly how your "ls -R" function should work. For example, you might decide that sorting the file names for output is too much work. For debugging purposes, it's probably better not to sort them. Similarly, you don't need to sort the contents of a directory (Windows does, Unix doesn't).

---

There is a Dropbox on ANGEL for Project 4. Attach all the necessary source files, a makefile named `Makefile`, and an optional text file named `pr4.txt`. The source files can have any names you want, but the files `Makefile` and `pr4.txt` must have exactly those names. Put the names of everyone on the team in each file, as a comment at the beginning of the file. If you want to bundle all your files into one, use Zip, or tar, or tar and gzip. Do not use winRAR (points will be deducted for this).

The makefile must be constructed so the grader can execute the command "`make pr4`" and have the program compiled on Linux. The name of the executable file must be `pr4`. These requirements will make it easier to do automated testing. Part of the grade will depend on how you wrote the makefile; for example, don't forget to turn on the compiler warning options.

The text file can contain any explanation that the program requires. Of course, comments in the program are preferred in most cases, but they often refer only to the specifics of some function. If you want to discuss the general design of the program, put it in the text file. The text file can b akes it difficult to provide a diagram, so text file that simply says "go read t



The Dropbox will close e date is 11 pm on Thursday, Apr. 24.

---

Please note - in these d". If the program doesn't compile on Lin vill be between 20 and 25 points, depend rogram compiles but fails to run to com s, depending on how far it got on the te mptly; for example, in C, each c `flush()`. This will ensure that script- t is lost on a program crash.

---

Last revised 9 Apr. 2014