Basic File Systems

CPSC 457: Principles of Operating Systems
Winter 2024

Contains slides from Pavol Federl, Mea Wang, Andrew Tanenbaum and Herbert Bos, Silberschatz, Galvin and Gagne

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Tuesday, 28 November 2024

UNIVERSITY OF CALGARY

Topics

- Filesystems
- File Operations
- Directories
- Directory Operations
- Assignment Help

Filesystems

Filesystem

- filesystem is a higher level abstraction of storage
- implemented using clever data structures, stored in storage + memory
- basic unit of a filesystem is a file
- files can be usually grouped in directories and subdirectories

File attributes

- files have contents but also attributes

- file attributes vary from one OS to another but typically consist of these:
  - filename: string to make it easy to reference the file
  - size: size of the file
  - time/date: time of creation/last modification/last access, used for usage monitoring
  - user ID, group ID: identifies owner(s) of the file
  - access control information (who can read/write/execute this file)

File format (file type)

- contents (sequence of bytes) of file determine file format (a.k.a. file type)
  - determined by file creator
  - if OS recognizes the file format, it can operate on the file in reasonable ways
    e.g. automatically using an appropriate program to open a file
- Windows uses file extension to guess file format, eg. ".jpg", ".xls"
- UNIX uses magic number technique to guess file format, extension is only a convention
  - format inferred by inspecting the contents of the file, often just first few bytes
  - eg. `#!/bin/bash` as the first line → file contains a bash script, `%PDF` → pdf file, ...

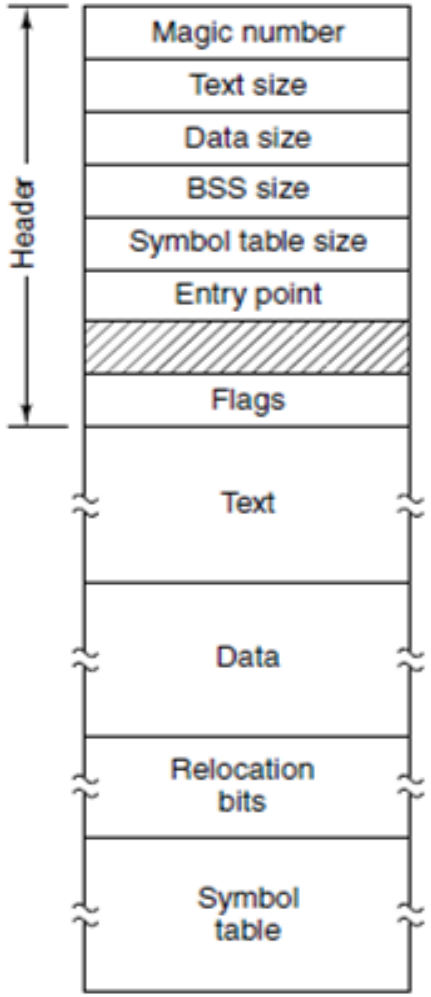
```
$ file file.c file /dev/hda .
file.c:    C program text
file:      ELF 32-bit LSB executable
/dev/hda: block special (3/0)
.:         directory
$ man file
$ man magic
```

Example file formats

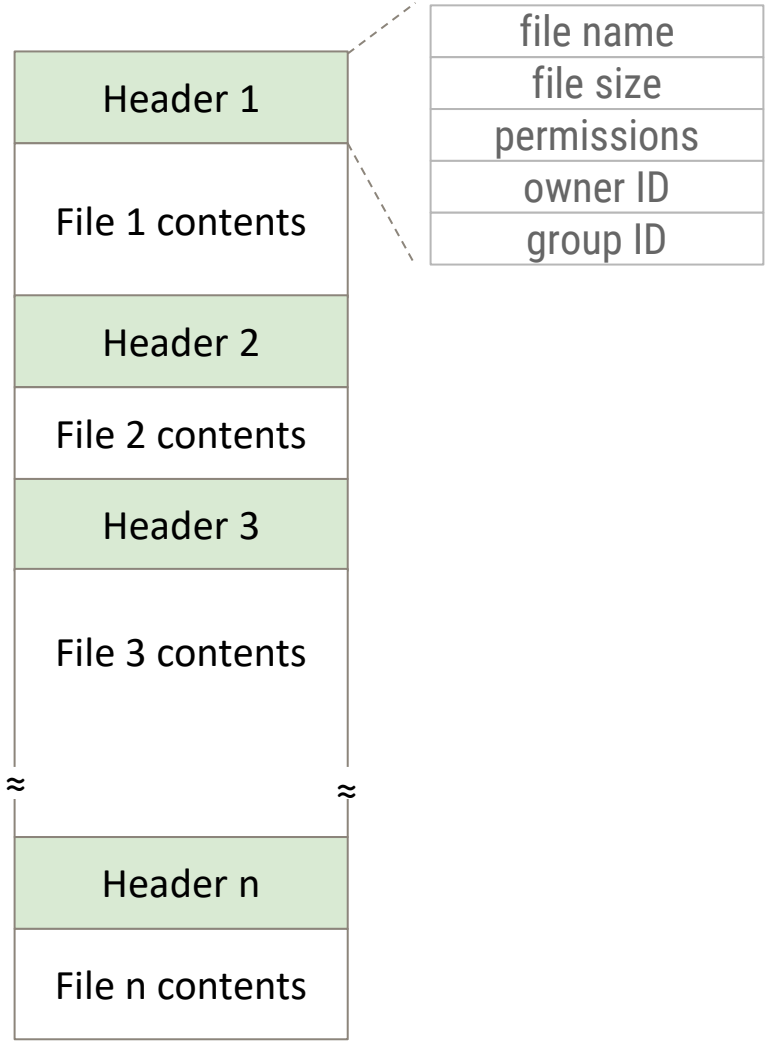
```
#include <stdio.h>

main()
{
    printf("Hello World");
}
```

#	i	n	c	l	u	d	e
	<	s	t	d	i	o	.
h	>	\n	\n	m	a	i	n
(	)	\n	{	\n			
	p	r	i	n	t	f	(
"	H	e	l	l	o		W
o	r	l	d	"	)	;	\n
}	\n						

text file



executable file



archive

File Operations

File operations

Most systems allow the following operations on regular files:

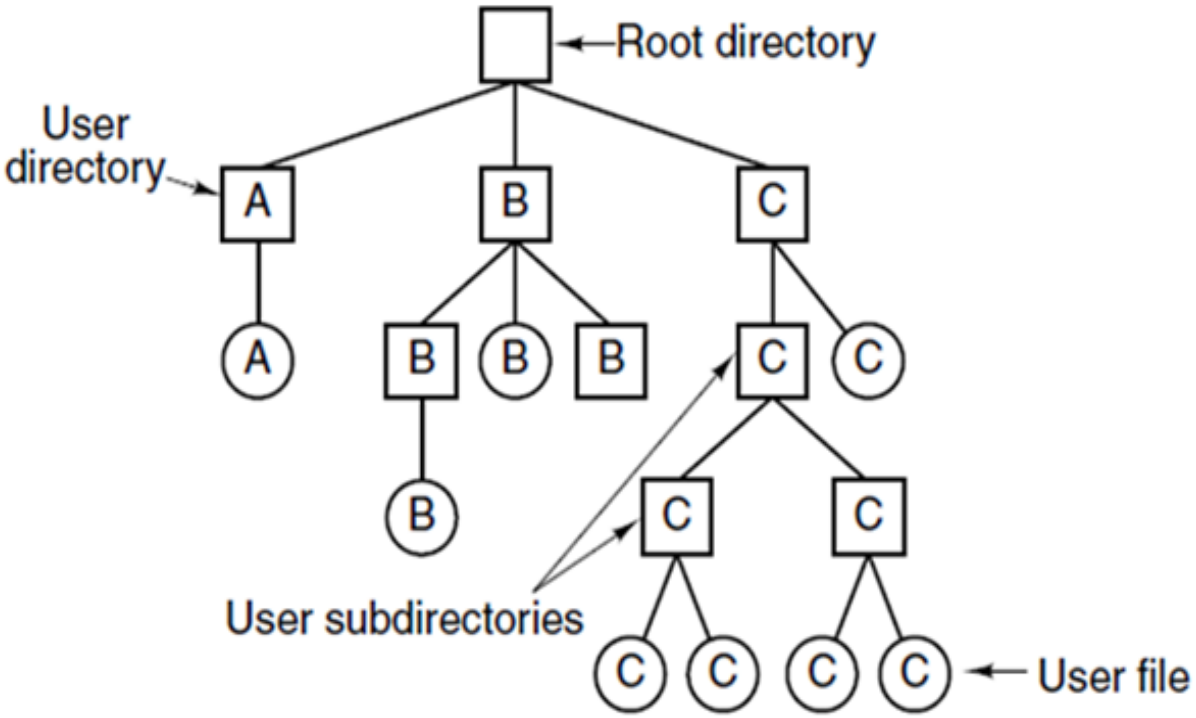
- create — empty file is created, with no data
- delete* — files can be deleted to free up disk space
- open — before using a file, a process must open it. OS can fetch and cache file attributes, such as list of disk addresses into main memory, for rapid access on subsequent calls
- close — free up space in memory associated with open file, flush unwritten data
- read — read contents of an opened file from current position
- write — overwrite data of an opened file at current position
- append — write new data at the end of file, results in file growing, usually implemented via write
- seek — change current position, affecting subsequent reads/writes
- get attributes* — eg. size
- set attributes* — eg. permissions
- rename* — change filename

** operation could be on a directory rather than a file*

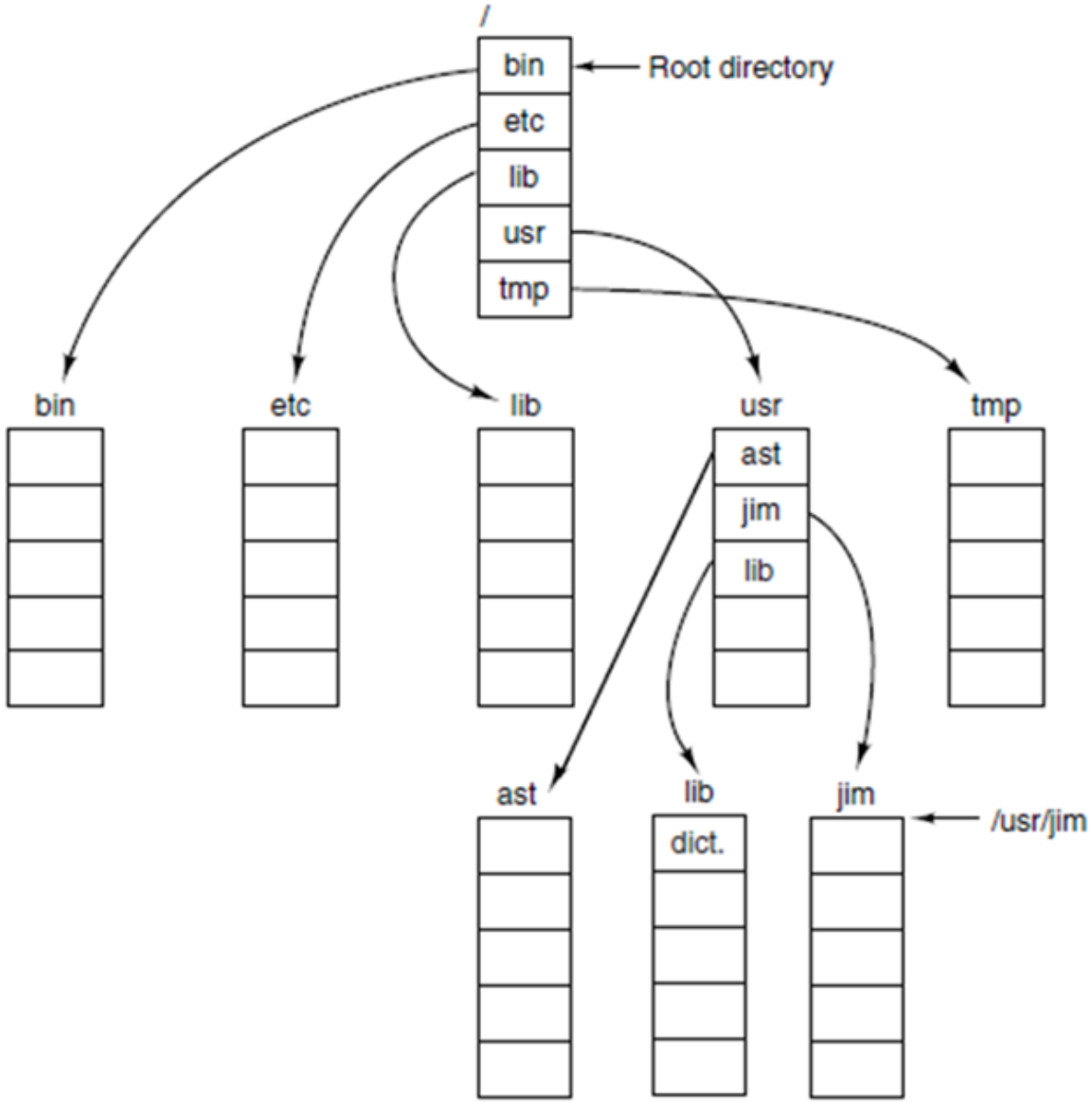
Directories

Directories

- filesystems use directories to help us with organizing files
- directories/subdirectories form a tree structure (directory structure)
- root node of the tree is the root directory
- internal nodes are directories
- leaf nodes are either files or empty directories
- path in a tree = filepath



Pathnames in a UNIX directory tree



- path separator: **/** (forward slash)
- pathname: **dir1/dir2/.../dirn/filename**
- root directory path: **/**
- an absolute path name begins at root, eg:

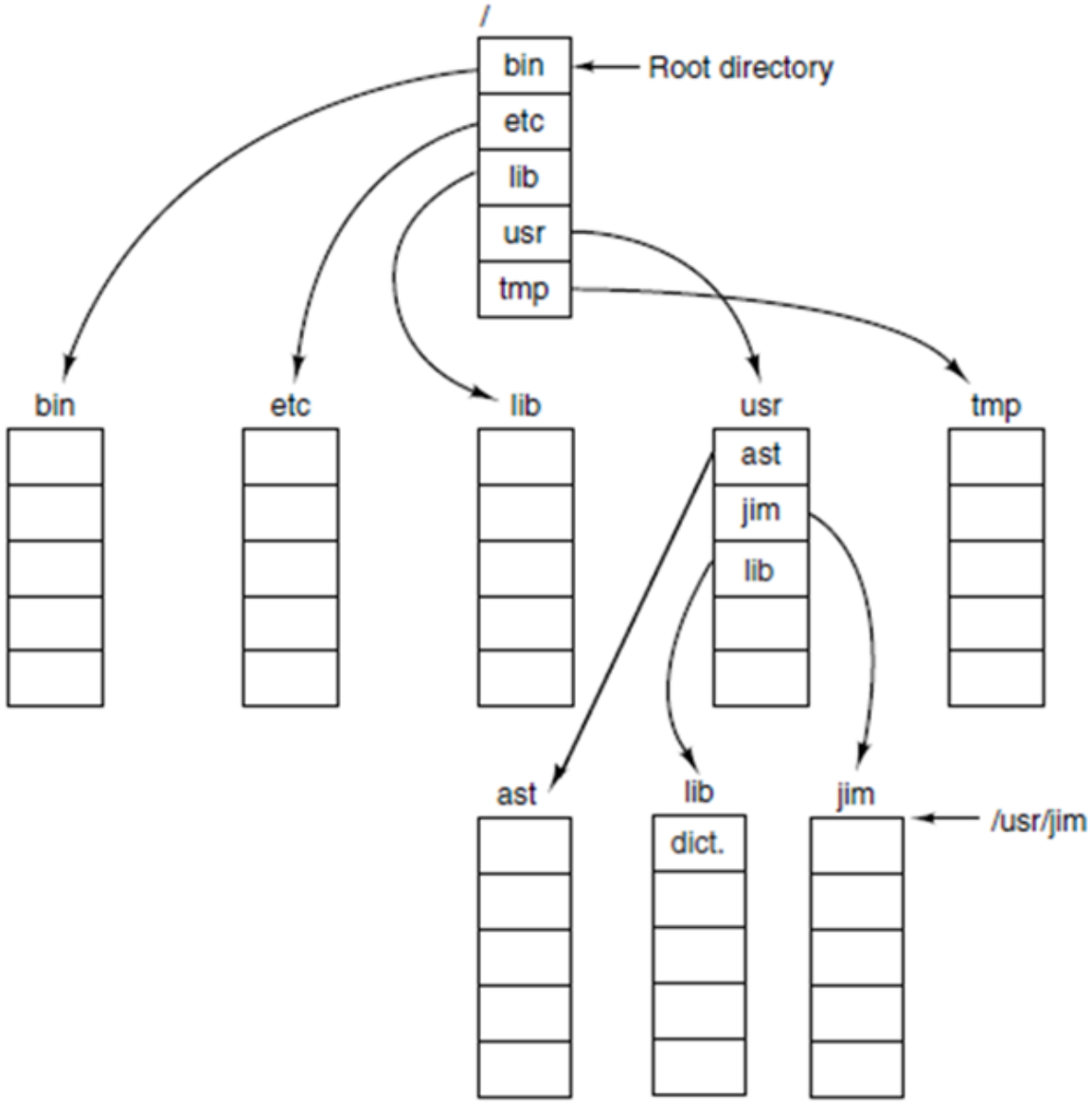
  /usr/jim
- a relative path name defines a path from the current directory, e.g.

  ./banker or **bin/cat** or **1.txt**
- every process has a working (current) directory
- can be changed using `chdir()` system call:

```
int chdir(const char *path);
```

UNIVERSITY OF CALGARY

Pathnames in a UNIX directory tree



- every directory has at least 2 entries:
  - pointer to current directory: `.` (dot)
  - pointer to parent directory: `..` (dotdot)
- dot and dotdot entries:
  - cannot be deleted
  - they are just pointers
  - directory containing only . and .. entries is considered empty
- weird but true example:
  - `/usr/jim`
  - `/./etc/../lib/./../usr/lib/../jim`
  - `../../../../../../../usr/jim`

  all refer to the same directory

UNIVERSITY OF CALGARY

Directory Operations

Directory operations in UNIX

- create — an empty directory is created (with `"."` and `".."` entries)
- delete — only empty directories can be deleted (`"."` and `"."` entries do not count)
- opendir — analogous to open for files
- closedir — analogous to close for files
- readdir — returns the next entry in an open directory
- rename — just like file rename for files
- link — technique that allows a file to appear in more than one directory
- unlink — a directory entry is removed. If the file being unlinked is only present in one directory (the normal case), it is removed from the file system. If it is present in multiple directories, only the path name specified is removed. In UNIX, the system call for deleting files (discussed earlier) is, in fact, `unlink`.

Assignment Help

Assignment help – converting string ⟶ 2 integers

```
void convert_using_sscanf(const std::string &s) {
  std::cout << "convert(\"" << s << "\")=";
  long n1, n2;
  if (2 != sscanf(s.c_str(), "%ld %ld", &n1, &n2)) {
    std::cout << "Error, could not convert\n";
  } else {
    std::cout << n1 << "," << n2 << "\n";
  }
}

int main(int, char **) {
  convert_using_sscanf("123 4883");
  convert_using_sscanf("  7   8   ");
  convert_using_sscanf("11");
  convert_using_sscanf("not a number 8");
  convert_using_sscanf("    ");
  convert_using_sscanf("");
  convert_using_sscanf("9,10");
  convert_using_sscanf("123 x");
  convert_using_sscanf("1 2 3");
  convert_using_sscanf("4 5x");
  return 0;
}
```

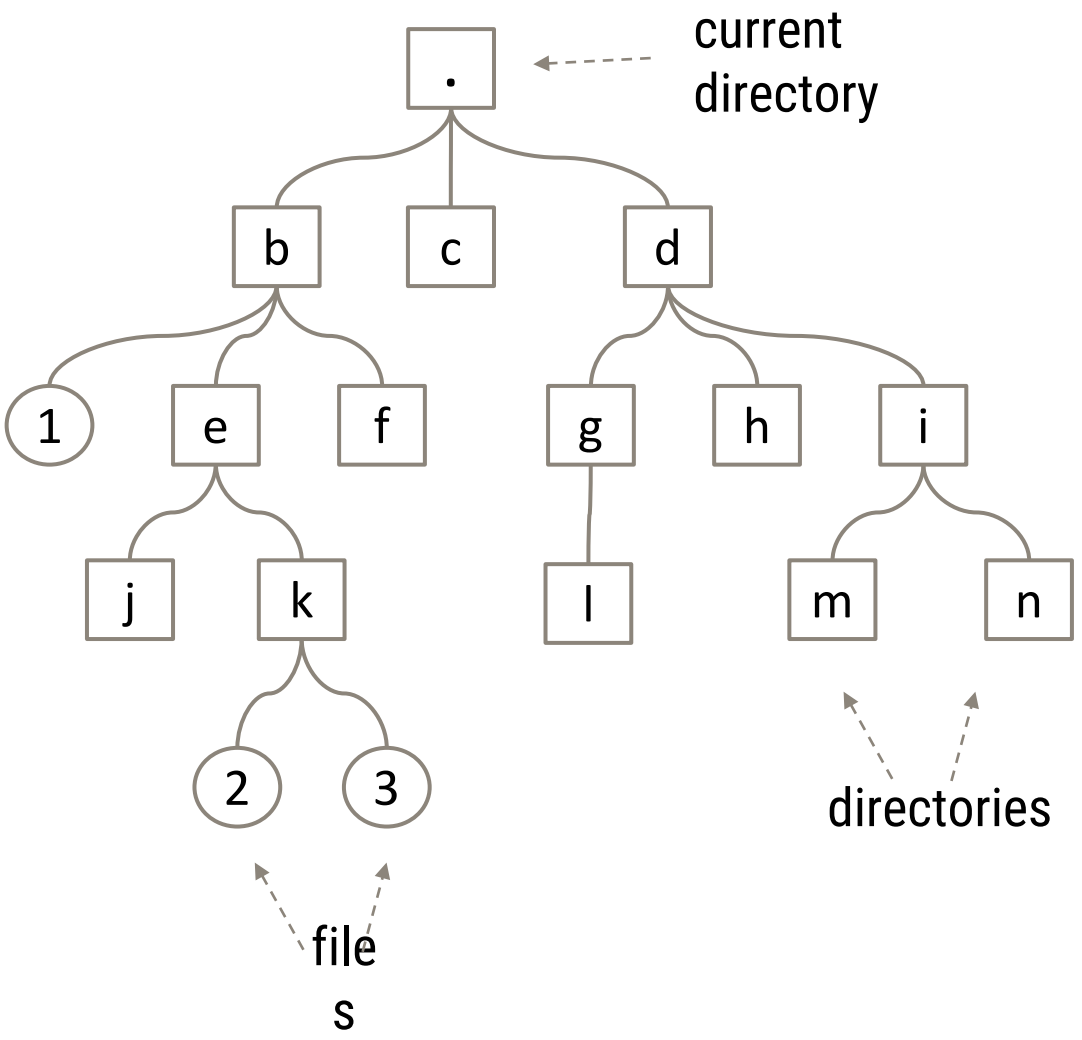
```
$ ./main
convert("123 4883")=123,4883
convert("  7   8   ")=7,8
convert("11")=Error, could not convert
convert("not a number 8")=Error, could not
convert
convert("    ")=Error, could not convert
convert("")=Error, could not convert
convert("9,10")=Error, could not convert
convert("123 x")=Error, could not convert
convert("1 2 3")=1,2
convert("4 5x")=4,5
```

https://replit.com/@jonathanwhudson/str22ints

Assignment help

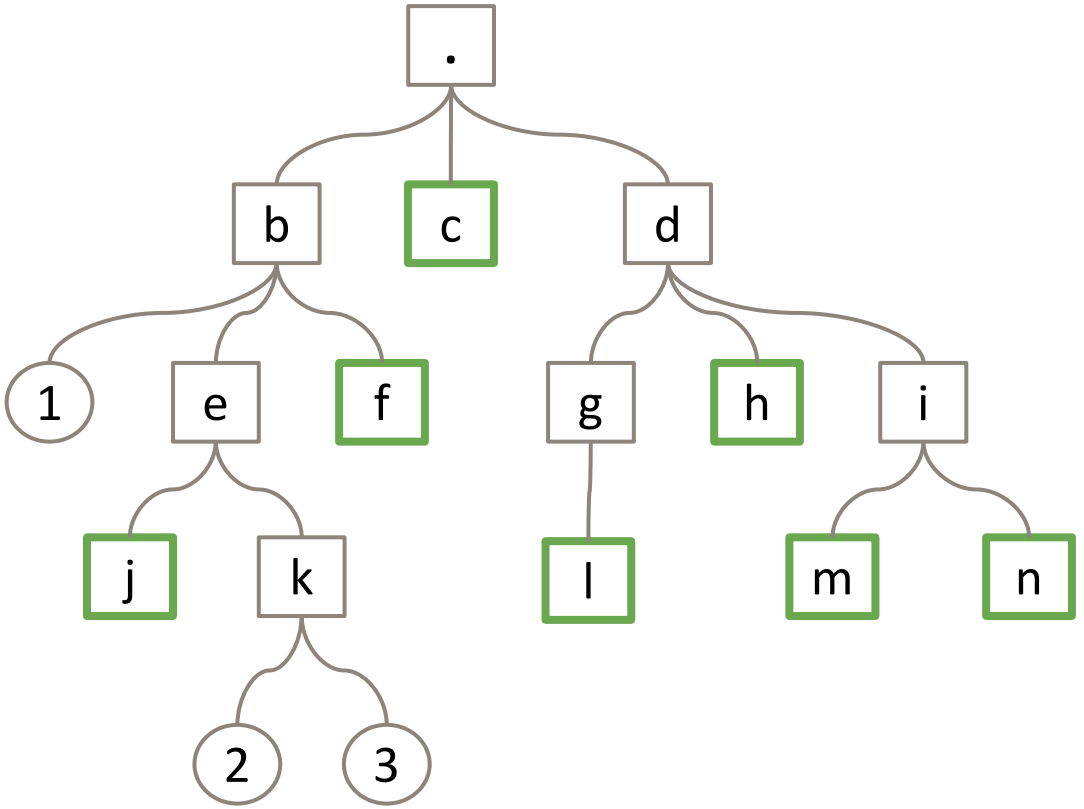
- files (1)
- directories [e]
- this directory has only 3 files:

  `"b/1"`, `"b/e/k/2"` and `"b/e/k/3"`



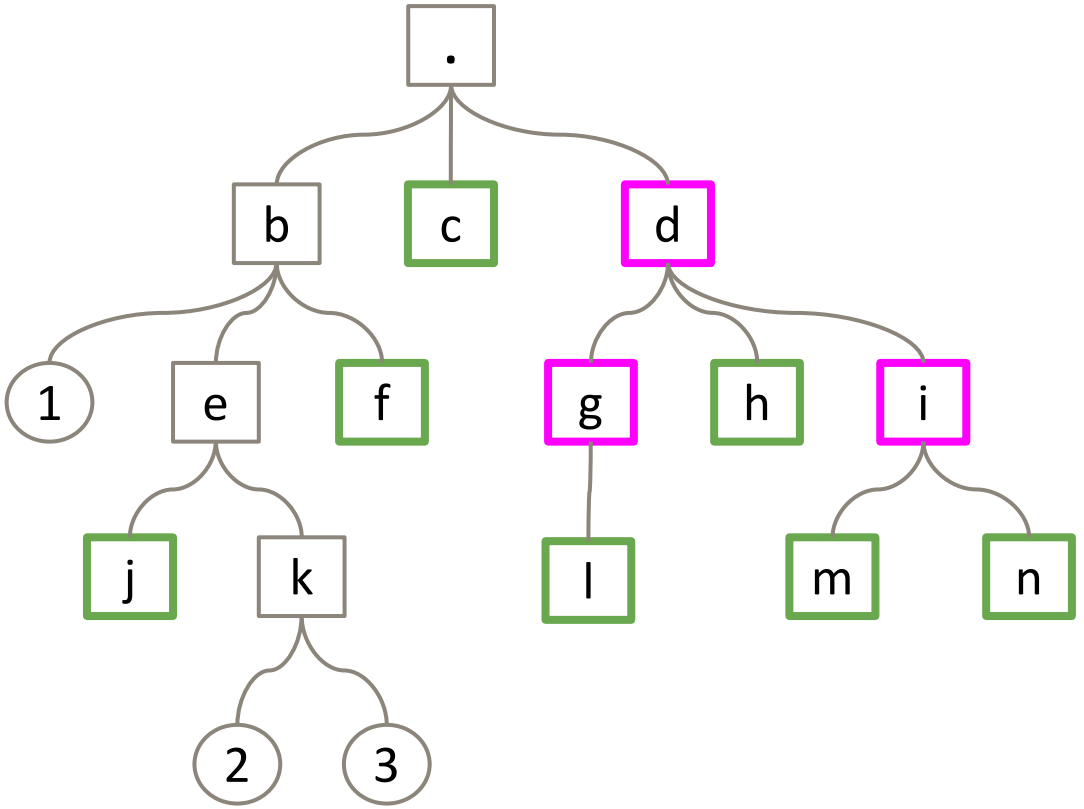
Assignment help

- 7 empty directories f



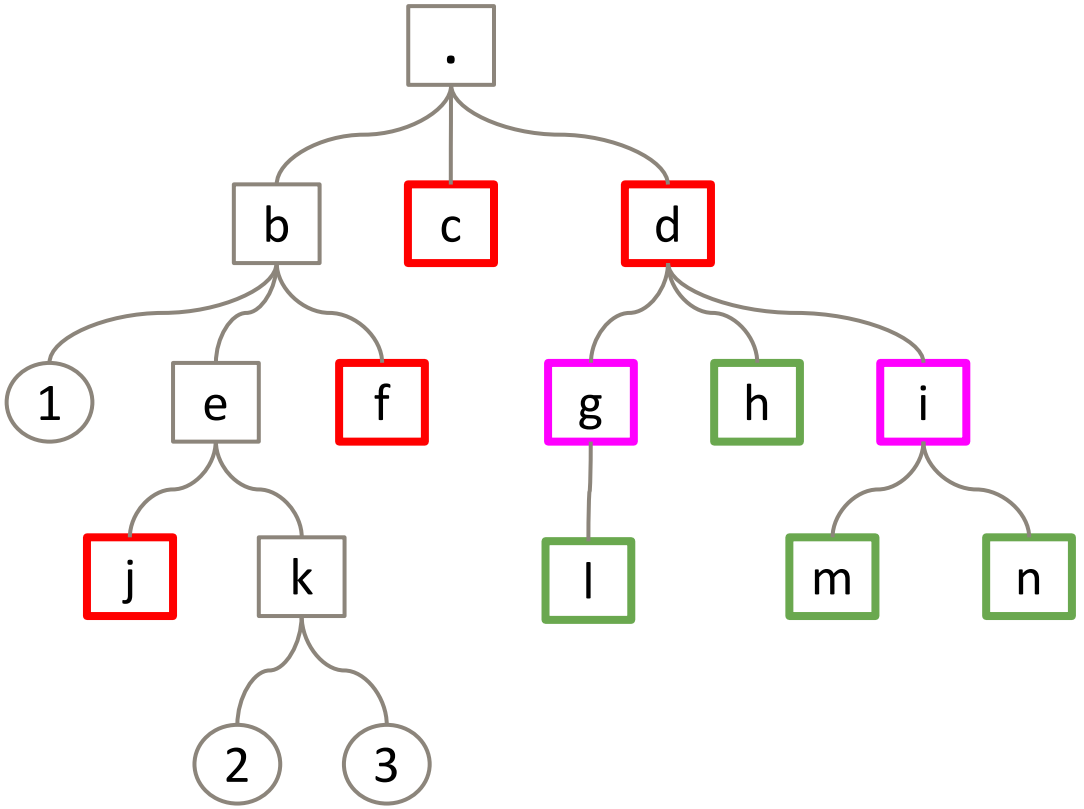
Assignment help

- 7 empty directories [f]
- 10 vacant directories [f] [d]
  - recursively contain no files
  - all empty directories are also vacant



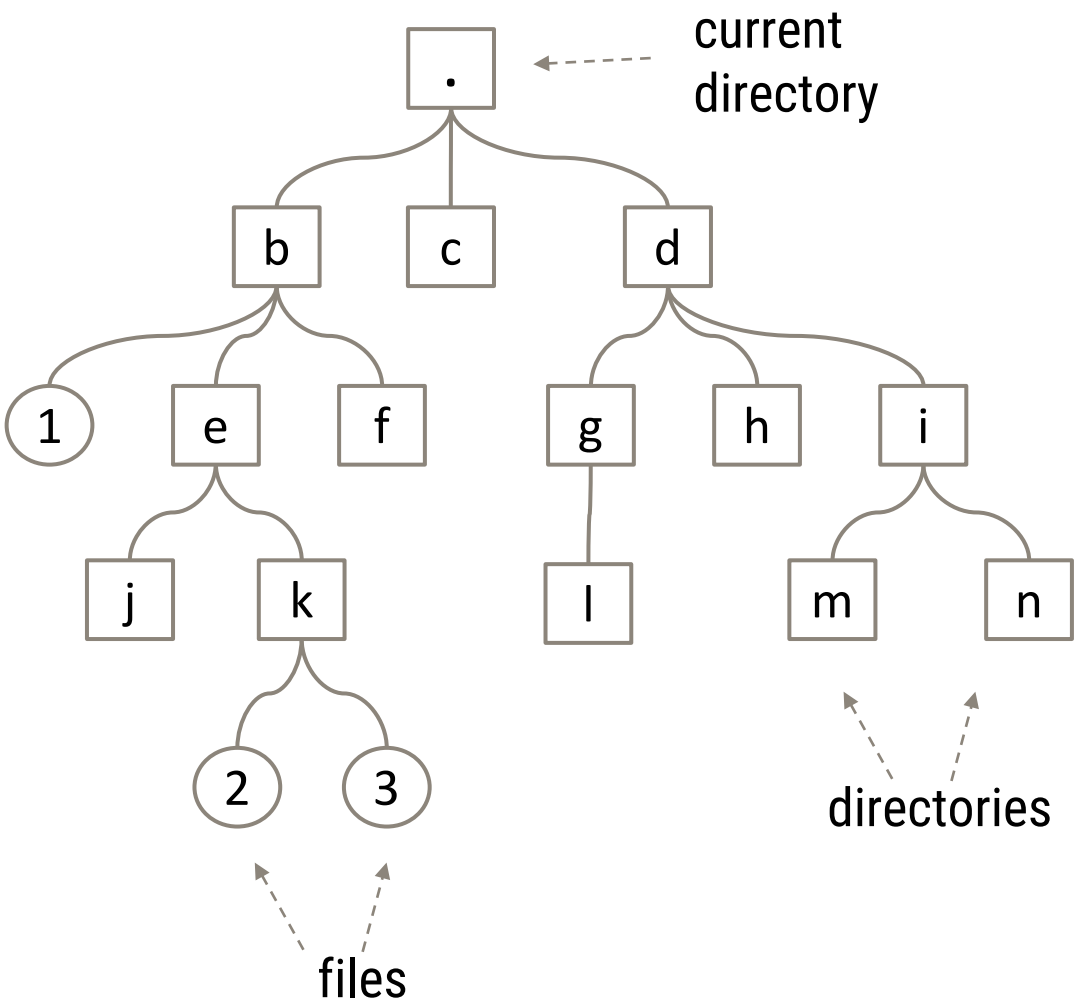
Assignment help

- 7 empty directories [f]
- 10 vacant directories [f] [d]
  - recursively contain no files
  - all empty directories are also vacant
- 4 top level vacant directories [d]
  - list of vacant directories, where descendants of already reported vacant directories are removed



Recursively counting all files

- how do we write a recursive function to count all files in current directory?
  - `countf(".")` should return `3`
  - `countf("d/g")` should return `0`

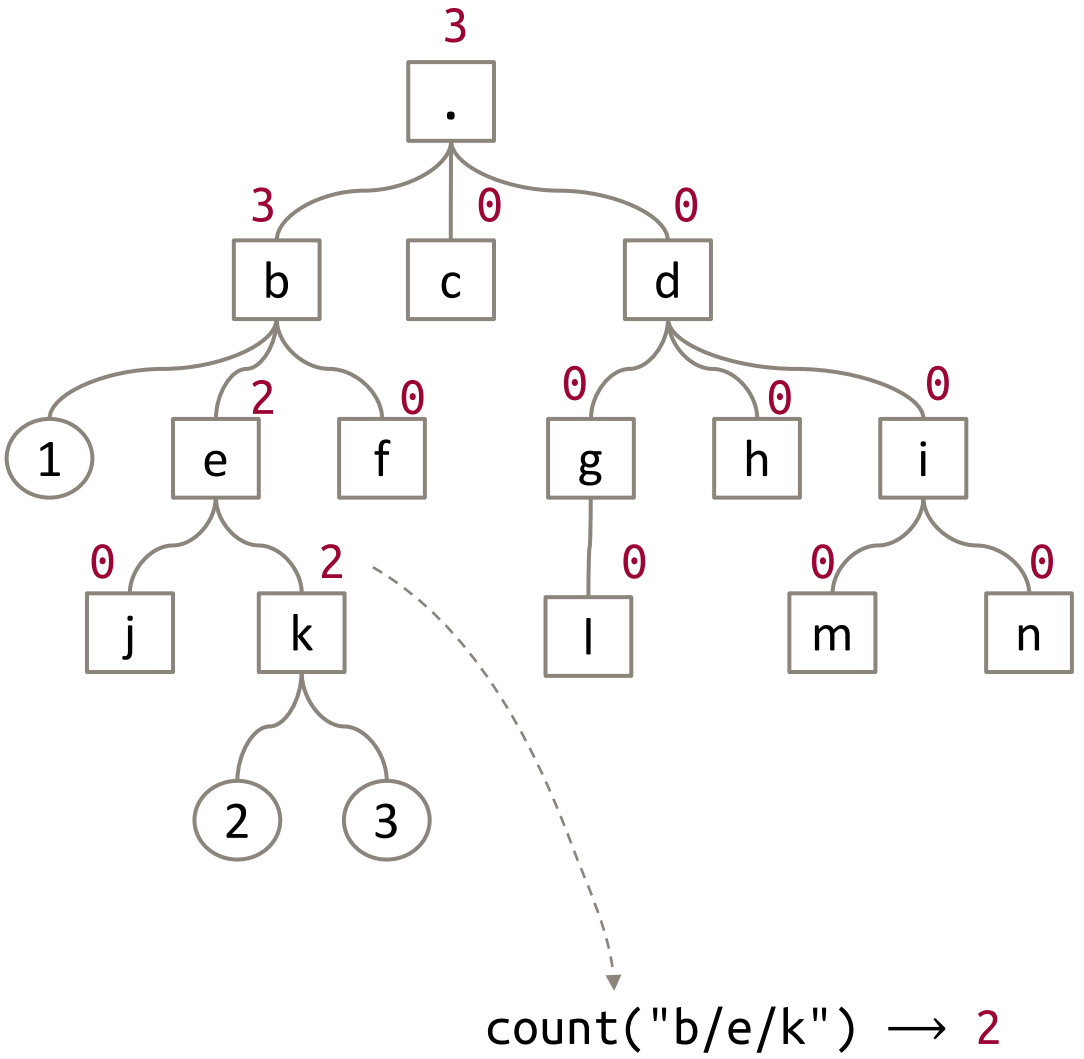


Recursively counting all files

Pseudocode:

```
int countf(path):
  N = 0
  for each entry in
opendir(path):
    skip "." and ".." entries
    if entry is file:
      N = N + 1
    else if entry is dir:
      N = N +
countf(path+"/"+entry)
  return N
```

Note: if `count(path) == 0` then `path` represents a vacant directory.

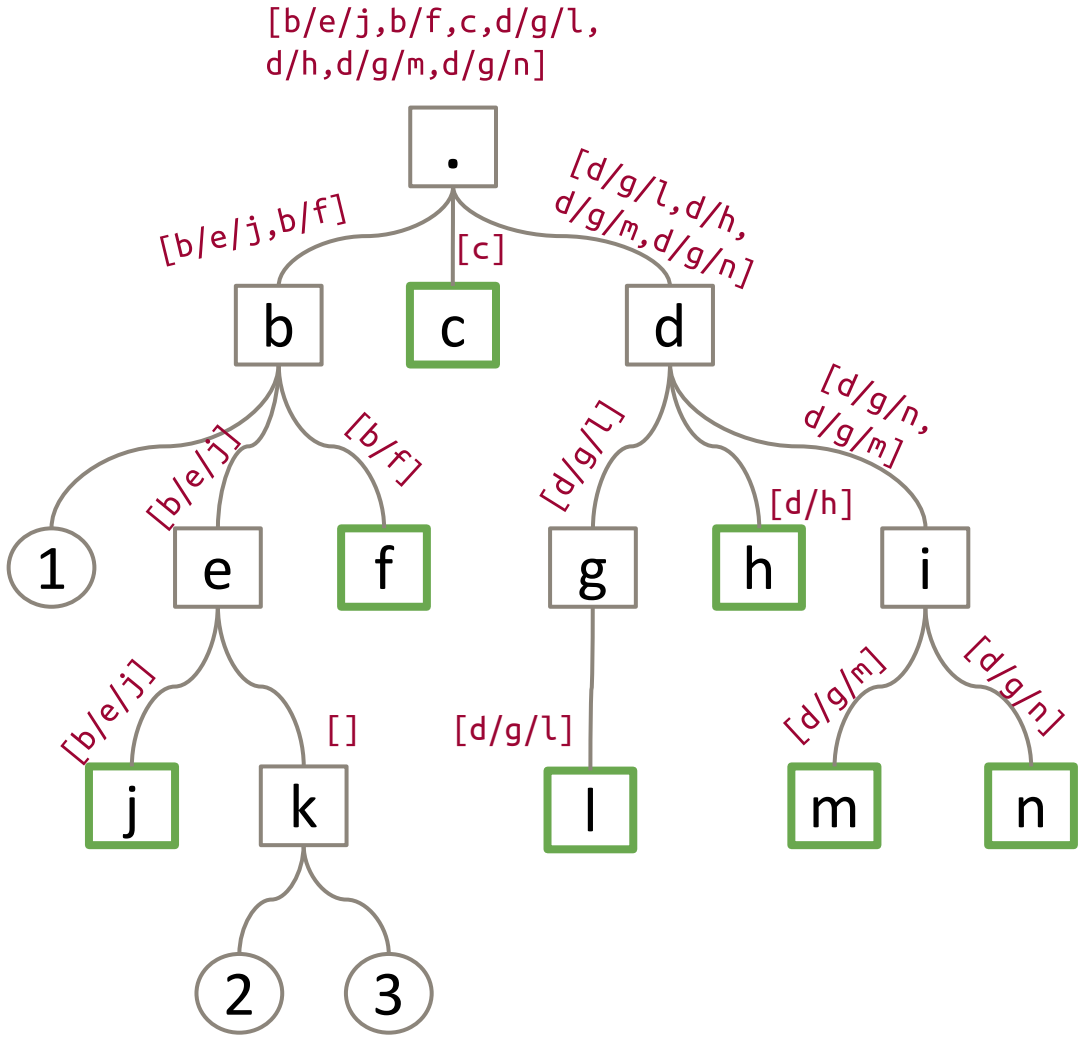


Recursively finding all empty directories

Pseudocode:

```
vector<string> findd(path):
  ourselves = [ path ]
  subdirs = []
  for each entry in opendir(path):
    skip "." and ".." entries
    if entry is file or dir:
      ourselves = []
    if entry is dir:

subdirs.append(findd(path+"/"+entry))
  return ourselves + subdirs
```

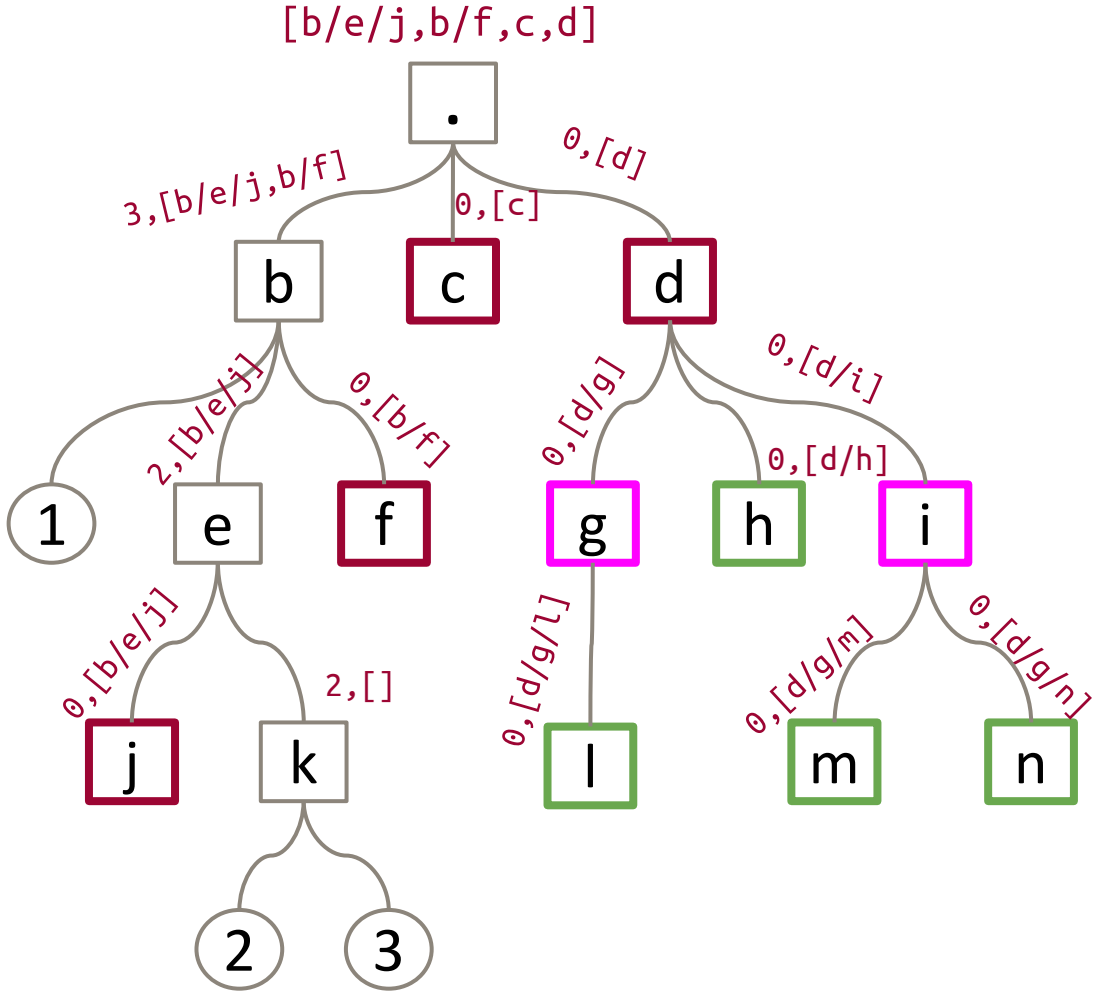
How would you need to adjust the code above to return all top level vacant directories?



Recursively finding all top-level vacant directories

Pseudocode:

```
struct VacantResult:
  /* recursive count of all files */
  int nfiles;
  /* top-level vacant directories */
  std::vector<std::string> dirs;

VacantResult vacant(path):
  combined code for finding
recursive
  file counts and empty directories

  if N == 0
    return {0, this directory}
  else
    return {N, subdirs from
recursion}
```



UNIVERSITY OF CALGARY

Review

Summary

- Filesystems
- File Operations
- Directories
- Directory Operations
- Assignment Help

Onward to …
Threads

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY