Resolving Pathnames

What to Do?

- Convert pathnames to common interface
  - Absolute pathname
- Different ways to access files
  - Relative to root directory
  - Relative to current directory
  - Relative to previous directory
  - Relative to $HOME
  - Relative to $PATH

Relative to Root Directory

- Absolute pathname
- Can only occur at the start of the path
- $PWD = /home/faculty/cop4610t
  - /bin/bash
  - /bin/bash
- Nothing to convert

Relative to Current Directory

- Default case
- Can occur anywhere in the path
- $PWD = /home/faculty/cop4610t
  - ./lectures/path_resolution.pdf
  - lectures/path_resolution.pdf
  - /home/faculty/cop4610t/lectures/path_resolution.pdf
- Can just ignore ./
  - Exception is when executing commands...

Relative to Previous Directory

- Access parent directory
- Can occur anywhere in path
- $PWD = /home/faculty/cop4610t/public_html
  - ../assignments
  - /home/faculty/cop4610t/assignments
- Use ../ to signal removing the last directory off of the current working directory
  - Take note of root directory...

Relative to $HOME

- Used to quickly access home directory
- Can only occur at start of path
- $PWD = /home/faculty/cop4610t/lectures
  - ~/assignments/project1
  - /home/faculty/cop4610t/assignments/project1
- Expand ~/ to value of $HOME

Relative to $PATH

- Only used for commands
- $PWD = /home/faculty/cop4610t
  - ls
  - /bin/ls
- If pathname satisfies both
  - Is a command (not a regular argument)
  - Does not contain any /'s
- Then you need to try each path in the $PATH

Relative to $PATH

- $PATH
  - ~/bin/git/:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin
- Need to split by ':' delimiter
  - ~/bin/git/
  - /usr/kerberos/bin
  - /usr/local/bin
  - /bin
  - /usr/bin
- Concatenate items in $PATH with the provided pathname
  - May need to convert these into absolute pathnames first (e.g. ~/bin/git)
- Test each until there is a match that is a regular file
  - Execute first match
  - If all fail, notify user

When to Convert the Paths?

- External commands
  - Expand the command
  - Do not expand the arguments
- Built-in commands
  - Do not expand the command
  - Do not expand the arguments
    - cd is the one exception
- However, you will have to look for environmental variables in all cases
  - e.g. ls $PWD/assignments

Parsing the Path

```
char **resolve_paths(char **args) {
    int i;
    for (i = 0; args[i] != NULL; i++) {
        args[i] = expand_path(args[i], is_command(args, i));
    }
    return args;
}

int is_command(char **args, int i) {}
    //returns 0 for argument, 1 for external command, 2 cd, 3 for other built-in commands
char *expand_path(char *path, int cmd_p) {}
    //returns expanded argument, does nothing in many cases (determined by is_command)
```

- Where to start???

Bottom-up Design

- We started planning the project design top-down
  - Decomposes task into smaller pieces
- However, it's less clear how to break this up
- Instead, lets build up with utility functions

Utility Function Ideas: Pathing

- Expand previous
  - Remove trailing directory from passed in path
- Expand home
  - Gets value in $HOME
  - Attaches it to passed in path
- Expand path
  - Gets value in $PATH
  - Tests each with passed in path
- Get current working directory
  - Gets value in $PWD

Utility Function Ideas: Strings

- Split
  - Breaks a string into an array of strings
  - Delimited by a character or string separator
- Concatenate pathnames
  - Combines two strings into one
  - Separated with '/' (may be included in first string...)
- Count
  - Counts the number of occurrences of a type of character in a string
- Is member?
  - Checks array for existence of an item

Utility Function Ideas: File Checking

- Exists?
  - Check if passed in file exists
- File?
  - Check if passed in file is a regular file
- Directory?
  - Check if passed in file is a directory file

Utility Function Ideas: Memory

- Array size
  - Compute number of items in 2D array
- Safe malloc
  - Checks for errors in calloc
- Big free
  - Frees all memory in a 2D array

Design Choice

- Personal preference
  - Start with top-down design
    - Breaks up work needed
    - Sets up milestones, use cases, general features
    - Makes task seem less daunting
    - Easier to do use case testing
  - Plan rest of the system with bottom-up design
    - Prevents continually remaking the same tools
    - Makes code cleaner
    - Allows dealing with memory management at the lowest level
    - Easier to do unit testing, and prove correct
- Can mix and match to fit your own methodology