PA2 Part 1: Executables

SKK UNIVERSITY

Update

- **2024-05-03**
  - Made the error message format much easier to understand.
    - » “command: ERROR MESSAGE FROM strerror()” ->
      - » “command: %s” where the string is the return value of strerror().
  - Added _GNU_SOURCE to mv
- **2024-05-02**
  - Removed the duplicate slide for longopts

PA2

- **Goal**
  - The first goal is to implement 6 executables found in GNU Core Utilities (coreutil).
    - » head
    - » tail
    - » cat
    - » cp
    - » mv
    - » rm
  - Although the executables you will implement are simplified, most of the details are based on the man page of these executables, so you can refer to these if needed.

PA2

- **Compilation**
  - These executables should be prefixed with pa2_ (i.e. pa2_head, pa_tail, etc.).
  - The source files should be in pa2/executable_src while the executables should be in pa2/bin.
  - *make* should be able to compile and remove these executables
  - pa2/
    - » **Makefile**
    - » **executable_src/**
    - » shell_src/
    - » **bin/**

PA2

- **Running commands**
  - If a command has [...], this means that the parameter is optional.
  - The file parameter is sometimes optional (i.e. [FILE]).
  - If that is the case, the command must read standard input instead.
  - If argument has ... , it means more than one argument can be passed.

Hint

SK UNIVERSITY

Extracting Arguments and Options

- **getopt (POSIX) vs getopt_long (GNU) vs argp (GNU)**
  - You can use argp, getopt, or getopt_long to get the arguments and options more easily.
  - argp and getopt_long more powerful, but they are not part of the standard C library, so you have to specify std=gnuXX in your Makefile instead of std=cXX.
  - The following example uses getopt_long, but you can use any functions you like.
  - Long option names are optional. If you plan on using these, refer to the manpage for the command to see the full name of the option.

getopt_long

- ```
  int getopt_long(int argc, char* const argv[],
      const char* optstring,
      const struct option* longopts, int* longindex);
  ```
  - Goes through each option when called repeatedly (like strtok)
  - You can use switch-case to check for the option.
  - argc, argv refer to the argc, argv in main()
    - » After calling getopt, argv would be changed so that the non-option arguments will be at the end.
    - » *optind* is the index of the first non-option argument.
  - optstring refers to the short names (letter) for the options (i.e. “abc:” for options –a –b –c VALUE)
    - » If “:” is after a letter, it means that the option requires an argument
  - longindex can be NULL
  ```

Iterating Through Options

- You can use switch-case to check for the option.
- If opt is ‘?’, it would print `argv[0]: invalid option – ‘...’`

```c
int opt;
while ((opt = getopt_long(argc, argv, "abc:h", long_options, NULL)) != -1) {
  switch (opt) {
    case 'a':
      a = 1; // a = true;
      break;
    case 'b':
      b = 1; // b = true;
      break;
    case 'c':
      c = strtoll(optarg, NULL, 10);
      break;
    case 'h':
      // print help message
      return 0;
    default: // actual value: ('?') -> unknown option
      fprintf(stderr, "Try '%s --help' for more information.\n", argv[0]);
      return 1;
  }
}
```

longopts

- **longopts is an array of options, which contains the long name and the type of option.**
  - NULL can be used for flag

```
struct option {
  const char* name; // long option name
  int has_arg; // whether the option takes an argument
               // (no_argument, required_argument, optional_argument)
  int* flag; // if not NULL, set the value of int to val
  int val; // short name for the option
};
```

Example

```c
#include <ctype.h>
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[]) {
  int uppercase_flag = 0;
  int lowercase_flag = 0;
  long long step = 1;

  // Not every option needs to be in here
  // if it does not have a long name (i.e. '-a')
  struct option long_options[] = {
      // {
      // name,
      // does option have arg?,
      // if you want getopt_long to automatically set the flag to a
value,
      // shortname
      // }
      {"uppercase", no_argument, NULL, 'u'},
      {"lowercase", no_argument, NULL, 'l'},
      {"step", required_argument, NULL, 'n'},
      {"help", no_argument, NULL, 'h'},
      {NULL, 0, NULL, 0}};
```

Example

```c
int opt;
while ((opt = getopt_long(argc, argv, "uln:h", long_options, NULL)) != -1) {
  switch (opt) {
    case 'u':
      uppercase_flag = 1;
      break;
    case 'l':
      lowercase_flag = 1;
      break;
    case 'n':
      step = strtoll(optarg, NULL, 10);
      break;
    case 'h':
      printf("Usage: %s [OPTION]... TEXT...\n", argv[0]);
      printf("Change the case of the text.\n");
      printf("\n");
      printf(
          "Mandatory arguments to long options are mandatory for short "
          "options too.\n");
      printf("  -u, --uppercase      change the text to uppercase\n");
      printf("  -l, --lowercase      change the text to lowercase\n");
      printf("  -n, --step=STRING    change the text step\n");
      printf("  -h, --help           display this help and exit\n");
      return 0;
    default:
      fprintf(stderr, "Try '%s --help' for more information.\n", argv[0]);
      return 1;
  }
}
```

Example

```
  // If the program requires arguments, check that there is at least one
  // optind refers to the argument index
  // while argc refers to the number of arguments
  // Error handling is also done in this way
  if (optind >= argc) {
    fprintf(stderr, "%s: missing input\n", argv[0]);
    return 1;
  }

  for (int i = optind; i < argc; i++) {
    char* text = argv[i];
    for (int j = 0; j < strlen(text); j += step) {
      if (uppercase_flag)
        text[j] = toupper(text[j]);
      else if (lowercase_flag)
        text[j] = tolower(text[j]);
    }
    printf("%s ", text);
  }

  return 0;
}
```

Errors

Errors

- **Handling errors**
  - When an error occurs in the program you implement, you have to print a message to the standard error.
  - The actual message format would be mentioned on the slide for the command
    - » But most of these errors can be derived from “command: %s” where the string is the return value of strerror(), so you do not have to hard-code most of these errors.
  - The format is based on the error handling of the actual commands, so please refer to that as well if needed.
  - If you encounter any other error not found in the Error section of the command, print “command: %s” where the string is the return value of strerror().

Programs

SK UNIVERSITY

head

- **Synopsis**
  - pa2_head [OPTION] [FILE]
- **Description**
  - Prints the first 10 lines of a FILE to standard output.
  - If the file has less than 10 lines, it will print the whole file without any padding.
  - If no FILE is provided, or FILE is -, read standard input.
  - -n NUM prints up to NUM lines instead of 10
  - It is guaranteed that NUM is not negative.
- **Error**
  - When FILE does not exist, print “pa2_head: cannot open 'FILE' for reading: No such file or directory”. Note that “No such file or directory” comes from errno. Also note that FILE refers to the actual filename, please do not print FILE, but the actual filename.
  - When K is not a number, print “pa2_head: invalid number of lines: ‘K’”

tail

- **Synopsis**
  - pa2_tail [OPTION] [FILE]
- **Description**
  - Prints the last 10 lines of a FILE to standard output.
  - If the file has less than 10 lines, it will print the whole file without any padding.
  - If no FILE is provided, or FILE is -, read standard input.
  - -n NUM prints up to NUM lines instead of 10
  - It is guaranteed that NUM is not negative.
- **Error**
  - When the file does not exist, print “pa2_tail: cannot open 'FILE' for reading: No such file or directory”.
  - When K is not a number, print “pa2_tail: invalid number of lines: ‘K’”

cat

- **Synopsis**
  - pa2_cat [FILE]...
- **Description**
  - Concatenate FILE(s) to standard output.
  - Note for concatenation: Do not add a newline or space when concatenating. Just print the contents of the files as is.
  - You don’t need to consider the maximum size of the files.
  - If no FILE is provided, or FILE is -, read standard input.
- **Error**
  - When a a file does not exist, print “pa2_cat: FILE: No such file or directory”.
  - When the file is a directory, print “pa2_cat: FILE: Is a directory”.
  - When you don’t have permissions to open a file, print “pa2_cat: FILE: Permission denied”.

cp

- **Synopsis**
  - pa2_cp SOURCE DEST
  - pa2_cp SOURCE... DIRECTORY
- **Description**
  - Copy SOURCE to DEST. If DEST exists and it is a file, overwrite DEST completely.
  - If it is an existing DIRECTORY, copy multiple SOURCE(s) to DIRECTORY.
  - Ignore cases when SOURCE is a directory. SOURCE is guaranteed to be a file
- **Error**
  - When no arguments are passed, print “pa2_cp: missing file operand”
  - When only one argument is passed, print “pa2_cp: missing destination file operand after ‘SOURCE’”
  - When a file does not exist, print “pa2_cp: cannot stat ‘FILE’: No such file or directory”
  - When you cannot access a file, print “pa2_cp: cannot open ‘FILE’ for reading: Permission denied”
  - When you cannot access a directory, print “pa2_cp: cannot stat ‘DIRECTORY’: Permission denied”

mv

- **Synopsis**
  - pa2_mv SOURCE DEST OR pa2_mv SOURCE... DIRECTORY
- **Description**
  - Rename SOURCE to DEST. If DEST exists and it is a file, overwrite DEST completely.
  - If it is an existing DIRECTORY, move multiple SOURCE(s) to DIRECTORY.
  - SOURCE can be a directory as well.
  - TARGET: DEST or DIRECTORY/basename(SOURCE)
- **Error**
  - When no arguments are passed, print “pa2_mv: missing file operand”
  - When only one argument is passed, print “pa2_mv: missing destination file operand after ‘SOURCE’”
  - When a file does not exist, print “pa2_mv: cannot stat ‘FILE’: No such file or directory”
  - When a directory cannot be accessed, print “pa2_mv: cannot move ‘SOURCE’ to ‘TARGET’: Permission denied”
  - When the SOURCE and TARGET are the same, print mv: ‘SOURCE' and ‘TARGET’ are the same file
  - When the DIRECTORY is a subdirectory of SOURCE, print “mv: cannot move ‘SOURCE’ to a subdirectory of itself, ‘TARGET’”

mv

- `int rename(const char* oldpath, const char* newpath);`
  - Renames a file, moving the file from oldpath to newpath, overwriting any file at newpath.
  - In stdio.h
  - Possible Errors:
    - » EACCES: Not enough write permissions for oldpath or the parent directory of oldpath or newpath; or not enough search (execute, x) permissions for one of the ancestors of oldpath/newpath.
    - » EISDIR: newpath is an existing directory, but oldpath is not a directory.
    - » EINVAL: newpath is a subdirectory of oldpath
    - » ENOENT: newpath, oldpath does not exist; some ancestor in newpath does not exist; newpath or oldpath is an empty string
    - » ENOTDIR: ancestor in newpath, oldpath is not a directory; oldpath is a directory, but newpath is not.
    - » ENOTEMPTY || EEXIST: newpath is a non-empty directory

mv

- `char* basename(const char* path);`
  - Gets the name of the file without the directory part (i.e. /home/spl/file -> file)
  - Use the GNU function for basename at string.h
    - » Requires you to use `std=gnuXX –D_GNU_SOURCE` in your Makefile
  - Do not include libgen.h as that uses the POSIX version instead of GNU.
    - » Issue with this function is that it may modify the path and cause segfault.

rm

- **Synopsis**
  - pa2_rm FILE...
- **Description**
  - Remove FILE(s); no need to consider directories and protected files (files with restricted permissions).
  - If an errors occur for one file, the other files should still be removed.
- **Error**
  - When no arguments are passed, print “pa2_rm: missing operand”
  - When a file does not exist, print “pa2_rm: cannot remove 'a4': No such file or directory”

rm

- `int unlink(const char *pathname);`
  - Deletes a name from the file system.
  - In unistd.h
  - Possible Errors:
    - » EACCES: Not enough write permissions for the parent of pathname or not enough search (execute) permissions for one of the ancestors of pathname
    - » EISDIR: pathname is a directory
    - » ENOENT: some ancestor in pathname does not exist; pathname is empty
    - » ENOTDIR: ancestor in pathname is not a directory;
    - » EPERM: Not enough permissions to unlink files/directorie