Project 0

Alexandre Tiard
tiard@cs.ucla.edu
Office hours : M 1.30-3.30 BH2432

CS 111

04/06/18

Outline

Installing Build tools

Basic tar functionality

File Descriptors

IO redirection

getopt

more getopt

more getopt

Basic functionality of gdb

Installing Build Tools

- For this project, you need:
  - Linux
  - gcc, libc, make, gdb

- Install gcc, libc, make and gdb with

```
sudo apt-get install build-essential
```

tar

- Create a tar:

```
tar -cvzf test.tar.gz file1.pdf file2.png source.c
```

- The `c` option tells tar to create the archive

- Extract a tar:

```
tar -xvf test.tar.gz
```

- The `x` option tells tar to extract the archive

- After you tar your assignment, untar it and make sure everything is there

File Descriptors

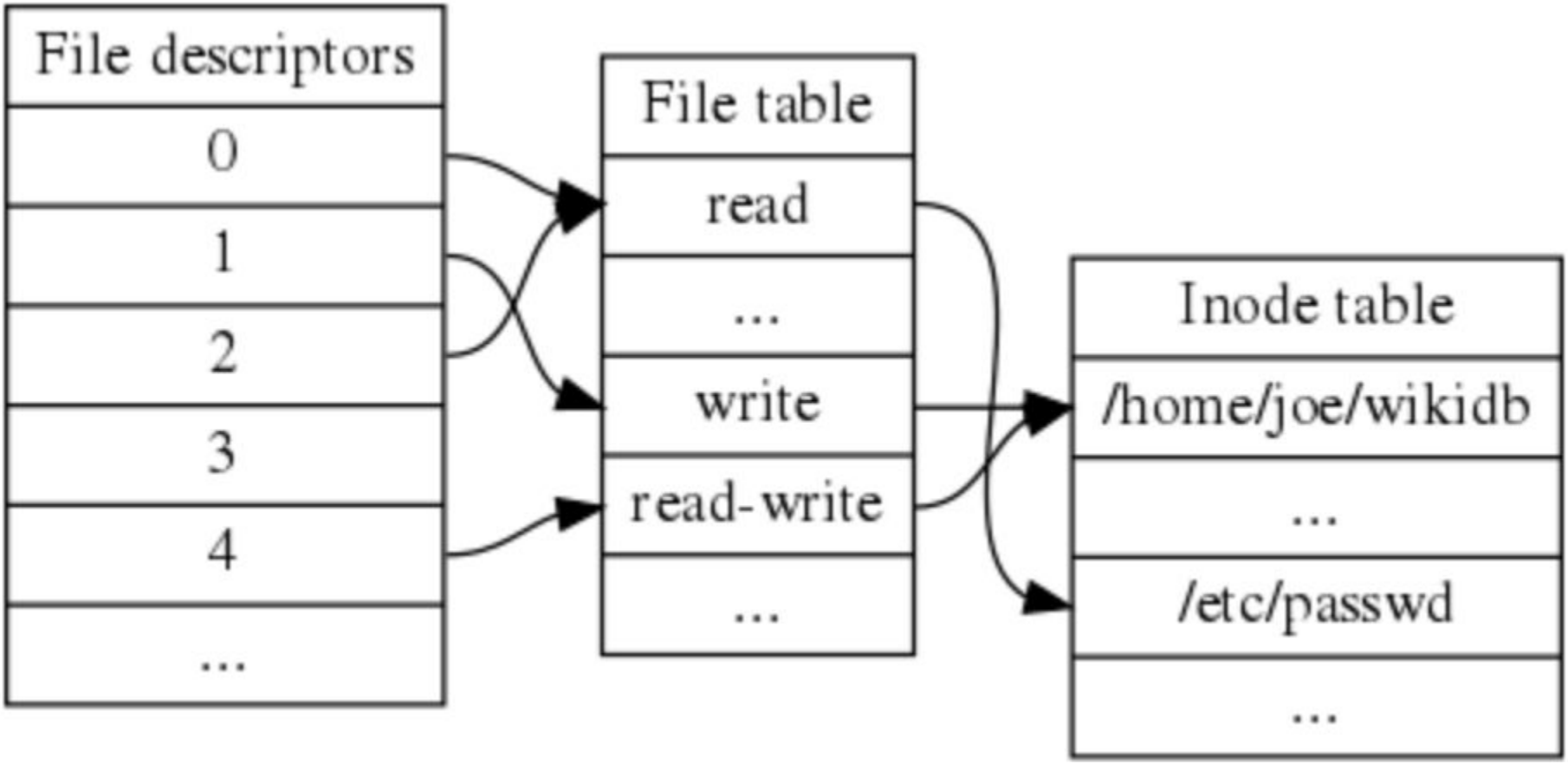
- An integer
- Used by file access API (read, write)
- When a process starts, it normally has access to :

0: standard input (read only)

1: standard output (write only)

2: standard error (write only)

File Descriptors : for a single process



I/O Redirection

- open(2) <- input the pathname, outputs fd
- close(2)

I/O Redirection

- open(2) <- input the pathname, outputs fd
- close(2) <- closes file descriptor, so it can be reused
- dup(2)

I/O Redirection

- open(2) <- input the pathname, outputs fd
- close(2) <- closes file descriptor, so it can be reused
- dup(2) <- duplicates file descriptor, to lowest available fd

How does this work for input redirection?

I/O Redirection

- open(2) <- input the pathname, outputs fd
- close(2) <- closes file descriptor, so it can be reused
- dup(2) <- duplicates file descriptor, to lowest available fd

How does this work for input redirection?

```
int ifd = open(newfile, O_RDONLY);
if (ifd >= 0) {
        close(0);
        dup(ifd);
        close(ifd);
}
```

getopt(3)

- Goal:

getopt(3)

- Goal: Processing the command line arguments using `argc` and `argv`
- Included in `unistd.h`
- Old way of processing command line arguments:

```
#include <stdio.h>

int
main(int argc, char **argv)
{
        int i;

        printf("argc = %d\n", argc);
        for (i=0; i<argc; i++)
                printf("arg[%d] = \"%s\"\n", i, argv[i]);
}
```

getopt(3)

- Goal: Processing the command line arguments using `argc` and `argv`
- Included in `unistd.h`
- Old way of processing command line arguments:

```
#include <stdio.h>

int
main(int argc, char **argv)
{
        int i;

        printf("argc = %d\n", argc);
        for (i=0; i<argc; i++)
                printf("arg[%d] = \"%s\"\n", i, argv[i]);
}
```

```
$ ./cmdline_basic test1 test2 test3 test4 1234 56789
cmdline args count=7
 exe name=./cmdline_basic
 arg1=test1
 arg2=test2
 arg3=test3
 arg4=test4
 arg5=1234
 arg6=56789
```

getopt(3)

Example : List files ‘ls’

ls -aFlL /etc
ls -a -l -FL /etc
ls -a -l -F -L /etc
ls -alLF /etc

getopt will process all of these, regardless of order

getopt(3)

int getopt(int *argc*, **char * const** *argv[]*, **const char ***optstring**);**

Inputs : argc and argv from the main, and the third is a string that defines the syntax. Options that require arguments are suffixed by a column (:)

Example : "df:mps:"

getopt(3)

Outputs:

getopt(3)

Outputs:

- If an option is successfully found, returns option character
- If all options have been parsed, returns -1
- If an unknown option , returns ‘?’
- If known option with missing argument, returns ‘?’

getopt(3)

- getopt is called repeatedly
- At each call, it returns the next command-line option that it found.
- If there a follow-on parameter, it is stored in optarg.
- If getopt runs in an undefined option, it returns a '?'
- When last command line option reached, returns -1
- Usually, calls to getopt are in a ‘while’ loop, with a ‘switch’ statement for each option

getopt(3)

Uses 2 external variables :

- extern char *optarg
- extern int optind

-optarg a is used when parsing options that take a name as a parameter, and points to that parameter.

-optind is the index to main()'s argument list

getopt_long

- Works like getopt, but accepts long options, with 2 dashes
- This is what you will have to use for the project
- Comes with a few extra arguments :

int getopt_long(int *argc*, **char * const** *argv[]*,
const char **optstring*,
const struct option **longopts*, **int** **longindex*);

getopt(3)

const struct option ***longopts**

- Struct option describes a single long option name
- The fields are :
  - const char *name : a string
  - int has_arg : does it take an argument (think of ':')
  - int *flag : a pointer used to determine how to act
  - int val : an int used to determine how to act (if the long option is equivalent to a short option, store short option in val)

If the flag is set to NULL, val is used to identify the option.

- longopts is an array of these structures, one per option
- The last element of the array should be {0, 0, 0, 0}

getopt(3)

int getopt_long(int *argc*, **char * const** *argv[]*,
const char **optstring*,
const struct option **longopts*, **int** **longindex*);

- Longindex stores the position of the current option in the longopts array

Getopt_long : an example

```
static struct option long_options[] = {
    {"add",     required_argument, 0,  0 },
    {"append",  no_argument,       0,  0 },
    {"delete",  required_argument, 0,  0 },
    {"verbose", no_argument,       0,  0 },
    {"create",  required_argument, 0, 'c'},
    {"file",    required_argument, 0,  0 },
    {0,         0,                 0,  0 }
};
```

getopt(3)

In while loop, while(1)

```
c = getopt_long(argc, argv, "abc:d:012",
          long_options, &option_index);
if (c == -1)
    break;

switch (c) {
case 0:
    printf("option %s", long_options[option_index].name);
    if (optarg)
        printf(" with arg %s", optarg);
    printf("\n");
    break;

case '0':
case '1':
case '2':
```

```
c = getopt_long(argc, argv, "abc:d:012",
          long_options, &option_index);
```

```
case 'a':
    printf("option a\n");
    break;

case 'b':
    printf("option b\n");
    break;

case 'c':
    printf("option c with value '%s'\n", optarg);
    break;

case 'd':
    printf("option d with value '%s'\n", optarg);
    break;

case '?':
    break;
```

gdb

- Debugger used to step through your program
- Start gdb with the executable you wish to debug:

```
gcc getopt_long.c -g -o getopt_long                    Home/Desktop
gdb getopt_long                                        Home/Desktop
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from getopt_long...done.
(gdb)
```

gdb

- Running the program from gdb:

```
(gdb) run --add option_added
Starting program: /media/psf/Home/Desktop/getopt_long --add option_added
option add with arg option_added
[Inferior 1 (process 13256) exited normally]
(gdb)
```

- `run` starts the program and you can pass arguments as you normally would from the command line

gdb

- Setting breakpoints:

```
(gdb) break getopt_long.c:32
Breakpoint 1 at 0x4007d9: file getopt_long.c, line 32.
(gdb)
```

- Running, we'll pause at the breakpoint

```
(gdb) run --add option_added
Starting program: /media/psf/Home/Desktop/getopt_long --add option_added

Breakpoint 1, main (argc=3, argv=0x7fffffffdaa8) at getopt_long.c:32
32                  printf(" with arg %s", optarg);
(gdb)
```

gdb

- `list` will show us the code surround the current line of execution:

```
(gdb) list
27          switch (c)
28          {
29            case 0:
30              printf("option %s", long_options[option_index].name);
31              if (optarg)
32                printf(" with arg %s", optarg);
33              printf("\n");
34              break;
35
36            case '0':
(gdb)
```

gdb

- `print` allows us to evaluate expressions from the gdb
- Most commonly used to print the value of a variable (your IDE does this nicely for you in a window)

```
(gdb) print this_option_optind
$2 = 1
(gdb) print option_index
$3 = 0
(gdb) print c
$4 = 0
```

gdb

- bt shows the backtrace, or call stack of the current execution

```
(gdb) bt
#0  main (argc=3, argv=0x7fffffffdaa8) at getopt_long.c:32
(gdb)
```

- `#0` refers to main's frame, and we can move around the call stack using `frame n`, where `n` is the frame we want to move to
  - In this example, we only have one function, but if we have multiple, we can jump around to each function in the call stack