Flinders UNIVERSITY
College of Science & Engineering

COMP2881 Operating Systems, DE
COMP9881 Operating Systems GE
Semester 1, 2020

CN Workshop04: Command line parsing, File metadata, Measurement

Due: (check submission time on FLO)

Lab 4 is comprised of 5 tasks; each task is worth 1 checkpoint. Following completion of the checkpoints, a copy of the source code and output for each task should be compiled into a single text file and uploaded to the relevant submission box on FLO for moderation. Speak to your demonstrator if you are unsure how to do this.

While discussion of programming tasks with fellow students in labs is welcome and encouraged, please be mindful of the University's **Academic Integrity** policy and refrain from simply copying another student's code (even if they agree!). You will gain a much greater understanding of the material if you take the time to work through your own solutions, and you will be expected to be able to explain your code before being awarded checkpoints. Remember, the demonstrators are there to help if you get stuck.

Objectives

Upon completing this lab, you should be able to:

- Write programs from scratch using the following tasks as a specification
- Complete each task with the code from the man pages
- Customise your programs with optional command line arguments
- Use data structures to collect data from system calls in your programs
- Measure how long file and network, operations take
- Use file operations to identify file metadata, open, read from, write to and close files

Learning out comes;
- LO2: Demonstrate the design and implementation of simple process and network level programs

Preparation

Create a new working directory for this workshop. Read the man pages on the functions included for each task. We searches for example usage of these functions. You may need to install valgrind from the command line with "apt-get install". It is recommended that you update the package repository before hand.

Task 1

Write a program that uses the function getopt() to parse command line options, print provided options, based on the example usage in the man page, see below.

Learning to use the command line parsing tools is an important step in creating flexible and useful programs for monitoring your environment. These optional arguments allow you to choose operations to perform, in the case below, selecting a filename for reading or writing.

- Call a function usage() if there are no command line arguments
- With in a suitable loop
  - Use getopt() to parse the command line options
  - With in a switch-case statement
    - Process arguments and print each option and argument
    - Set the default case and '?' to call usage() if incorrect arguments are provided

```
void usage ( char * str) {
        printf( "Usage: %s -f filename", str ); /* change options as appropriate */
}
```

See;
```
man 3 getopt
```

Example;
```
    #include <unistd.h>
    #include <stdlib.h>
    #include <stdio.h>
    int main(int argc, char *argv[]) {
      int flags, opt;
      int nsecs, tfnd;
      nsecs = 0;
      tfnd = 0;
      flags = 0;
      while ((opt = getopt(argc, argv, "nt:")) != -1) {
        switch (opt) {
        case 'n':
          flags = 1;
          break;
        case 't':
          nsecs = atoi(optarg);
          tfnd = 1;
          break;
        default: /* '?' */
          /* create new usage( ... ) and move the following there */
          fprintf(stderr, "Usage: %s [-t nsecs] [-n] name\n",  argv[0]);
          exit(EXIT_FAILURE);
        }
      }

      printf("flags=%d; tfnd=%d; nsecs=%d; optind=%d\n",
          flags, tfnd, nsecs, optind);
```

```
    if (optind >= argc) {
        fprintf(stderr, "Expected argument after options\n");
        exit(EXIT_FAILURE);
    }
    printf("name argument = %s\n", argv[optind]);
    /* Other code omitted */
    exit(EXIT_SUCCESS);
}
```

Task 2

Extend the program from Task1, use `open()`, `read()`, `printf()`, `close()` to print out the contents of a file.

- Use getopt() to parse the command line for an argument option; -i filename
- Print a line displaying the file name
- Print out the file to the console line by line

See;
`man 2 open`
`man 2 read`
`man 2 close`

Task 3

Extend the program from Task2, that calculates and prints the elapsed time of a loop to an accuracy of microseconds. Each loop should allocate a read buffer, open the a file, read its contents and close the file, free the read buffer.

- Use the structure `timeval` to store times
- Use `gettimeofday()` to get the time
- Iterate through the main loop 10,000 times (If this is “slow” run less iterations and document)
- Use `gettimeofday()` to get the elapsed time
- Calculate and print the *total* and average *per instance* time taken in microseconds

See;
`man 2 gettimeofday`

Example;

```
int main (int argc, char*argv[]) {
 char filename[...]; /* alternatively dynamically allocate space for the input filename */

 /* use getopt to acquire the filename, as per task1 */
 while ( ... ) {
  switch( ... ) {
    ...
  }
 }

 /* declare variables and get start time of operation */
 gettimeofday( ... )

 /* main loop */
 for ( ... ) {
  /* allocate memory */
  /*open file*/
   while ( ... ) {
     /* read content of file into buffer*/
     ...
     /* print buffer*/
     ...
    }
  /* close file*/
  /* free memory */
 }
 /* get finish time of operation */
 gettimeofday( ... )

 /* display elapsed time for operations */

 printf( ... )
 return ( ... )
}
```

Task 4

Extend the program from task3, replace the main loop, record the time it takes to use stat() to collect the metadata about a file and print out each of the fields, pay particular attention to the date-time, mode and file size fields.

- Use getopt() to provide a the command line argument of the file name
- Use gettimeofday() to record the time taken to do the following step
  - Use stat() and the structure stat to find the metadata about the provided file
- Print all of the elements of the structure with the appropriate format and description

```
struct stat {
        dev_t   st_dev;   /* device inode resides on */
        ino_t   st_ino;   /* inode's number */
        mode_t  st_mode;  /* inode protection mode */
        ...
}
```

For example, using the above comments in the structure in the man page print out all elements of the structure to describe the file, including correct time units;

```
        printf("device inode resides on: '%d'\n", file->st_dev);
        printf("inode's number: '%d'\n", file->st_ino);
        printf("inode's number: '%d'\n", file->st_ino);
        ...
```

See;
```
man 2 stat
```

Task 5

Write a unit test program that calculates and prints the elapsed time of a loop to an accuracy of microseconds, with ctap.h

- Create a function that Iterates through a loop 100,000,000 times
- Create a pair of unit test functions that record the time taken in seconds and microseconds
- Create a unit test comparison to evaluate the run time of the function
- Set the limit of run time to 60 seconds, adjust the iterations to complete with in that time
- Document the changes/additions/reductions made to loop iterations in comments in the code

See;
man 2 gettimeofday
https://github.com/jhunt/ctap/blob/master/README.md

This example is a starting point for the task;

```
#include < /* time related include */ >
#include < /* sting handling related include */ >
#include <ctap.h>

/* function prototypes, choose appropriate types */
type iterate( args... );
type start( args... );
type stop( args... );
type difference( args... );

TESTS {

 TODO("to implement") {  /* move from todo to above in main body when done */

   /* choose a more appropriate types */
   char loops = 1;
   char * timetaken;

   /* define appropriate structure */
   type start, stop, difference;

   note( "Time compute steps" );
   ok(   start(...), "get the start time" );
   ok(   iterate( loops ), "iterate through the loop" );
   ok(   stop(...), "get the stop time" );

   /* compute elapsed time and convert to string */
   note( "Compute time taken %s", timetaken );

   /* use the appropriate time unit for comparison */
   ok(   difference < 60, "time was less than 60 seconds" );
  }
}
```