CS3104 - Operating Systems
Assignment: P1 - A system utility - Report

Matriculation Number: 160001362

22/10/2018

Contents

1 Overview

1.1 System Calls

In modern computer systems, mechanisms to protect system resources are necessary to avoid disruption to the execution of operating system code as well as to ensure that systems remain secure.

A common approach used to achieve these goals is to implement a dual-mode of operation in which user and kernel level processes have different privilege levels. This ensures that certain resources/operations available to the operating system kernel are not made directly available to the user. Instead, in order for user processes to be able to carry out necessary restricted operations, they must invoke an operating system service known as a system call.

System calls are the way in which a user-process requests that the operating system kernel perform a task on its behalf. System call invocations cause a trap into the operating system resulting in the protection mode bit being flipped from user to kernel mode in the system architecture, transferring control to the operating system. The system call can then be executed by first looking for a particular entry in the operating system's interrupt vector table, and then running the interrupt service routine corresponding to the address of that entry. Control can then be returned back to the user, by once again flipping the mode bit, thereby switching from kernel to user mode.

In Linux, to be able to identify which entry in the interrupt vector table corresponds with the system call in question, the value of rax (on 64 bit systems) is used. If the system call in question requires parameters, these are then stored in subsequent registers whose values are used by the operating system when the system call is invoked. The system call identifier and parameters must therefore be written to registers prior to the system call invocation. In contrast to the method described previously, many modern systems use what are known as "fast system calls", a different method to invoke a system call that avoids the overhead of the interrupt mechanism (e.g. SYSCALL/SYSRETURN is used by the x86 instruction set).

Prior background reading was done using the text book [1] and the lecture slides.

1.2 Aim of Practical

This practical specified the development of a custom implementation of a shell command similar to that of the Unix command "ls", whose output resembles "ls" run with the "-n" flag. The command had to be implemented solely using system calls, avoiding any use of C standard library functions[1]. A solution to the

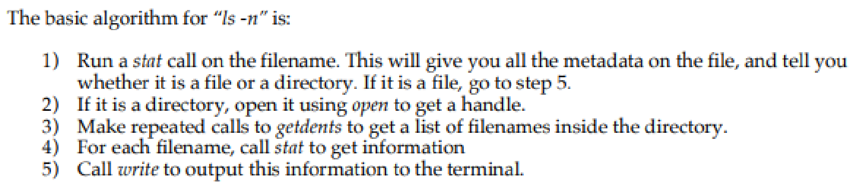
[1]Apart from the C library function localtime(), allowed by the specification.

practical should be implemented using a series of custom system call wrappers, each individually implemented using inline assembly code, invoking system calls using the system call codes found in /usr/include/asm/unistd_64.h. Ultimately this exercise aims to provide experience using Linux system calls which as a result will provide valuable experience of C system level programming, as well as reinforcing an understanding of file systems.

2 Design and Implementation

2.1 Summary

The overarching design of this solution was modelled after the basic algorithm for "ls -n" listed in the practical specification:



The basic algorithm for *"ls -n"* is:

1) Run a *stat* call on the filename. This will give you all the metadata on the file, and tell you whether it is a file or a directory. If it is a file, go to step 5.
2) If it is a directory, open it using *open* to get a handle.
3) Make repeated calls to *getdents* to get a list of filenames inside the directory.
4) For each filename, call *stat* to get information
5) Call *write* to output this information to the terminal.

The following are different execution scenarios influenced by the design of the solution:

- When the "myls" executable is run with an single argument representing a file/directory name, the functionality imitating "ls -n" is executed, printing meta data information about a file or list of files in a directory.[2]
- When it is run with no arguments, the unit tests implemented to test the solution are run. This decision was made for the sake of simplicity in order to have a single executable for both running the program as well as for running the unit tests. The same functionality as running "ls -n" with no arguments can be achieved by running "./myls" with '.' as an argument.[3]
- When "myls" is run with an invalid file/directory name, an error message is printed to the user which resembles the error message printed by "ls -n" when a non-existent file is used as an argument.[4]

2.2 Design Decisions

The main method of the solution executes the correct code based on which of the above scenarios is taking place. If a valid file/directory name has been passed, a stat structure is populated with the meta data of that file/directory using the myStat() system call implementation.

[2]Figure 2 in section 8.1 of the Appendix
[3]Figure 3 in section 8.1 of the Appendix
[4]Figure 4 in section 8.1 of the Appendix

2.2.1 Printing the Meta Data of a Single File

For printing the meta data of a single file, the function printMetaData() is called which takes the stat structure as a parameter and then proceeds to extract and print the meta data elements required to imitate "ls -n". The file elements printed are as follows:

- File type code ('-' for file, 'd' for directory).
- File permissions (e.g. 'rwxrwxrwx' would be a file with read, write and execute permissions for user, group and others).
- The number of hard links to the file.
- The numeric user ID of the file.
- The numeric group ID of the file.
- The time the file was last modified.

After extracting each these values from the stat meta data structure, the myWrite() system call implementation is repeatedly invoked which prints the values of each element to standard out. The file name is then output.

2.2.2 Extracting the File Type and Permissions

The function getDirChar() uses the POSIX macro S_ISDIR to determine whether a file is a directory or not, and returns '-' or 'd' accordingly.

The function responsible for returning permissions of a file, getFilePermissions(), makes use of the permission macros from sys/stat.h. When bit-wise masked with the mode element of the file meta data, these values return whether a particular permission is set or not. For example (S_IRUSR & meta_data.mode) determines whether the user has read permissions on the file. This operation, repeated for each mode-bit, is used to populate an array of permissions for any given file. The knowledge to use the bit-masks was gained from reading an example on stackoverflow[2].

2.2.3 Formatting and Printing the Modification Time of a File

A separate function printModifiedTime() was implemented to handle the output of file modification time, as this data is made up of many values and requires more formatting than the other data elements. The function makes use of the custom system call implementation myTime() to get the number of seconds since the Epoch and the standard library function localtime() to convert the number of seconds into a date. The current date/time is computed as well as the date/time the file was modified. These dates are stored in time structures. The following time elements are extracted and printed from the file time structure:

- Month of modification (e.g. "Jan", "Feb", ... , "Dec")

- Day of modification (e.g. 1 - 31)
- One of the following:
  - Time of modification in twenty-four hour notation (e.g. 22:31), if file was modified in the current year.
  - Year of modification (e.g. 2017) if file was modified prior to the current year.

The month returned by localtime() is an integer from 0 - 11. To enable convenient integer to string conversion, a function monthToStr() has been implemented which indexes into a constant array of month strings (e.g. "Jan", "Feb") and returns the corresponding string given a month integer.

2.2.4 Printing the Meta Data of Files in a Directory

In order to print the meta data of all files in a directory when a directory name is given as an argument to the program, the printDirEntries() function has been implemented.

The function makes uses of the custom system call wrapper myGetDents() which reads a number of bytes equal to the size of all of the directory entries in a given directory into a buffer. The function then iterates through the buffer populated by myGetDents through each directory entry by calculating the offset of the next directory entry by using the size of the current entry.

During each iteration, the file name of the directory entry is combined with the directory name (e.g. directory/file) and then a myStat() call is invoked to get the meta data of the file. This meta data is then used to call printMetaData() to print the data for that particular file.

The system call wrappers myOpen() and myClose() are also used to open/close the directory.

2.2.5 String Functions Implemented

Three string library functions were re-implemented from scratch for use in this solution, myStrCpy() for copying strings, myStrLen() for getting the length of strings, and myitoa() for converting unsigned integers to strings. A new string function was also implemented for use in the unit testing of the solution, strEqual(), which checks two strings for equality (essentially a more simple version of strcmp()).

2.2.6 Data Structures

Due to the restriction of not being able to use malloc and similar memory allocation functions, fixed size character arrays are used as buffers for storing strings where required. Also, as discussed briefly, the stat, linux_dirent, and tm structs are used to store file meta data, directory entries, and file date/time data respectively.

2.3 Implementation of System Calls

2.3.1 Design Methodology

The process for implementing system call wrappers using inline assembly involved the following process:

- **man pages** - To assess the behaviour of each system call, the man page entry for the system call was studied. The parameters, return type and functionality were noted in order to determine how best to incorporate the system call into the solution.
- **POSIX basic implementation** - Initially to produce a working solution, the implementation of a system call wrapper (e.g. myStat) would simply contain the corresponding wrapper function provided by POSIX (e.g. stat in this case).
- **Unit tests** - Once the solution worked with the provided POSIX system call functions, unit tests were written to ensure that the inline assembly system calls behaved as expected when implemented.
- **Inline assembly implementation** - The system call wrapper implementations were then converted to inline assembly by adapting the example given in starter.c. The long form of inline assembly was used as it was easier for me to understand. /usr/include/asm/unistd_64.h was used to find the identifier for each system call. After each wrapper was implemented, the unit tests were run to identify and then rectify any potential logical errors.

2.3.2 Table of System Calls

System Call List			
System Call Wrapper Function	Name of System Call	System Call Code	Purpose in Program
myStat	stat	4	Retrieves file meta data to be printed
myWrite	write	1	Writes file data to stdout
myGetDents	getdents	78	Gets directory entry names to call myStat on
myOpen	open	2	Opens directory to get file handle for myGetDents
myClose	close	3	Closes above directory
myTime	time	201	Gets seconds since Epoch for current date/time
myCreat	creat	85	Creates test files for unit tests
myUnlink	unlink	87	Deletes test files created for unit tests
mymkdir	mkdir	83	Creates test directories for unit tests
myrmdir	rmdir	84	Removes test directories created for unit tests

Figure 1: List of system calls used for the basic specification

2.4 File Structure Discussion

The code for the solution of the basic specification is contained within the the file ”myls.c” as required by the specification. The file itself is very dense due to the restriction of having to use a single source file. Ideally the macro declarations and function header declarations would be contained in a separate header file. Other code that would be better placed in a separate source file includes the implemented system call wrappers that use inline assembly. Were these functions placed in another file, they would then resemble a C style library file which could be reused by multiple programs via an ”include” statement. The custom unit test suite functions and associated tests could also be placed in their own respective files.

2.5 Compilation and Execution

The README.md file contained within the submission directory contains useful compilation, execution, and troubleshooting information. In particular it distinguishes between executing the ”ls -n” utility and executing the unit tests.

3 Testing

Screenshots showing evidence of testing are available in section 8.2 of the Appendix

3.1 Approach to Testing and Debugging

To ensure the robustness of this solution, a combination of unit tests and functional tests were carried out. As recommended by the practical specification, the approach to testing and debugging this solution was to implement functionality piece by piece and to test each piece when implemented. This approach lent itself well to test-driven development, where failing unit tests are implemented, and then the functionality required to pass those tests is implemented. For this practical, unit tests were especially suited to testing system call wrappers, as well as custom string functions such as myStrCpy and myitoa whose expected behaviour is well documented.

Not all functions were suited to unit testing. The functions writeErrorMessage(), printModifiedTime(), printMetaData(), and printDirEntries() all rely on writing data to standard out and so were harder to test using unit tests. As a result, functional tests have been carried out instead. This involved running the program as an end user would, observing the output, and comparing it with the expected output produced by "ls -n".

3.2 Unit Testing

In order to be able to run unit tests without the use of any external libraries, the functions initTests() and runTests() have been implemented.

The function initTests() populates an array with pointers to the functions of each unit test.

The runTests() function then takes the list and number of functions and then iterates through each one, displaying whether it has passed or failed using the myWrite system call wrapper.[5]

The unit test functions return true or false if the test has passed or failed. These tests are located at the end of the "myls.c" source file.

3.3 Functional Testing

The output of functional testing is available in the appendix under section 8.2.2.

For each test carried out with "myls", the same test has been carried out with "ls -n" to compare the output.

[5]Figures 5 and 6 in section 8.2 of the Appendix

The following are the functional tests that have been carried out:[6]

- Running the program with "myls.c" as an argument.
- Running the program with "." as an argument.
- Running the program with ".." as an argument.
- Running the program with "./myls.c" as an argument (from relative path).
- Running the program with " /Documents/CS3104/practicals/CS3104-P1-Sysutil/myls.c" as an argument.
- Running the program with a file modified earlier than 2018.
- Running the program with "/" as an argument.
- Running the program on an adjacent directory to the executable (in this case the ".git" directory of the program repository) as an argument.
- Running the program with "*" as an argument.
- Running the program with a symbolic link as an argument.

3.3.1 Results of Functional Tests

Based on the output of the aforementioned functional tests, the "myls" executable successfully manages to replicate the main features of the command "ls -n". However some notable formatting present in "ls -n" that is missing from "myls" includes the lack of the '*' displayed at the end of files that are executable, the lack of the total number of 512 bytes blocks used by the files in a directory (e.g. "total 52" in figure 8), as well as the lack of alphabetisation.

Functional differences between "myls" and "ls -n" include the ability of "ls -n" to recursively list all of the files in a directory using '*' as a parameter ("myls" outputs nothing when given '*'), and the ability of "ls -n" to process files that are not a regular file or directory such as symbolic links or pipes ("myls" returns an error message for those files). Finally "myls" outputs hidden files such as '.' and '..' when run where as "ls -n" does not. This feature of "myls" was left in, in order to provide a full view of all of the files in a directory.

[6]Figures 7 - 16 in section 8.2.2 of the Appendix

4 Evaluation

Overall, this solution meets the requirements laid out in the practical specification extensively. The specification states that an "ls -n" style system utility should be developed using the C programming language and system calls implemented using inline assembly code. The specification also states that the solution should be well-tested, stable, correct, and well written.

This solution fulfills this criteria by making extensive use of inline assembly system calls via the C programming language which are well tested as demonstrated by the output of the automated unit tests and manual functional tests. The suggested algorithm outlined in the specification is followed closely and the output of the solution closely resembles the provided example output. The solution is well commented, avoids magic numbers and literals where pragmatic, and attempts to be modular. The implemented solution also uses no standard library functions (aside from localtime) as required.

5 Conclusion

5.1 Achievements

In conclusion, this solution has achieved the development of a system utility similar to the command-line tool "ls -n" that is correct, robust, and well tested, and that makes extensive use of inline assembly system calls.

5.2 Difficulties Overcome

Initially, difficulties were encountered when attempting to use the getdents system call, as the man pages were slightly vague as to how it worked. This difficulty was overcome by studying the example code in the man pages, and analysing the output it produced, stripping away functionality until it produced the correct output. Finding the motivation to implement unit tests was also a struggle, as it is tempting to simply dive into the implementation of a practical without first writing tests.

5.3 Future Goals

Ideally given more time, the minor formatting differences discussed in section 3.3.1 would be implemented, as well as the ability to process files such as symbolic links and pipes.

6 Extensions

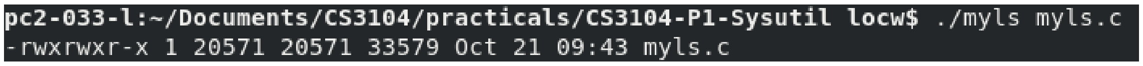
7 References

[1] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating Systems Concepts*. Wiley, Hoboken, New Jersey, 2013.

[2] Chalasani Arjith - Getting file permissions from a stat structure using bit masks.
https://stackoverflow.com/questions/8812959/how-to-read-linux-file-permission-programmatically-in-c-

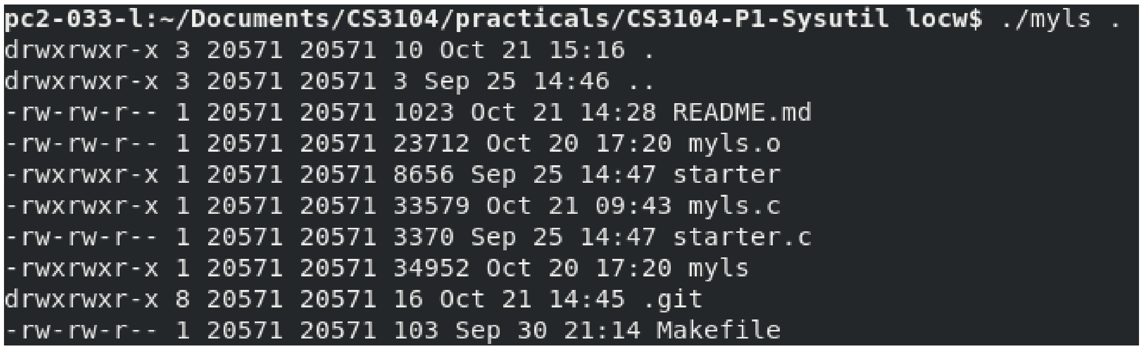
8 Appendix

8.1 Program Running



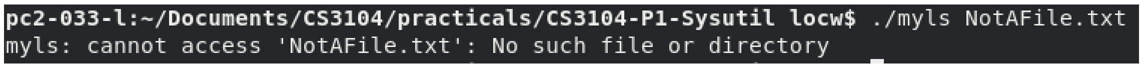
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls myls.c
-rwxrwxr-x 1 20571 20571 33579 Oct 21 09:43 myls.c
```

Figure 2: Command running with a valid file name



```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls .
drwxrwxr-x 3 20571 20571 10 Oct 21 15:16 .
drwxrwxr-x 3 20571 20571 3 Sep 25 14:46 ..
-rw-rw-r-- 1 20571 20571 1023 Oct 21 14:28 README.md
-rw-rw-r-- 1 20571 20571 23712 Oct 20 17:20 myls.o
-rwxrwxr-x 1 20571 20571 8656 Sep 25 14:47 starter
-rwxrwxr-x 1 20571 20571 33579 Oct 21 09:43 myls.c
-rw-rw-r-- 1 20571 20571 3370 Sep 25 14:47 starter.c
-rwxrwxr-x 1 20571 20571 34952 Oct 20 17:20 myls
drwxrwxr-x 8 20571 20571 16 Oct 21 14:45 .git
-rw-rw-r-- 1 20571 20571 103 Sep 30 21:14 Makefile
```

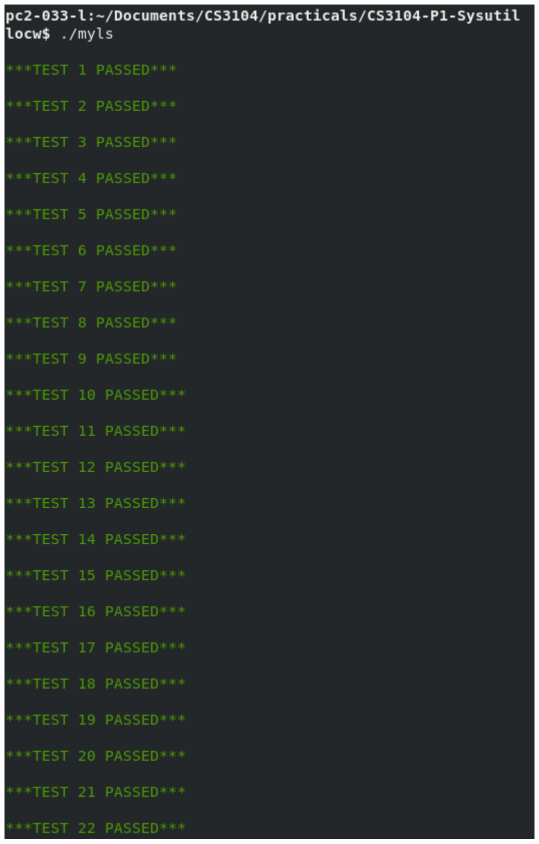
Figure 3: Command running with a valid directory name



```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls NotAFile.txt
myls: cannot access 'NotAFile.txt': No such file or directory
```

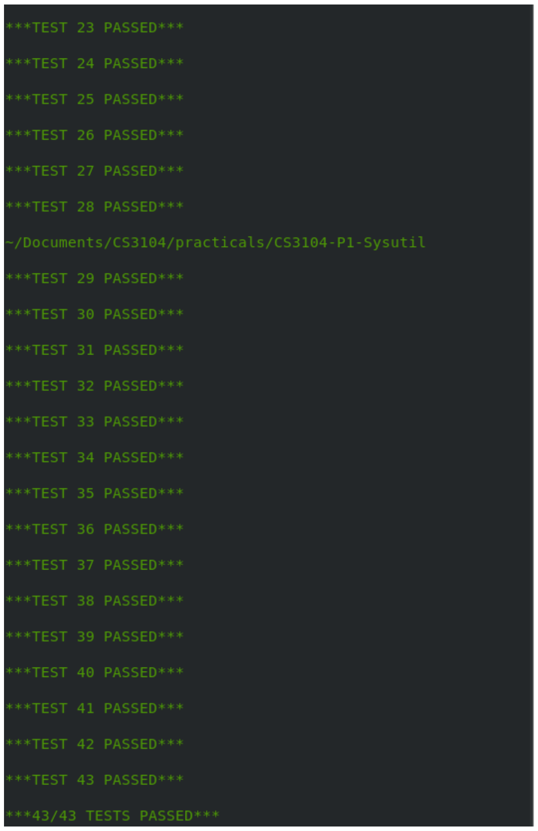
Figure 4: command running with an invalid file name

8.2 Evidence of Testing



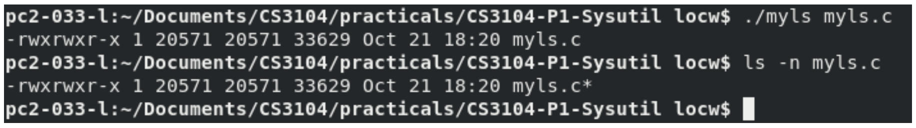
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil
locw$ ./myls

***TEST 1 PASSED***

***TEST 2 PASSED***

***TEST 3 PASSED***

***TEST 4 PASSED***

***TEST 5 PASSED***

***TEST 6 PASSED***

***TEST 7 PASSED***

***TEST 8 PASSED***

***TEST 9 PASSED***

***TEST 10 PASSED***

***TEST 11 PASSED***

***TEST 12 PASSED***

***TEST 13 PASSED***

***TEST 14 PASSED***

***TEST 15 PASSED***

***TEST 16 PASSED***

***TEST 17 PASSED***

***TEST 18 PASSED***

***TEST 19 PASSED***

***TEST 20 PASSED***

***TEST 21 PASSED***

***TEST 22 PASSED***
```

Figure 5: Output of Unit Tests



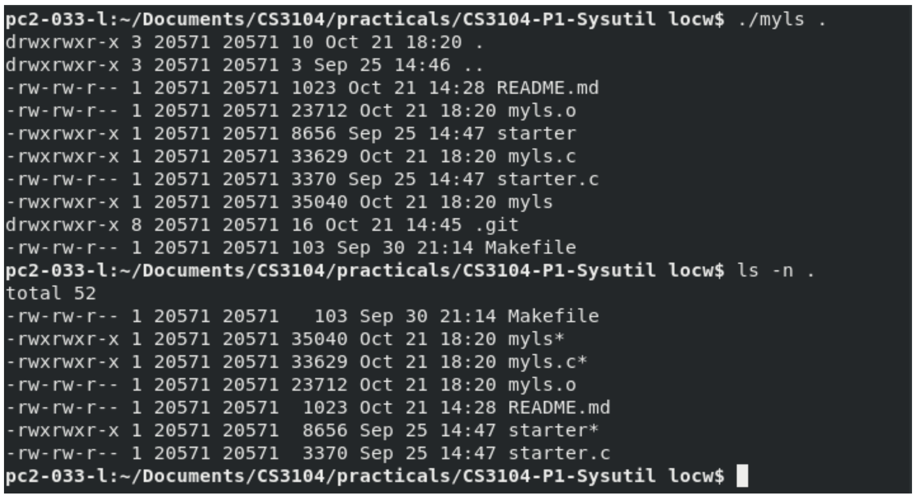
```
***TEST 23 PASSED***

***TEST 24 PASSED***

***TEST 25 PASSED***

***TEST 26 PASSED***

***TEST 27 PASSED***

***TEST 28 PASSED***

~/Documents/CS3104/practicals/CS3104-P1-Sysutil

***TEST 29 PASSED***

***TEST 30 PASSED***

***TEST 31 PASSED***

***TEST 32 PASSED***

***TEST 33 PASSED***

***TEST 34 PASSED***

***TEST 35 PASSED***

***TEST 36 PASSED***

***TEST 37 PASSED***

***TEST 38 PASSED***

***TEST 39 PASSED***

***TEST 40 PASSED***

***TEST 41 PASSED***

***TEST 42 PASSED***

***TEST 43 PASSED***

***43/43 TESTS PASSED***
```

Figure 6: Output of Unit Tests (continued)



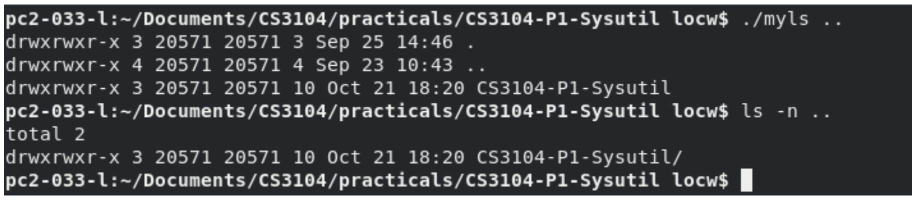
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls myls.c
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 myls.c
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n myls.c
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 myls.c*
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 7: Running the program with ”myls.c” as an argument



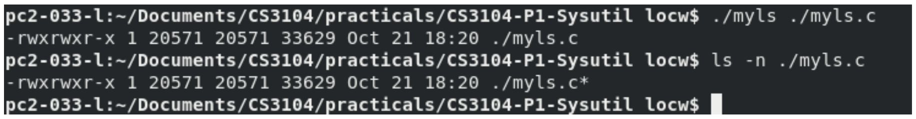
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls .
drwxrwxr-x 3 20571 20571 10 Oct 21 18:20 .
drwxrwxr-x 3 20571 20571 3 Sep 25 14:46 ..
-rw-rw-r-- 1 20571 20571 1023 Oct 21 14:28 README.md
-rw-rw-r-- 1 20571 20571 23712 Oct 21 18:20 myls.o
-rwxrwxr-x 1 20571 20571 8656 Sep 25 14:47 starter
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 myls.c
-rw-rw-r-- 1 20571 20571 3370 Sep 25 14:47 starter.c
-rwxrwxr-x 1 20571 20571 35040 Oct 21 18:20 myls
drwxrwxr-x 8 20571 20571 16 Oct 21 14:45 .git
-rw-rw-r-- 1 20571 20571 103 Sep 30 21:14 Makefile
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n .
total 52
-rw-rw-r-- 1 20571 20571   103 Sep 30 21:14 Makefile
-rwxrwxr-x 1 20571 20571 35040 Oct 21 18:20 myls*
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 myls.c*
-rw-rw-r-- 1 20571 20571 23712 Oct 21 18:20 myls.o
-rw-rw-r-- 1 20571 20571  1023 Oct 21 14:28 README.md
-rwxrwxr-x 1 20571 20571  8656 Sep 25 14:47 starter*
-rw-rw-r-- 1 20571 20571  3370 Sep 25 14:47 starter.c
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 8: Running the program with "." as an argument



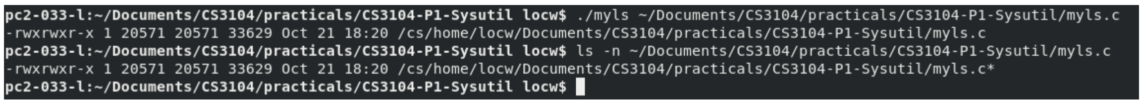
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls ..
drwxrwxr-x 3 20571 20571 3 Sep 25 14:46 .
drwxrwxr-x 4 20571 20571 4 Sep 23 10:43 ..
drwxrwxr-x 3 20571 20571 10 Oct 21 18:20 CS3104-P1-Sysutil
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n ..
total 2
drwxrwxr-x 3 20571 20571 10 Oct 21 18:20 CS3104-P1-Sysutil/
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 9: Running the program with ".." as an argument



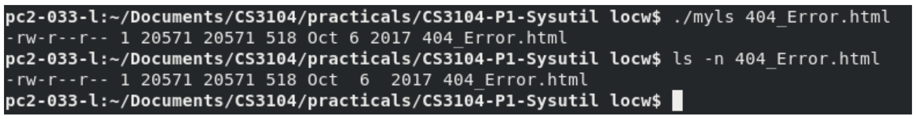
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls ./myls.c
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 ./myls.c
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n ./myls.c
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 ./myls.c*
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 10: Running the program with "./myls.c" as an argument (from relative path)



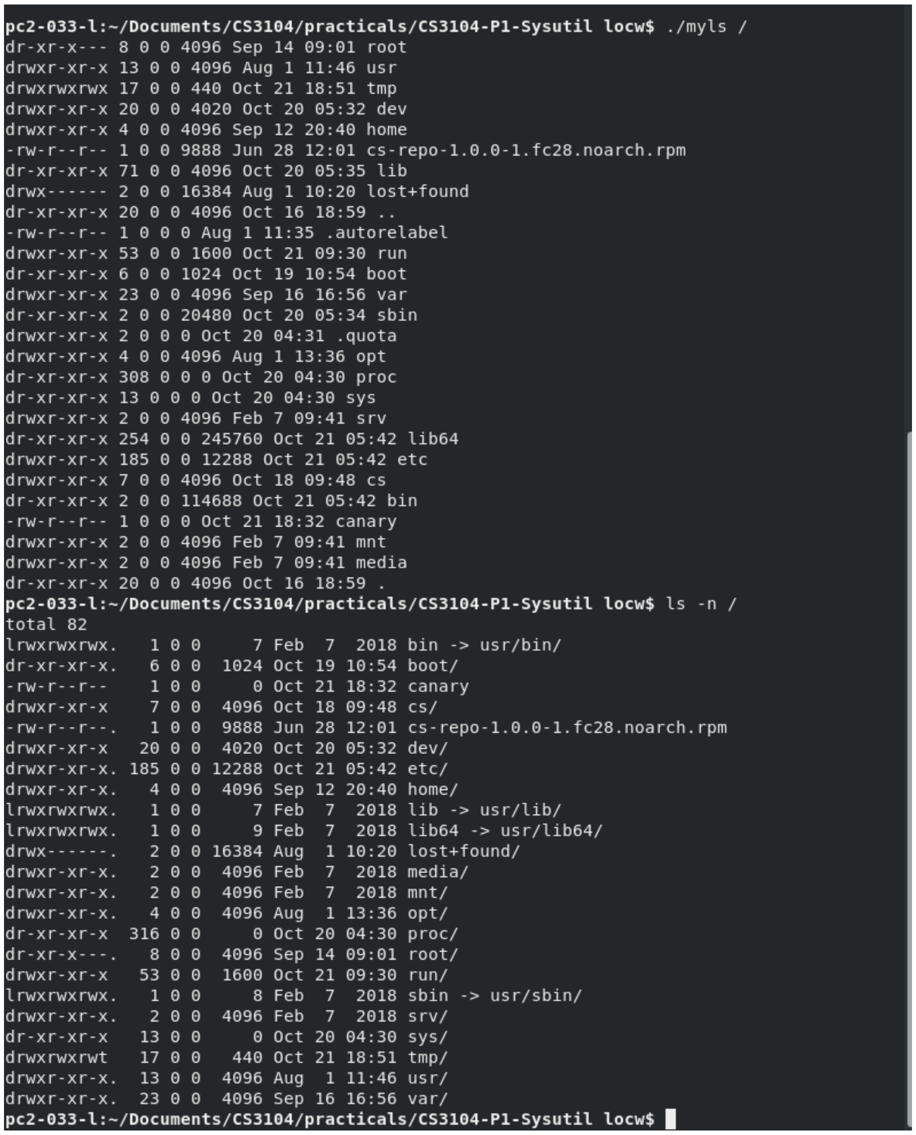
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls ~/Documents/CS3104/practicals/CS3104-P1-Sysutil/myls.c
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 /cs/home/locw/Documents/CS3104/practicals/CS3104-P1-Sysutil/myls.c
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n ~/Documents/CS3104/practicals/CS3104-P1-Sysutil/myls.c
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 /cs/home/locw/Documents/CS3104/practicals/CS3104-P1-Sysutil/myls.c*
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 11: Running the program with "~/Documents/CS3104/practicals/CS3104-P1-Sysutil/myls.c" as an argument



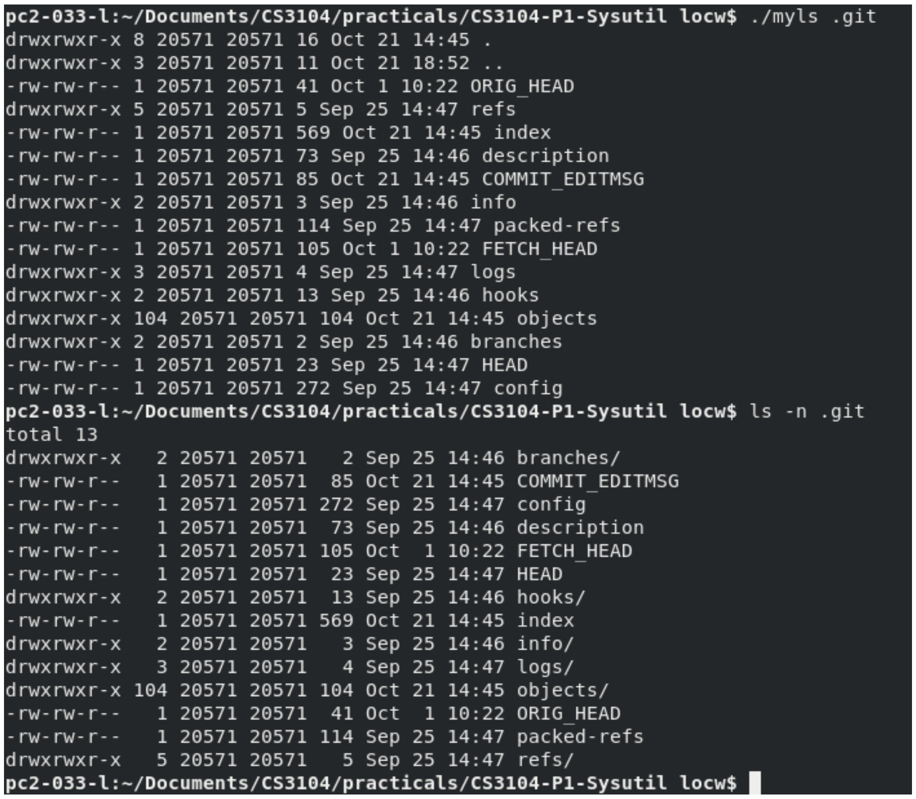
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls 404_Error.html
-rw-r--r-- 1 20571 20571 518 Oct 6 2017 404_Error.html
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n 404_Error.html
-rw-r--r-- 1 20571 20571 518 Oct  6  2017 404_Error.html
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 12: Running the program with a file modified earlier than 2018



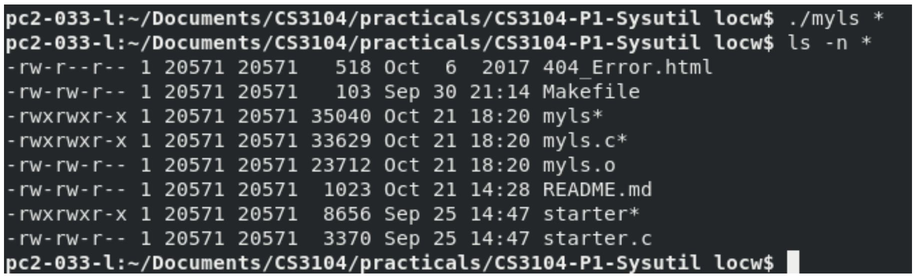
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls /
dr-xr-x--- 8 0 0 4096 Sep 14 09:01 root
drwxr-xr-x 13 0 0 4096 Aug 1 11:46 usr
drwxrwxrwx 17 0 0 440 Oct 21 18:51 tmp
drwxr-xr-x 20 0 0 4020 Oct 20 05:32 dev
drwxr-xr-x 4 0 0 4096 Sep 12 20:40 home
-rw-r--r-- 1 0 0 9888 Jun 28 12:01 cs-repo-1.0.0-1.fc28.noarch.rpm
dr-xr-xr-x 71 0 0 4096 Oct 20 05:35 lib
drwx------ 2 0 0 16384 Aug 1 10:20 lost+found
dr-xr-xr-x 20 0 0 4096 Oct 16 18:59 ..
-rw-r--r-- 1 0 0 0 Aug 1 11:35 .autorelabel
drwxr-xr-x 53 0 0 1600 Oct 21 09:30 run
dr-xr-xr-x 6 0 0 1024 Oct 19 10:54 boot
drwxr-xr-x 23 0 0 4096 Sep 16 16:56 var
dr-xr-xr-x 2 0 0 20480 Oct 20 05:34 sbin
drwxr-xr-x 2 0 0 0 Oct 20 04:31 .quota
drwxr-xr-x 4 0 0 4096 Aug 1 13:36 opt
dr-xr-xr-x 308 0 0 0 Oct 20 04:30 proc
dr-xr-xr-x 13 0 0 0 Oct 20 04:30 sys
drwxr-xr-x 2 0 0 4096 Feb 7 09:41 srv
dr-xr-xr-x 254 0 0 245760 Oct 21 05:42 lib64
drwxr-xr-x 185 0 0 12288 Oct 21 05:42 etc
drwxr-xr-x 7 0 0 4096 Oct 18 09:48 cs
dr-xr-xr-x 2 0 0 114688 Oct 21 05:42 bin
-rw-r--r-- 1 0 0 0 Oct 21 18:32 canary
drwxr-xr-x 2 0 0 4096 Feb 7 09:41 mnt
drwxr-xr-x 2 0 0 4096 Feb 7 09:41 media
dr-xr-xr-x 20 0 0 4096 Oct 16 18:59 .
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n /
total 82
lrwxrwxrwx.   1 0 0     7 Feb  7  2018 bin -> usr/bin/
dr-xr-xr-x.   6 0 0  1024 Oct 19 10:54 boot/
-rw-r--r--    1 0 0     0 Oct 21 18:32 canary
drwxr-xr-x    7 0 0  4096 Oct 18 09:48 cs/
-rw-r--r--.   1 0 0  9888 Jun 28 12:01 cs-repo-1.0.0-1.fc28.noarch.rpm
drwxr-xr-x   20 0 0  4020 Oct 20 05:32 dev/
drwxr-xr-x. 185 0 0 12288 Oct 21 05:42 etc/
drwxr-xr-x.   4 0 0  4096 Sep 12 20:40 home/
lrwxrwxrwx.   1 0 0     7 Feb  7  2018 lib -> usr/lib/
lrwxrwxrwx.   1 0 0     9 Feb  7  2018 lib64 -> usr/lib64/
drwx------.   2 0 0 16384 Aug  1 10:20 lost+found/
drwxr-xr-x.   2 0 0  4096 Feb  7  2018 media/
drwxr-xr-x.   2 0 0  4096 Feb  7  2018 mnt/
drwxr-xr-x.   4 0 0  4096 Aug  1 13:36 opt/
dr-xr-xr-x  316 0 0     0 Oct 20 04:30 proc/
dr-xr-x---.   8 0 0  4096 Sep 14 09:01 root/
drwxr-xr-x   53 0 0  1600 Oct 21 09:30 run/
lrwxrwxrwx.   1 0 0     8 Feb  7  2018 sbin -> usr/sbin/
drwxr-xr-x.   2 0 0  4096 Feb  7  2018 srv/
dr-xr-xr-x   13 0 0     0 Oct 20 04:30 sys/
drwxrwxrwt   17 0 0   440 Oct 21 18:51 tmp/
drwxr-xr-x.  13 0 0  4096 Aug  1 11:46 usr/
drwxr-xr-x.  23 0 0  4096 Sep 16 16:56 var/
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 13: Running the program with "/" as an argument



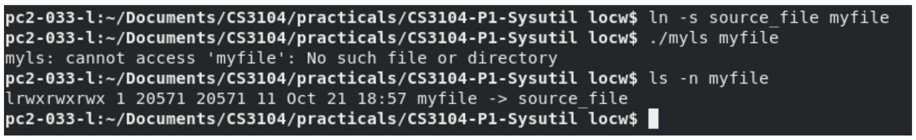
```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls .git
drwxrwxr-x 8 20571 20571 16 Oct 21 14:45 .
drwxrwxr-x 3 20571 20571 11 Oct 21 18:52 ..
-rw-rw-r-- 1 20571 20571 41 Oct 1 10:22 ORIG_HEAD
drwxrwxr-x 5 20571 20571 5 Sep 25 14:47 refs
-rw-rw-r-- 1 20571 20571 569 Oct 21 14:45 index
-rw-rw-r-- 1 20571 20571 73 Sep 25 14:46 description
-rw-rw-r-- 1 20571 20571 85 Oct 21 14:45 COMMIT_EDITMSG
drwxrwxr-x 2 20571 20571 3 Sep 25 14:46 info
-rw-rw-r-- 1 20571 20571 114 Sep 25 14:47 packed-refs
-rw-rw-r-- 1 20571 20571 105 Oct 1 10:22 FETCH_HEAD
drwxrwxr-x 3 20571 20571 4 Sep 25 14:47 logs
drwxrwxr-x 2 20571 20571 13 Sep 25 14:46 hooks
drwxrwxr-x 104 20571 20571 104 Oct 21 14:45 objects
drwxrwxr-x 2 20571 20571 2 Sep 25 14:46 branches
-rw-rw-r-- 1 20571 20571 23 Sep 25 14:47 HEAD
-rw-rw-r-- 1 20571 20571 272 Sep 25 14:47 config
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n .git
total 13
drwxrwxr-x   2 20571 20571   2 Sep 25 14:46 branches/
-rw-rw-r--   1 20571 20571  85 Oct 21 14:45 COMMIT_EDITMSG
-rw-rw-r--   1 20571 20571 272 Sep 25 14:47 config
-rw-rw-r--   1 20571 20571  73 Sep 25 14:46 description
-rw-rw-r--   1 20571 20571 105 Oct  1 10:22 FETCH_HEAD
-rw-rw-r--   1 20571 20571  23 Sep 25 14:47 HEAD
drwxrwxr-x   2 20571 20571  13 Sep 25 14:46 hooks/
-rw-rw-r--   1 20571 20571 569 Oct 21 14:45 index
drwxrwxr-x   2 20571 20571   3 Sep 25 14:46 info/
drwxrwxr-x   3 20571 20571   4 Sep 25 14:47 logs/
drwxrwxr-x 104 20571 20571 104 Oct 21 14:45 objects/
-rw-rw-r--   1 20571 20571  41 Oct  1 10:22 ORIG_HEAD
-rw-rw-r--   1 20571 20571 114 Sep 25 14:47 packed-refs
drwxrwxr-x   5 20571 20571   5 Sep 25 14:47 refs/
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$
```

Figure 14: Running the program with the ".git" directory as an argument



```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls *
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n *
-rw-r--r-- 1 20571 20571   518 Oct  6  2017 404_Error.html
-rw-rw-r-- 1 20571 20571   103 Sep 30 21:14 Makefile
-rwxrwxr-x 1 20571 20571 35040 Oct 21 18:20 myls*
-rwxrwxr-x 1 20571 20571 33629 Oct 21 18:20 myls.c*
-rw-rw-r-- 1 20571 20571 23712 Oct 21 18:20 myls.o
-rw-rw-r-- 1 20571 20571  1023 Oct 21 14:28 README.md
-rwxrwxr-x 1 20571 20571  8656 Sep 25 14:47 starter*
-rw-rw-r-- 1 20571 20571  3370 Sep 25 14:47 starter.c
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$
```

Figure 15: Running the program with "*" as an argument



```
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ln -s source_file myfile
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ./myls myfile
myls: cannot access 'myfile': No such file or directory
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ ls -n myfile
lrwxrwxrwx 1 20571 20571 11 Oct 21 18:57 myfile -> source_file
pc2-033-l:~/Documents/CS3104/practicals/CS3104-P1-Sysutil locw$ 
```

Figure 16: Running the program with a symbolic link as an argument