ICSI333 System Fundamentals

Note: Students are expected to start the activities as soon as the description is available and seek feedback as needed. Although some activities are not graded for credit, they are contiguous study of the lecture or used as stepping-stones for the projects. Skipping any activities would impact the learning significantly.

Objectives:

- Practice and understand kernel level file manipulation
- Practice how to create and manage a file
- Practice how to read a directory
-

Reading:

- Lecture notes 12 and 13
- Reading materials

Submission (5 points):

- All required C programs must be submitted on Duifene on time.

Instructions:

Task #1. Study kernel-level file handling.
Go to the Linux Programmer's Manual and study the following system call functions using the corresponding examples listed below.

System calls	Description
`int creat(const char *pathname, mode_t mode);`	Create a new file.
`int open(const char *pathname, int flags, mode_t mode);`	Open a file for reading, writing or both.
`int close(int fd);`	Close a specific file descriptor.
`ssize_t read(int fd, void *buf, size_t count);`	Read from a specific file descriptor.
`ssize_t write(int fd, const void *buf, size_t count);`	Write to a specific file descriptor.

Example 1: An example for `open` system call.

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
extern int errno;

int main(){
    // if file does not have in directory
    // then file foo.txt is created.
    int fd = open("foo.txt", O_RDONLY | O_CREAT);
    printf("fd = %d/n", fd);

    if (fd == -1){
      // print which type of error have in a code
      printf("Error Number % d\n", errno);

      // print program detail "Success or failure"
     perror("Program");
    }
```

```
    return 0;
}
```

Example 2: An example for `close` system call.

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
    // assume that foo.txt is already created.
    int fd1 = open("foo.txt", O_RDONLY, 0);
    close(fd1);

    // assume that baz.txt is already created
    int fd2 = open("baz.txt", O_RDONLY, 0);
    printf("fd2 = % d\n", fd2);

    exit(0);
}
```

Example 3: An example for `read` system call.

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(){
    int fd, sz;
    char *c = (char *) calloc(100, sizeof(char));

    fd = open("foo.txt", O_RDONLY);

    if (fd < 0) {
      perror("r1");
      exit(1);
    }

   sz = read(fd, c, 10);
   printf("called read(%d, c, 10)returned and  %d bytes were read.\n",
          fd, sz);
   c[sz] = '\0';

   printf("Those bytes are as follows: %s\n", c);

   return 0;
}
```

Example 4: An example for `write` system call.

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
```

```
#include <unistd.h>

int main(){
    int sz;
    int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

    if (fd < 0){
        perror("r1");
        exit(1);
    }

    sz = write(fd, "hello world\n", strlen("hello world\n"));

    printf("called write(%d, \"hello world\\n\", %zu)returned and  %d\n ",
           fd, strlen("hello world\n"), sz);

    close(fd);
}
```

Task # 2. Calculate the sum of a list randomly generated integers.
Write a program that randomly generates 100 numbers between 0 and 99(0 and 99 are inclusive), calculates the sum of the numbers, prints the sum, and writes the numbers to a new file using `open`, `close` and `write` system call functions. Here is the more information:

- *Input*: there are no user-entered numbers. Use `rand` or `srand` to generate the numbers.
    - Go to `man 3 rand` to study `rand` and `srand`.
    - Store the numbers in an array.
    - Remember to initialize any sum variable to `0`.
    - `rand` may return a bigger number than 99. Use the modulus operator to reduce the range.
    - `rand` or `srand` can be called with a seed value. You can use `time(0)` as the seed value. For example, `srand(time(0));`
        - `time(0)` returns the current time in seconds.
- *Output*: print the sum of the numbers and creates a file called `numbers.xxxx` where `xxxx` is the sum of the numbers in the file.
- *main:* this is the only function that is required.

Task #3. Process files whose names match the pattern `numbers.xxxx`.
Write a program that searches for the files whose names match the pattern "`numbers.xxxx`" in the current working directory. For each file that matches, open the file, read the file, calculates the sum the integers, and print the file name and the sum. You can assume that a file will contain 100 integers.
Hints:

- Create a function to deal with open/read/sum/print/close.
- `opendir()` requires a starting directory. Use the current working directory as the starting directory.
- Loop over the directory entries(as if waking in a linked list) to check every file to see if it starts with "`numbers.`"

Linux Basic Commands

`ls [option(s)] [file(s)]`	If you run ls without any additional parameters, the program will list the contents of the current directory in short form. `-l` detailed list `-a` displays hidden files
`cp [option(s)] sourcefile targetfile`	Copies sourcefile to targetfile. `-I` waits for confirmation, if necessary, before an existing targetfile is overwritten `-r` copies recursively (includes subdirectories)
`mv [option(s)] sourcefile targetfile`	Copies sourcefile to targetfile then deletes the original sourcefile. `-b` creates a backup copy of the sourcefile before moving. `-I` waits for confirmation, if necessary, before an existing targetfile is overwritten.
`rm [option(s)] file(s)`	Removes the specified files from the file system. Directories are not removed by rm unless the option `-r` is used. `-r` deletes any existing subdirectories `-I` waits for confirmation before deleting each file
`cd [options(s)] [directory]`	Changes the current directory. cd without any parameters changes to the user's home directory.
`mkdir [option(s)] directoryname`	Creates a new directory.
`rmdir [option(s)] directoryname`	Deletes the specified directory, provided it is already empty.
`cat [option(s)] file(s)`	The cat command displays the contents of a file, printing the entire contents to the screen without interruption. `-n` numbers the output on the left margin
`cal`	Displays the calendar of the current month.
`date`	Displays current time and date.
`whoami`	Reveals the user who is currently logged in.
`whatis`	Gives a one-line description about the command. It can be used as a quick reference for any command.

`man`	Manual Pages, for more detailed information, Linux provides man pages and info pages. To see a command's manual page, man command is used.
`pwd`	Prints the absolute path to current working directory.
`vi`	A text editor for Linux operating system.
`gedit`	The default GUI text editor in the Ubuntu operating system.
`emacs`	Another text editor for Linux operating system.

Vi commands:

To save and quit:

Commands	Action
:wq	Save and quit
:w	Save
:q	Quit
:w fname	Save as fname
ZZ	Save and quit
:q!	Quit discarding changes made
:w!	Save (and write to non-writable file)

More commands:

Copy-pasting within the terminal: Ctrl+Shift+C/V

Resources and Credits:

- Teaching materials from Professor Kuperman at UAlbany
- Vi reference: https://www.ele.uri.edu/faculty/vetter/Other-stuff/vi/vi-quick-ref.pdf