System Calls CLI

Generated by Doxygen 1.9.6

Chapter 1

System Calls CLI

This is a command-line program that can be used to demonstrate system calls in UNIX. It is written in C and uses the GNU C Library and the POSIX API.

This project was done as a part of the Advanced Operating Systems (MCSC202) course at the University of Delhi.

1.1 Features

This program supports the following features as of now:

- creating a file/named pipe with permissions as given by user
- reading from a file and printing the contents to `stdout`
- writing to a file from `stdin`
- displaying statistical information about a file
- copying the contents of one file to another using an unnamed pipe
- using a named pipe to communicate between two processes

Note: The reading and writing operations also support offsets and number of bytes to be read or written.

1.2 Commands

The various commands supported by this program can be listed by running the program without any arguments.

```
$ ./main
Usage: ./main [command] [options...] [file1] [file2...]
Commands:
  create: create a file or a named pipe
  read: read a file and print to stdout
  write: write to a file
  copy: copy a file using an unnamed pipe
  info: information about a file
  pipe: create a pipe and simulate communication between processes
```

1.3 Usage

Each command has its own set of options and arguments. The usage of each commmand can be known by executing `./main [command]`. The details of each command are given below.

1.3.1 Creating Files

The `create` command should be used to create a file or a named pipe. The permissions of the file or named pipe have to be specified after the path as an octal number. The `-p` flag can be used to create a named pipe.

1.3.1.1 Example: Creating a Regular File

```
$ ./main create file.txt 0644
```

1.3.1.2 Example: Creating a Named Pipe

```
$ ./main create -p pipe 0666
```

Note: In case the file or named pipe already exists, the program will exit with an error asking the user to pass the `-f` flag if they really want to do so.

1.3.2 Reading Files

The `read` command should be used to read a file and print its contents to `stdout` i.e. the console. The `-o` flag can be used to specify the offset from which the file should be read. The `-n` flag can be used to specify the number of bytes to be read from the file. If the `-n` flag is not used, the entire file will be read. If the `-o` flag is not used, the file will be read from the beginning.

1.3.2.1 Example: Reading a File

```
$ ./main read file.txt
lorem ipsum dolor amet
```

1.3.2.2 Example: Reading a File (With Offset and Number of Bytes)

```
$ ./main read file.txt -o 6 -n 5
ipsum
```

1.3.3 Writing to Files

The `write` command should be used to write to a file from `stdin` - the console or a shell pipe. The `-o` flag can be used to specify the offset from which the file should be written to. The `-n` flag can be used to specify the number of bytes to be written to the file. If the `-n` flag is not used, the entire file will be written to. If the `-o` flag is not used, the file will be written to from the beginning.

The user is prompted with a message to enter the text to be written to the file. The user can enter the contents of the file. To end the input, the user has to type `:w` followed by the `Enter` key.

1.3.3.1 Example: Writing to a File

```
$ ./main write file.txt
type :w to exit
---------------
lorem ipsum dolor amet
:w
```

1.3.4 Copying Files

The `copy` command should be used to copy the contents of one file to another using an unnamed pipe. If the destination file exists, the program will force the user to pass the `-f` flag to overwrite it.

1.3.4.1 Example: Copying a File

```
$ ./main copy file.txt file2.txt
Copied file.txt to file2.txt...
```

1.3.5 File Information

The `info` command should be used to display statistical information about a file. The information displayed includes the file type, permissions, number of hard links, owner, group, size, block size, number of blocks, last access time, last modification time, and last status change time.

1.3.5.1 Example: Displaying File Information (Regular File)

```
$ ./main info a.txt
File: a.txt
Type: regular file
Device: 66311
Major Device: 259
Minor Device: 7
Size: 15
Blocks Allocated (in 512B units): 8
Filesystem Block Size: 4096
Inode: 15994677
Links: 1
Owner Permissions: rwx
Group Permissions: rwx
Other Permissions: rwx
Owner User: sudipto (UID: 1000)
Owner Group: sudipto  (GID: 1001)
Last Accessed: Tue Jun 27 11:16:42 2023
Last Modified: Tue Jun 27 15:04:14 2023
Last Changed: Tue Jun 27 15:04:14 2023
```

1.3.5.2 Example: Displaying File Information (Special File)

```
$ ./main info /dev/nvme0n1p1
File: /dev/nvme0n1p1
Type: block device
Device: 5
Major: 0
Minor: 5
Size: 0
Blocks Allocated (in 512B units): 0
Filesystem Block Size: 4096
Inode: 477
Links: 1
Owner Permissions: rw-
Group Permissions: rw-
Other Permissions: ---
Owner User: root (UID: 0)
Owner Group: disk  (GID: 995)
Last Accessed: Sun Jun 25 08:18:52 2023
Last Modified: Sun Jun 25 08:18:06 2023
Last Changed: Sun Jun 25 08:18:06 2023
```

1.3.6 Pipes

The `pipe` command should be used to create a pipe and simulate communication between two processes. This program can demonstrate this using both unnamed as well as named pipes.

1.3.6.1 Unnamed Pipes

If used without arguments, it creates an unnamed pipe. The user is prompted with a message to enter the text to be written to the pipe. The user can enter the contents of the pipe. To end the input, the user has to type `:q` followed by the `Enter` key. Internally, the program forks a child process and the parent process writes to the pipe and the child process reads it from the pipe and prints the contents to `stdout`.

```
$ ./main pipe
type :q to exit
----------------
Message: Hello
Received from Parent: Hello
Message: :q
Received Bye from Parent...
```

1.3.6.2 Named Pipes

If used with the `-p` flag, it uses a named pipe in the mode specified by the argument after the path of the pipe. The user is prompted with a message to enter the text to be written to the pipe. The user can enter some text and press `Enter` to send the message. To end the input, the user has to type `:q` followed by the `Enter` key.

Multiple instances of the program have to be run to demonstrate communication between processes. In case multiple readers are waiting for a writer, the first reader to read from the pipe will receive the message and the rest will receive an empty string. This is because the message is removed from the pipe after it is read. Which reader receives the message is determined by the operating system scheduler.

```
# writer process
$ ./main pipe pipe w
type :q to exit
----------------
Message: Hello World
Message: :q

# reader process
$ ./main pipe pipe r
Waiting for writer...
Received: Hello World
Received Bye...
```

1.4 Author

Sudipto Ghosh (51)
Department of Computer Science
University of Delhi

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 copy_file.h File Reference

Module to copy a file using system calls.

Functions

- int copy_file (const char ∗, const char ∗, const int)

  Copies a regular file using an unnamed pipe and the `read()` and `write()` system calls.

3.1.1 Detailed Description

Module to copy a file using system calls.

Author

Sudipto Ghosh

3.1.2 Function Documentation

3.1.2.1 copy_file()

```
int copy_file (
            const char * src,
            const char * dest,
            const int force_flag )
```

Copies a regular file using an unnamed pipe and the `read()` and `write()` system calls.

It copies the content of the source file specified by `src` to the target specified by the `dest` parameter.

This function is invoked by the `copy` command.

In case the target file already exists, the user is prompted for confirmation and needs to pass the `-f` flag to force this operation.

Parameters

src	the absolute or relative path to the source file
dest	the absolute or relative path to the destination file
force_flag	flag to force overwrite if the destination file exists

Returns

0 on success, -1 on failure

3.2 copy_file.h

Go to the documentation of this file.

```
#ifndef SYSCALL_COPY_FILE
#define SYSCALL_COPY_FILE

int copy_file(const char *, const char *, const int);

#endif
```

3.3 create_file.h File Reference

Module to create a file using system calls.

Functions

- int create_file (const char *, const int, const int, const int)

  Creates a regular file or a named pipe using the `creat()` or `mknod()` system call.

3.3.1 Detailed Description

Module to create a file using system calls.

Author

Sudipto Ghosh

3.3.2 Function Documentation

3.3.2.1 create_file()

```
int create_file (
            const char * path,
            const int perms,
            const int pipe_flag,
            const int force_flag )
```

Creates a regular file or a named pipe using the `creat()` or `mknod()` system call.

It also sets the file permissions to `perms` by using the mode bits supplied.

This function is invoked by the `create` command.

In case the target file already exists, the user is prompted for confirmation and needs to pass the `-f` flag to force this operation.

Parameters

path	the absolute or relative path of the file to create with filename
perms	the permissions to set for the file using mode bits
file_type	flag to denote regular file or named pipe
force_flag	flag to force create if the file exists

Returns

0 on success, -1 on failure

3.4 create_file.h

Go to the documentation of this file.

```
#ifndef SYSCALL_CREATE_FILE
#define SYSCALL_CREATE_FILE

int create_file(const char *, const int, const int, const int);

#endif
```

3.5 helpers.h File Reference

Module containing helper functions for the program.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
```

Functions

- int parse_command (const char ∗)

  Parses a command string and returns the corresponding command code or -1 if the command is invalid.
- int parse_perms (const char ∗)

  Parses a permissions string and returns the corresponding permissions or -1 if the permissions are invalid.
- void flush_buffer (char ∗)

  Flushes the buffer by filling null values in it.

3.5.1 Detailed Description

Module containing helper functions for the program.

Author

Sudipto Ghosh

3.5.2 Function Documentation

3.5.2.1 flush_buffer()

```
void flush_buffer (
            char * buf )
```

Flushes the buffer by filling null values in it.

Parameters

buffer	buffer to flush

3.5.2.2 parse_command()

```
int parse_command (
            const char * command )
```

Parses a command string and returns the corresponding command code or -1 if the command is invalid.

Parameters

command	string to parse

Returns

command code or -1 if invalid

3.5.2.3 parse_perms()

```
int parse_perms (
            const char * permissions )
```

Parses a permissions string and returns the corresponding permissions or -1 if the permissions are invalid.

Parameters

perms	string to parse

Returns

permission mode bits or -1 if invalid

3.6 helpers.h

Go to the documentation of this file.

```
#ifndef HELPERS_AOS
#define HELPERS_AOS

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int parse_command(const char *);

int parse_perms(const char *);

void flush_buffer(char *);

#endif
```

3.7 named_pipe.h File Reference

Module to demonstrate use of named pipes for message passing.

Functions

- int named_pipe (const char *, const int)

  Uses named pipes for message passing between reader and writer processes.

3.7.1 Detailed Description

Module to demonstrate use of named pipes for message passing.

Author

Sudipto Ghosh

3.7.2 Function Documentation

3.7.2.1 named_pipe()

```
int named_pipe (
            const char * path,
            const int read_flag )
```

Uses named pipes for message passing between reader and writer processes.

If invoked in write mode, the program will create a named pipe at the specified path if it doesn't exist and write to it. If invoked in read mode, the program will wait for a writer to write to the named pipe and then read from it.

In write mode, the user is prompted to enter a message to write to the named pipe. The user can exit the program by entering `:q` as the message.

This function is invoked by the `pipe` command.

In read mode, the program will wait for a writer to write to the named pipe and then read from it. The process will exit when it receives `:q` from the writer.

Parameters

path	the path to the named pipe
read_flag	0 for write mode, 1 for read mode

Returns

0 on success, -1 on failure

3.8 named_pipe.h

Go to the documentation of this file.

```
#ifndef SYSCALL_NAMED_PIPE
#define SYSCALL_NAMED_PIPE

int named_pipe(const char *, const int);

#endif
```

3.9 read_file.h File Reference

Module to read a file using system calls.

Functions

- int read_file (const char ∗, const int, const int)

  Reads a regular file using the `read()` system call and writes the content to `stdout`.

3.9.1 Detailed Description

Module to read a file using system calls.

Author

Sudipto Ghosh

3.9.2 Function Documentation

3.9.2.1 read_file()

```
int read_file (
            const char * path,
            const int offset,
            const int nBytes )
```

Reads a regular file using the `read()` system call and writes the content to `stdout`.

This function is invoked by the `read` command.

Parameters

path	the absolute or relative path to the file to be read
offset	the offset from the beginning of the file
nBytes	the number of bytes to read from the file, -1 to read the entire file

Returns

nBytes denoting number of bytes read from file, -1 on failure

3.10 read_file.h

Go to the documentation of this file.

```
#ifndef SYSCALL_READ_FILE
#define SYSCALL_READ_FILE

int read_file(const char *, const int, const int);

#endif
```

3.11 stat_file.h File Reference

Module to get statistical information about a file.

Functions

- int stat_file (const char ∗)

  Queries information about a file using `stat()` system call.

3.11.1 Detailed Description

Module to get statistical information about a file.

Author

Sudipto Ghosh

3.11.2 Function Documentation

3.11.2.1 stat_file()

```
int stat_file (
            const char * path )
```

Queries information about a file using `stat()` system call.

It prints the following information about the file to the console:

- File type
- Device ID
- Major device ID
- Minor device ID
- File size (in bytes)
- Blocks allocated
- Filesystem block size
- Inode
- Links
- Owner permissions
- Group permissions
- Other permissions
- Owner user and UID
- Owner group and GID
- Last accessed
- Last modified
- Last changed

This function is invoked by the `info` command.

Parameters

path	the absolute or relative path of the file

Returns

0 on success, -1 on failure

3.12 stat_file.h

Go to the documentation of this file.

```
#ifndef SYSCALL_STAT_FILE
#define SYSCALL_STAT_FILE

int stat_file(const char *);

#endif
```

3.13 syscalls.h File Reference

Module to import dependencies to demonstrate use of system calls.

```
#include "copy_file.h"
#include "create_file.h"
#include "named_pipe.h"
#include "read_file.h"
#include "stat_file.h"
#include "unnamed_pipe.h"
#include "write_file.h"
```

3.13.1 Detailed Description

Module to import dependencies to demonstrate use of system calls.

Author

Sudipto Ghosh

3.14 syscalls.h

Go to the documentation of this file.

```
#ifndef SYSCALLS_AOS
#define SYSCALLS_AOS

#include "copy_file.h"
#include "create_file.h"
#include "named_pipe.h"
#include "read_file.h"
#include "stat_file.h"
#include "unnamed_pipe.h"
#include "write_file.h"

#endif
```

3.15 unnamed_pipe.h File Reference

Module to demonstrate use of unnamed pipes for message passing.

Functions

- int unnamed_pipe (void)

  Uses unnamed pipes for message passing between a parent and a child process.

3.15.1 Detailed Description

Module to demonstrate use of unnamed pipes for message passing.

Author

Sudipto Ghosh

3.15.2 Function Documentation

3.15.2.1 unnamed_pipe()

```
int unnamed_pipe (
             void  )
```

Uses unnamed pipes for message passing between a parent and a child process.

It creates a pipe using the `pipe()` system call and forks a child process. The parent process writes to the pipe and the child process reads from it.

The user is prompted to enter a message to write to the pipe. The user can exit the program by entering `:q` as the message.

This function is invoked by the `pipe` command.

Returns

0 on success, -1 on failure

3.16 unnamed_pipe.h

Go to the documentation of this file.

```
#ifndef SYSCALL_UNNAMED_PIPE
#define SYSCALL_UNNAMED_PIPE

int unnamed_pipe(void);

#endif
```

3.17 usage.h File Reference

Module to print usage information for the program and its commands.

Functions

- void **print_usage** (void)

  Prints usage information for the program.
- void print_usage_command (char ∗)

  Prints usage information for the commands of the program.

3.17.1 Detailed Description

Module to print usage information for the program and its commands.

Author

Sudipto Ghosh

3.17.2 Function Documentation

3.17.2.1 print_usage_command()

```
void print_usage_command (
            char * command )
```

Prints usage information for the commands of the program.

Parameters

command	command to print usage information for

3.18 usage.h

Go to the documentation of this file.

```
#ifndef USAGE_AOS
#define USAGE_AOS

void print_usage(void);

void print_usage_command(char *);

#endif
```

3.19 write_file.h File Reference

Module to write to a file using system calls.

Functions

- int write_file (const char *, const int, const int, const int)

  Writes to a regular file from `stdin` *using the* `write()` *system call.*

3.19.1 Detailed Description

Module to write to a file using system calls.

Author

Sudipto Ghosh

3.19.2 Function Documentation

3.19.2.1 write_file()

```
int write_file (
            const char * path,
            const int offset,
            const int nBytes,
            const int force_flag )
```

Writes to a regular file from `stdin` using the `write()` system call.

It uses `getline()` to read ther user input from `stdin` and writes to the file using `write()` system call. The user has to enter `:w` to stop writing to the file and close its file descriptor.

This function is invoked by the `write` command.

In case the file already exists, the user is prompted for confirmation and needs to pass the `-f` flag to force this operation. It either truncates the file or creates the file with the default permissions of `0644`.

Parameters

path	the absolute or relative path to the file to be written
offset	the offset from the beginning of the file to start writing
nBytes	the number of bytes to write to the file, -1 to read the entire file
force_flag	flag to force truncate first if the file exists

Returns

0 on success, -1 on failure

3.20 write_file.h

Go to the documentation of this file.

```
#ifndef SYSCALL_WRITE_FILE
#define SYSCALL_WRITE_FILE

int write_file(const char *, const int, const int, const int);

#endif
```

Index