Deign Document: ASGN-0 mycat

Hang Yuan
CruzID: hyuan3

1 Goals

The goal of this program is to perform functionality similar to that of mycat. mycat will reads files sequentially, writing them to standard output, without support for any flags and functional operators. mycat will handle unlimited arguments.

mycat will print out the error messages when face specific circumstances and skip the file that cannot be opened. Only three cases exist in basic mycat.

2 Design

There are two parts to the design. The program first does initialization, processing arguments, and then based on the validation of the passed argument, mycat will either print them in standard output on by one, or print out error messages.

2.1 Handling arguments

Following the mycat command, all arguments of file_name will be stored in argv[] automatically. Based on the number of passed arguments, two situations (1 and more than 1) are considered. If the argc is 1, meaning that only mycat and no arguments, then mycat will continually read the standard input from the keyboard and write standard output on the console. If the argc is more than 1, meaning there are arguments that need to be executed.


```
Input  : Array of arguments: arguments
Input  : Array length: arg_count
1.  buf ← 64KiB
2.  if arg_count = 1 then
3.      if read() = 0 then // no arguments
4.          break ;
5.      end
6.      if read() = -1 then // fail to read file → it's a directory_path
7.          error_message (buf, "-", errno) ;
8.          break ;
9.      end
10.     buf ← std input
11.     std output ← buf
12. else
13.     for i ← 1 to arg_count do
14.         file_check (arguments i) ;
15.     end
16. end
```

Algorithm 1: Initialization code and main program loop to check each argument

2.2 Check argument validation

With the inner functions `open(2)`, `read(2)`, `write(2)`, `mycat` can check the validation of the argument as a filename. Based on the returned result of functions, we have five basic different cases:

1. `mycat`
   read and write standard I/O continually
2. `mycat exist_file_name`
   Print out the content of files
3. `mycat not_exist_file_name`
   Prints out Error Message: "`mycat: not_exist_file_name: No such file or directory`"
4. `mycat directory_path`
   rints out Error Message: "`mycat: directory_path: Is a directory`"
5. `mycat no_read_permission_file_name`
   Prints out Error Messahe: "`mycat: no_read_permission_file_name: Permission denied`"

Adopted functions

`open(2)`: int open(const char *pathname, int flags) ← if return -1, fail to open the file, meaning not exist

`read(2)`: ssize_t read(int fd, void *buf, size_t count) ← if return -1, fail to read file, meaning it's a directory
if return 0, either touch the end of file or fill up the fixed buffer

`write(2)`: ssize_t write(int fd, const void *buf, size_t count) ← no matter what value is returned

Size of file

In terms of the size of file, because the buffer is fixed at 64KiB, if the file is over than 64KiB, `mycat` should run the `read(2)` and `write(2)` again until finish reading the entire file content. In this case, adding one do-while will control the loop.

Size of standard input

`mycat` can handle unlimited standard input at one time for `mycat` without any other arguments. The `read(2)` function will automatically handle this by offset and make sure the entire input can be read and written.

```
1.  function file_check (filename):
2.      open (filename)
3.      if open() = -1 then
4.          error_message (buf, "-", errno) ;     /* "mycat: not_exist_file_name: No such file or directory" */
5.      else
6.          do
7.              buf ← read(filename)
8.              if read() = -1 then
9.                  error_message (buf, "-", errno) ;     /* "mycat: directory_path: Is a directory" */
10.             else
11.                 write(buf)
12.             end
13.         while read_result != 0 and read_result != 1
14.     end
```

Algorithm 2: Check argument validation

2.3 Error Message Control

`mycat` uses `<errno.h>` to decide the type of error and give proper error message. In order to satisfy all situations, including unix special command arguments, mycat uses a method to generate error message and print it out.

The error messages include:

1. Not such file or directory
2. Is a directory

Permission issue of file

When `mycat` reaches the file that the current user doesn't have permission to read or write, `UNIX` will prints out an error message directly.

```
1.  function error_message (buffer, filename, errno):
2.      memset(buffer, 0, sizeof(buffer))
3.      strcat(buffer, "mycat: filename: strerror(error_number)")
```

Algorithm 3: Generate error message

