CS232 Operating Systems
Assignment 02: Introduction to System Calls

Mudasir Hanif Shaikh (ms03831) Kainat Abbasi (ka04051)

Fall 2019

Question No. 1: mycat

```c
/*
Question1: mycat
Mudasir Hanif Shaikh (ms03831) and Kainat Abbasi (ka04051)
CS 2021, Habib University
Assignment 2, OS, Fall 2019
*/


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //for close
#include <fcntl.h> // for open
#include <string.h>
#include <errno.h>
/* References:
https://c-for-dummies.com/blog/?p=1758
https://stackoverflow.com/questions/19472546/implicit-declaration-of-function-close */

extern int errno ;

int isNumbered(char* arg)
{
  if (!(strcmp(arg, "-n"))) return 1;
  return 0;
}

int main(int argc, char *argv[])
{
  int i;
  int line_count;
  FILE* file;
  size_t buffer = 120;
  char *line = (char *) malloc(buffer);
  int check = isNumbered(argv[1]);
  if (isatty(0) || argc > 1){
    for (i = 1; i < argc; i++) {
      line_count = 1;
```

```
38        //if isNumbered(argv[i]) continue;
39        file = fopen(argv[i],"r"); //opens the file in readonly mode
40        if(file == NULL) {
41        fprintf(stderr, "mycat: %s %s\n", argv[i], strerror( errno ))
      ;
42          continue;
43          }
44    while ( getline ( &line, &buffer, file ) >= 0 )
45    {
46    if (check) printf("     %d %s", line_count, line);
47    else printf("%s", line);
48    line_count++;
49    }
50    fclose(file);
51    }
52    }
53    else{
54      char ch;
55      while ( read(STDIN_FILENO, &ch, 1) > 0 ) {
56        printf("%c", ch);
57      }
58    }
59    free(line);
60    return 0;
61  }
```

Question No. 2: Mini shell HUsh

```
1
2  /*
3  Question2
4  Mudasir Hanif Shaikh (ms03831) and Kainat Abbasi (ka04051)
5  CS 2021, Habib University
6  Assignment 2, OS, Fall 2019
7  */
8
9
10 #include <stdio.h>
11 #include <sys/wait.h>
12 #include <errno.h>
13 #include <string.h>
14 #include <stdlib.h>
15 #include <stdbool.h>
16 #include <limits.h>
17 #include <fcntl.h>
18 #include <unistd.h>
19 #include <sys/types.h>
20 #include <signal.h>
21
22 int getStrSize(char** str){
23 int size = 0;
24 while (str[size] != 0) size++;
25 size++; //(for null termination)
26 return size;
27 }
28
29
30 int getSizeCharArray(char* str){
31 int size = 0;
32 while (str[size] != 0) size++;
33 size++; //(for null termination)
34 return size;
35 }
36
37
38 int getCountOfSubstrings1(char** const str, const char*
       inpDelimiter){
39 int cDelimiter = 0;
40 int i;
41 int strSize = getStrSize(str);
42 for (i = 0; i < strSize; i++) {
43 if (str[i] == inpDelimiter) cDelimiter++;
44 } return cDelimiter + 2; // "A & b" : delimiter count = 1,
       substrings = 2, additional 1 for null terminating character;
45 }
46
47 int getCountOfSubstrings(char* const str, const char inpDelimiter){
48 int cDelimiter = 0; int i;
49 int strSize = getSizeCharArray(str);
50 for (i = 0; i < strSize; i++) {
51 if (str[i] == inpDelimiter) cDelimiter++;
52 } return cDelimiter + 2; // "A & b" : delimiter count = 1,
       substrings = 2, additional 1 for null terminating character;
```

```
}


char** split(char* str, const char inpDelimiter, int spaces)
{
char delimiter[3];
delimiter[0] = inpDelimiter;
delimiter[1] = '\n';
delimiter[2] = 0;
int substrings = getCountOfSubstrings(str, inpDelimiter);
int sizeChar = sizeof(char*);
char ** substr = malloc(substrings*sizeChar);
int i;
substr[0] = strtok(str, delimiter);
//printf("%s \nsubtr0 ", substr[0]);
for (i = 1; i < substrings; i++)
{
substr[i] = strtok(NULL, delimiter);

}
return substr;
}


char** split1(char* str, const char * inpDelimiter, int spaces)
{
int substrings = getCountOfSubstrings1(&str, inpDelimiter);
int sizeChar = sizeof(char*);
char ** substr = malloc(substrings*sizeChar);
int i;
substr[0] = strtok(str, inpDelimiter);
for (i = 1; i < substrings; i++)
{
substr[i] = strtok(NULL, inpDelimiter);
}
return substr;
}

struct node {
int PID;
char* name;
struct node *next;
};


struct node *head = NULL;


void add(int PID, char* name) {
struct node *newProcess = (struct node*) malloc(sizeof(struct node)
    );
newProcess->PID = PID;
newProcess->name = name;
newProcess->next = head;
head = newProcess;
}

```

```
109
110 void printRunningProcesses(){
111 struct node *temp = head;
112 while (temp!=NULL){
113 int a = kill(temp->PID, 0);
114 if (a == 0 && errno != ESRCH ){
115 printf("name: %s, PID: %i\n", temp->name, temp->PID);
116 }
117 temp = temp->next;
118 }
119 }
120
121 void killAllProcesses(){
122 struct node *temp = head;
123 while (temp!=NULL){
124 kill(temp->PID, SIGTERM);
125 temp = temp->next;
126 }
127 }
128
129 int runCommand(char* command, int wt, int ad){
130
131 int rc = fork();
132 if (rc < 0) {
133 printf("fork failed\n");
134 exit(1);
135 } else if (rc == 0) {
136 char name[strlen(command) + 1];
137 strcpy(name, command);
138 int i = 0;
139 int redirection = 0;
140 char **myargs = split(name, ' ', 0);
141
142 while( myargs[i] != 0 )
143 {
144 if( strncmp(myargs[i], ">>", 2) == 0 )
145 {
146 redirection = 2;
147 close(STDOUT_FILENO);
148 open(myargs[i+1], O_CREAT | O_WRONLY | O_APPEND , S_IRWXU);
149 break;
150 }
151 else if( strncmp(myargs[i], ">", 1) == 0 ) {
152 redirection = 1;
153 close(STDOUT_FILENO);
154 open("a.txt", O_CREAT | O_WRONLY, 0777);
155 break;
156 }
157 else {
158 redirection = 0;
159 }
160 i++;
161 }
162
163 int size = 0;
164 while(myargs[size] != 0)
165 {
```

```
size++;
}
//if( strncmp(myargs[i], ">>", 2)
if (redirection > 0) size--;
char * myargs1[ size + 1];
i = 0;
int y = 0;
for( i = 0; i < size; i ++)
{
if( strncmp(myargs[i], ">>", 2)  == 0 ||  strncmp(myargs[i], ">",
    1)  == 0){
continue;
}
myargs1[y] = strdup(myargs[i]);
y++;
}
myargs1[y] = 0;
free(myargs);
if (execvp(myargs1[0], myargs1) == -1) {
printf("Error executing command\n");
}

} else {
if (ad != 0) add(rc, command);
if (wt){
int rcWait = wait(NULL);
if (rcWait < 0){
printf("Error while executing the wait command\n");
}
}

} return 0;
}




int callExit(){
killAllProcesses();
printf("%s\n", "Exiting program in 3....2.....1 \nExit Successful")
    ;
return 0;
}

int main(int argc, char * argv[], char *envp[]) {
char * path = getenv("PATH");
strcat(path, ":.");
setenv("PATH", path, 1);
char *input = NULL;
size_t buffer = 120;
int ampercents = 0;
int i = 0;
printf("********************************* \n %s \n
    *********************************\n", "Welcome to Habib
    University Shell");
```

```
while (1) {
input = NULL;
printf("%s", "(<--HUsh-->)>>> ");
getline(&input, &buffer, stdin);
ampercents = 0;
int wt = 1;
for(i = 0; input[i] != '\0'; ++i)
{
if('&' == input[i]) ampercents++;
}

char ** listOfCommands = split(input, '&', 0);
int commands = 0;

while(listOfCommands[commands] != 0)
{
commands++;
}

if (ampercents == 1 && commands == 1){
wt = 0;
}

for(int i = 0; i < ampercents + 1; i++) {

if (listOfCommands[i] != 0){
char commandCopy[strlen(listOfCommands[i]) + 1];
strcpy(commandCopy, listOfCommands[i]);
char ** strs = split(commandCopy, ' ', 0);
int sizeCommand = 0;

while(strs[sizeCommand] != 0)
{
sizeCommand++;
}

if ((strstr(strs[0], "cd")) != NULL){

if (sizeCommand < 2) {
printf("%s\n", "Nothing to change, please specify a directory");
continue;
}

if (chdir(strs[1]) < 0 ){
perror("Error occured while changing directory.\n");
}
else{
printf("directory change successful\n");
}
}

else if ((strstr(strs[0], "pwd")) != NULL){
char cwd[PATH_MAX];
//getting current directory
if (getcwd(cwd, sizeof(cwd))) {
printf("Dear user, your current working directory is %s\n", cwd);
}
```

```
276 else {
277 perror("Error occured while printing current directory.\n");
278 }
279 }
280
281 else if ((strstr(strs[0], "mylsenv")) != NULL){
282 int en = 0;
283 while(envp[en] != 0)
284 {
285 printf("%s\n", envp[en]); en++;
286 }
287 }
288
289 else if ( (strstr(strs[0], "myps")) != NULL ){
290 printRunningProcesses();
291 }
292
293 else if ( (strstr(strs[0], "showVAR")) != NULL ){
294 char commandCopy[strlen(listOfCommands[i]) + 1];
295 strcpy(commandCopy, listOfCommands[i]);
296 char ** tokens = split(commandCopy, ' ', 0);
297 int numTokens = 0;
298
299 while(tokens[numTokens] != 0)
300 {
301 numTokens++;
302 }
303 if (numTokens < 2) {
304 printf("%s\n", "Can't show var, not enough arguments");
305 continue;
306 }
307 else {
308 char * environment = getenv(tokens[1]);
309 if (environment) printf("%s \n", environment);
310 else printf("The command was unsuccesful in fetching environment
    vars for %s ", tokens[1]);
311 }
312 free(tokens);
313 }
314
315 else if ((strstr(listOfCommands[i], "=")) != NULL ) {
316 char commandCopy[strlen(listOfCommands[i]) + 1];
317 strcpy(commandCopy, listOfCommands[i]);
318 char ** tokens = split(commandCopy, '=', 0);
319 int numTokens = 0;
320
321 while(tokens[numTokens] != 0)
322 {
323 numTokens++;
324 }
325 if (numTokens == 2){
326 int rc = setenv(tokens[0], tokens[1], 1);
327 printf("%s%s\n", tokens[0], tokens[1]);
328 if ( rc < 0)
329 {
330 printf("%s\n", "Error while setting environment variable");
331 continue;
```

```
}
}
free(tokens);
}

else if (strstr(listOfCommands[i], "exit") != NULL) {
callExit();
free(listOfCommands);
free(input);
return 0;
}

else{
printf("%s\n", listOfCommands[i]);
runCommand(listOfCommands[i], wt, 1);
}

free(strs);
}
}

if (strstr(input, "exit") != NULL) {
callExit();
free(listOfCommands);
free(input);
return 0;
}
free(listOfCommands);
free(input);
}
return 0;
}

/*
refs:
OSTEP
string concat: https://www.codingame.com/playgrounds/14213/how-to-play-with-strings-in-c/string-concatenation
getcwd, chdir: https://stackoverflow.com/questions/298510/how-to-get-the-current-directory-in-a-c-program/
PATH_MAX: https://www.systutorials.com/241453/maximum-allowed-file-path-length-for-c-programming-on-linux/
Kill(2): http://man7.org/linux/man-pages/man2/kill.2.html;
how to find if process is running: https://stackoverflow.com/questions/11785936/how-to-find-if-a-process-is-running-in-c
*/
```