Barış Ayyıldız
1901042252

System Programming
Homework 2 - Report

mystring.c

The mystring.c file contains several string manipulation functions used by main.c:

Returns the length of a string

```
int strlength(char* s){
  int length = 0;
  while(s[length] != '\0'){
    length++;
  }
  return length;
}
```

Compares two strings and returns 0 if they are equal otherwise returns -1

```
int strcompare(char* s, char* s2){
  int length = strlength(s);
  int length2 = strlength(s2);
  if(length != length2){
    return -1;
  }
  for(int i=0; i<length; i++){
    if(s[i] != s2[i]) return -1;
  }
  return 0;
}
```

Removes the leading and trailing whitespaces in a string

```c
char* strstrip(char* str){
  int length = strlength(str);

  if(length == 0){
    return str;
  }

  int startIndex = 0;
  int endIndex = length-1;

  while(str[startIndex] == ' '){
    startIndex++;
  }
  while(str[endIndex] == ' '){
    endIndex--;
  }

  char *res = malloc(sizeof(char) * (endIndex - startIndex + 1));

  for(int i=startIndex; i<=endIndex; i++){
    res[i-startIndex] = str[i];
  }
  return res;
}
```

Tokenizes string with the given character

```c
char** stringTokenizer(char *str, char c){
  char** res;
  int counter = 0;
  int indexArray[20];
  int isSingleQuoteOpen = 0;
  int isDoubleQuoteOpen = 0;

  if(strlength(str) == 0){
    res = malloc(sizeof(char*));
    res[0] = malloc(sizeof(char));
    res[0] = str;
    return res;
  }

  for(int i=0; i<strlength(str); i++){
    if(c == ' '){
      if(str[i] == '\''){
        isSingleQuoteOpen = !isSingleQuoteOpen;
      }else if(str[i] == '\"'){
        isDoubleQuoteOpen = !isDoubleQuoteOpen;
      }else if(str[i] == ' ' && !isSingleQuoteOpen && !isDoubleQuoteOpen){
        indexArray[counter++] = i;
      }
    }else{
      if(str[i] == c){
        indexArray[counter++] = i;
      }
    }
  }
```

Input:

str="input string"

c=' '

Output: ["input", "string", NULL]

Returns the size of tokenizer array

```c
int sizeOfTokenizer(char** tokenizer){
  int counter = 0;
  while(tokenizer[counter] != NULL){
    counter++;
  }
  return counter;
}
```

Slices tokenizer array by the start end end indexes

```
char** sliceTokenizer(char** tokenizer, int start, int end){
  char** res = malloc(sizeof(char*)*(end-start+2));
  int counter = 0;
  for(int i=0; i<=(end-start); i++){
    res[i] = tokenizer[start+i];
    counter++;
  }
  // [a,b,c,d], (0,2) => [a,b,c]
  res[counter] = NULL;
  return res;
}
```

Concatenates two tokenizer arrays

```
char** concatTokenizer(char** tokenizer1, char** tokenizer2){
  int size1 = sizeOfTokenizer(tokenizer1);
  int size2 = sizeOfTokenizer(tokenizer2);

  printf("size1 : %d\nsize2 : %d\n", size1, size2);

  char** res = malloc(sizeof(char*) * (size1+size2+1));
  int counter = 0;
  for(int i=0; i<size1; i++){
    printf("->%s\n", tokenizer1[i]);
    res[counter++] = tokenizer1[i];
  }
  printf("..........\n");
  printf("#%s\n", tokenizer2[0]);
  printf("#%s\n", tokenizer2[1]);
  for(int i=0; i<size2; i++){
    printf("->%s\n", tokenizer2[i]);
    res[counter++] = tokenizer2[i];
  }
  res[counter] = NULL;
  return res;
}
```

Concatenates two strings

```c
char* concatStrings(char* str, char* str2){
  int size = strlength(str);
  int size2 = strlength(str2);
  int idx = 0;

  char* res = malloc(sizeof(char) * (size+size2+1));
  for(int i=0; i<size; i++){
    res[idx++] = str[i];
  }
  for(int i=0; i<size2; i++){
    res[idx++] = str2[i];
  }
  res[idx] = '\0';
  return res;
}
```

Returns the index of a token in the given tokenizer array, returns -1 if its not exists

```c
int indexOf(char** tokens, char* token){
  for(int i=0; i<sizeOfTokenizer(tokens); i++){
    if(strcompare(tokens[i], token) == 0){
      return i;
    }
  }
  return -1;
}
```

mysyscall.c

This function returns parameters for the **execv** function. It simply adds NULL to the and of the tokens array

```
char** generateParameters(char** tokens){
  int size = sizeOfTokenizer(tokens);
  char* command;
  char** rest;
  char** res;        You, 2 days ago • updat

  if(size == 0){
    return;
  }else if(size == 1){
    res = malloc(sizeof(char*)*2);
    res[0] = tokens[0];
    res[1] = NULL;
    return res;
  }

  res = malloc(sizeof(char*)*(size+1));
  for(int i=0; i<size; i++){
    res[i] = tokens[i];
  }
  res[size] = NULL;
  return res;
}
```

Generates path for the given command using concatStrings function

```
char* generatePath(char* command){
  return concatStrings("/bin/", command);
}
```

main.c

This is the main function. It has a while loop and inside it waits for a user input. Then it tokenizes the input with the character **'|'.** Then it strips the **first element** inside tokenpipes and clears the whitespaces. The reason I'm only working with the first element is because I couldn't implement pipes, so the user is able to give only one command at a time. After that it tokenizes the string, but this time with ' '.

```
while(1){
  printf("$ ");
  scanf("%[^\n]", buffer);
  getchar();

  tokenPipes = stringTokenizer(buffer, '|');
  int sizeTokenPipes = sizeOfTokenizer(tokenPipes);

  tokenPipes[0] = strstrip(tokenPipes[0]);
  commandTokens = stringTokenizer(tokenPipes[0], ' ');
```

Then it looks for **'>'** or **'<'** characters inside tokens for redirection. If they exist they enter their related condition and open a file descriptor to read or write operations and create a new process with the function **fork**. Also if they exist, the sliceTokenizer function gets called here to slice the tokens before the redirection character
For example:

Tokens : [echo, hello, >, input.txt]
Sliced : [echo, hello]

```
indexInput = indexOf(commandTokens, "<");  // input
if(indexInput != -1){
  fd = open(commandTokens[indexInput+1], O_RDONLY);
  char** sliced = sliceTokenizer(commandTokens, 0, indexInput-1);
  params = generateParameters(sliced);
  pid = fork();
  if(pid == 0){
    // child process
    dup2(fd, 0);
    close(fd);

    execv(generatePath(commandTokens[0]), params);
    perror("execl error");
    exit(1);
  }
}
indexOutput = indexOf(commandTokens, ">");  // output
if(indexOutput != -1){
  fd = open(commandTokens[indexOutput+1], O_WRONLY | O_CREAT, 0777);
  char** sliced = sliceTokenizer(commandTokens, 0, indexOutput-1);
  params = generateParameters(sliced);
  pid = fork();
  if(pid == 0){
    // child process
    dup2(fd, 1);
    close(fd);

    execv(generatePath(commandTokens[0]), params);
    perror("execl error");
    exit(1);
  }
}
```

If they don’t exists we simply create the parameters and call the execv function

```
if(indexInput == -1 && indexOutput == -1){
  params = generateParameters(commandTokens);
  command = commandTokens[0];

  pid = fork();
  if(pid == 0){
    // child process
    execv(generatePath(command), params);
    return 1;
  }
}
```

After the loop we call **wait(NULL)** to wait all child processes and check if the command is equal to **:q**, which is used for termination.

```
wait(NULL);
if(strcompare(":q", buffer) == 0){
  break;
}
```

Outputs

```
$ ls -la
total 32
drwxr-xr-x 1 barisayyildiz barisayyildiz   512 Apr 14 19:51 .
drwxr-x--- 1 barisayyildiz barisayyildiz   512 Apr 14 19:16 ..
drwxr-xr-x 1 barisayyildiz barisayyildiz   512 Apr 14 19:16 .git
drwxr-xr-x 1 barisayyildiz barisayyildiz   512 Apr 10 19:32 .vscode
-rw-r--r-- 1 barisayyildiz barisayyildiz   165 Apr 13 21:24 Makefile
drwxr-xr-x 1 barisayyildiz barisayyildiz   512 Apr 10 19:42 include
-rwxr-xr-x 1 barisayyildiz barisayyildiz 21096 Apr 14 19:51 main
-rw-r--r-- 1 barisayyildiz barisayyildiz  2006 Apr 14 19:21 main.c
drwxr-xr-x 1 barisayyildiz barisayyildiz   512 Apr 10 19:42 src
-rw-r--r-- 1 barisayyildiz barisayyildiz  2540 Apr 14 19:18 test.c
$
```

```
$ pwd
/home/barisayyildiz/system_hw2
$
```

```
input.txt M ×   C main.c M
input.txt
You, 6 seconds ago | 1 author (
1  hello world
2  this is input.txt
3
```

```
$ cat < input.txt
hello world
this is input.txt
$
```

```
$ echo "overwrite input.txt" > input.txt
$
```

input.txt M × C main.c M C te

input.txt

You, 12 seconds ago | 1 author (You)

```
1 "overwrite input.txt"
2 put.txt
3
```

How to Run

make compile, to compile
./main, to run