**Student Name**: Thanh Tin Nguyen
**Student ID**: 904285164
**Email:** ttn0011@auburn.edu

# PROJECT 2: A Pipe-based WordCount Tool

**Project 2 report:**
**// I finished it independently**
**// I used sample codes from Canvas**
**https://auburn.instructure.com/courses/1474680/assignments/13357620**
**In this report, I will describe what I already did in Project 2.**

1. **How to run my project.**
2. **How to create 2 processes.**
3. **How to send data between two processes.**
4. **How to load a file in C**
5. **How to count the number of words in a string**
6. **Handling error cases,**
7. **Describe some results and discussion.**

## 1. How to Run my Program

The structure of my program is:

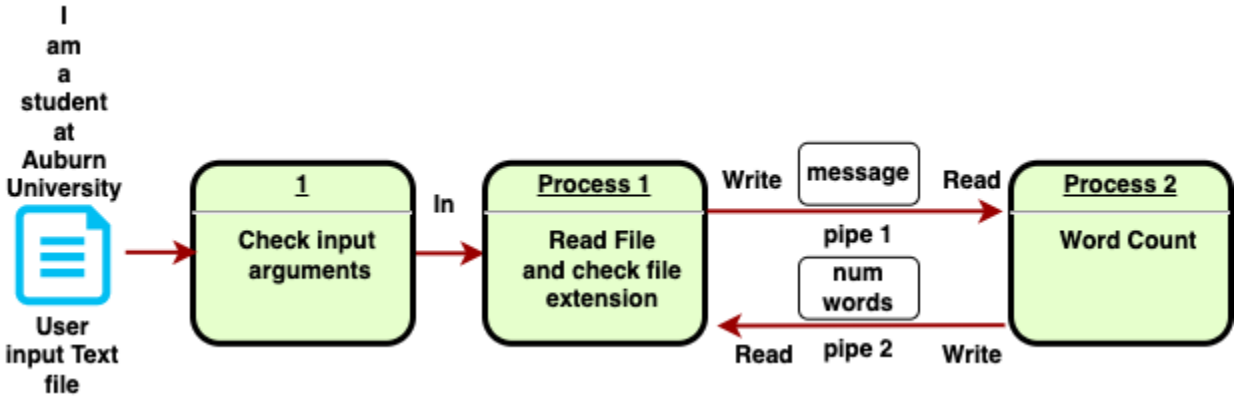
```
project2/
|-- Makefile
|-- pwordcount
|-- pwordcount.c
|-- test.txt
|-- test1.txt
|-- test3.txt
|-- test4.txt
|-- utils.c
`-- utils.h
```

- File Description:
  - The **pwordcount.c** contains the main() function used to run the entire program.
  - The **utils.h** contains the initialization of all the support functions, the **utils.c** contains the implementation of all the function in **utils.h.**
  - The **Makefile** contains the code to create the executable file **pwordcount,** and the code to clean (remove) the **pwordcount** file.
  - 
```
# compile pwordcount
pwordcount: pwordcount.c utils.c
    echo "Compiling pwordcount..."
    gcc -o pwordcount pwordcount.c utils.c -std=c99
    echo "Finished Compiling."

# delete old pwordcount
clean:
    echo "Removing pwordcount executable..."
    rm -f pwordcount
    echo "Finished Cleaning."
```
  - **Pwordcount:** The executable file that runs the WordCount program.
  - **test*.txt** files are the text files used to conduct testing.

- Run this command:
  ***gcc -g  -lm -o pwordcount utils.h utils.c pwordcount.c -std=c99***
- Or using Makefile, just type: ***make***
- If you type: ***make clean***, this will remove the *pwordcount* execution file.

## 2. Data Flow Diagram




## 3. Function Prototypes, Data Structure

| Function Prototype | Functionality |
|---|---|
| int count_words(char *word); | used to count the number of words in a file |
| char* check_filename_extension(char *filename); | check filename's extension |
| void error_checking(int error_type, char *additional_msg); | print the error, and any additional message |

### a. Creating two processes

This project aims to design and implement a program where two processes cooperate through Unix pipes.
There are two processes: (1) 1st process (parent process) and (2) 2nd process (parent process). And they are created using **fork()** function.

If the pid > 0, it will return to the 1st process.
If the pid == 0, the 2nd process will be called.
If the pid < 0, the pipe creation fails.

## b. Create Unix pipes

To make two processes communicate with each other, I created two **pipes**. The first pipe is in charge of communication from process 1 to process 2, the second pipe is in charge of communication from process 2 to process 1.
They are created using the **pipe(int fd[])** function and **fd[2]** is the **file descriptor**. Concretely, fd[0] and fd[1] are denoted for the **READ_END** and the **WRITE_END**, respectively.
**The data structure of fd[] is an array.**

```
int fd1[2]; // file descriptors for pipe 1
int fd2[2]; // file descriptors for pipe 2

printf("Creating a pipe ...\n");
// create the 1st pipe to send message from process 1 to process 2
if (pipe(fd1) == -1)
{
        fprintf(stderr,"Failed to create a pipe\n");
        return 1;
}
// create the 2nd pipe to send message from process 2 to process 1
if (pipe(fd2) == -1)
{
        fprintf(stderr,"Failed to create a pipe\n");
        return 1;
}
```

The **read()** and **write()** functions are used to read the data in the pipe and write the data to the pipe. The unused end of the write-end and read-end of pipes must be closed after finishing the read and write.

## c. Loading an Input File

Loading the input text file is taken inside process 1. The data of the file will be loaded into the variable **char content[BUFFER_SIZE]**,

the **BUFFER_SIZE** is defined as **30000**, and the **data structure is a string array.**

```
        // 1st process, which is also the parent process
        // The 1st process will read the content of a file and save into a buffer
        printf("Process 1 is reading file %s now ...\n", filename);
        FILE *file;
        static char content[BUFFER_SIZE];
        int errnum; // stores error number when attempting to open a file
        file = fopen(filename, "r");
        if(file == NULL) // if can not open the file
        {
                errnum = errno;
                // The strerror might be:
// 1 Operation not permitted
                // 2 No such file or directory
                // 3 No such process
                // 4 Interrupted system call
                error_checking(4, strerror(errnum)); // load file failed
                exit(0);
```

As you can see from the image above, the variable **int errnum** will contain the error number **(errno)** from the system. The **errno** is from the library **errno.h** which is included in the **pwordcount.c** file. And the function **strerror(int)** will convert the error number **errnum** to a string that describe the error. The **errno** number might be:

| errno | Error String |
|---|---|
| 1 | Operation not permitted |
| 2 | No such file or directory |
| 3 | No such process |

After that, the data will be sent to the 2nd process via a pipe. And using the write function to write the data into the pipe, note that, the unused read end of pipe 1 will be closed. After that, process 1 will wait for process 2 to process counting the number of words and then send this information back to process 1, still, the write end of pipe 1 will be closed.

```
// copy the data into an array which has the size of BUFFER_SIZE
strcpy(write_msg, content);

// close the read end of the pipe1
close(fd1[READ_END]);

// write to a pipe
printf("Process 1 starts sending data to Process 2 ...\n");
write(fd1[WRITE_END], write_msg, strlen(write_msg)+1);

// close the write end of the pipe1
close(fd1[WRITE_END]);

// wait for the 2nd (child) process
wait(0);
```

### d. Counting words

In this step, process 2 will first receive the data sent from process 1. Then, do word counting, and send the number of words back to process 1 via pipe 2 using **write()** function. Note that, the write end of pipe 1, the read end, and the write end of pipe 2 will be closed.

```
// read from the pipe1
read(fd1[READ_END], read_msg, BUFFER_SIZE);
printf("Process 2 finishes receiving data from Process 1 ...\n");
// close the write end of the pipe1
close(fd1[READ_END]);


// count words
printf("Process 2 is counting words now ...\n");
int num_words;
num_words = count_words(read_msg);

// close the read end of the pipe2
close(fd2[READ_END]);

// write to the pipe2
printf("Process 2 is sending the result back to Process 1 ...\n");
write(fd2[WRITE_END], &num_words, sizeof(num_words));

// close the write end of the pipe2
close(fd2[WRITE_END]);
```

The **count_words(msg)** is used to count the number of words in the message. This function is implemented in **utils.c.** The **data structure of msg is a string array.**
To count the number of words, I have a variable **int pos,** which will tell the current index while traversing the **msg** string**,** I also created a variable called **int pointer_still_in_a_word,** if it is **true**, it will tell that the **current pos** is still pointing to a word, otherwise, it will tell that the **current pos** is out of a word.

While traversing the message string array:

- If the character at a position is one of the **special characters** such as ‘\0’ (null) or ‘ ‘ (space) or ‘\t’ (tab) or ‘\n’ (new line) or ‘\r’ (return), then check if the pointer_still_in_a_word is true or not, if it is true, set it to false because a special character at this position indicates a **separation**, and then **increase the number of word by 1.**

- If the character at a position is not one of the special characters, it indicates that the current pointer is still in a word, so the value of pointer_still_in_a_word = true.

```
int count_words(char *word)
{
        int num_words = 0; // this var is to store the number of words
        int len = strlen(word); // get the length of string word

        int pos; // this var shows the position of the pointer in the string
        bool pointer_still_in_a_word = false; // this var is to tell if the current position of the pointer
                // is still pointing to a character except
                // '\0' (null), ' ' (space), '\t' (tab), '\n' (new line), '\r' (return)

        for(pos = 0; pos < len; pos++) // the pointer starts traversing the string from the beginning
        {
                char cur_char = word[pos]; // this var is to store the current character
                if(cur_char == '\0' || cur_char == ' ' || cur_char == '\t'
                 || cur_char == '\n' || cur_char == '\r')
                {
                        if (pointer_still_in_a_word)
                        {
                                pointer_still_in_a_word = false;
                                num_words++;
                        }
                }
                else
                {
                        pointer_still_in_a_word = true;
                }
        }

        return num_words;
}
```

I used references from the following links to count the word in a text file:

***strlen:***
https://www.tutorialspoint.com/c_standard_library/c_function_strlen.html

***strcmp***:
https://stackoverflow.com/questions/22527152/strcmp-giving-segmentation-fault

***strrchr:***
https://www.ibm.com/docs/en/i/7.4?topic=functions-strrchr-locate-last-occurrence-character-in-string

***strchr:***
https://www.tutorialspoint.com/c_standard_library/c_function_strchr.htm

***C error handling:***
https://www.tutorialspoint.com/cprogramming/c_error_handling.html
**Errno meaning:**
https://stackoverflow.com/questions/503878/how-to-know-what-the-errno-means
**Use bool in C**: https://www.geeksforgeeks.org/bool-in-c/
**Makefile**: https://opensource.com/article/18/8/what-how-makefile

## e. Checking possible errors

There will be some unexpected errors. The following list shows all the errors that I handled in my project. The **data structure of additional_msg here is a string array.**

```
void error_checking(int error_type, char *additional_msg)
{
        switch(error_type)
        {
                case 1: // Forget filename
                        printf("Please enter a file name.\nUsage: ./pwordcount <file_name>\n");
                        break;
                case 2: // too many input arguments
                        printf("Too many input arguments, please check again.\nUsage: ./pwordcount <file_name>\n");
                        break;
                case 3: // wrong file extension name (in this project, only using text file (*.txt))
                        printf("Wrong input file type, please use a text file.\nUsage: <file_name>\n");
                        break;
                case 4: // can't load file
                        // The additional message might be:
                        // 1 Operation not permitted
                        // 2 No such file or directory
                        // 3 No such process
                        // 4 Interrupted system call
                        printf("Error opening file: %s\n", additional_msg);
                        printf("Load file failed.\n");
                        break;
        }
}
```

- **Forget the file name (case 1)**: This error occurs when the user forgets to input the filename.
- **Too many input arguments (case 2)**: This error occurs when the user inputs too many arguments.
- **The extension file name is wrong (case 3)**: This error occurs when the user inputs a file that does not have an extension (for example: “abc”) or a file that has an extension other than “txt” (for example: “abc.rar”)
- **Can not load file (case 4)**: This error occurs might be because the program can not find the file or the file is locked.

In case 3, I check the file name extension by a function called **char* check_filename_extension(char* filename)**

```
char* check_filename_extension(char *filename)
{
        //This function will return empty string denoted that if there is no filename extension
        //It will return the file name extension if there is a filename extension

        //Note that: If "." is the first character in filename. Ex: ".abcdef",
        //then, this file is a hidden file, and the filename extension will be "abcdef"

        char *dot = strrchr(filename, '.'); // search for the last occurrence of the character "."
        int len = strlen(filename); // get the length of filename

        if(!dot)// can not find the "." in filename. Ex: "abcdef"
        {
                printf("Can not find the \".\" in filename.\n");
                return "";
        }
        else if('.' == filename[len-1]) // "." is the last character in filename. Ex: "abcdef."
        {
                printf("\".\" is the last character in filename\n");
                return "";
        }

        return dot + 1; // get the string (filename extension)
}
```

In this function, firstly, it finds the dot "." in the string, if there is no ".", the function will return an empty string. If there is a "." at the end of the file name, it is also a wrong name, so the function still returns an empty string. Otherwise, the function will return the filename extension.

```
// check file extension name, the extension must be "txt" (text file)
char* filename, *filename_extension;
filename = argv[1];
filename_extension = check_filename_extension(filename);

if(strcmp(filename_extension, "txt") != 0)
{
        error_checking(3, "no additional message");
        return 0;
}
```

The **check_filename_extension(filename)** function is called here to check the filename extension.

## 4. Results and Discussion on Design Issues

I tested the program with some text files:

**test.txt**: The file contains the text: **"I am a student at Auburn University"**.

The following figure shows the output of the program:

```
[root@fa24daec5d50 project2]# ./pwordcount test.txt
Creating a pipe ...
Forking a child process ...
Process 1 is reading file test.txt now ...
Process 1 starts sending data to Process 2 ...
Process 2 finishes receiving data from Process 1 ...
Process 2 is counting words now ...
Process 2 is sending the result back to Process 1 ...
The total number of words is 7.
```

**test1.txt**: The file contains the text: **"ghatGPT is very dangerous, we should be careful when using it. Moreover, another tool called DALLE-2 can generate images that look very similar to real images. If we combine these two technology, it can generate an image based on an input text."**. The following figure shows the output of the program:

```
[root@fa24daec5d50 project2]# ./pwordcount test1.txt
Creating a pipe ...
Forking a child process ...
Process 1 is reading file test1.txt now ...
Process 1 starts sending data to Process 2 ...
Process 2 finishes receiving data from Process 1 ...
Process 2 is counting words now ...
Process 2 is sending the result back to Process 1 ...
The total number of words is 42.
```

Some error cases:

a. Forget the file name:

```
[root@fa24daec5d50 project2]# ./pwordcount
Not enough input arguments, please enter a file name.
Usage: ./pwordcount <file_name>.txt
```

b. Too many inputs:

```
[root@fa24daec5d50 project2]# ./pwordcount test1.txt abcde
Too many input arguments, please check again.
Usage: ./pwordcount <file_name>.txt
```

c. Wrong file name extension:

```
[root@fa24daec5d50 project2]# ./pwordcount test1.rar
Wrong input file type, please use a text file.
Usage: <file_name>.txt
```

d. Can not find the input file:

```
[root@fa24daec5d50 project2]# ./pwordcount abc.txt
Creating a pipe ...
Forking a child process ...
Process 1 is reading file abc.txt now ...
Error opening file: No such file or directory
Load file failed.
Process 2 finishes receiving data from Process 1 ...
Process 2 is counting words now ...
Process 2 is sending the result back to Process 1 ...
```

In terms of Design Issues:

- For designing the word count function, if the comma is right after a character, for example, “I am a boy, I like cars”, the comma is attached to the word “boy” so “boy,” is counted as 1 word. But if the comma is separated from the word boy (by a space) then, the comma will be counted as 1. In this work, I did not take this into account.
- For designing the check file extension function, if the file name is “.txt”, this file is still created as a hidden file. In my project, I allow the creation of such a hidden file.
- This work can be improved by using object-oriented programming.
- In future work, this project can be used to handle large files, multiple processes, etc.