Project 2: pWordcount: A Pipe-based WordCount Tool

Satyam P. Todkar

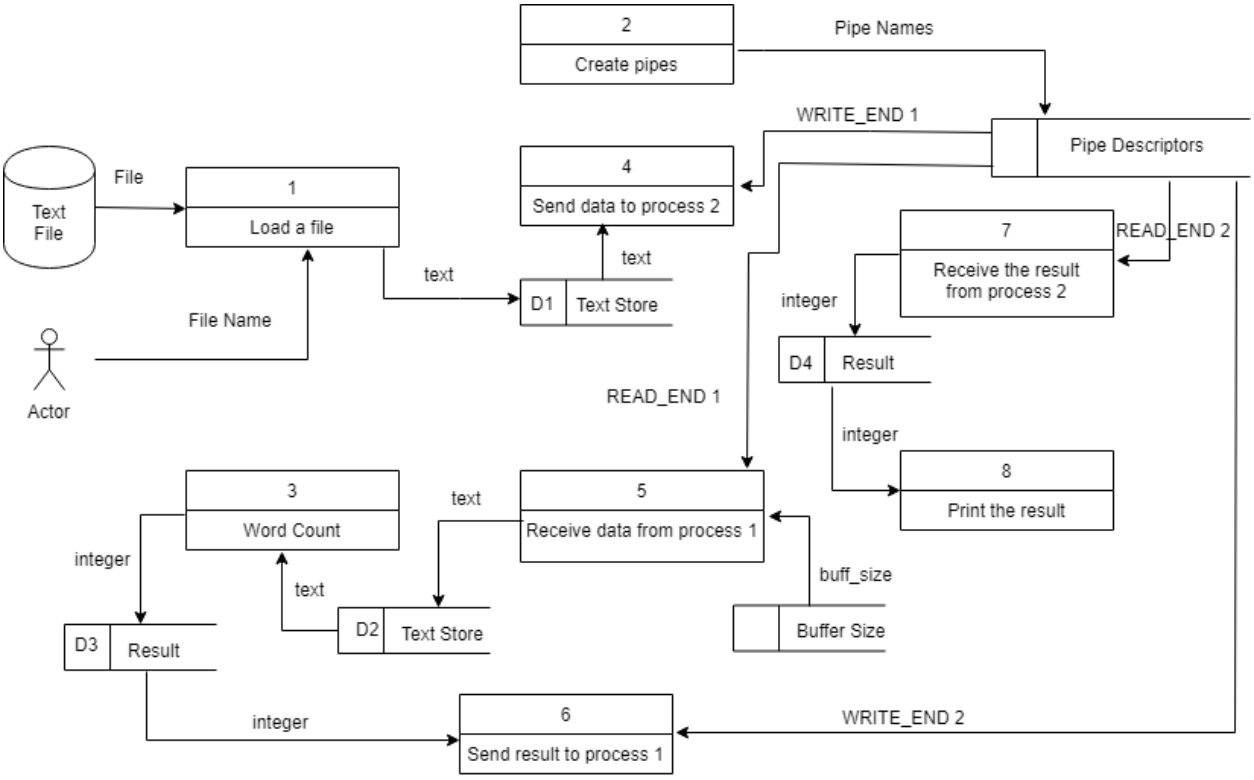
szt0064@auburn.edu

Department of Computer Science & Software Engineering, Auburn University

Introduction:

This report is about building a tool named pwordcount which accepts a text file as input and outputs the number of words in the file. The problem is to be dealt in a producer-consumer fashion and the fork system call is used to create those processes viz parent and child. A pipe forms the channel of communication between these processes and is used to send the contents of the file from the parent process to the child process. The child process then counts the words and using another pipe sends the result to the parent process. The parent process then prints the same

Data Flow Diagram:



The above figure is the data flow diagram for the process that needs to be followed in order to create the tool. It shows the interactions between the modules by stating how two modules communicate. The last module i.e. 8 should finally print the result

Functions Needed:

1. **fopen:** Used to open a file

   FILE *fopen(const char *restrict filename, const char *restrict mode);

   r or rb: Open file for reading.
   w or wb: Truncate to zero length or create file for writing.
   a or ab: Append; open or create file for writing at end-of-file.
   r+ or rb+ or r+b: Open file for update (reading and writing).
   w+ or wb+ or w+b: Truncate to zero length or create file for update.
   a+ or ab+ or a+b: Append; open or create file for update, writing at end-of-file.

   Upon successful completion, fopen() shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned and errno shall be set to indicate the error.

2. **pipe:** A pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process.

   int pipe(int fd[2]);
   where fd[0] is the read end of the pipe and fd[1] is the write end of the pipe

   Returns 0 on success and -1 on error

3. **fork:** Used to create a new process, which is called child process

   int fork();

   Returns Negative if the creation of a child process fails
   Returns Zero to the newly created child process
   Returns Positive value to the process that created the child i.e. parent process

4. **write:** A sender writes the message to the pipe using this command. write() writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf.

   ssize_t write (int fd, const void *buf, size_t count);

   Returns the number of bytes written on success. On error -1 is returned

5. **read:** A receiver reads the messages from the pipe using this command. read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

   ssize_t read(int fd, void *buf, size_t count);

   The number of bytes read is returned (zero indicates the end of file), and the file position is advanced by this number on success. On error -1 is returned. It may be possible that read() reads less number of bytes than that specified and this happens in the case where the file size is not a multiple of the number of bytes to be read

6. **close:** It is used to close the appropriate end of the pipe. The end that is unused or the one which is no longer used is closed using this function

   int close(int fd);

   where fd is an array and clos accepts fd[0] (read end) or fd[1](write end)

   Returns zero on success and -1 on failure

7. **fgets:** It is used to read a line of data from an external source. It checks that the incoming data does not exceed the buffer size

   char *fgets(char *str, int n, FILE *stream);

   On success returns the same str parameter. Returns a null pointer if the end of file is encountered or no characters have been read or an error occurs.

8. **stat:** Gets the file attributes for a file that exists.

   int stat(const char *path, struct stat *buf);

   Returns a negative value on failure

Describing the Program:

The program firstly checks for the conditions specified viz file name not given, file does not exists, failure while opening the file etc. and then performs the appropriate actions. The fork function creates a child process. The parent and child process communicate with each other using the pipe. As described in the introduction the task to count the number of words in the input file is performed by these coordinating processes.

The program is named as pwordcount.c and the input text file is named as input.txt. The function countWords and fileExists is defined and implemented in order to follow the function-oriented approach.

The program is implemented and tested for a text file with size less than 1024 bytes. The usage of fgets was used in order to read 1024 bytes from the input.txt and store them in the buffer write_msg. As the fgets function reads a line it reads until a “\n” is encountered. The program thus implemented cannot handle lines which are separated by a newline or “\n”.

The program uses a modified version of counting spaces and tabs. It makes use of a flag in order not to count spaces or tabs as words. The input in the input.txt is less than 1024 bytes in size and words are separated by a combination of space(s) and tab(s).

The output on the console i.e. execution of printf statements in code may not print in order depending on the CPU load, scheduling algorithm etc.