CS3841 Lab 4
A Multiprocessed UNIX Word counter

Dr. Walter Schilling

Due: October 11, 2012 23:59 CDT

1 Lab Objectives

1. Use command line parameters to pass data between programs.
2. Manage dynamic memory and heap allocation using C methods.
3. Manage the spawning of additional processes from within a UNIX program using the fork method.
4. Perform interprocess communication between forked processes using pipes.
5. Analyze the performance of a program in the UNIX operating environment.

2 Recommended Reference Readings

Topic	Book
POSIX Pipe	`http://www.opengroup.org/onlinepubs/9699919799/functions/pipe.html`
Unix getrusage	`http://rabbit.eng.miami.edu/info/functions/time.html#getrusage`

3 Introduction

Last week, you created a program which would count the number of words in a given file and provide results at the end. This was very useful. But, it does not solve everything. For example, what if a person wanted to count words in multiple files?

The purpose of this lab is to expand upon last week's lab to perform yet another meaningful task. In particular, you are being asked to modify [1] last week's program so that multiple text files can have their words counted. Like last week, user will run the program, passing in a command line argument indicating the names of the text file that is to have words counted.

The program is to be tested by running a set of HG Wells novels through the word counter and analyzing his writing to determine the most commonly used words in his texts. The files to be counted are available on the course website.

4 Step 1

The part of this lab is to modify your code from last week. In short, instead of accepting a single command line parameter listing the single file that is to be counted, you now must accept multiple filenames. For example, you might enter

```
wws@WWS-Ubunto:~/CS3841/Labs/WordCounterPart2/solution$ ./wordCounter C test1.txt test2.txt test3.txt
```

[1] Or start over, if necessary.

to count the words in files test1.txt, test2.txt, and test3.txt. When all is finished, a report of the total words shall be provided.

In addition to counting multiple files, you program also must determine how long it took to execute. In UNIX, there are three types of times that an application will take. Wall-clock time is the amount of time that passes on your wall clock while your process runs. This would be equivalent to starting a stopwatch at the start of execution and stopping the stopwatch when the program is finished. Because of background processes and other aspects, this is really the least useful time for us as engineers (though users are concerned about this time). System time is the amount of time the system spends on behalf of supporting the given process. This includes operating systems calls and other similar items. The user time is the amount of time spend executing in user mode. The getrusage() function provides a direct interface into the unix getrusage function. It returns a detailed array containing many values related to resource consumption.

The following code segment will return, as a double, the amount of time, in seconds, spent by a process and all child processed.

```
#include <sys/time.h>
#include <sys/resource.h>

static double getcputime();

static double getcputime()
{
    struct timeval tim;
    struct rusage ru;
    double t = 0.0;

    /* Determine the time for the process. */
    getrusage(RUSAGE_SELF, &ru);
    tim=ru.ru_utime;

    t=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;

    tim=ru.ru_stime;
    t+=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;

    /* Determine the tiume for the child processes. */
    getrusage(RUSAGE_CHILDREN, &ru);
    tim=ru.ru_utime;
    t+=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;
    tim=ru.ru_stime;
    t+=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;
    return t;
}
```

When you program completes execution, in addition to displaying the the heap memory which is still allocated, the program shall also display the amount of execution time taken by printing to standard error the sum of the user and system time. Further details can be obtained from `http://rabbit.eng.miami.edu/info/functions/time.html#getrusage`.

5 Step 2

Now that you have completed modification of your code to handle multiple files, you must again modify your code so that each file is processed in a separate process. In short, your main process will spawn, using the fork command, n children to process the n files. Upon completion, each of the n process shall pipe the results back

to the parent process. The parent process shall reconcile all of the lists, resulting in a single linked list of word counts representing the total count of all words in each file.

When you are finished, I'd like you to use the rusage structure to tell me something else about the execution of your code. In addition to time, there are many other useful performance statistics built into this structure which may be useful when one is looking at performance. Use the documentation at `http://rabbit.eng.miami.edu/info/functions/time.html#getrusage` to determine something I might like to know about your code, and print it out.

6 Specific Requirements

1. Your program shall execute under 32 bit UBUNTU Linux.
2. The program shall accept multiple command line options. The first option shall be W if the output is to show the word first or C if the output is to show the count first. The rest of the command line parameters shall be the names of the files which are to be word counted.
3. The word which is being stored, as well as the word count, shall be stored in a C structure.
4. Your program shall use the linked list from two week's ago to store word count entries.
5. The program shall operate by spawning a set of n child processes, where n represents the number of files to be word counted. Each file shall be processed in a separate process.
6. For each process that is spawned, a pipe shall be created between the parent process and the child process.
7. When the child has completed processing, the child shall pipe the results back to the parent.
8. When each child finishes processing, a message "Child finished!" shall be printed to stderr.
9. Immediately prior to program exit, your program shall print to the console the remaining dynamic memory allocated by including the following code. This should always be zero at program exit. If not, there is a memory leak within your program.

```
struct mallinfo veryend = mallinfo();
fprintf(stderr, "Final Dynamic Memory used: %d\n", veryend.uordblks);
```

10. All words shall be converted to upper case before being counted.
11. All words shall be stripped of non-alphabetic characters before being counted. For example, "it's" shall be counted as "IT" and "Drs." would be counted as "DRS".
12. Your code shall not contain any magic constants (i.e. random numbers assigned to the source code.)
13. All functions which are externally visible shall be defined in an appropriate header file.
14. All methods and variables which are limited in scope to the given file shall be declared with a "static" prefix.
15. All code must be fully commented. At the top of each file shall contain a comment block with your name, your course, the assignment name, the date, and what the file is responsible for doing. Each function shall have a function block declaring the purpose for the function, the parameters accepted, and the return value (if one exists).
16. Header files must be protected against multiple inclusion using an appropriately constructed ifdef or ifndef construct.
17. Output to the screen shall be manipulated using the C printf function.
18. File input and output shall be handled using the stdio.h library and fscanf.

7 Design

As with last weeks project, the design is left up to you. However, Figure 1 does include a simple UML diagram for a sample solution.

8 Development Process

Like last week, the development process is entirely up to you.

9 Sample Program Executions

A sample program execution is given in Figure 2.

10 Development Process

The process you use to develop this package is entirely up to you. However, it is recommended that you consider using the gnu debugger (gdb) to debug your software. A complete manual on this package can be found at http://www.gnu.org/software/gdb/documentation/. To do this, you will need to make certain your makefile compiles your program with the -g option.

11 Experimental Analysis

Once your project has been completed and is functioning properly, you are to analyze several works by HG Wells, including First Men In The Moon, In the Days of the Comet, The Invisible Man,The Island of Doctor Moreau, The Time Machine, The War In The Air, and The War Of The Worlds.

Run each of your two programs on this set of files and report the results. How long does each take to execute? Which is faster? Which non-trivial word (i.e. not A, An, the, etc.) is used most often by HG Wells in the sample writings? Alphabetically, which is the last word he uses and how often does it appear? In a table, list the top 10 words in HG Wells writings as well as the words which he only uses once in his writings.

When you are running your code, what performance are you seeing from the rusage structure? Does it change as you change the number of files processed (i.e. 1 file to 2 files to n files?)

12 Deliverables

1. Tar file including all of your source code, makefile, etc. Your source code must compile without any compiler warnings or errors.
2. Lab report, submitted in pdf format, with the following
   (a) Name, date, and lab title
   (b) What things went right and what things went wrong with this lab.
   (c) What did you learn from this lab.
   (d) Output captures of the program executing properly.
   (e) Answers to questions (including appropriate tables)
   (f) Conclusions
   (g) Source code as an appendix (In order that it can be commented)

Deliverables are to be uploaded to the course website.

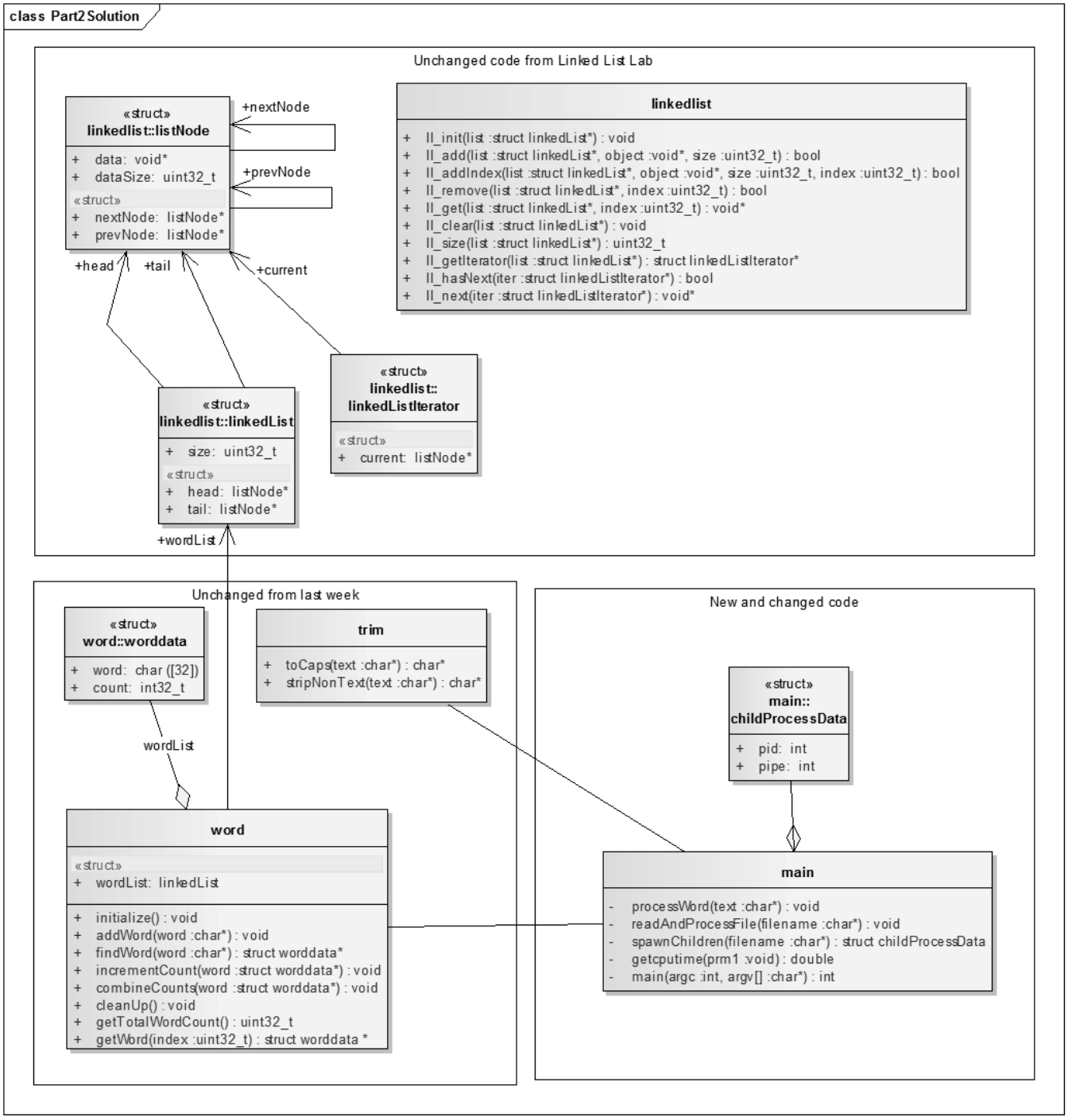

Figure 1: Sample UML diagram showing a potential design solution.

```
wws@WWS-Ubunto:~/Dropbox/ClassPreps/Fall2010/CS3841/Labs/WordCounterPart2/Part2Solution$ ./wordCounter W test1.txt test2.txt test3.txt
child finished
child finished
child finished
A 10
BIRD 1
CAME 1
DOWN 1
THE 14
WALK 1
.
.
.
LET 1
BEETLE 1
PASS 1
GLANCED 1
WITH 2
RAPID 1
EYES 1
THAT 2
HURRIED 1
ALL 1
.
.
.
VENTURE 1
BUT 1
ETERNITY 1
ENABLES 1
ENDEAVORING 1
AGAIN 2
Final Dynamic Memory Used after program completion: 0
Total Execution Time 0.004000
wws@WWS-Ubunto:~/Dropbox/ClassPreps/Fall2010/CS3841/Labs/WordCounterPart2/Part2Solution$
```

Figure 2: Example output 1 without sorting. (Note: This is a set of 3 Emily Dickinson poems.)