Midterm #1 Programming Test (10%)

CS5432 Advanced UNIX Programming, Instructor: Cheng-Hsin Hsu

Department of Computing Science, National Tsing Hua University, Taiwan

15:30 – 17:10, October 27th, 2023

- Before leaving the PC room, each group should submit the following files via **eeclass**:

  (i) **Source code** (named `q1.c` for **(1)**, and `q2.c` for **(2)**)

  (ii) **Makefile** (a single makefile should be able to compile both `q1.c` and `q2.c`)

  (iii) **A concise report** (named Group_[group_id].pdf)

  You will not be able to re-submit the code after that.

- Please use the **C language** for your answers (C++ is not allowed).
- You are allowed to search online for tips.
- You are **NOT allowed to use ChatGPT**.
- You are not allowed to copy and paste source code from the Internet.
- You cannot exchange (online/offline) messages with other groups during the exam.
- Any academic dishonesty (including but not limited to the ones mentioned above) automatically leads to zero point. In addition, we will have no choice but to report this incident to the university.

(1) (4 pts) `uniq(1)` reads the specified input file comparing adjacent lines and writes a copy of each unique input line to the output file. For example, consider an ASCII file called `animal.txt` with the following lines:

```
alligator
bear
bear
camel
camel
camel
dolphin
elephant
bear
fish
```

Running `uniq` against it gives:

```
bear$ uniq animal.txt
alligator
bear
camel
dolphin
elephant
bear
fish
```
Notice that the second last line contains `bear`, since it is not adjacent to the first two occurrences of `bear`. If you are not sure about how `uniq` behaves, please see its man page.

To "get around" the adjacency requirements, one common trick is to `sort(1)` the file, so that identical lines are next to each other. Then, the sorted file is piped to `uniq`. Please see a real example below:
```
bear$ sort animal.txt | uniq -
alligator
bear
camel
dolphin
elephant
fish
```
Note that `bear` only appears once now.

In this problem, you are required to **implement a new utility called `suniq`:**
(1) **(1 pt)** First **sorts the lines**
(2) **(3 pts)** Then acts the same as `uniq`. Please make sure that you implement the following arguments: `-c`, `-i`, and `-u`, **1 pt for each**. (They will be called separately, ex: `-c -i`. You don't need to handle combined cases like `-ci`.)

Your program should handle all these three arguments in any order. We may not exhaustively test the error handling features of your program; but please try to cover most common errors, such as "unknown arguments" and so on. **The outputs should be identical to the `uniq` utility in FreeBSD. (Please be careful that the results may vary on different systems.)**

In the report, you only need to mention how to run your program.
We provide a `source1.txt` for you to test your implementation.

(2) (6 pts) In this problem, you have to compare different types of time with different I/O routines.

Please copy the 20 MB binary file `source2` we provided to a memory stream created by `fmemopen`.

Please create a table with four columns (similar to the table below) in your report. The four columns are: Functions, User CPU, System CPU, and Clock Time. Implement a utility (i.e., a single binary executable) that supports the following I/O routines:

(1) **(1 pt)** `fread/fwrite` with a buffer size of 1 byte

(2) **(1 pt)** `fread/fwrite` with a buffer size of 32 bytes

(3) **(1 pt)** `fread/fwrite` with a buffer size of 1024 bytes

(4) **(1 pt)** `fread/fwrite` with a buffer size of 4096 bytes

(5) **(1 pt)** `fgets/fputs` with a buffer size of 4096 bytes

(6) **(1 pt)** `fgetc/fputc`

For each pair of the I/O routines, please output the data from the memory stream as a binary file named `output1`, `output2`, … and so on, we will compare the output file with `source2`. (You can use an additional `fwrite` to output the file, but be sure that you do not include the output time in your time measurement.)

In the report, in addition to the table, please make at least three observations on your timing results.

(**Hint**: Please use `times` to calculate the elapsed time.)

Function	User CPU (s)	System CPU (s)	Clock Time (s)
`fread/fwrite` with 1 byte	1.462	0.159	1.700
`fread/fwrite` with 32 bytes	0.048	0.128	0.192
`fread/fwrite` with 1024 bytes	0.006	0.146	0.187
`fread/fwrite` with 4096 bytes	0.004	0.131	0.140
`fgets/fputs` with 4096 bytes	0.001	0.124	0.130
`fgetc/fputc`	1.381	0.133	1.544

(The numbers in the table are for reference only.)