Assignment 1

Due: 8:30 A.M. Monday, May 31, 1999.
Weight: 8% of your course grade.

Preliminaries

As with all assignments in this course, a small part of the assessment for this assignment reflects how your work is presented. Your work must be easy to read. This means:

- Your work must be clear, succinct, grammatically correct, and correctly spelled.
- Your work must be laid out sensibly. In text, lines should be short enough to be readable (i.e., don't fill the entire width of the page with text—appropriate use of margins and other blank space on a page can dramatically improve the readability of your work).
- Unless there is a compelling reason to do otherwise, all pages should be on letter-size paper, with content in the portrait orientation. Pages should be held together with a single staple in the top lefthand corner.
- Avoid unnecessary embellishment—you should not use presentation folders.

Most students find it easier to create their answers to assignments using a word processor or equivalent program. Although you may handwrite your answers, it is much easier to check and correct grammar and spelling on a computer screen than a handwritten page. If you choose to handwrite your answers, they must be neat, tidy and legible.

The late policy for all assignments is as follows: 10% penalty for being one day late, 20% penalty for being two days late. Assignments may be handed in early. No assignments will be accepted more than two days late. The late period begins immediately after the beginning of class on the due date (i.e., handing in your work at 9 A.M. on the Monday it is due counts as one day late).

This assignment may be refined or clarified in class or e-mail.

Programming Exercises

For this programming assignment, you will be modifying C++ source code. This source code is available on the Web at http://www.cs.sfu.ca/CC/300/oneill/Homework/assign1.tar.gz and also in /gfs1/CMPT/300/src/assign1.tar.gz on CSIL. You can expand this archive file using the command gzcat assign1.tar.gz | tar –xvf – on any UNIX system. You can also find the contents of this archive file in the directory /gfs1/CMPT/300/src/assign1.

For each of the programming exercises below, develop the program as indicated in the question, and then include the source listing for the program in your report along with any commentary you feel is necessary to explain it. Your programs must be *short* (your changes should easily fit on a single page), but *clear* and *robust*. If the program detects an error (e.g., premature end of file), it should return exit status 1 (but no error message).

You may program in either C or C++. The source code provided is in C++, but does not use advanced C++ features—it should be straightforward to port the code to plain C.

P1. Read the UNIX manual page for the system program cat. Write your own version, called mycat, that performs the basic functionality of cat (i.e., it does not need to take option switches). Copy filecopy.cc to a new file called mycat.cc, and adapt it as necessary. You *must* use the copyout function as provided–do not modify it.

On your hardcopy source listing, use a highlighter to indicate the code you have written.

Tip: Standard input is file descriptor zero. Standard output is file descriptor one.

P2. The filecopy program does not include any handling for interrupt signals. If the user presses Control-C while the file is being copied, the program will exit and leave a partial copy behind.

Adjust the code to catch the SIGINT signal and remove the partial copy (if the file has not been completely copied) before exiting.

On your hardcopy source listing, use a highlighter to indicate the code you have written.

P3. The filecopy program incurs the overheads of the C++ iostream library even though cerr is only used to print a short error message in the event of failure.

Adjust the code to print the same error message using a single invocation of the write system call.

On your hardcopy source listing, use a highlighter to indicate the code you have written.

P4. Read the UNIX manual page for the system program wc. Write your own version, called mywc, that performs the basic functionality of wc (i.e., it does not need to take option switches or command-line arguments and only reads from standard input).

Your program may not (explicitly) call any functions in the standard C or C++ libraries except for isspace—everything else must be accomplished through system calls or functions you have written yourself.

Written Exercises

W1. If you modify BUFFER_SIZE in filecopy.cc you will find that increasing the buffer size does not make much of an impact on performance, but reducing it can make a significant difference (try setting BUFFER_SIZE to 8 or even 1).

(a) Why does a small buffer size cause such poor performance?

(b) Why doesn't increasing the buffer size beyond 4096 help much?

(c) Explain why a program performing one system call and 1000 subroutine calls should perform better than a similar program that performs no subroutine calls and 1000 system calls.

W2. Recently, some operating systems have added a system call identical in functionality to the copyout function in filecopy.cc. Give the tradeoffs involved in providing this functionality as a system call versus providing it as a subroutine in a programmer's library.

W3. You should have been reading the *Kernel Traffic* Web site during the first few weeks of the course. Pick one topic discussed there, describe the issue briefly and make constructive comments of your own. (Include the date of the original discussion and the issue number of *Kernel Traffic*).