CSC 3350 Systems Programming

Lab 5 – Concurrent Processes with Pipe Interprocess Communication: Source-Filter-Sink

Due: Wednesday, May 25, 2016 9:20am Blackboard Submit, printed lab in class

Design and implement a Source-Filter-Sink pipe application in C. The goal is to achieve concurrent execution of 3 child processes that use anonymous pipes to communicate and synchronize with each other. You must implement 3 separate small programs for this lab (Source.c, Filter.c, Sink.c) plus an additional main program (lab5.c) that creates the pipe objects and initializes and creates the 3 child processes that execute utilizing the pipes.

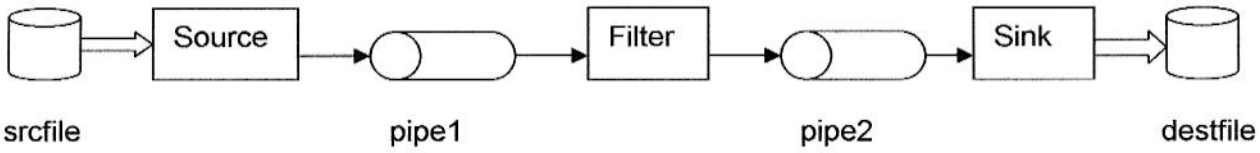
First, each of the Source, Filter and Sink programs are standalone programs that are capable of running by themselves directly from the command line (with proper commandline arguments).

- The **Source** program obtains a source text-file filename from its commandline, opens the file, and reads and copies the file contents one character at a time directly to standard output (stdout). When the file has been copied, Source terminates (closing all of its open file handles).
- The **Filter** program does not utilize any filename commandline parameters. Instead, Filter reads a text file from standard input (stdin) and writes to standard output (stdout) a copy of the input with all **upper-case** letters converted to **lower-case**. Filter specifically must be designed to read one character, convert it, output it, and then loop until the incoming data is finished.
- The **Sink** program obtains a destination text-file filename from its commandline, opens the file for writing, and then reads characters one at a time from the standard input file (stdin) and writes each incoming character directly to the destination sink file.
- Design each of the Source, Filter, and Sink applications to simulate doing “more work” by delaying (e.g., using a DELAY_A_WHILE() busy-loop) inside their copy loop, so that the copy operations proceed more slowly. The busy-loop iteration count can be adjusted to an appropriate value so that each application takes a few seconds to complete and you can visually observe the applications starting and stopping. Note that "busy-looping" is not the same as "SLEEP( )ing", which will cause a process to become "not ready" and force an O/S scheduling event.
- You can test each of the Source, Filter, and Sink applications individually by themselves running from the commandline.

```
// Example Source.c
#include <stdio.h>
#define DELAY_A_WHILE() {volatile int j; for(j=0; j<1000000; ++j) ; }

int main(int argc, char *argv[])
{
        FILE *fin;

        if(argc != 2 || fopen_s(&fin, argv[1], "rb"))
        {
                fprintf(stderr, "Usage: source inputfilename\n");
                return(1);
        }

        // read chars and copy to standard out
        while(1)
        {
                int  ch;
                if( (ch = fgetc(fin)) == EOF )
                        break;
                fputc(ch, stdout);
                // fputc(ch, stderr);  // optional for debugging: output to console too
                DELAY_A_WHILE();       // Slow down the program: simulate more "work"
        }
        fclose(fin);
        return(0);
}
```

The overall objective of utilizing a concurrent process execution pipeline could be accomplished directly by the CMD.exe shell window using the '|' pipe operator:

e.g., C:\-> **source.exe *srcfile*** | **filter.exe** | **sink.exe *destfile***



Instead of using the CMD.exe shell to create and execute this pipeline, this lab assignment involves implementing a main driver program (lab5.exe) that creates 2 pipes and spawns the 3 independent child applications with inputs and outputs configured to carry out the indicated concurrent execution and dataflow.

The **lab5 main** program requires 2 command line parameters:

e.g., C:\-> **lab5.exe *srcfile* *destfile***

where *srcfile* is an existing data text-file, and *destfile* is the filename of a new destination file that is to be created by the Sink application.

- The lab5 main program creates and executes three separate processes:
  the Source process, the Filter process and the Sink process.
  As previously described, these applications are not explicitly aware that they will be utilizing pipe resources. From their perspective, they are simply using their standard I/O streams.

- The lab5 main program also creates two anonymous file 'pipes'.
  The pipes must be used to establish communication between the three processes to link the Source output to the Filter input, and the Filter output to the Sink input.

- The lab5 main must configure and start the execution of the three processes with CreateProcess( ) system calls using STARTUPINFO parameters to appropriately enable and map the fields for the standard input and output handles that each child process will use.
  Remember, any process always can assume that all 3 handles are mapped to valid streams.

- It is important to carefully design lab5 main so that only the necessary handles (particularly write handles) are passed along and inherited by each of the child processes.

  Remember that as long as the "write" handle for a pipe is still open, the "read" handle for that pipe will not signal the end-of-file condition. To keep unnecessary handles from being inherited by created child processes, do not create a pipe before it is needed, and be sure to have lab5 main explicitly close any handles that it no longer needs as soon as possible.

- It is important to design lab5 main to enable the 3 child processes to execute concurrently, in parallel. lab5 main should not wait for any of the child processes to complete until all 3 are created and executing concurrently. Pipe objects have a "capacity", and the O/S may block a process trying to write into a pipe until a process reading from the pipe can execute and consume some of the data in the pipe. You can design lab5 main to wait at the end for all 3 children to terminate, or just let lab5 terminate with the 3 children still executing (a scenario which is supported with Win32).

Turn in:

- ZIP file containing the source files and ready-to-run executables for all 4 programs.
  Submit the ZIP using Blackboard file attachment for this lab assignment.
- Printed source code listing turned in during class on the due date.
- Printed Lab summary report:
  - Include instructions for running your lab.
    Also include a brief 1-2 paragraph summary statement that describes how well the lab works, and what you learned in doing the lab.