Operating Systems

University at Albany
Department of Computer Science
Chongqing University of Posts and Telecommunications
Computer Science, International College
ICSI 412

Assignment-3
Assigned: Monday, March 24th, 2025.
Due: To be determined by your co-instructor.

Student Name:

OBJECTIVES

To develop a C program that uses **fork(), exec(),** and **pipe()** process system calls to implement a *full duplex* communication between two processes.

PROBLEM

You are to use any distribution of the **Linux operating system** to create a producer/consumer-based service. The referred service will use two shared **Ordinary Pipes** (Lecture 04 – Interprocess Communication; slides 37-39) to allow full duplex communications between a child/parent pair of processes. Such processes are to be named *consumerProducerParent* and *producerConsumerChild*. These two processes will share information through a full duplex communication structure only. Details regarding the services provided by both *producerConsumerChild and consumerProducerParent* are described as follows:

The *producerConsumerChild* process will

1) open a text file with the name *editSource.txt* and will write all contents of the created file to one pipe to be read by the *consumerProducerParent* process. The *editSource.txt* file is to be created and populated by the user.

The *consumerProducerParent* process will

1) read from the pipe used by the *consumerProducerParent* process to write the contents of file *editSource.txt*, and it will
2) count the number of characters,
3) count the number of words,
4) count the number of lines in the received sentences, and
5) change all uppercase letters to lowercase and place the resulting contents in a new file named *noUpper.txt*.
   It will then
6) create a file named *theCount.txt*,
7) populate *theCount.txt* file with the results obtained in items 2) to 4), and it will

8) write both the file names *theCount.txt* and *noUpper.txt* as well as their full paths on a second pipe to be read by the *producerConsumerChild* process.
9) The *producerConsumerChild* process will also
   a. display the contents of the received file *theCount.txt* on standard output according to the following format
      i. *Number of characters:* number of characters found in *editSource.txt*
      ii. *Number of words:* number of words found in *editSource.txt*
      iii. *Number of lines:* number of lines found in *editSource.txt*
      and
   b. it will also display on the standard output the result of the ***diff*** linux command as invoked with both files *editSource.txt* and *noUpper.txt* as arguments.

You are to create a library to support your producer/consumer process activities. The file "**encDec.h**" is to be created and it must contain the prototypes of all code required to support the producer/consumer activities. The following services must be provided through your library:

a. Read data from a pipe,
b. Write data to a pipe,
c. Count the number of characters,
d. Count the number of words,
e. Count the number of lines, and
f. Change all uppercase letter to lowercase.

All access to the services provided by your library must be done only through the use of any of the members of the exec() family of system calls. The sample code that follows illustrates the use of a service provided by my version of the proposed library.

Code to Illustrate the *encDec.h* File.

encDec.h

```
int toUpper( char *inData );
```

toUpperClient.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
  int pid = fork();
  if( pid == 0 ){
    printf("Child!!!!\n");
    execl("toUpperService", "toUpperService",  "z", NULL );
  }
  else if( pid > 0){
    wait(NULL);
    printf("Parent!!!!\n");
  }
  else
```

```
    printf(" No fork this time! \n");
}
```

toUpperService.c

```
#include <stdio.h>
#include <ctype.h>
#include "encDec.h"

int main(int argc, char *argv[]){
   printf( "%c\n", toUpper( argv[1] ));
}

int toUpper( char *inData ){
  if( (int)*inData >= 'a' && (int)*inData <= 'z' )
    return( toupper( *inData ));
}
```

You may use the following text data as the contents of the *editSource.txt* file for the initial testing of your prototype:

Joke:
McAfee-Question: Is Windows a virus?
No, Windows is not a virus. Here's what viruses do:
1. They replicate quickly-okay, Windows does that.
2. Viruses use up valuable system resources, slowing down the system as they do so-okay, Windows does that.
3. Viruses will, from time to time, trash your hard disk-okay, Windows does that too.
4. Viruses are usually carried, unknown to the user, along with valuable programs and systems. But, Windows does that, too.
5. Viruses will occasionally make the user suspect their system is too slow (see 2.) and the user will buy new hardware. Yup, that's with Windows, too.
Until now it seems Windows is a virus but there are fundamental differences:
Viruses are well supported by their authors, are running on most systems, their program code is fast, compact and efficient and they tend to become more sophisticated as they mature.
So, Windows is not a virus. It's a bug.

WHAT TO SUBMIT

1. Your solution must be placed in a Microsoft compressed (zipped) folder (.zip). Your .zip folder must be named: *412 Assignment-3 Your Name*. Marks will be deducted if you do not follow this requirement. Your solution must be submitted to your co-instructor.

2. You are to use the documentation format provided below to collect all required information about the solution you developed. You are to include, in your documentation document, the source code of all your .c files well as any output produced by your solution that clearly shows:

a) The contents of the file *editSource.txt* and a screenshot of the displayed contents of *theCount.txt* file*,* and the result of the *diff* command.

b) Both pids of the parent and the child processes as well as the file descriptors (fds) of *editSource.txt, theCount.txt*, and *noUpper.txt* files. In essence you are to show the results generated by both your *consumerProducerParent* and *producerConsumerChild* processes.

Your program should be developed using GNU versions of the C compiler. Your solution must use **Ordinary Pipes** (Lecture 04 – Interprocess Communication; slides 37-39). It should be layered, modularized, and well commented. The following is a tentative marking scheme and what is expected to be submitted for this exercise:

1. External Documentation (as many pages necessary to fulfill the requirements listed below.) including the following:

   a. Title page.
   b. A table of contents.
   c. [10%] System documentation.
      i. [5%] A high-level data flow diagram for the system.
      ii. [3%] A list or routines and their brief description.
      iii. [2%] Implementation details.
   d. [15%] Test documentation.
      i. [5%] How you tested your program.
      ii. [10%] A list of your test sets including the results obtained by each set.
   e. [5%] User documentation.
      i. [3%] How to run your program.
      ii. [2%] Describe any parameters (if any).

2. Source Code.

   a. [65%] Correctness.
   b. [5%] Programming style.
      i. [2%] Layering.
      ii. [1%] Readability.
      iii. [1%] Efficiency.
      iv. [1%] Comments.