Dean Cosanella
Flavio dos Santos-Ross
CSE 460
Dr. Tong Lai Yu
Lab 7 Report

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

2. Process Pipes

```
#include <stdio.h>

FILE * popen(const char *command, const char *type);
int  pclose(FILE *stream);
```

Use "man" to study popen() and pclose().

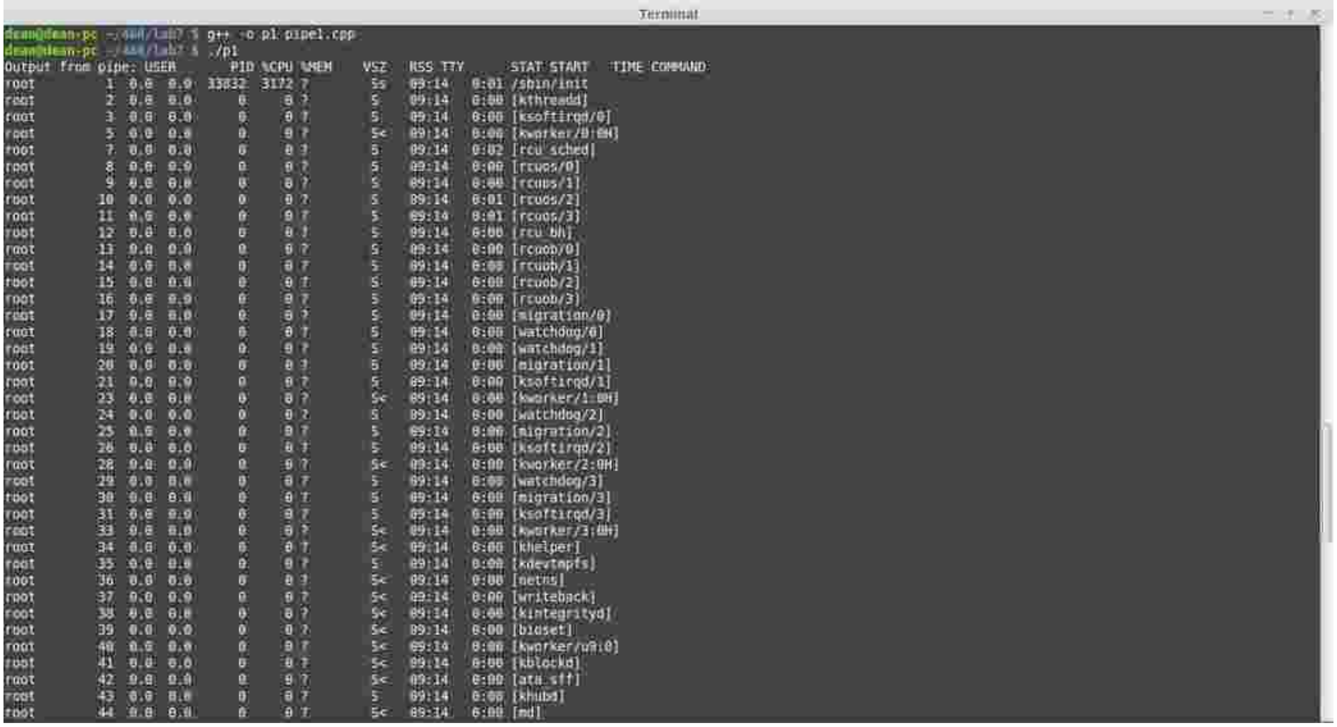
popen()
The popen() function opens a process by creating a pipe, forking, and invoking the shell. Since a pipe is by definition unidirectional, the type argument may specify only reading or writing, not both; the resulting stream is correspondingly read-only or write-only.

pclose()
The pclose() function waits for the associated process to terminate and returns the exit status of the command as returned by wait4(2).

```
//pipe1.cpp
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <iostream>

using namespace std;

int main()
{
  FILE *fpi;                            //for reading a pipe

  char buffer[BUFSIZ+1];                //BUFSIZ defined in <stdio.h>

  int chars_read;
  memset ( buffer, 0,sizeof(buffer));   //clear buffer
  fpi = popen ( "ps -auxw", "r" );      //pipe to command "ps -auxw"
  if ( fpi != NULL ) {
    //read data from pipe into buffer
    chars_read = fread(buffer, sizeof(char), BUFSIZ, fpi );
    if ( chars_read > 0 )
     cout << "Output from pipe: " << buffer << endl;
    pclose ( fpi );                     //close the pipe
    return 0;
  }

  return 1;
}
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

When we compile pipe1.cpp, we get the following:



```
dean@dean-pc ~/460/lab7 $ g++ -o p1 pipe1.cpp
dean@dean-pc ~/460/lab7 $ ./p1
Output from pipe: USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  33832  3172 ?        Ss   09:14   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    09:14   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    09:14   0:02 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuos/0]
root         9  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuos/1]
root        10  0.0  0.0      0     0 ?        S    09:14   0:01 [rcuos/2]
root        11  0.0  0.0      0     0 ?        S    09:14   0:01 [rcuos/3]
root        12  0.0  0.0      0     0 ?        S    09:14   0:00 [rcu_bh]
root        13  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/0]
root        14  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/1]
root        15  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/2]
root        16  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/3]
root        17  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/0]
root        18  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/0]
root        19  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/1]
root        20  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/1]
root        21  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/1]
root        23  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/1:0H]
root        24  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/2]
root        25  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/2]
root        26  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/2]
root        28  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/2:0H]
root        29  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/3]
root        30  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/3]
root        31  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/3]
root        33  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/3:0H]
root        34  0.0  0.0      0     0 ?        S<   09:14   0:00 [khelper]
root        35  0.0  0.0      0     0 ?        S    09:14   0:00 [kdevtmpfs]
root        36  0.0  0.0      0     0 ?        S<   09:14   0:00 [netns]
root        37  0.0  0.0      0     0 ?        S<   09:14   0:00 [writeback]
root        38  0.0  0.0      0     0 ?        S<   09:14   0:00 [kintegrityd]
root        39  0.0  0.0      0     0 ?        S<   09:14   0:00 [bioset]
root        40  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/u9:0]
root        41  0.0  0.0      0     0 ?        S<   09:14   0:00 [kblockd]
root        42  0.0  0.0      0     0 ?        S<   09:14   0:00 [ata_sff]
root        43  0.0  0.0      0     0 ?        S    09:14   0:00 [khubd]
root        44  0.0  0.0      0     0 ?        S<   09:14   0:00 [md]
```

We believe that this output is showing the information of all the pipes currently on the computer. We think that the pipes that have a PID that is not 0 are associated with commands that are connected to that particular pipe and are currently running on the computer. The reason why these pipes would have a PID greater than 0 is because it is a child process which always have a nonzero PID.

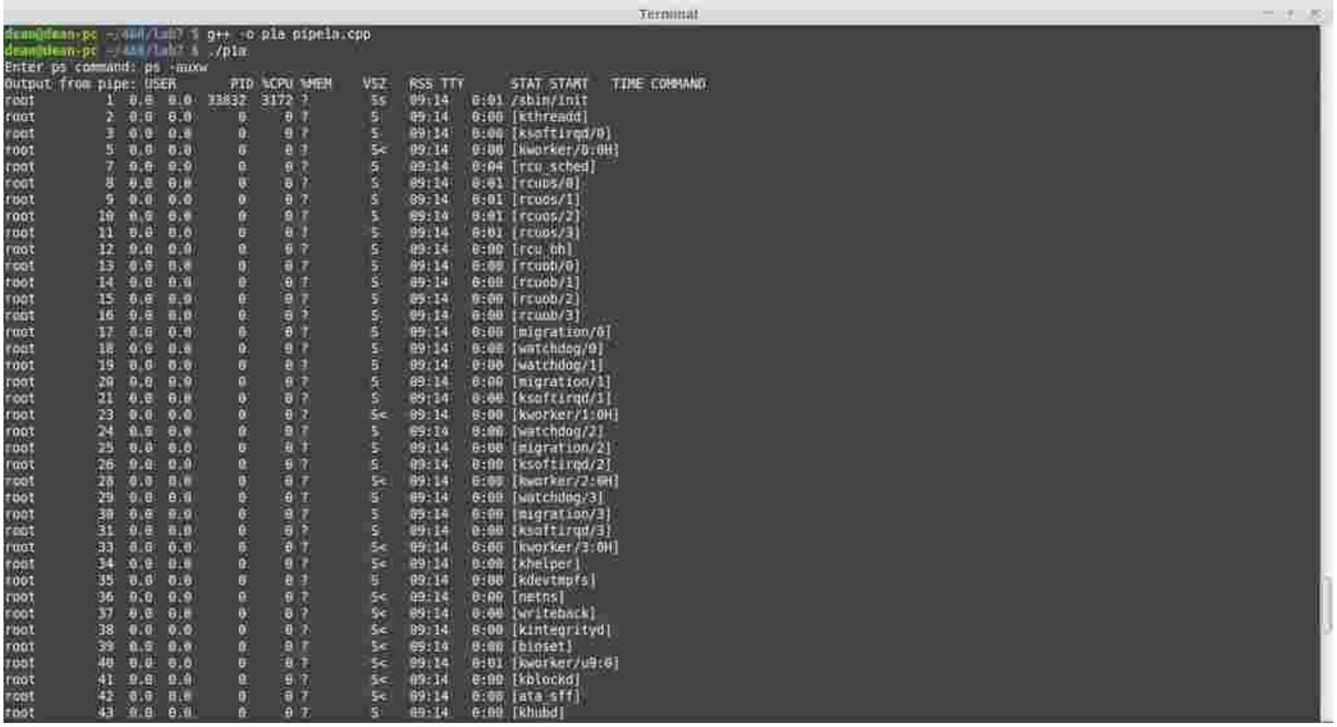
Modification of pipe1.cpp. New program is called pipe1a.cpp:

```
//pipe1a.cpp
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <iostream>

using namespace std;

int main()
{
  FILE *fpi;                            //for reading a pipe

  char buffer[BUFSIZ+1];                //BUFSIZ defined in <stdio.h>
  char s[50];
  char r[50];

  int chars_read;
  memset ( buffer, 0,sizeof(buffer));   //clear buffer
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
  cout << "Enter ps command: ";
  cin.getline(s, 50);
  strcpy(r, s);

  fpi = popen ( r, "r" );
  if ( fpi != NULL ) {
       //read data from pipe into buffer
    chars_read = fread(buffer, sizeof(char), BUFSIZ, fpi );
    if ( chars_read > 0 )
      cout << "Output from pipe: " << buffer << endl;
    pclose ( fpi );                          //close the pipe
    return 0;
  }

  return 1;
}
```

Output of pipe1a.cpp:



```
dean@dean-pc ~/460/lab7 $ g++ -o p1a pipe1a.cpp
dean@dean-pc ~/460/lab7 $ ./p1a
Enter ps command: ps -auxw
Output from pipe: USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  33832  3172 ?        Ss   09:14   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    09:14   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    09:14   0:04 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    09:14   0:01 [rcuos/0]
root         9  0.0  0.0      0     0 ?        S    09:14   0:01 [rcuos/1]
root        10  0.0  0.0      0     0 ?        S    09:14   0:01 [rcuos/2]
root        11  0.0  0.0      0     0 ?        S    09:14   0:01 [rcuos/3]
root        12  0.0  0.0      0     0 ?        S    09:14   0:00 [rcu_bh]
root        13  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/0]
root        14  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/1]
root        15  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/2]
root        16  0.0  0.0      0     0 ?        S    09:14   0:00 [rcuob/3]
root        17  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/0]
root        18  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/0]
root        19  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/1]
root        20  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/1]
root        21  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/1]
root        23  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/1:0H]
root        24  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/2]
root        25  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/2]
root        26  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/2]
root        28  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/2:0H]
root        29  0.0  0.0      0     0 ?        S    09:14   0:00 [watchdog/3]
root        30  0.0  0.0      0     0 ?        S    09:14   0:00 [migration/3]
root        31  0.0  0.0      0     0 ?        S    09:14   0:00 [ksoftirqd/3]
root        33  0.0  0.0      0     0 ?        S<   09:14   0:00 [kworker/3:0H]
root        34  0.0  0.0      0     0 ?        S<   09:14   0:00 [khelper]
root        35  0.0  0.0      0     0 ?        S    09:14   0:00 [kdevtmpfs]
root        36  0.0  0.0      0     0 ?        S<   09:14   0:00 [netns]
root        37  0.0  0.0      0     0 ?        S<   09:14   0:00 [writeback]
root        38  0.0  0.0      0     0 ?        S<   09:14   0:00 [kintegrityd]
root        39  0.0  0.0      0     0 ?        S<   09:14   0:00 [bioset]
root        40  0.0  0.0      0     0 ?        S<   09:14   0:01 [kworker/u9:0]
root        41  0.0  0.0      0     0 ?        S<   09:14   0:00 [kblockd]
root        42  0.0  0.0      0     0 ?        S<   09:14   0:00 [ata_sff]
root        43  0.0  0.0      0     0 ?        S    09:14   0:00 [khubd]
```

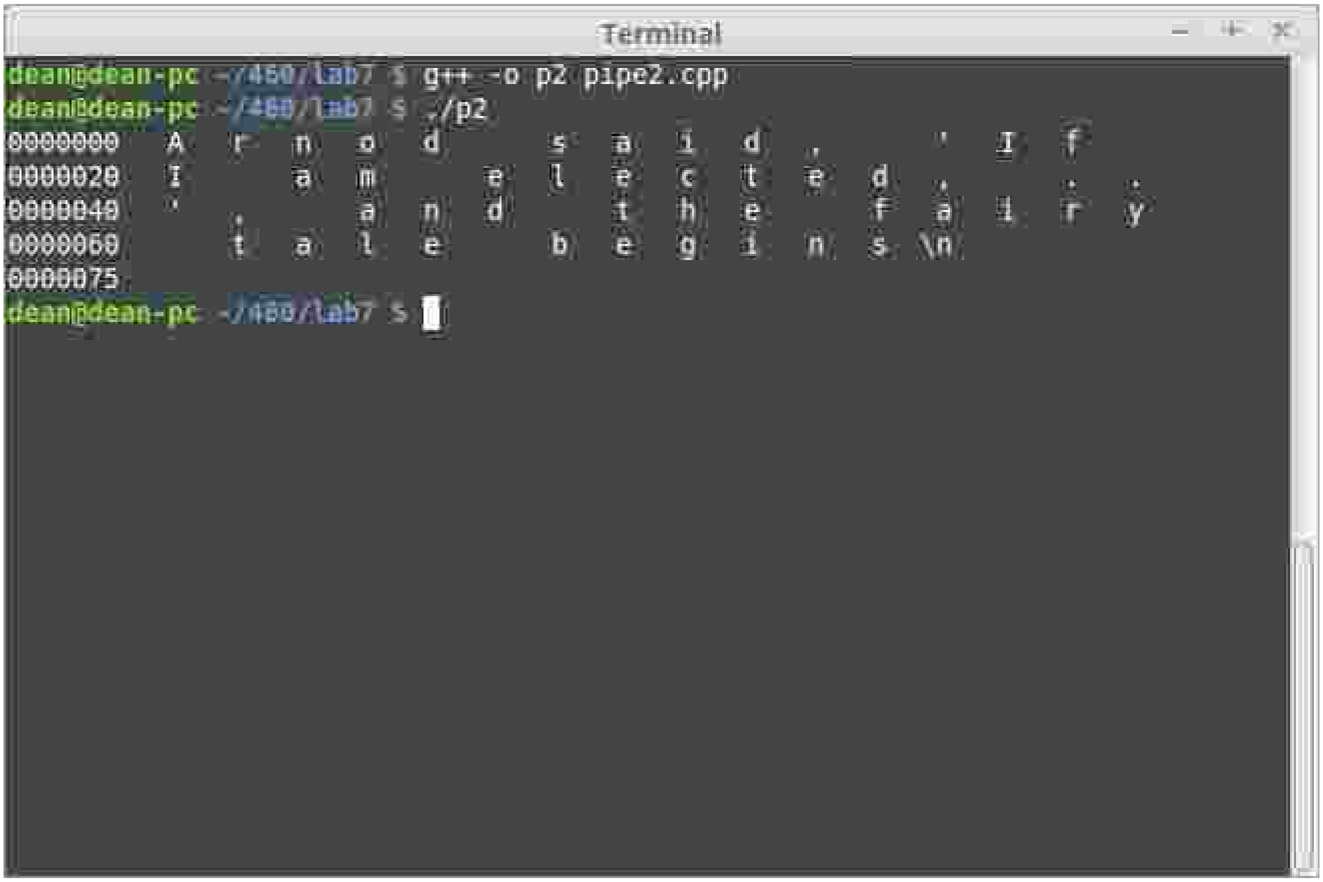
pipe2.cpp code:

```
//pipe2.cpp
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <iostream>
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
using namespace std;

int main()
{
  FILE *fpo;                                //for writing to a pipe

  char buffer[BUFSIZ+1];                    //BUFSIZ defined in <stdio.h>

  //Write buffer a message
  sprintf(buffer, "Arnod said, 'If I am elected, ..', and the fairy tale
begins\n");

  fpo = popen ( "od -c", "w" ); //pipe to command "od -c"
                                            //od -- output dump, see "man od"
  if ( fpo != NULL ) {
    //send data from buffer to pipe
    fwrite(buffer, sizeof(char), strlen(buffer), fpo );
    pclose ( fpo );                         //close the pipe
    return 0;
  }
  return 1;
}
```

output for pipe2.cpp:



```
dean@dean-pc ~/460/lab7 $ g++ -o p2 pipe2.cpp
dean@dean-pc ~/460/lab7 $ ./p2
0000000   A   r   n   o   d       s   a   i   d   ,       '   I   f    
0000020   I       a   m       e   l   e   c   t   e   d   ,       .   .
0000040   '   ,       a   n   d       t   h   e       f   a   i   r   y
0000060       t   a   l   e       b   e   g   i   n   s  \n
0000075
dean@dean-pc ~/460/lab7 $ 
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

The output for pipe2.cpp looks this way because the “od -c” command selects ASCII characters or backslash escapes and prints it out like this. The program takes the buffer that was passed into fpo and runs the “od -c” command.

Modification of pipe2.cpp:
To modify pipe2.cpp to output the first three words in reverse order, we changed the following line:

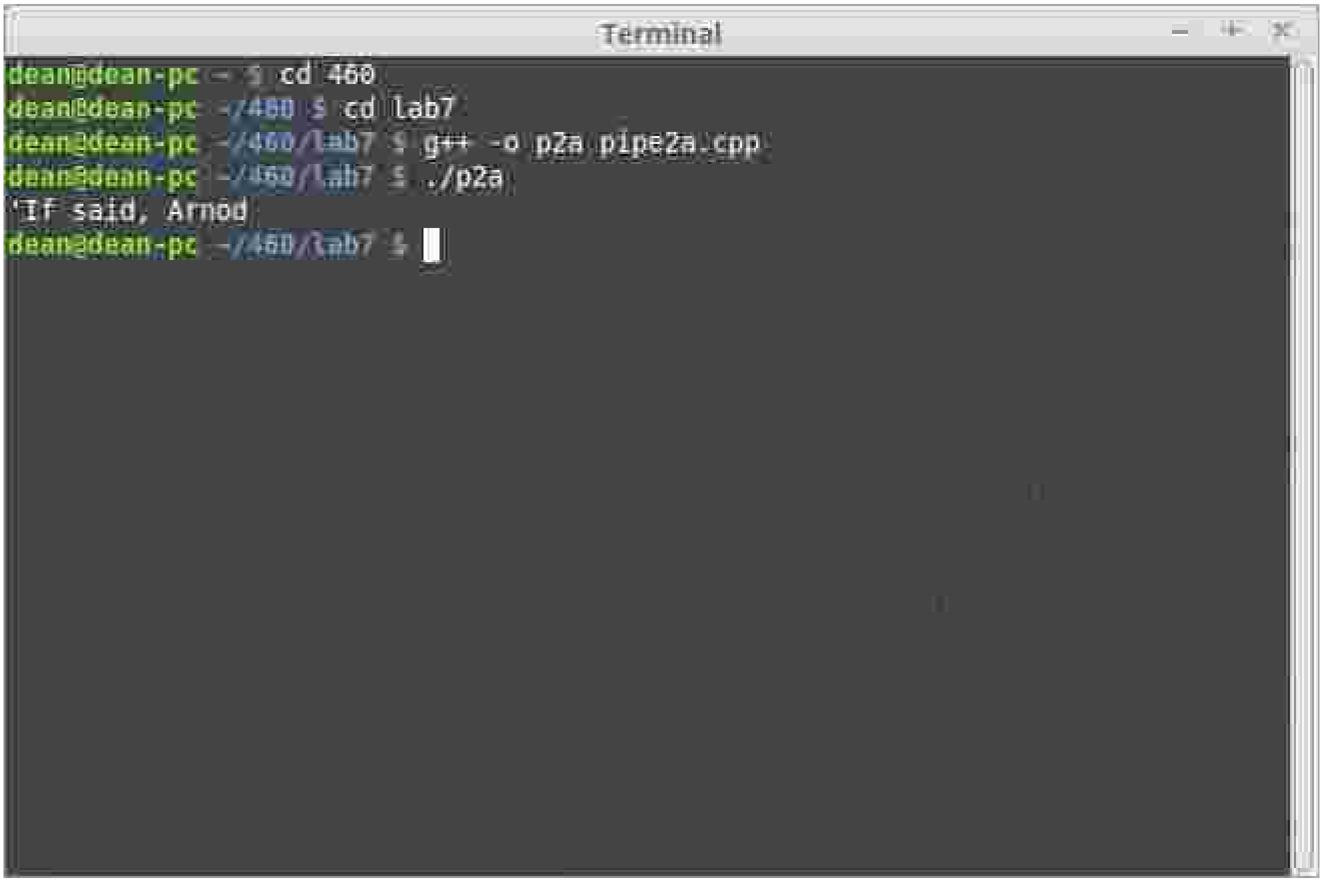
```
fpo = popen ( "od -c", "w" );
```

to this:

```
fpo = popen ( "awk '{print $3, $2, $1}'", "w" );
```

and we also changed the name of the program to pipe2a.cpp.

Output for pipe2a.cpp:



```
dean@dean-pc ~ $ cd 460
dean@dean-pc ~/460 $ cd lab7
dean@dean-pc ~/460/lab7 $ g++ -o p2a pipe2a.cpp
dean@dean-pc ~/460/lab7 $ ./p2a
'If said, Arnod
dean@dean-pc ~/460/lab7 $ 
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

3. The pipe Call

The lower-level pipe() function provides a means of passing data between two processes, without the overhead of invoking a shell to interpret the requested command.

```
#include <unistd.h>

int pipe ( int fd[2] );
```

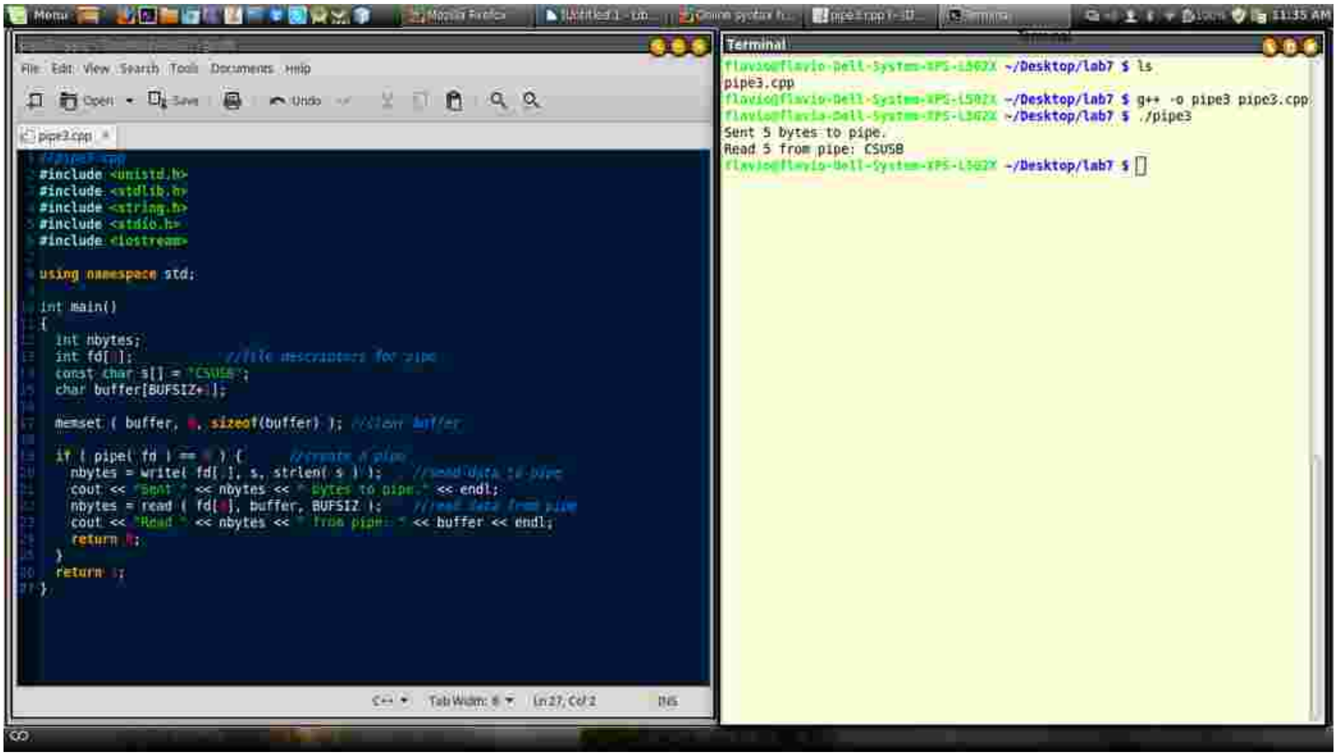
Here is an example of a program that illustrates an array address of two integer file descriptors; it fills the array with two new file descriptors and returns 0 if successful. The two file descriptors returned are connected in a special way. Any data written to *fd[1]* can be read back from *fd[0]*. The data are processed on a first in, first out (FIFO) basis.

Code:

```
//pipe3.cpp
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <iostream>

using namespace std;

int main()
{
  int nbytes;
  int fd[2];            //file descriptors for pipe
  const char s[] = "CSUSB";
  char buffer[BUFSIZ+1];

  memset ( buffer, 0, sizeof(buffer) ); //clear buffer

  if ( pipe( fd ) == 0 ) {      //create a pipe
    nbytes = write( fd[1], s, strlen( s ) );    //send data to pipe
    cout << "Sent " << nbytes << " bytes to pipe." << endl;
    nbytes = read ( fd[0], buffer, BUFSIZ );    //read data from pipe
    cout << "Read " << nbytes << " from pipe: " << buffer << endl;
    return 0;
  }
  return 1;
}
```

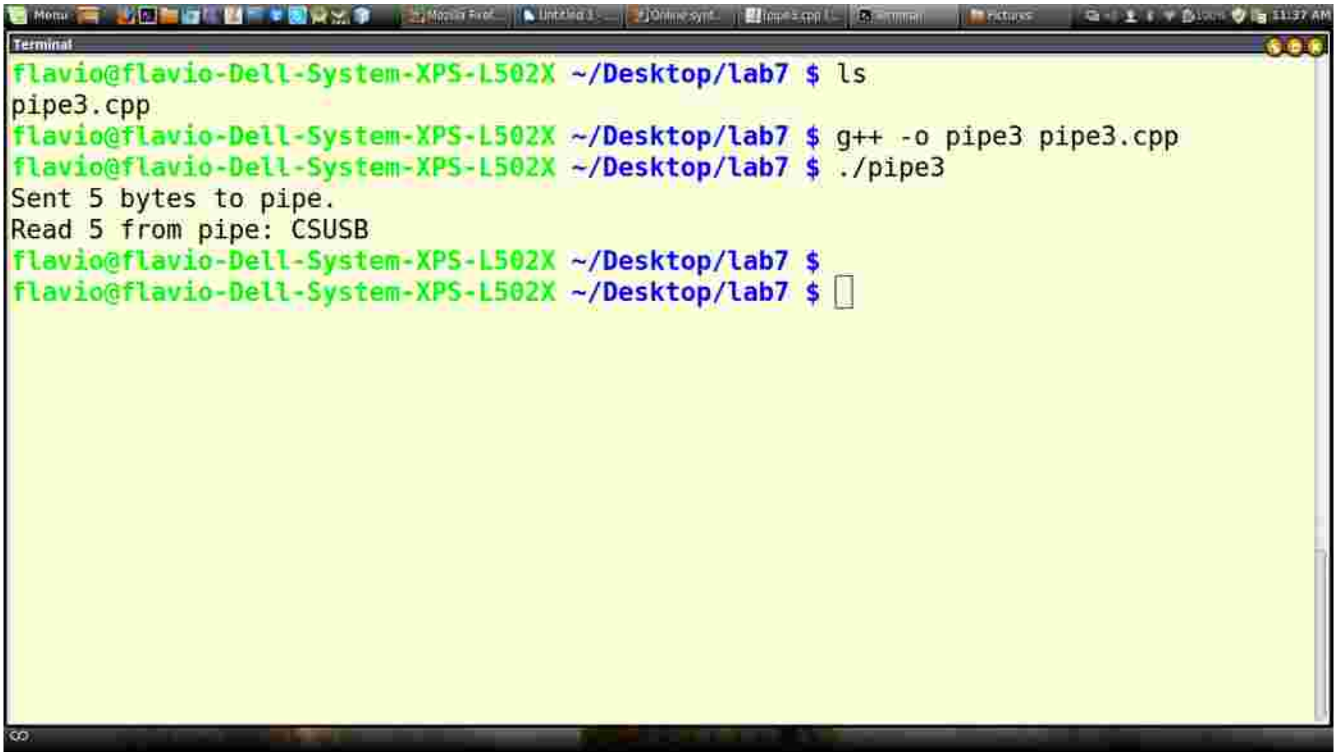
Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Here is the compilation and output:



```
//pipe3.cpp
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <iostream>

using namespace std;

int main()
{
  int nbytes;
  int fd[2];               //file descriptors for pipe
  const char s[] = "CSUSB";
  char buffer[BUFSIZ+1];

  memset ( buffer, 0, sizeof(buffer) ); //clear buffer

  if ( pipe( fd ) == 0 ) {       //create a pipe
    nbytes = write( fd[1], s, strlen( s ) );    //send data to pipe
    cout << "Sent " << nbytes << " bytes to pipe." << endl;
    nbytes = read ( fd[0], buffer, BUFSIZ );    //read data from pipe
    cout << "Read " << nbytes << " from pipe: " << buffer << endl;
    return 0;
  }
  return 1;
}
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
pipe3.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o pipe3 pipe3.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./pipe3
Sent 5 bytes to pipe.
Read 5 from pipe: CSUSB
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```



```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
pipe3.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o pipe3 pipe3.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./pipe3
Sent 5 bytes to pipe.
Read 5 from pipe: CSUSB
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Since a pipe is a connection between two processes, the standard output from one process becomes the standard input of the other process. Therefore, it is possible to have a series of processes arranged in a a pipeline, with a pipe between each pair of processes in the series, and that's exactly what pipe3 does, it creates a pipe and gets back the input of the other process, in this case from the same pipe.

4. Parent and Child Processes

We can use **exec()** to create a child process running a different program. After an **exec** call, the old process has been replaced by the new child process. The following two programs demonstrate the concept.

The first is the "data producer", which creates the pipe and then invokes the child, the "data consumer".

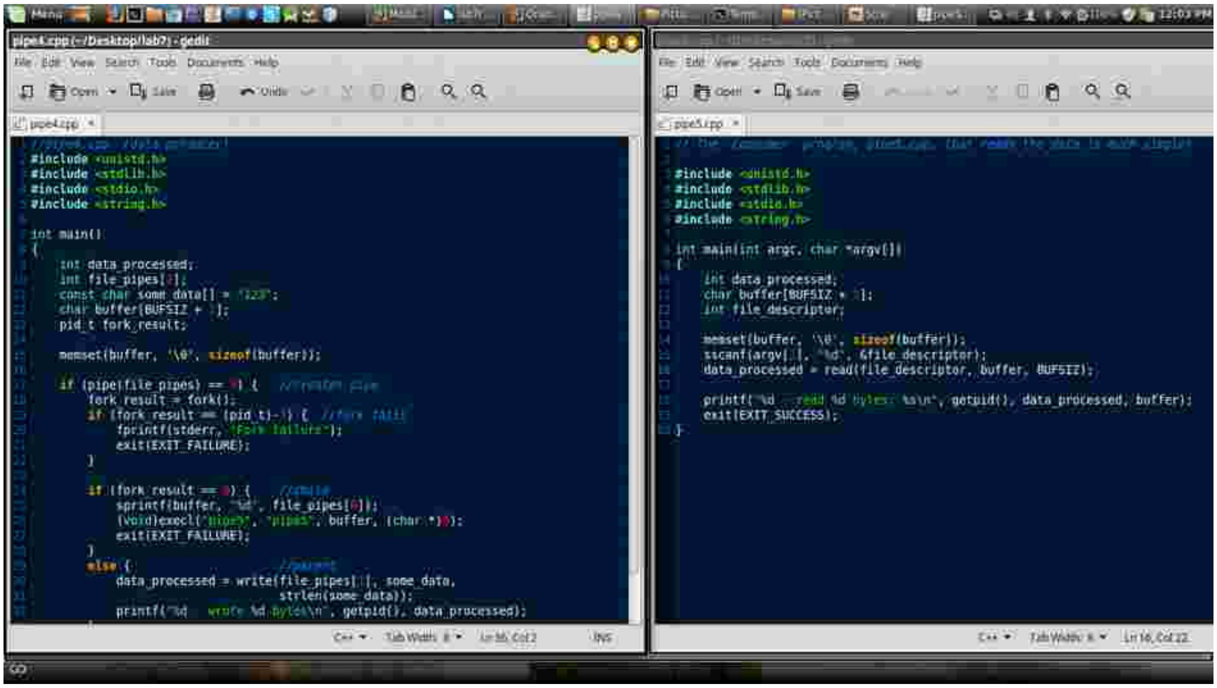
```
//pipe4.cpp
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main()
{
    int data_processed;
    int file_pipes[2];
    const char some_data[] = "123";
    char buffer[BUFSIZ + 1];
    pid_t fork_result;

    memset(buffer, '\0', sizeof(buffer));

    if (pipe(file_pipes) == 0) {   //creates pipe
        fork_result = fork();
        if (fork_result == (pid_t)-1) {  //fork fails
            fprintf(stderr, "Fork failure");
            exit(EXIT_FAILURE);
        }

        if (fork_result == 0) {    //child
            sprintf(buffer, "%d", file_pipes[0]);
            (void)execl("pipe5", "pipe5", buffer, (char *)0);
            exit(EXIT_FAILURE);
        }
        else {                     //parent
            data_processed = write(file_pipes[1], some_data,
                                   strlen(some_data));
            printf("%d - wrote %d bytes\n", getpid(), data_processed);
        }
    }
    exit(EXIT_SUCCESS);
}
```

The 'consumer' program, pipe5.cpp, that reads the data is much simpler.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int data_processed;
    char buffer[BUFSIZ + 1];
    int file_descriptor;

    memset(buffer, '\0', sizeof(buffer));
    sscanf(argv[1], "%d", &file_descriptor);
    data_processed = read(file_descriptor, buffer, BUFSIZ);

    printf("%d - read %d bytes: %s\n", getpid(), data_processed, buffer);
    exit(EXIT_SUCCESS);
}
```

now compiling:

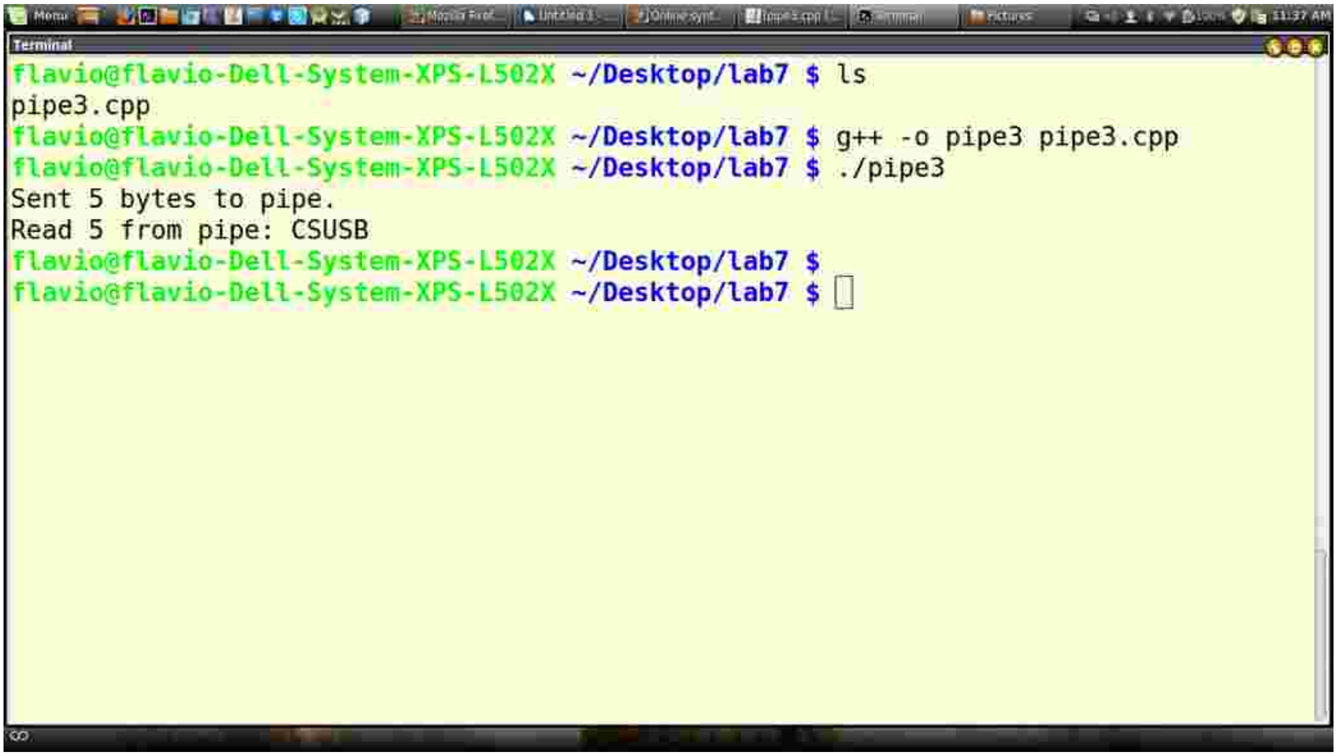
```
$ g++ -o pipe4 pipe4.cpp
$ g++ -o pipe5 pipe5.cpp
$ ./pipe4
```


pipe4.cpp (~/Desktop/lab7) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
pipe4.cpp
C++ Tab Width: 8 Ln 36, Col 2 INS
File Edit View Search Tools Documents Help
Open Save
pipe5.cpp
C++ Tab Width: 8 Ln 16, Col 22

```
//pipe4.cpp //data producer
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main()
{
    int data_processed;
    int file_pipes[2];
    const char some_data[] = "123";
    char buffer[BUFSIZ + 1];
    pid_t fork_result;

    memset(buffer, '\0', sizeof(buffer));

    if (pipe(file_pipes) == 0) {    //creates pipe
        fork_result = fork();
        if (fork_result == (pid_t)-1) {  //fork fails
            fprintf(stderr, "Fork failure");
            exit(EXIT_FAILURE);
        }

        if (fork_result == 0) {     //child
            sprintf(buffer, "%d", file_pipes[0]);
            (void)execl("pipe5", "pipe5", buffer, (char *)0);
            exit(EXIT_FAILURE);
        }
        else {                      //parent
            data_processed = write(file_pipes[1], some_data,
                                   strlen(some_data));
            printf("%d - wrote %d bytes\n", getpid(), data_processed);
```

```
// the 'consumer' program, pipe5.cpp, that reads the data is much simpler

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int data_processed;
    char buffer[BUFSIZ + 1];
    int file_descriptor;

    memset(buffer, '\0', sizeof(buffer));
    sscanf(argv[1], "%d", &file_descriptor);
    data_processed = read(file_descriptor, buffer, BUFSIZ);

    printf("%d - read %d bytes: %s\n", getpid(), data_processed, buffer);
    exit(EXIT_SUCCESS);
}
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460



```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
pipe3.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o pipe3 pipe3.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./pipe3
Sent 5 bytes to pipe.
Read 5 from pipe: CSUSB
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

Here is the writing code to read a msg:

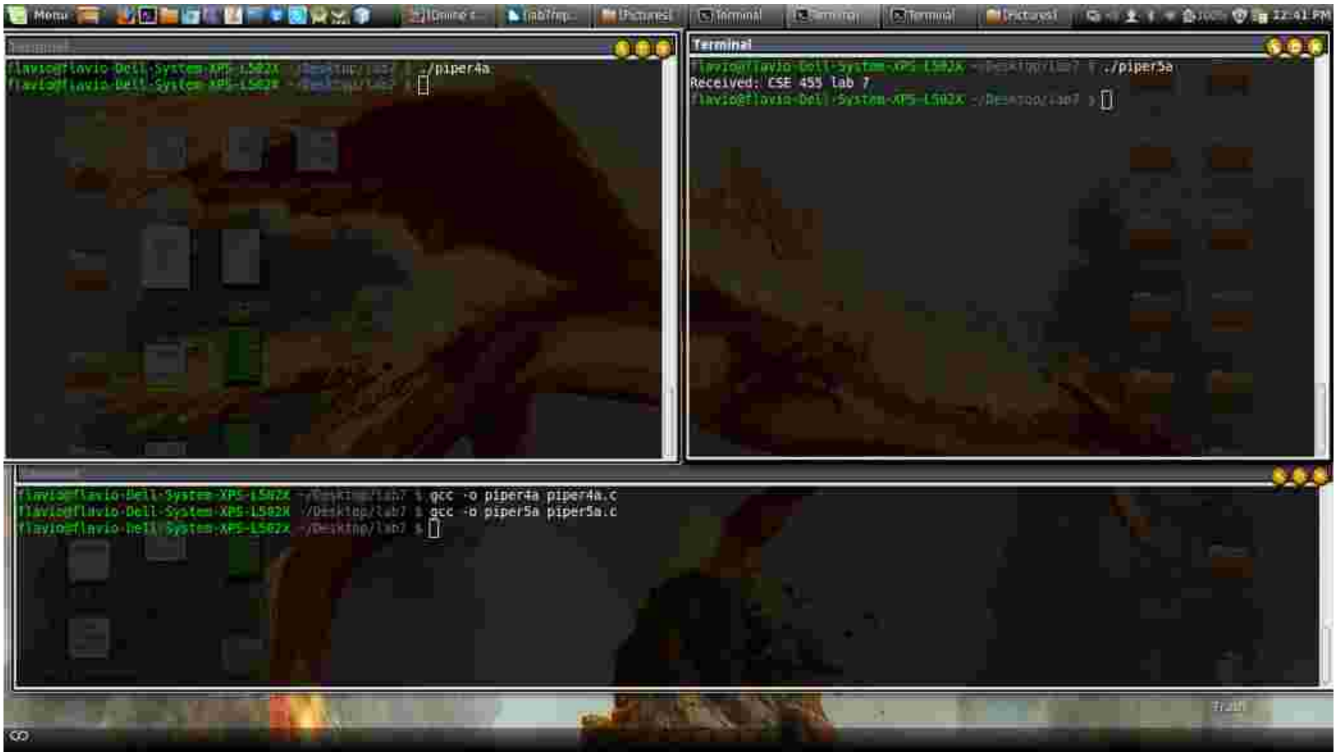
```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int fd;
    char * myfifo = "/tmp/myfifo";

    /* create the FIFO (named pipe) */
    mkfifo(myfifo, 0666);

    /* write "Hi" to the FIFO */
    fd = open(myfifo, O_WRONLY);
    write(fd, "CSE 455 lab 7", sizeof("CSE 455 lab 7"));
    close(fd);

    /* remove the FIFO */
    unlink(myfifo);

    return 0;
}
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

here is the reading code:

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>

#define MAX_BUF 1024

int main()
{
    int fd;
    char * myfifo = "/tmp/myfifo";
    char buf[MAX_BUF];

    /* open, read, and display the message from the FIFO */
    fd = open(myfifo, O_RDONLY);
    read(fd, buf, MAX_BUF);
    printf("Received: %s\n", buf);
    close(fd);

    return 0;
}
```

here is the output:



```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./piper4a
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./piper5a
Received: CSE 455 lab 7
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ gcc -o piper4a piper4a.c
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ gcc -o piper5a piper5a.c
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

5. Special Pipes

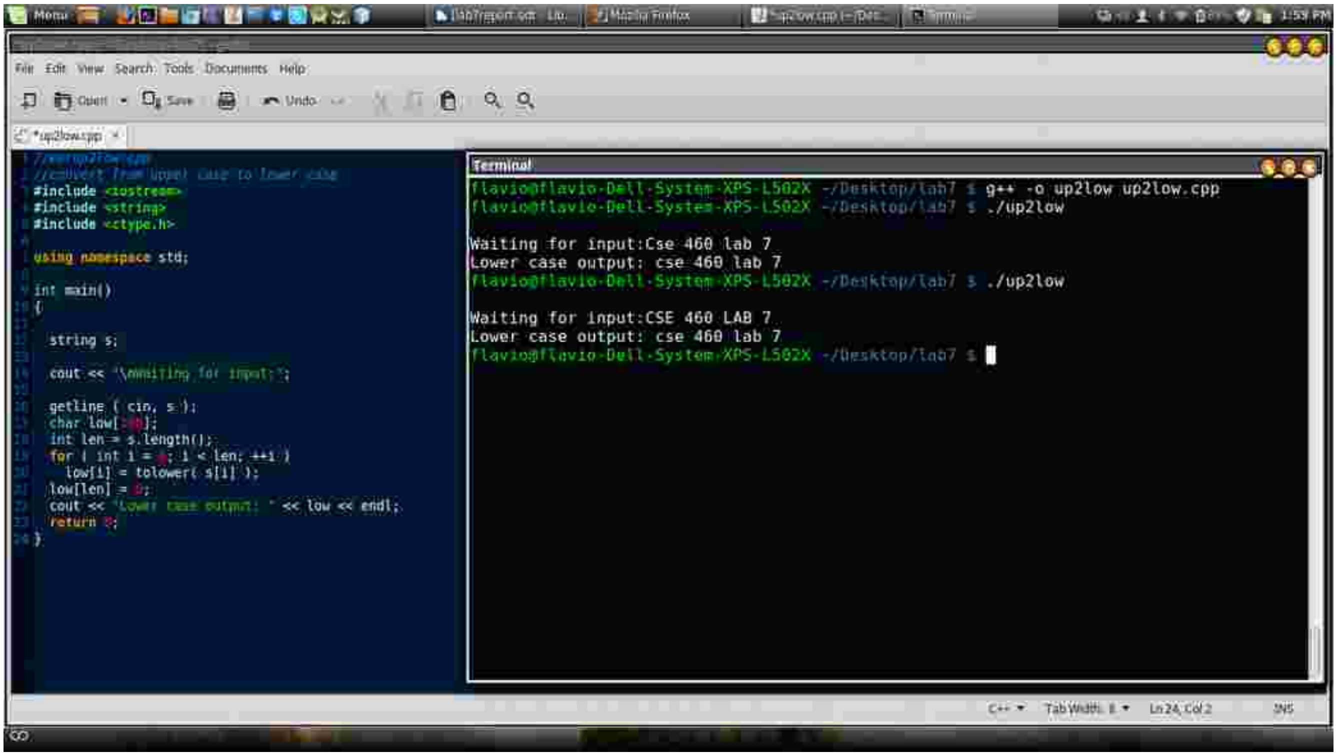
A read on a pipe that is not open for writing returns zero, rather than blocking. Note that this is not the same as reading an invalid file descriptor, which **read** considers an error, returning -1. If we use a pipe across a **fork** call, there are two different file descriptors that we can use to write to the pipe, one in the parent and one in the child. We have to close the write file descriptors of the pipe in both parent and child processes. The **dup**() system call creates a copy of the file descriptor *oldfd*, using the lowest-numbered unused descriptor for the new descriptor. The **dup2**() system call performs the same task as **dup**(), except that it uses the descriptor number specified in *newfd* instead of the lowest-numbered unused descriptor. If the descriptor *newfd* was previously open, it is silently closed before being reused.

Code:

```
//up2low.cpp
//convert from upper case to lower case
#include <iostream>
#include <string>
#include <ctype.h>

using namespace std;

int main()
{

  string s;

  cout << "\nWaiting for input:";

  getline ( cin, s );
  char low[100];
  int len = s.length();
  for ( int i = 0; i < len; ++i )
    low[i] = tolower( s[i] );
  low[len] = 0;
  cout << "Lower case output: " << low << endl;
  return 0;
}
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

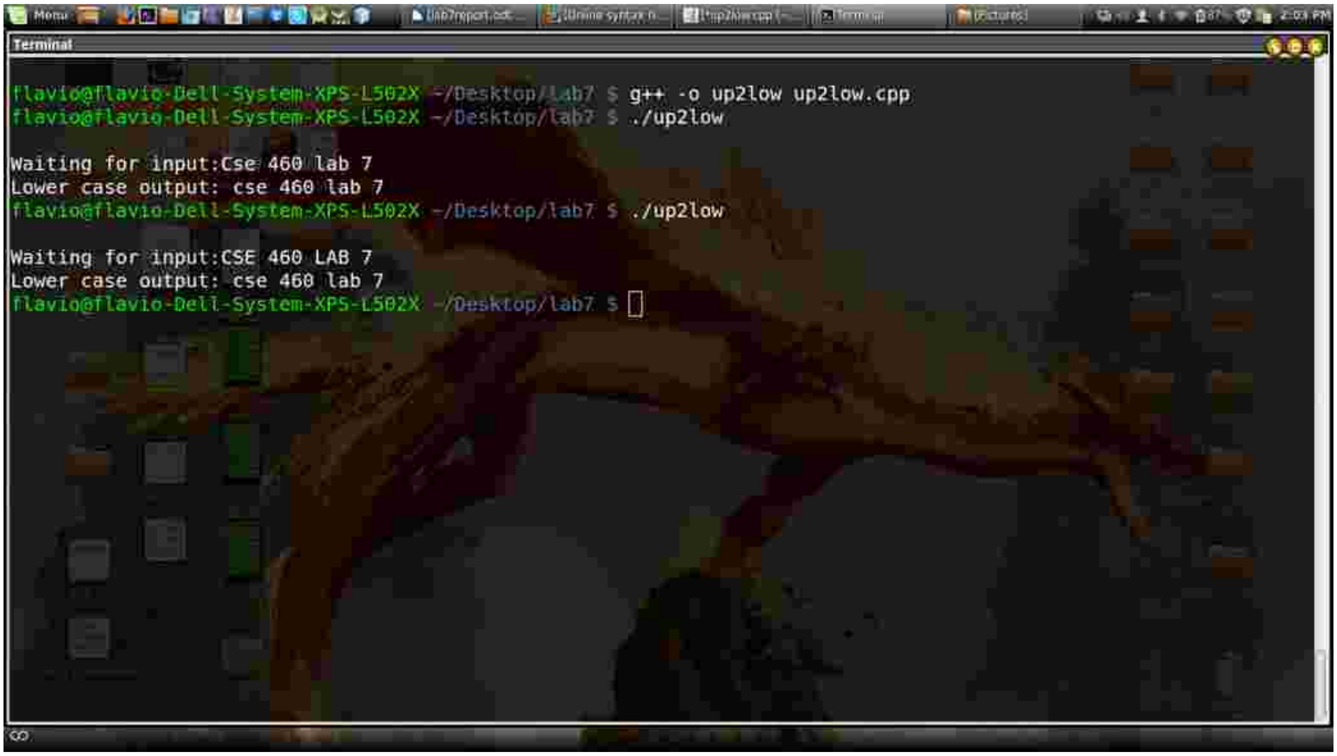
Program compiles and executes:



File Edit View Search Tools Documents Help
Open Save Undo
*up2low.cpp
Terminal
C++ Tab Width: 8 Ln 24, Col 2 INS

```
//up2low.cpp
//convert from upper case to lower case
#include <iostream>
#include <string>
#include <ctype.h>

using namespace std;

int main()
{

  string s;

  cout << "\nWaiting for input:";

  getline ( cin, s );
  char low[80];
  int len = s.length();
  for ( int i = 0; i < len; ++i )
    low[i] = tolower( s[i] );
  low[len] = 0;
  cout << "Lower case output: " << low << endl;
  return 0;
}
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o up2low up2low.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./up2low

Waiting for input:Cse 460 lab 7
Lower case output: cse 460 lab 7
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./up2low

Waiting for input:CSE 460 LAB 7
Lower case output: cse 460 lab 7
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```



```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o up2low up2low.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./up2low

Waiting for input:Cse 460 lab 7
Lower case output: cse 460 lab 7
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./up2low

Waiting for input:CSE 460 LAB 7
Lower case output: cse 460 lab 7
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

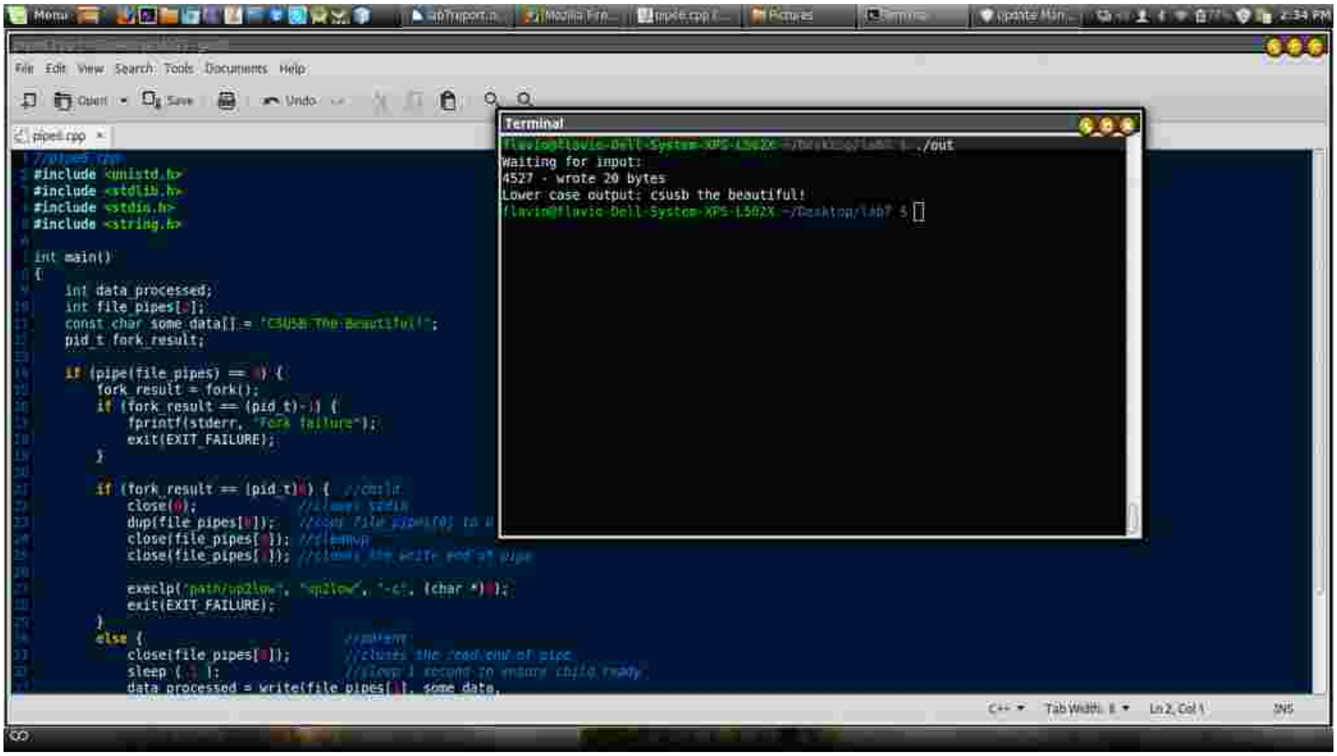
The program above display any input into “lower case”

Pipe6 script code:

```
//pipe6.cpp
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main()
{
    int data_processed;
    int file_pipes[2];
    const char some_data[] = "CSUSB The Beautiful!";
    pid_t fork_result;

    if (pipe(file_pipes) == 0) {
        fork_result = fork();
        if (fork_result == (pid_t)-1) {
            fprintf(stderr, "Fork failure");
            exit(EXIT_FAILURE);
        }

        if (fork_result == (pid_t)0) {  //child
            close(0);              //closes stdin
            dup(file_pipes[0]);    //copy file_pipes[0] to 0
            close(file_pipes[0]); //cleanup
            close(file_pipes[1]); //closes the write end of pipe

            execlp("path/up2low", "up2low", "-c", (char *)0);
            exit(EXIT_FAILURE);
        }
        else {                          //parent
            close(file_pipes[0]);       //closes the read end of pipe
            sleep ( 1 );                //sleep 1 second to ensure child ready
            data_processed = write(file_pipes[1], some_data,
                                       strlen(some_data));
            close(file_pipes[1]);
            printf("\nProcess %d wrote %d bytes\n", (int)getpid(), data_processed);
        }
    }
    exit(EXIT_SUCCESS);
}
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Compiling an executing:



```
//pipe6.cpp
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main()
{
    int data_processed;
    int file_pipes[2];
    const char some_data[] = "CSUSB The Beautiful!";
    pid_t fork_result;

    if (pipe(file_pipes) == 0) {
        fork_result = fork();
        if (fork_result == (pid_t)-1) {
            fprintf(stderr, "Fork failure");
            exit(EXIT_FAILURE);
        }

        if (fork_result == (pid_t)0) {  //child
            close(0);                   //closes stdin
            dup(file_pipes[0]);   //copy file_pipes[0] to s
            close(file_pipes[0]); //cleanup
            close(file_pipes[1]); //closes the write end of pipe

            execlp("path/up2low", "up2low", "-c", (char *)0);
            exit(EXIT_FAILURE);
        }
        else {                          //parent
            close(file_pipes[0]);       //closes the read end of pipe
            sleep ( 1 );                //sleep 1 second to ensure child ready
            data_processed = write(file_pipes[1], some_data,
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./out
Waiting for input:
4527 - wrote 20 bytes
Lower case output: csusb the beautiful!
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $
```

Modify pipe6.cpp so that the child process executes a program written by you rather than **up2low:**

pipe6.cpp code modified:

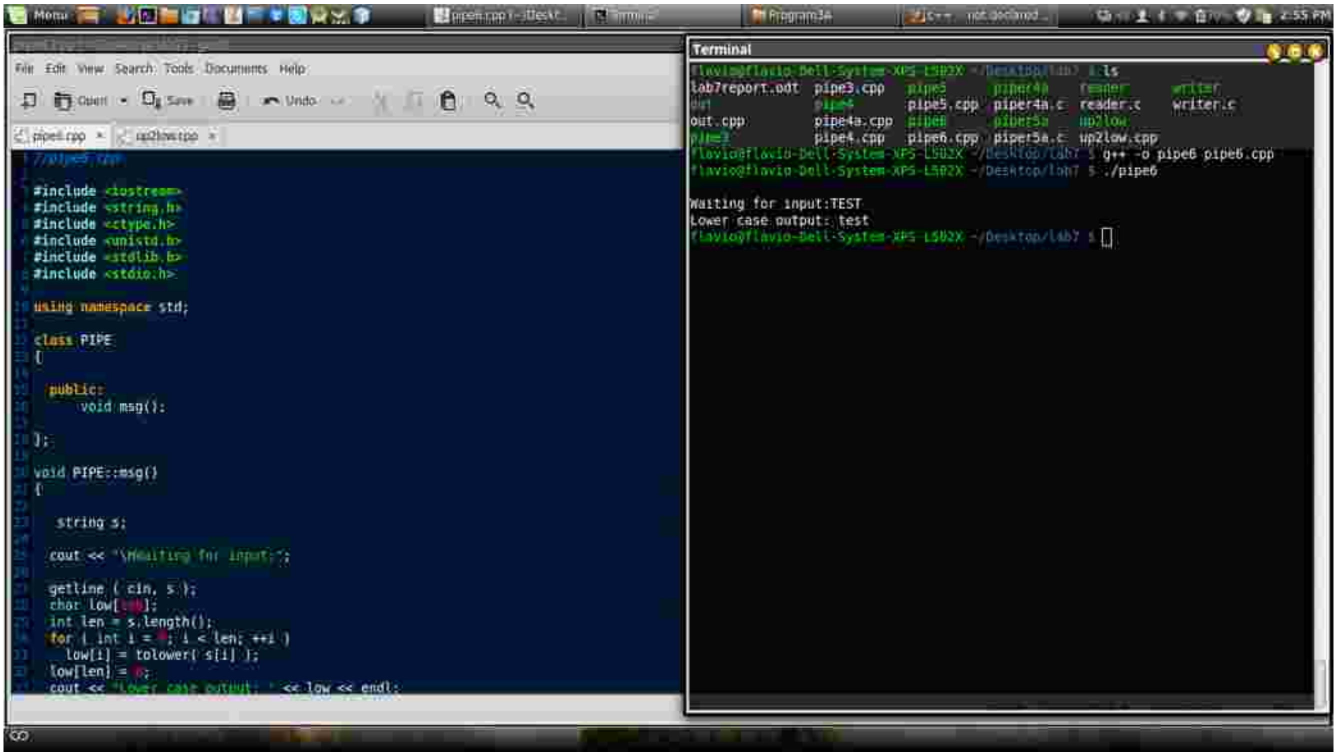
We found easier to create a class called PIPE (random name) and create a function “msg” which executes “up2low” script:

```
//pipe6.cpp
#include <iostream>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

using namespace std;

class PIPE
{

  public:
      void msg();
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
};

void PIPE::msg()
{

   string s;

  cout << "\nWaiting for input:";

  getline ( cin, s );
  char low[100];
  int len = s.length();
  for ( int i = 0; i < len; ++i )
    low[i] = tolower( s[i] );
  low[len] = 0;
  cout << "Lower case output: " << low << endl;
}


int main()
{
    PIPE up2low;

    int data_processed;
    int file_pipes[2];
    const char some_data[] = "CSUSB The Beautiful!";
    pid_t fork_result;

    up2low.msg();

    if (pipe(file_pipes) == 0) {
        fork_result = fork();
        if (fork_result == (pid_t)-1) {
            fprintf(stderr, "Fork failure");
            exit(EXIT_FAILURE);
        }

        if (fork_result == (pid_t)0) {  //child
            close(0);             //closes stdin
            dup(file_pipes[0]);   //copy file_pipes[0] to 0
            close(file_pipes[0]); //cleanup
            close(file_pipes[1]); //closes the write end of pipe

            execlp("path/up2low", "up2low", "-c", (char *)0);
            exit(EXIT_FAILURE);
        }
        else {                          //parent
            close(file_pipes[0]);       //closes the read end of pipe
            sleep ( 1 );                //sleep 1 second to ensure child ready
            data_processed = write(file_pipes[1], some_data,
                                   strlen(some_data));
            close(file_pipes[1]);
            printf("\nProcess %d wrote %d bytes\n", (int)getpid(), data_processed);
        }
    }

 exit(EXIT_SUCCESS);
}
```

After compiling and executing the program does what it suppose to do.



```
//pipe6.cpp

#include <iostream>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

using namespace std;

class PIPE
{

  public:
      void msg();

};

void PIPE::msg()
{

   string s;

  cout << "\nWaiting for input:";

  getline ( cin, s );
  char low[100];
  int len = s.length();
  for ( int i = 0; i < len; ++i )
    low[i] = tolower( s[i] );
  low[len] = 0;
  cout << "Lower case output: " << low << endl;
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
lab7report.odt  pipe3.cpp   pipe5      piper4a    reader     writer
out             pipe4       pipe5.cpp  piper4a.c  reader.c   writer.c
out.cpp         pipe4a.cpp  pipe6      piper5a    up2low
pipe3           pipe4.cpp   pipe6.cpp  piper5a.c  up2low.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o pipe6 pipe6.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./pipe6

Waiting for input:TEST
Lower case output: test
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $
```

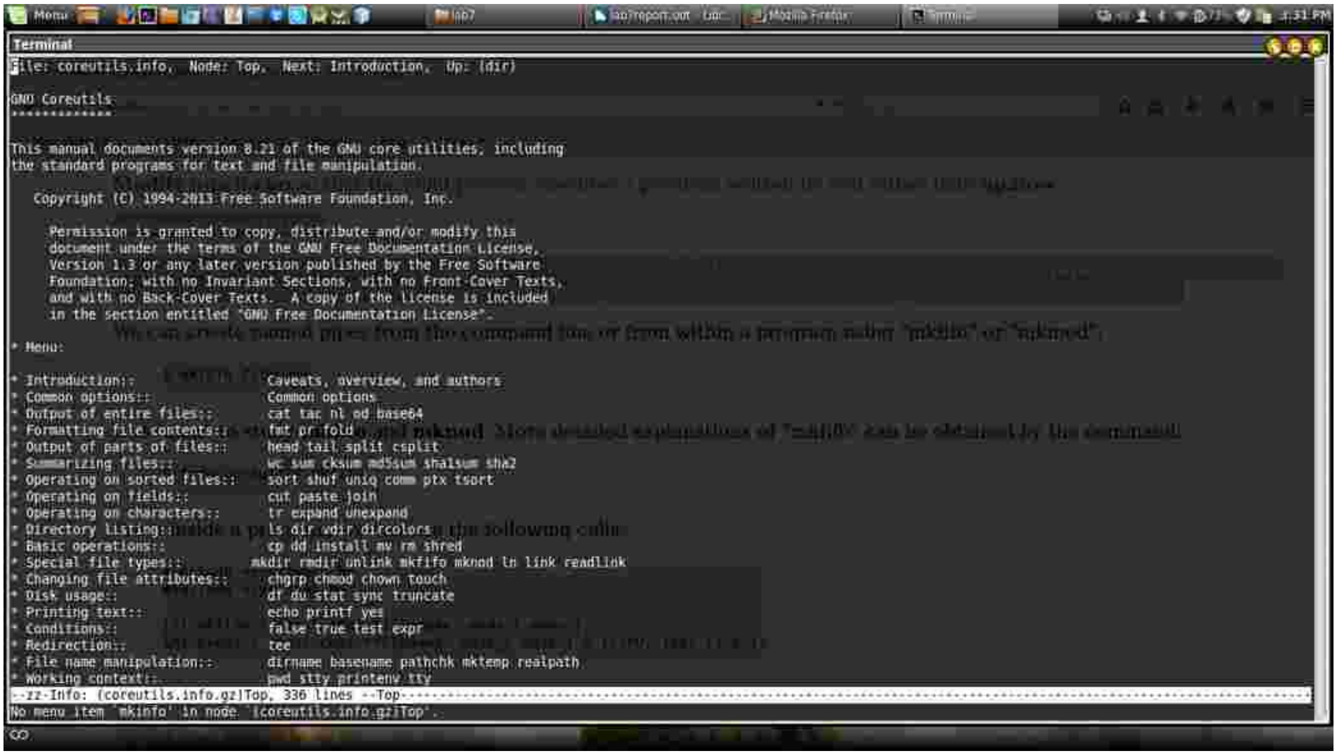
Named Pipes: FIFOs

Data exchange between unrelated processes can be done using **FIFO**s, often referred to as **named pipes**. A named pipe is a special type of file that exists as a name in the file system, but behaves like unamed pipes.

Use "man" to study **mkfifo** and **mknod**. More detailed explanations of "mkfifo" can be obtained by the command:

```
$ info coreutils mkfifo
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460



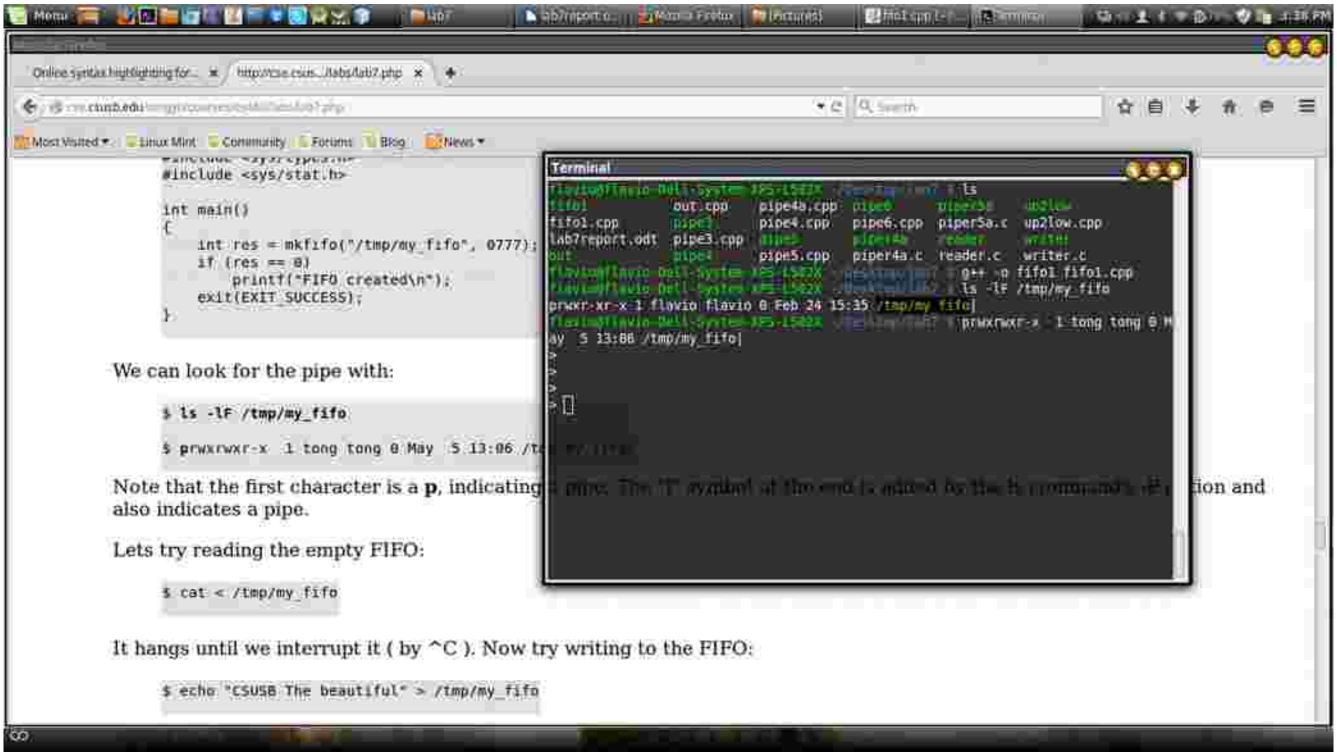
```
File: coreutils.info,  Node: Top,  Next: Introduction,  Up: (dir)

GNU Coreutils
*************

This manual documents version 8.21 of the GNU core utilities, including
the standard programs for text and file manipulation.

   Copyright (C) 1994-2013 Free Software Foundation, Inc.

     Permission is granted to copy, distribute and/or modify this
     document under the terms of the GNU Free Documentation License,
     Version 1.3 or any later version published by the Free Software
     Foundation; with no Invariant Sections, with no Front-Cover Texts,
     and with no Back-Cover Texts.  A copy of the license is included
     in the section entitled "GNU Free Documentation License".

* Menu:

* Introduction::                 Caveats, overview, and authors
* Common options::               Common options
* Output of entire files::       cat tac nl od base64
* Formatting file contents::     fmt pr fold
* Output of parts of files::     head tail split csplit
* Summarizing files::            wc sum cksum md5sum sha1sum sha2
* Operating on sorted files::    sort shuf uniq comm ptx tsort
* Operating on fields::          cut paste join
* Operating on characters::      tr expand unexpand
* Directory listing::            ls dir vdir dircolors
* Basic operations::             cp dd install mv rm shred
* Special file types::         mkdir rmdir unlink mkfifo mknod ln link readlink
* Changing file attributes::     chgrp chmod chown touch
* Disk usage::                   df du stat sync truncate
* Printing text::                echo printf yes
* Conditions::                   false true test expr
* Redirection::                  tee
* File name manipulation::       dirname basename pathchk mktemp realpath
* Working context::              pwd stty printenv tty
--zz-Info: (coreutils.info.gz)Top, 336 lines --Top-------------------------------
No menu item `mkinfo' in node `(coreutils.info.gz)Top'.
```

We can look for the pipe with:

```
$ ls -lF /tmp/my_fifo

$ prwxrwxr-x  1 tong tong 0 May  5 13:06 /tmp/my_fifo|
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Note that the first character is a **p**, indicating a pipe. The "|" symbol at the end is added by the ls command's -F option and also indicates a pipe.



```
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    int res = mkfifo("/tmp/my_fifo", 0777);
    if (res == 0)
        printf("FIFO created\n");
    exit(EXIT_SUCCESS);
}
```

We can look for the pipe with:

```
$ ls -lF /tmp/my_fifo

$ prwxrwxr-x  1 tong tong 0 May  5 13:06 /tmp/my_fifo|
```

Note that the first character is a **p**, indicating a pipe. The "|" symbol at the end is added by the ls command's -F option and also indicates a pipe.

Lets try reading the empty FIFO:

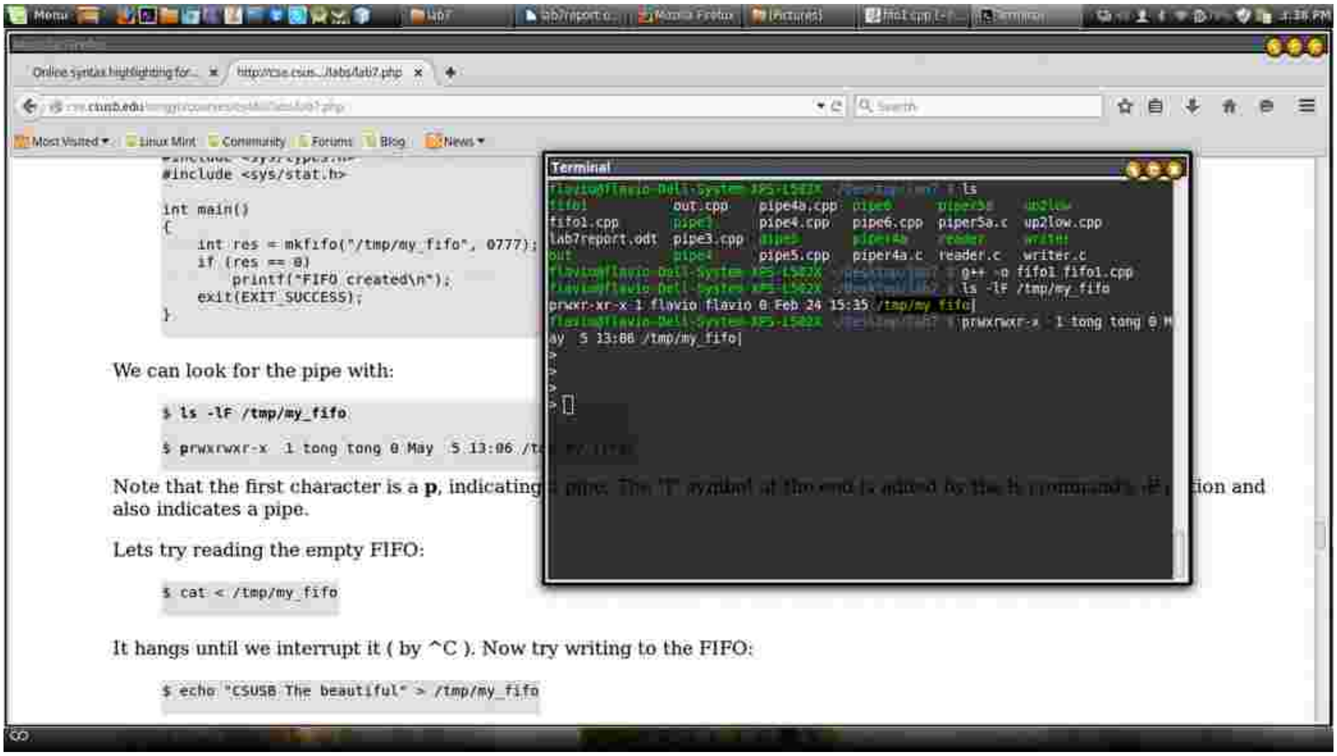
```
$ cat < /tmp/my_fifo
```

It hangs until we interrupt it (by ^C). Now try writing to the FIFO:

```
$ echo "CSUSB The beautiful" > /tmp/my_fifo
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
fifo1           out.cpp    pipe4a.cpp  pipe6      piper5a    up2low
fifo1.cpp       pipe3      pipe4.cpp   pipe6.cpp  piper5a.c  up2low.cpp
lab7report.odt  pipe3.cpp  pipe5       piper4a    reader     writer
out             pipe4      pipe5.cpp   piper4a.c  reader.c   writer.c
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o fifo1 fifo1.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls -lF /tmp/my_fifo
prwxr-xr-x 1 flavio flavio 0 Feb 24 15:35 /tmp/my_fifo|
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ prwxrwxr-x  1 tong tong 0 May  5 13:06 /tmp/my_fifo|
>
>
>
> 
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460



```
#include <sys/stat.h>

int main()
{
    int res = mkfifo("/tmp/my_fifo", 0777);
    if (res == 0)
        printf("FIFO created\n");
    exit(EXIT_SUCCESS);
}
```

We can look for the pipe with:

```
$ ls -lF /tmp/my_fifo

$ prwxrwxr-x  1 tong tong 0 May  5 13:06 /t
```

Note that the first character is a **p**, indicating ion and also indicates a pipe.

Lets try reading the empty FIFO:

```
$ cat < /tmp/my_fifo
```

It hangs until we interrupt it (by ^C). Now try writing to the FIFO:

```
$ echo "CSUSB The beautiful" > /tmp/my_fifo
```

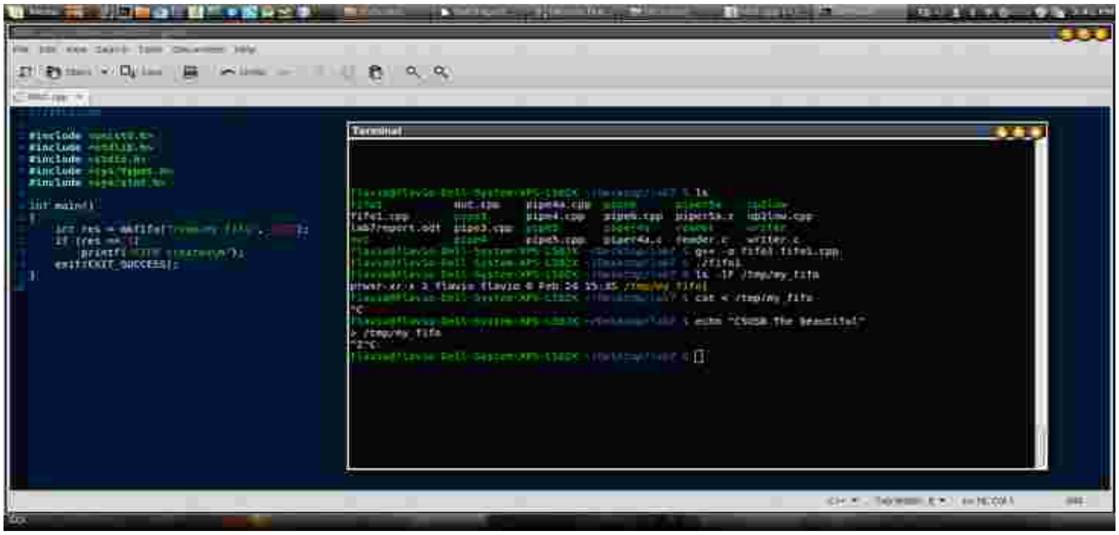
```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
fifo1            out.cpp    pipe4a.cpp  pipe6      piper5a    up2low
fifo1.cpp        pipe3      pipe4.cpp   pipe6.cpp  piper5a.c  up2low.cpp
lab7report.odt   pipe3.cpp  pipe5       piper4a    reader     writer
out              pipe4      pipe5.cpp   piper4a.c  reader.c   writer.c
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o fifo1 fifo1.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls -lF /tmp/my_fifo
prwxr-xr-x 1 flavio flavio 0 Feb 24 15:35 /tmp/my_fifo|
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ prwxrwxr-x  1 tong tong 0 M
ay  5 13:06 /tmp/my_fifo|
>
>
>
> 
```

Lets try reading the empty FIFO:

```
$ cat < /tmp/my_fifo
```

It hangs until we interrupt it (by ^C). Now try writing to the FIFO:

```
$ echo "CSUSB The beautiful" > /tmp/my_fifo
```



```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    int res = mkfifo("/tmp/my_fifo", [illegible]);
    if (res == 0)
        printf("FIFO created\n");
    exit(EXIT_SUCCESS);
}
```

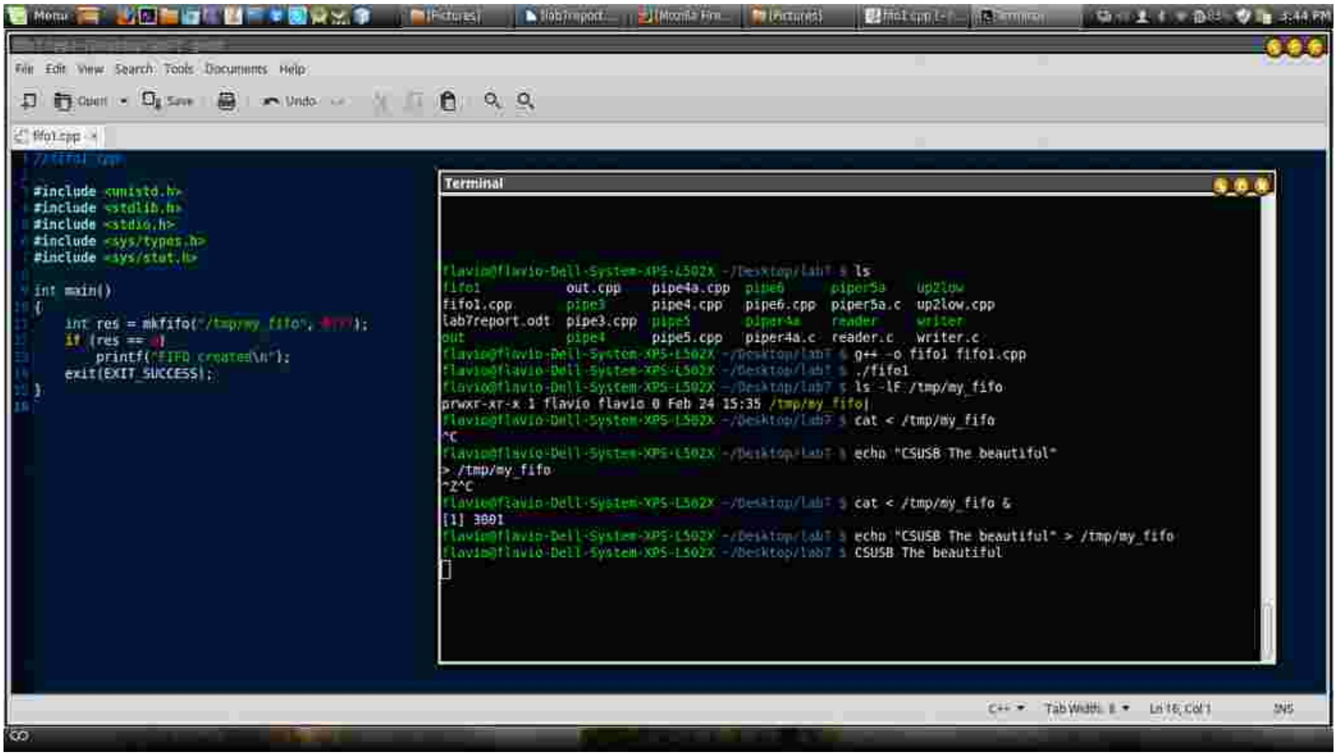
```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
fifo1            out.cpp    pipe4a.cpp  pipe6      piper5a    up2low
fifo1.cpp        pipe3      pipe4.cpp   pipe6.cpp  piper5a.c  up2low.cpp
lab7report.odt   pipe3.cpp  pipe5       piper4a    reader     writer
out              pipe4      pipe5.cpp   piper4a.c  reader.c   writer.c
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o fifo1 fifo1.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./fifo1
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls -lF /tmp/my_fifo
prwxr-xr-x 1 flavio flavio 0 Feb 24 15:35 /tmp/my_fifo|
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ cat < /tmp/my_fifo
^C
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ echo "CSUSB The beautiful"
> /tmp/my_fifo
^Z^C
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Again, it hangs. This is because there were no data in the FIFO, the **cat** and **echo** programs block, waiting some data to arrive and some other process to read the data, respectively.

If we do both at once, we can pass information through the pipe:

```
$ cat < /tmp/my_fifo &
[1] 5513
$ echo "CSUSB The beautiful" > /tmp/my_fifo
CSUSB The beautiful
```



```
//fifo1.cpp

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>

int main()
{
    int res = mkfifo("/tmp/my_fifo", 0777);
    if (res == 0)
        printf("FIFO created\n");
    exit(EXIT_SUCCESS);
}
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
fifo1          out.cpp    pipe4a.cpp  pipe6       piper5a    up2low
fifo1.cpp      pipe3      pipe4.cpp   pipe6.cpp   piper5a.c  up2low.cpp
lab7report.odt pipe3.cpp  pipe5       piper4a     reader     writer
out            pipe4      pipe5.cpp   piper4a.c   reader.c   writer.c
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ g++ -o fifo1 fifo1.cpp
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./fifo1
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls -lF /tmp/my_fifo
prwxr-xr-x 1 flavio flavio 0 Feb 24 15:35 /tmp/my_fifo|
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ cat < /tmp/my_fifo
^C
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ echo "CSUSB The beautiful"
> /tmp/my_fifo
^Z^C
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ cat < /tmp/my_fifo &
[1] 3001
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ echo "CSUSB The beautiful" > /tmp/my_fifo
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ CSUSB The beautiful
```

The following is a client/server application that makes use FIFOs to communicate. The server program **server.cpp** creates and opens the server pipe which is set to read-only, with blocking. After sleeping (for demonstration purposes), the server reads in any data sent by a client, which has the **data_to_pass_st** structure. In the next stage, it performs some processing on the data just read from the client, converting all characters recieved to uppercase and combine the **CLIENT_FIFO_NAME** with the received **client_pid**. Finally, it sends the data back, opening the client pipe in write-only, blocking mode, and then shut down the FIFO server by closing the file and unlinking the FIFO.

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
//server.cpp
#include <ctype.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>

#define SERVER_FIFO_NAME "/tmp/serv_fifo"
#define CLIENT_FIFO_NAME "/tmp/client_fifo"

#define BUFFER_SIZE 20

struct data_to_pass_st {
    pid_t  client_pid;
    char   some_data[BUFFER_SIZE - 1];
};

int main()
{
    int server_fifo_fd, client_fifo_fd;
    struct data_to_pass_st my_data;
    int read_res;
    char client_fifo[256];
    char *tmp_char_ptr;

    mkfifo(SERVER_FIFO_NAME, 0777);
    server_fifo_fd = open(SERVER_FIFO_NAME, O_RDONLY);
    if (server_fifo_fd == -1) {
        fprintf(stderr, "Server fifo failure\n");
        exit(EXIT_FAILURE);
    }

    sleep(10); /* lets clients queue for demo purposes */

    do {
        read_res = read(server_fifo_fd, &my_data, sizeof(my_data));
        if (read_res > 0) {

// In this next stage, we perform some processing on the data just read from the
client.
// We convert all the characters in some_data to uppercase and combine the
CLIENT_FIFO_NAME
// with the received client_pid.

            tmp_char_ptr = my_data.some_data;
            while (*tmp_char_ptr) {
                *tmp_char_ptr = toupper(*tmp_char_ptr);
                tmp_char_ptr++;
            }
            sprintf(client_fifo, CLIENT_FIFO_NAME, my_data.client_pid);
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
// Then we send the processed data back, opening the client pipe in write-only,
blocking mode.
// Finally, we shut down the server FIFO by closing the file and then unlinking the
FIFO.

            client_fifo_fd = open(client_fifo, O_WRONLY);
            if (client_fifo_fd != -1) {
                write(client_fifo_fd, &my_data, sizeof(my_data));
                close(client_fifo_fd);
            }
        }
    } while (read_res > 0);
    close(server_fifo_fd);
    unlink(SERVER_FIFO_NAME);
    exit(EXIT_SUCCESS);
}
```

The client program **client.cpp** opens the server FIFO, if it already exists, as a file. It then gets its own process ID, which forms some of the data that will be sent to the server. The client FIFO is also created and opened (read-only, blocking mode for reading back data.

Compile with the commands:

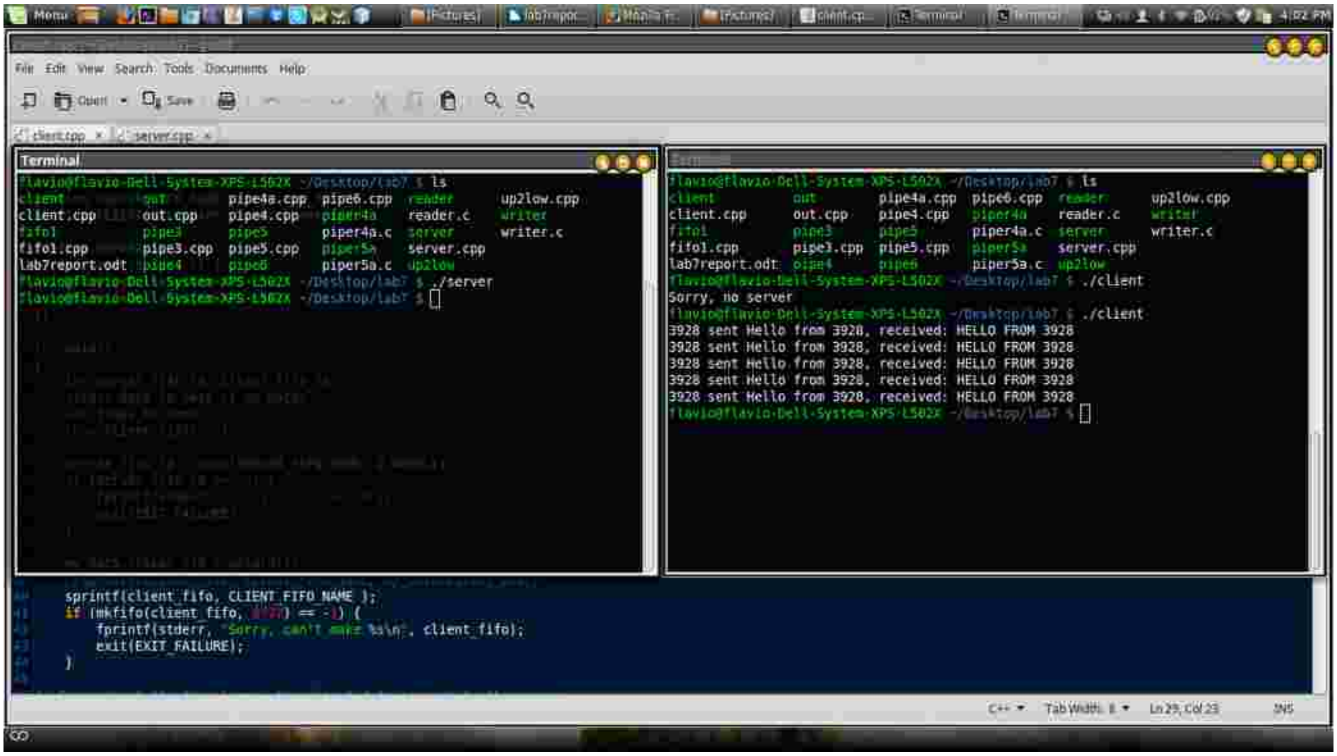
```
$ g++ -o server server.cpp
$ g++ -o client client.cpp
```

To test this out, we need to run a single copy of the server and serveral clients. To make them all started at approximately the same, we may use the following shell commands:

```
$ server &
$ for i in 1 2 3 4 5
> do
> client &
> done
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Here it is:



```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
client          out        pipe4a.cpp  pipe6.cpp   reader      up2low.cpp
client.cpp      out.cpp    pipe4.cpp   piper4a     reader.c    writer
fifo1           pipe3      pipe5       piper4a.c   server      writer.c
fifo1.cpp       pipe3.cpp  pipe5.cpp   piper5a     server.cpp
lab7report.odt  pipe4      pipe6       piper5a.c   up2low
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./server
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $
```

```
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
client          out        pipe4a.cpp  pipe6.cpp   reader      up2low.cpp
client.cpp      out.cpp    pipe4.cpp   piper4a     reader.c    writer
fifo1           pipe3      pipe5       piper4a.c   server      writer.c
fifo1.cpp       pipe3.cpp  pipe5.cpp   piper5a     server.cpp
lab7report.odt  pipe4      pipe6       piper5a.c   up2low
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./client
Sorry, no server
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./client
3928 sent Hello from 3928, received: HELLO FROM 3928
3928 sent Hello from 3928, received: HELLO FROM 3928
3928 sent Hello from 3928, received: HELLO FROM 3928
3928 sent Hello from 3928, received: HELLO FROM 3928
3928 sent Hello from 3928, received: HELLO FROM 3928
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $
```

```
    sprintf(client_fifo, CLIENT_FIFO_NAME );
    if (mkfifo(client_fifo, 0777) == -1) {
        fprintf(stderr, "Sorry, can't make %s\n", client_fifo);
        exit(EXIT_FAILURE);
    }
```

Try the scripts and then **modify** them so that received characters are converted to lower case rather than upper case.

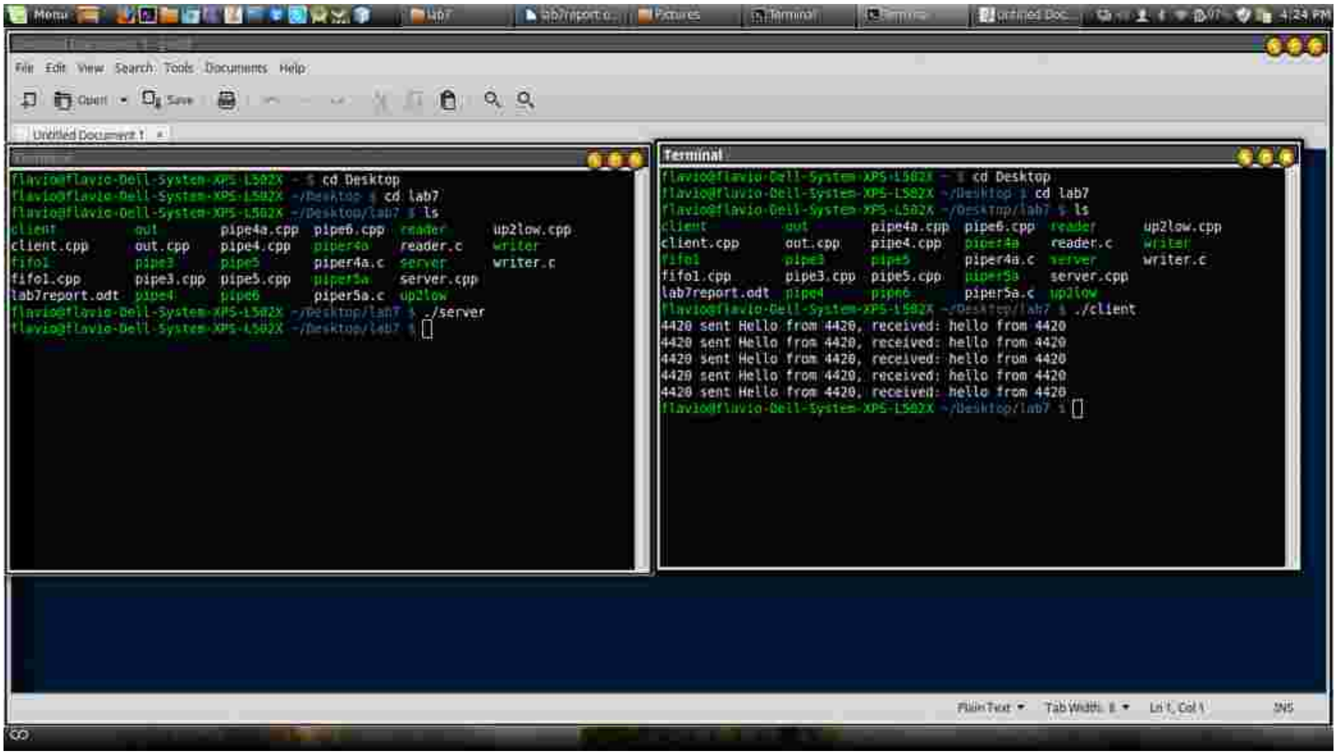
Here is the code:

```
// In this next stage, we perform some processing on the data just read from the client.
// We convert all the characters in some_data to uppercase and combine the CLIENT_FIFO_NAME
// with the received client_pid.

            tmp_char_ptr = my_data.some_data;
            while (*tmp_char_ptr) {
                *tmp_char_ptr = tolower(*tmp_char_ptr);
                tmp_char_ptr++;
            }
```

The only part we had to change to display all lower case characters was to change the process of the data read from the client, which means "toupper" into "tolower."

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Here are the results:



```
flavio@flavio-Dell-System-XPS-L502X ~ $ cd Desktop
flavio@flavio-Dell-System-XPS-L502X ~/Desktop $ cd lab7
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
client          out         pipe4a.cpp  pipe6.cpp   reader      up2low.cpp
client.cpp      out.cpp     pipe4.cpp   piper4a     reader.c    writer
fifo1           pipe3       pipe5       piper4a.c   server      writer.c
fifo1.cpp       pipe3.cpp   pipe5.cpp   piper5a     server.cpp
lab7report.odt  pipe4       pipe6       piper5a.c   up2low
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./server
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

```
flavio@flavio-Dell-System-XPS-L502X ~ $ cd Desktop
flavio@flavio-Dell-System-XPS-L502X ~/Desktop $ cd lab7
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ls
client          out         pipe4a.cpp  pipe6.cpp   reader      up2low.cpp
client.cpp      out.cpp     pipe4.cpp   piper4a     reader.c    writer
fifo1           pipe3       pipe5       piper4a.c   server      writer.c
fifo1.cpp       pipe3.cpp   pipe5.cpp   piper5a     server.cpp
lab7report.odt  pipe4       pipe6       piper5a.c   up2low
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ ./client
4420 sent Hello from 4420, received: hello from 4420
4420 sent Hello from 4420, received: hello from 4420
4420 sent Hello from 4420, received: hello from 4420
4420 sent Hello from 4420, received: hello from 4420
4420 sent Hello from 4420, received: hello from 4420
flavio@flavio-Dell-System-XPS-L502X ~/Desktop/lab7 $ 
```

We worked very hard to finish the lab and learned a lot about pipes. Because we were able to finish everything in this lab, we give ourselves a score of 20/20.