LAB 2

CO327 – Operating Systems

NILUCSHAN . S
E/13/239
SEMESTER 6
23/08/2017

Interprocess Communication

DEPARTMENT OF COMPUTER ENGINEERING

1. Lower level I/O

Exercise1.1:
a. Explain what the flags O_*WRONLY*, *O_APPEND* and *O_CREAT* do.
b. Explain what the modes *S_IRUSR*, *S_IWUSR* do.

a. Argument flags generally define in which way a file needed to be opened. So by using the flag **O_WRONLY** the file will be opened in the write only mode. **O_APPEND** makes the file to be appended. (The file is opened in the append mode, i.e. if it is set, the file offset will be set to the end of the file prior to each write.) **O_CREATE** is used to create the file if the file does not exist. If the file exists, this flag has no effect.

b. S_IRUSR mode will give read permission to the owner whereas S_IWUSR will give write permission to the owner.

Exercise1.2:
a. Write a program called `mycat` which reads a text file and writes the output to the standard output.
b. Write a program called `mycopy` using *`open()`*, *`read()`*, *`write()`* and *`close()`* which takes two arguments, viz. source and target file names, and copy the content of the source file into the target file. If the target file exists, just overwrite the file.

Attached as **mycat.c** and **mycopy.c**

2. Pipes

Exercise2.1:
Look at the code in example2.2.c and answer the following questions.
a. What does *`write(STDOUT_FILENO, &buff, count);`* do?
b. Can you use a pipe for bidirectional communication? Why (not)?
c. Why cannot unnamed pipes be used to communicate between unrelated processes?
d. Now write a program where the parent reads a string from the user and send it to the child and the child capitalizes each letter and sends back the string to parent and parent displays it. You'll need two pipes to communicate both ways.

a. Every program is expected to have three file streams. One for getting input, one for writing output and one for writing errors. STDOUT_FILENO is one of them and has a file descriptor of "1", this is like an index to the kernel-resident array, containing the information about all the open files. To edit a file on a Linux system we obviously need a buffer location. It is an abstract memory space, created in the RAM for that particular file. So by using **write(STDOUT_FILENO, &buff, count)**, we are writing an output to the given **buff** location **count** number of bytes.

b. If we use the same pipe there will be difficulties for child in separating its messages from the parent's messages and vice versa. For example if parent writes to pipe and tries to read from pipe hoping to get message from child, it gets its own message.

Therefore a pipe should be used for **parent-child** communication and another pipe for **child-parent** communication.

c. Related processes have common parent processes. When considering unrelated processes pipes cannot be used since there is no way of referring to pipes which have been created by another process.

d. Attached as **capitalize.c**

3. *dup()*and *dup2()*System Calls

Exercise3.1:
Write a program that uses *`fork()`* and *`exec()`* to create a process of *`ls`* and get the result of *`ls`* back to the parent process and print it from the parent using pipes. If you cannot do this, explain why.

A program cannot be written.
When executing **fork()** and **exec()** the relation will be lost. This is because original code will be edited. Unnamed pipes cannot be used to communicate between unrelated processes.

Exercise3.2:

a. What does 1 in the line *`dup2(out,1);`* in the above program stands for?

b. The following questions are based on the example3.2.c
   i. Compare and contrast the usage of *`dup()`* and *`dup2()`*. Do you think both functions are necessary? If yes, identify use cases for each function. If not, explain why.
   ii. There's one glaring error in this code (if you find more than one, let me know!). Can you identify what that is (hint: look at the output)?
   iii. Modify the code to rectify the error you have identified above.

c. Write a program that executes `"cat fixtures | grep <search_term> | cut -b 1-9"` command. A skeleton code for this is provided as exercise3.2.c_skel.c. You can use this as your starting point, if necessary.

a. **dup2()** is a system call which duplicates one file descriptor, making them aliases, and then deleting the old file descriptor. It is similar to the dup call. This is useful when attempting to redirect output, as it automatically takes care of closing the old file descriptor, performing the redirection in one elegant command.

b.
   i. **dup()** - takes one file descriptor as its argument and duplicates the next lowest available file descriptor with the same file table pointer. It returns the new file descriptor value.

   dup2() - takes two file descriptors as its arguments.

   dup2() is useful in scenarios where we need to duplicate a file descriptor as another file descriptor. When using dup(), it is not sure about the return value. It will automatically return the next lowest available file descriptor.

4. Named pipes

Exercise4.1:

a. Comment out the line "*mkfifo(fifo,0666);*" in the reader and recompile the program. Test the programs by alternating which program is invoked first. Now, reset the reader to the original, comment the same line in the writer and repeat the test. What did you observe? Why do you think this happens? Explain how such an omission (i.e., leaving out *mkfifo()* function call in this case) can make debugging a nightmare.

b. Write two programs: one, which takes a string from the user and sends it to the other process, and the other, which takes a string from the first program, capitalizes the letters and send it back to the first process. The first process should then print the line out. Use the built in command *tr()* to convert the string to uppercase.

a.

Commenting out in reader

reader program invokes first and no messages are printed when the **writer** is run first.

Commenting out in writer

writer invoked first. Here also no messages are printed when the **reader** is run first.

Explanation

mkfifo can be used to create a named pipe from within a program. It creates a new FIFO special file by the path name that path name points to. If a file is created as a named pipe then the reader program will not wait until the FIFO gets filled which results in an empty file to be read.

b. Attached as **exercise4.1.reader.c** and **exercise4.1.writer.c**