LAB03-Interprocess Communication

DEPARTMENT OF COMPUTER ENGINEERING

Amasith K.T.D. - E/16/025

Exercise1.1:

a. Explain what the flags O_WRONLY, O_APPEND and O_CREAT do.

All these are Flags which can set in open system call.

• O_WRONLY:
  Open for writing only.

• O_APPEND:
  The new data that is written to the end of the file every time it write after opening file with this flag. This is done by setting the offset to the end of the file. The modification of the file offset and the write operation are performed as a single atomic step

• O_CREATE
  A new file is created by the file name is the file is not existing. If the file exists, nothing happens. The owner (user ID) of the new file is set to the effective user ID of the process.

b. Explain what the modes S_IRUSR, S_IWUSR do

These are mode setting when open system call flaged with O_CREATE.
The mode argument specifies the file mode bits be applied when a new file is created.

- S_IRUSR (00400) - user has read permission
- S_IWUSR (00200) - user has write permission

Exercise1.2:

mycat.c

mycopy.c

Exercise 2.1

(a). What does write(STDOUT_FILENO, &buff, count); do?

This is write system call which get file descriptor as constant value which reserved with file descriptor of standard out put. This is same as we print with printf function.

(b). Can you use a pipe for bidirectional communication? Why (not)?

Can't. The pipes cannot be used for bi directional communication . One can write into one side .and others can others can received it on other side. So Pipe is only for unidirectional communication.

(c). Why cannot unnamed pipes be used to communicate between unrelated processes?

The unnamed pipes are accessed by file descriptors. There is no way to copy the file descriptor.
Only forking the process after assigning the file descriptors in memory.

(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.

Ex21d.c

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.

Can't. As exec will replace everything in the parent process and replace the old file or program from the process with a new file or program.

Exercise3.2:

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

1 is file descriptor of the standard output

b. The following questions are based on the example

3.2.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.

dup() and dup2() takes in a file descriptor, assigns another file descriptor to it and returns it.

- dup(int oldFileDescriptor) This function assigns the smallest numbered file descriptor that is not being used.

- dup2(int oldFileDescriptor, int newFileDescriptor) The function tries to assign the oldFileDescriptor to the newFileDescriptor.

They are differentiating with how they select new file descriptor.

Both dup() and dup2() usecases :

- dup() is useful if we want to get a new file descriptor without overlaping with the file descriptors that are in use at the moment
- dup2() is useful if we want to assign a file descriptor to a known file descriptor (usually to modify an existing file descriptor)

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)?

close()
dup()
This sequence of code will lead child and parent processes to have a race condition. When one processes will use the file descriptor from another process.

iii. Modify the code to rectify the error you have identified above.

exampleModify3.2.c

c.

Ex32c.c

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.**

- mkfifo is removed in the reader -
  When read running first, it will wait until the writer is run and the 4 message is written from writer. Then the reader will display the message and both programs will close. – If the writer is run first, it will wait until the reader is run and reads the message. Then the reader will display the message and both programs will close.

- mkfifo is removed in the writer -
  When read running first,it will wait until the writer is run and the message is written from the writer. Then the reader will display the message and both programs will close. If the writer is run first, it will not wait until the reader reads the message. The message will be lost. mkfifo makes a special file which needs to be open on both ends (reading and writing) for a process to write to it. This makes read() write() functions blocking until open() is called in the other process. if mkfifo is omitted, it can occur race conditions. And this is hard to debug.

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.**

Ex42user.c Ex42capitalizer.c