LEARNING DIARY__ASSIGNMENT1.VERSION1 <without exec() code replacement>

PROGRAM DESIGN:

This program is mainly about the communication between two processes. The program read a file in one process and try to transfer the file content to another process, with the content being morse coded. The process who receive the file content would save it into a new file.

Communication Conventions:

Since morse code only use two digits - dit and dah, it can be easily replaced by two signals. SIGUSR1 and SIGUSR2 are chosen to be the digit dit and dah, respectively. One character can be represented by a specific code sequence. Because each code sequence of a character may have a different length, it shall have an end, indicating the receiver process that one character has been sent completely. SIGALRM is chosen to be the character dividing signal. When the file reaches the end at the sender process, the sender shall inform the receiver process to finish decoding and close the copied file. SIGTERM is picked to be the signal for end of file indication.

Standard morse code does not include every symbol on the modern computer keyboard, this is why the code sequences of many keys and symbols have been defined by myself. Even though, there's possibility that the sender process encounters an undefined character and could not able to translate it into morse code. In this condition, my program is designed to end the entire copy work gracefully. First, another signal SIGCONT is used by the the communication to indicate unrecognized character. When the sender process encounter a character, it calls a function named sendcode(), if this character is not defined in the sendcode() function, then the function send SIGCONT directly to the receiver process and the parent process exits(with all file closed). On receiving SIGCONT, the receiver process would log a message about undefined character and exit.

Communication Mechanism:

In designing the communication mechanism, I have tested different APIs, from the simplest signal() to sigaction(), sigprocmask() and sigwait() etc. I have well understood the early unreliable signal and the reliable signal. All these tests work can be found in my tests folder.

together with another part of my learning diary-the test diary. After writing several version of signal test programs, I found that signals are really easy to get lost in the window time of calling functions, say sigwait() or pause(). Finally, I decide to use your signalbetter model. In this model, signal handler would just use a self owned pipe to buffer incoming signal. signal is read only when it's needed. In this way, signal would no longer get lost. sigwait() or pause() to wait for pair process to give trigger signal is just a bad idea!!!!Unreliable!!!

The final design is like this:

Before forking and communicating happens between the two processes, signal handlers are set for both the parent process and the child process. Obviously, signal handlers can inherit across fork(). After forking, the parent process would call the senderprocess() function, which read file and send morse code; The child process then call the receiverprocess() function, which receive morse code, parse it and write character onto a newly created file.

The flow control is design to follow a one-signal-at-a-time model. After opening the file and doing all preparations, the parent(Also the sender) would hang there waiting for 'Go'(defined as SIGINT) signal from the child. The child would send SIGINT as soon as it finish certain preparation. In this sense, the child is the one who initialize the entire transfer process. Then the parent begins to read file into a buffer, read a character, call the sendcode() function and send the morse code sequence. every time, the sendcode() function would only send one signal, the receiver process would check the signal, take corresponding action and send back a SIGINT as a 'Go', indicating the sender to send another signal.

Sender logic: simply put, the sender call sendcode() for every character it read, when sendcode() returns, the sender knows that the code sequence for one character has been sent. At this, the sender would send SIGALRM as the character dividing signal. When finish reading the file, the sender would send SIGTERM to terminate both processes. Undefined character is processed also by sendcode(), who send the receiver a SIGCONT and terminate the parent process.

Receiver logic: If the receiver process get SIGUSR1 or SIGUSR2, it save them into a char variable. This is done by bitwise operation. SIGUSR1 is recorded as digit 1, SIGUSR2 is

recoded as digit 0. The char variable was initialized to be 0x01, so that if the first morse digit for a character is dah(SIGUSR2), this digit would not get lost. Whenever received SIGALRM, the receiver would call readmorse(). readmorse() would compare the value of the char variable ,which hold the morse code sequence for a single character, with a pre-calculated integer value, and match it with a certain character. readmorse() returns this character and the receiver process put this character onto file. SIGTERM signal cause the receiver to close file and shut down normally, SIGCONT cause it to log a message of "undefined character" and then shut down the process.

PROBLEMS ABOUT LOGGING FILE:(SOLVED)

- Before forking, I use fopen to open a file for both processes to writting log; I encounter error while compiling. this is because fopen returns a pointer rather than a file descriptor, which cannot be inheritated across fork(); So I call fopen to open the same file in appending mode in both processes.
- My intention is to design the log system so that log messages are written in order of time.
- Since two processes are accessing the same log file,using file lock to lock the file before writting to it and release the lock afterward seems fine. But it won't work because file lock is not inheritated across fork(); then I begin to consider using mutex since in some implementation it does work with fork(). The problem is that using mutex is useless even the mutex itself is working. Given that I am using C stand library I/O, each process has a underlayer buffer for fprintf(). even if I use mutex to make the write in order of time, the content in the logfile is probably not in order. In addition to mutex, I have to use fflush each time I call fprintf, then it should work.
- In my program, the consideration of two processes writing to the same log file,in order of time, is not a problem at all! Since my morse transcoder works in a way that each log message is written only when the corresponding process get the signal from the pair process. Before the process could finish the log message and send back a signal, the pair process would just hangs there and wait for that signal.
- All I have to do is to fflush() the stream every time right after I use fprintf() to write a log.

No file locking needed and no mutex needed.

MEMORY LEAKAGE PROBLEM:(SOLVED)

- When I forgot to call fclose(), valgrind shows memory leakage caused by fopen()'s internal malloc. So, keep in mind that flcose() make sure no memory leakage is possible from fopen().
- Since both the parent and child processes access the same logfile, the logfile must be closed in both processes. A file handler is bond to a certain process. each process has a handler of it's own.(remind me of that table entry stuff)