MORSE_TRANSCODER - USER GUIDE

/* By Yunfeng He; 2012.11.13; UnixC Assignment 1 */

Also check the folder exec to review the exec() version of the programme, diary and guide are also in that folder

INSTALLATION:

- In the folder morsetranscoder, there's a Makefile for doing the compiling and clean work.
- Open a terminal and change into the morsetranscoder directory, then make. An executable file called morsetranscoder will be generated. The related source files are main.c morse.c morse.h.
- make clean will clean morsetranscoder, the copied files and LOGFILE.

PROGRAMME FUNCTION:

- The program morsetranscoder takes a command line argument, which is a file to be copied. after successfully open the file, the program would fork a child process. The parent process then acts as a morse code sender who read the file and translate the characters into morse code and send the code sequence to the child process. The child on the other hand would behave as the decoder, who receive the morse code sequences and translate it into characters. After parsing the morse code sequence, the child process write the corresponding character into a file whose name is that of the original file with a COPY suffix attached.
- Both processes write log messages into the same file called LOGFILE. The log messages are written in LOGFILE in order of generated time, giving a hint about what the programme is doing while copying. LOGFILE will be created the first time morsetranscoder is launched.
- Do not press ctrl-c to interrupt the program, this could probably disrupt the program and block both processes.
- Be patient when the program is copying. It may take a short while if the file to be copied is too large(too many white space and lines for example). The log system somehow stalls the program, I need feedback on conditional compilation, since the log system does nothing but making the program much slower.

USAGE:

- In terminal, under the directory of morsetranscoder, type in *./morsetranscoder [filename] to run the morsetranscoder grogram.*
- For example, *./morsetranscoder main.c* will open main.c and make a copy of it named main.cCOPY; log messages will be attached on the file LOGFILE.

DESIGN DETAILS:

The design details are listed and documented in the Assignment1Diary within the learningdiary folder.

MET REQUIREMENTS:

- Writing to a file and reading from a file.(C library I/O used. No system level I/O used, in order to improve performance.)
- Signal handling.(SIGINT is used as flow control code, so please do not press Ctrl-C to mess things up, otherwise the program would probably block forever.)
- Creation of a child process.

ADDITIONAL REQUIREMENTS:

- Code replacement with exec (): [implemented!]

  The exec folder contains the exec version. please check! only necessary and minor changes made. please review the morsetranscoder program first; In version2(the exec version), morsecoder.c morse.c morse.h are the source for the main program; decoder.c decoder.h are the sources for the replacement code
- Use of memory mapped files : [not implemented!]

  I am using C library I/O which has buffer system, the performance is ok. Also, I don't think this would actually work on my logfile system. This is explained in the learning diary in detail.
- File locking : [not implemented!]

  Not useful since it seems to work only with system level I/O. My logging system is already killing performance, I see no point of using it. this is explained in the learning diary in detail
- Asynchronous or non-blocking I/O :[not implemented!]

  Non-blocking I/O only making things more annoying. not writing to or reading from socket.

FUFILLED CODING REQUIREMENTS:

- Personally think the code is modular enough and is well commented.
- Valgrind mem test passed.
- No warning with -Wall -pedantic, except for mixed declaration and comment style
- CTRL-C is not my consideration here. However the program does exit well under error conditions.(Failed opening files; undefined character encountered by my morse decoder; exec() failure;)
- Makefile works well.

LIMITATIONS:

- At the decoder side, upper or lower case letters will all be understood as lower case letters! (it's just laborious work to define those morse code sequences, defeat the purpose of the assignment)
- However, all signs and symbols on the keyboard are well understood by my decoder.
- The logging system hit the program performance hard, since log messages are written each time a character is sent and decoded. fflush() after each write cause this performance problem but it is the necessary way to make the log messages in right order.(it was a lot worse since I used to give log message each time a

process send or receive a signal)

- How do I write conditional compiling? My logging system problem has proved it useful. Please give feedback and simple example if possible. You've showed it during lecture but it's not easy to remember those stuff in five minutes.