# S-38.3600 UNIX Application Programming, Fall 2014

# Assignment 1 - Diary

31.10.2014

Jaume Benseny, 463922, jaume.benseny@aalto.fi

## INTRODUCTION

The present document describes chronologically how the assignment 1 was programmed. All of the stages describe a part of the program and more specifically include: how I though about it in the beginning, problem and solution while coded, how it was tested and what it could be improved in the future.

General description and short usage instructions could be found in the main directory as file “short usage instructions.txt”. In addition older versions of the code can be found in the folder /old/.

## FIRST STARGE

I started my programming creating the basic CHILD - PARENT structure based on the fork() instruction. After that, I opened one FILE for the CHILD so it could read and another FILE for the PARENT to write. In order to validate that everything worked fine, I created a pipe that allowed the CHILD i) to read from FILE and ii) WRITE to pipe and the PARENT to iii) READ from pipe and iv) WRITE to file. (Implementation can be found in folder /old/code1/forkrwpipe.c).

Afterwards, I started working with MORSE and SIGNALS. First of all I wrote a high level pseudo-code scheme to help me design the basic modules and libraries. The scheme follows:

```
1. CHILD
            OPEN LOCAL FILE
            GET CHAR FROM FILE
              TRANSCODE TO MORSE
              SEND SIGNALS ACCORDING TO CHAR
            NEXT CHAR
            ..
            CLOSE FILE

2. PARENT
            START LISTENNIG FOR SIGNALS
              DECODE SIGNALS TO CHAR
              WRITE CHAR TO FILE
              ..
                      ..
            CLOSE FILE
```

## MORSELIB.C

My initial idea was to make the CHILD translate each char to a vector of MORSE UNITS composed of ./- that would be used as a mask to send the signals USR1 and USR2. Therefore, when I created the library morselib.c I included a unique vector which contained vectors of all possible chars and its codification in a MORSE WORD, one after the other (like "E","-",). In order to be able to make the translation work I created two functions chartomorse() and morsetochar() within the same library. The main problem of this approach is that I forced myself to work with pointers every time I had to seek one char or MORSE WORD. It's been tricky and I wasted a lot of time looking for the end of the vector inside the vector, but it works well. Another option could have been to use structs and work in more abstract and clear way. I would like to emphasize that chars were introduced in the vector following the MORSE tree to optimize its search. (An early implementation of the lib can be found in /old/code2/codetomorse2.c)

## END OF CHAR, END OF LINE AND SPACE

Regarding how to indicate i) the end of transmission of a char (from now on ENDOFCHAR), ii) the blank space and iii) change of line I thought I could code them as an new MORSE WORD and include them in the dictionary. However, I realized that was not possible for the ENDOFCHAR so I decided to instead use SIGALRM. Blank space and change of line are coded and exchanged as MORSE WORDS. I have to admit they are long codes that require a lot of time to be transmitted and due to the fact that are used frequently, they slow down my program. It could be improved by the usage of more SIGNALS.

## MORSE ERROR CODE

I included in my dictionary the chars like Ä or Ü because they have their own MORSEWORD. However, when testing the chartomorse() function I saw that the standard out didn't print them correctly. I realized they have a particular representation as Extended Chars[1] in the UNIX domain. To correctly respond to this chase and some other chars which are not considered in MORSE I introduced the morse error code. In case the CHILD reads a char which is not present in our morselib dictionary, the CHILD sends MORSE ERROR CODE (........). When the PARENT receives the code, it writes to the out FILE the char '?'. I admit it's not the original functionality of the MORSE ERROR CODE but it's helpful in this particular implementation.

## SIGNALLIB.C

In order to map SIGNALS with MORSE UNITS I created the library SIGNALLIB.C and the functions sendsignals() and signalstomorse(). In this case, as I mentioned before, I used SIGALRM to indicate ENDOFCHAR to the PARENT (which returned \0 to close the vector that

1 http://www.gnu.org/software/libc/manual/html_node/Extended-Char-Intro.html

defines MORSE WORD) and SIGINT in order to indicate END OF FILE. This last fact, allows the user to terminate the program by using CTL+ C because it's interpreted by the PARENT as END OF FILE (from the file1) and it closes the program.

In order to avoid race-conditions, I used a pipe to buffer signals as shown in class. The main problem I found is that exchanged SIGNALS were not received by the PARENT in the proper order (not number either) and therefore it was impossible to recover the original value of the sent char. This behavior is caused by the nature of SIGNALS, order or number of received signals is not preserved. To deal with this problem, I introduced one pipe called pipeflow[] which allows the PARENT to inform the CHILD when a new SIGNAL can be sent. This solution is much more effective compared to the one that make the CHILD sleep until the PARENT gets CPU time. As introduced in the assignment announcement **I concluded that a flow control mechanism is required.**

I used an stream to transfer information through the pipe. This fact forced me to use fflush() because it was blocking the CHILD from sending the next SIGNAL. Another possible option it could have been to implement file locking. For example, the PARENT could block a shared file until we was ready to receive the next SIGNAL.

**LOG FUNTION AND FILE BLOCK**

Finally, a log function was implemented. The FILE is created before executing fork() to make sure PARENT and CHILD share the same file descriptor. In order to avoid concurrent write to the log file a FILE LOCK was implemented. I tried to get the file descriptor from a FILE with fileno() and apply the block after writing to the FILE without success. I have finally decided to skip the FILE and just work with the file descriptor. Functionality has been tested implementing a “wait for user” function that allowed to see how the other process could not write until block was freed.

**FINAL TEST**

As a final test, I passed the assignment 1 description document to the program as file1. Due to the fact that the majority of chars are low letters, file2 had 90% of characters as question marks. The structure of the document remained intact. I changed the assignment 1 description to upper case (can be found in the main directory as assign1_UP.txt) and the program worked fine. File1 and file2 looked exactly the same.