Assignment 02 - Processes

Download the coding canvas provided for this assignment

Submission Format

When submitting your work, please stick to the conventions described below.
You will prepare a working directory that will contain the following:
- `bin` directory: empty directory for all the executable files created by editing the links between your object files
- `include` directory: directory containing all of your header files (* .h headers)
- `lib`: directory containing all of your pre-compiled libraries (* .a)
- `obj` directory: empty directory for all the object files (* .o) created by compiling your source files
- `src`: directory containing all of your source code files (*.c)
- `Makefile`: file containing all of your compiler directives
- `README.txt` file: plain text file that reports on your work submission

Of all the elements in the working directory, the `README` file is the most important. It shall contain:
- Your first name, your last name, and your NYU ID
- The detailed list of files you submitted in the directories `include`, `lib`, and `src` (for each file, list the questions it provides a solution for)
- The explanation of the compilation rules provided in your `Makefile`
- Comments about the work you submitted, if any (what does not run, what is encoded but produces errors at compile / execution, what feature is incomplete, ...)
- Textual answers to some questions in the set (eg. Q7 in this set)

Before submitting your working directory, please rename it like so:
`[lastname-firstname-nyuid].os.[assignment#]` (*eg. marin-olivier-ogm2.os.01*)
archive it in TAR format and compress it in GZIP format. The submission should be a single file:
`[lastname-firstname-nyuid].os.[assignment#].tgz` (*eg. marin-olivier-ogm2.os.01.tgz*)

This is the file you attach to your submission ***before*** the deadline.

Question 1 - Program trace

Consider the following program:

```
int main (int arg, char * argv [])
{
      int i, j, p;
      for (i = 0; i <3; i ++)
            if ((p = fork ()) == 0){
                  printf( "i =% d \ n", i)
                  j = 0;
                  while (j < i && ((p = fork ()) == 0))
                        j ++;
                  if (p == 0)
                        printf ( "j =% d \ n", j);
                  exit(j);
            }/ * if * /
      return (0);
 }
```

Note: Assume that all process creations succeed

How many child processes are created upon executing this program?
Represent the family tree of processes and give the output of each process.
Write a modified version of this code which guarantees that the original parent process (the one created by function `main`) will only terminate once all its descendants have terminated.

Question 2 - Hacking the magic square

A magic square is a square matrix where adding up the values in each row, column, or diagonal will produce the same result. For example, a matrix whose entries all share the same value is necessarily magic.
Consider the following 3x3 matrix of integer values in [0 .. 9]:

```
{{4 a, 8}, {b, c, d}, {2, e, 6}};
```

Write a program that searches for sets of values *{a, b, c, d, e}* that make this matrix a magic square. Your program will use brute force: the main program creates 10 child processes, assigns each of them a different value for *a*, and then waits for their termination. Each child will then fill the array depending on its assigned value, and check whether the resulting square is magic: if it is, the child will then display the matrix.
You can use the skeleton code `decoder.c` to implement your solution.

Question 3 - Multi-Currency Converter

Write a program `multi_converter` that converts an amount expressed in any one of a set of predefined currencies (see file `converters.h` and the corresponding implementation `converters.c`), and displays the result of converting this amount to all the currencies in the set.
The conversion parameters are passed to the program by the user via the command line in the following format:

```
$ multi_converter <currency> <amount>
```

`currency` represents the input currency
`amount` represents the amount to be converted in the target currencies.

Your program shall operate as follows. The parent process retrieves the parameter values entered by the user, creates one child process per target currency, and then waits until all of its children terminate to notify the end of the conversion. Each child process handles a different target currency and displays the conversion result.

Output example:

```
./multi_converter CNY "100.0"
     Conversion for: CNY 100.000
          EUR 13.336
          GBP 11.186
          USD 15.009
          JPY 1521.564
          CNY 100.000
     End of conversion
```