TP 7 : Processes and Signals

1 Warming up : Forks in C

To write a C program using `fork`, you must include the following libraries in your programs :

```
#include <unistd.h>
```

You will probably also need the following libraries later :

```
#include <sys/types.h>
#include <sys/wait.h>
```

1. How many times is "Hello World" printed ?

   ```
   #include <stdio.h>
   #include <sys/types.h>
   int main()
   {
   fork();
   fork();
   fork();
   printf("Hello World\n");
   return 0;
   }
   ```

2. Given a macro `BOUND`, write a program that can create $2^{\texttt{BOUND}}$ processes.
3. Predict the output of the following code snippet :

   ```
   #include <stdio.h>
   #include <sys/types.h>
   #include <unistd.h>
   void forkexample()
   {
   // child process because return value zero
   if (fork() == 0)
   printf("Hello from Child!\n");

   // parent process because return value non-zero.
   else
   printf("Hello from Parent!\n");
   }
   int main()
   {
   forkexample();
   return 0;
   }
   ```

4. Consider the following code fragment :

```
if (fork() == 0) {
a = a + 5;
printf("a = %d, &a = %d\n", a, &a);
}
else {
a = a -5;
printf("a = %d, &a = %d\n", a, &a);
}
```

   Is the value of `a` same in both print statements ? What about the value of `&a` ?

2 Exit status

In this exercise we study the use and evaluation of the exit code of a process. Normally the exit code is used to indicate, for example, the success or failure of a program. Here, we use it to provide the results of a calculation. (Note : the exit code must be between 0 and 255, so it is really not very effective for this sort of thing.)

1. We start with a simple example `simple.c` which calculates the sum of two integers. Complete the program (you can access the skeleton from the website). The objective is that the child uses exit to communicate the sum to the parent and the parent receives this value by wait.
2. More complicated : download the archive `calc`, which contains a simple calculator that evaluates expressions with natural numbers, additions, multiplications and subtractions. In the current state the program converts the expression to a tree, then it traverses the nodes of the tree one by one to calculate the values of all the sub-expressions. Your task is to allow the program to compute the sub-expressions in parallel. When the program evaluates a sub-expression, it should start two child processes which evaluate their sub-expressions and which reports the result to the parent process by their exit codes. The parent process waits for the two results and applies the operation corresponding to get its own result. It (should) suffice to modify the compute function.

3 Signals in ANSI and POSIX

Write a program with a parent process that
— creates a child process
— transmits a sequence of keyboard-entered bits (the user enters 0's and 1's), bit by bit, to this child process.

The child process must display the sequence in the correct order as received from the parent. To do this, use signals. The difficulty with this exercise is that the order is not guaranteed : a signal A sent before signal B by the parent process can be received in the reverse order by the child process. Take care to test several patterns of sequences (alternations of 0's and 1's, successions of 0, successions of 1). To enter the bits, you can use

```
for (c = 0; c != 0 && c != 1 && c != EOF; c = getchar());
if (c == 0) // Do something
if (c == 1) // Do something
if (c == EOF) break;
```

1. First implement with the ANSI C API (kill, pause, signal functions). What issue do you face ?
2. Improve your program by using POSIX.