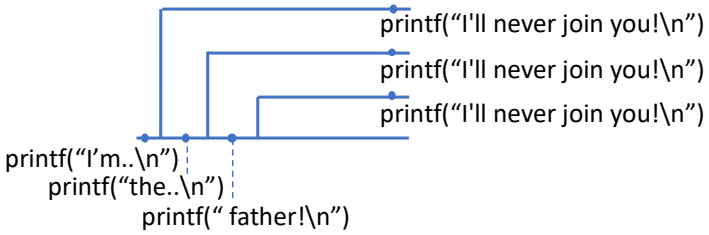
Creating Processes with `fork()` — SCOMP-PL1

1. Analyse the following code.

```
1  int main(void) {
2    int x = 1;
3    pid_t p = fork(); /*pid_t: sys/types.h; fork(): unistd.h*/
4    if (p == 0) {
5      x = x+1;
6      printf("1. x = %d\n", x);
7    } else {
8      x = x-1
9      printf("2. x = %d\n", x);
10   }
11   printf("3. %d; x = %d\n", p, x);
12 }
```

   a) Assume that the PID of the child process is 1234. What is the output of this code in the terminal? Justify your answer.
   b) Are you always guaranteed to see the output of the `printf()` in line 9 before the one in line 11? Explain.

2. Write a program that creates the process tree below and where each process executes the `printf()` calls as depicted.



3. Analyse the following code.

```
1  int main(void) {
2    fork(); fork(); fork();
3    printf("SCOMP!\n");
4  }
```

   a) How many processes are created by this code?
   b) Draw a process tree that describes the processes created.
   c) How many times is "SCOMP" printed?

4. Analyse the following code.

```
1  int main(void) {
2    int a=0, b, c, d;
3    b = (int) fork();
4    c = (int) getpid(); /* getpid(), getppid(): unistd.h*/
5    d = (int) getppid();
6    a = a + 5;
7    printf("\na=%d, b=%d, c=%d, d=%d\n",a,b,c,d);
8  }
```

   a) Which of the variables **a**, **b**, **c** and **d** will have the same value in both processes?
   b) Draw a process tree that describes the processes created.

Creating Processes with `fork()` — SCOMP-PL1

5. Write a program that creates two child processes, the father waits for each one of them to terminate and then outputs their exit value. The first child process created should sleep for 1 second and then exit with a return value of 1. The second child created should sleep for 2 seconds and exit returning 2.

6. Given the following code.

```
1  void main()
2  {
3   int i;
4   int status;
5
6   for (i = 0; i < 4; i++) {
7       if (fork() == 0) {
8           sleep(1); /*sleep(): unistd.h*/
9       }
10  }
11
12  printf("This is the end.\n").
13 }
```

http://codepad.org/GKp8fDKM

a) How many processes will be created by this code? Justify by drawing a process tree that describes the processes created.
b) What change, if any, would you do to this code so that exactly 4 child processes are created?
c) Assuming the changes in b), change the code so that the parent process waits for child processes with an even PID.
d) Assuming the changes in b) and c), change the code so that the child processes return a number that reflects their creation order (that is, the first child process returns 1, the second returns 2, …).

7. Given the code below, write the code necessary to create a child process to find how many times the number **n** is found in half of the elements of the **`numbers`** array. While the child is processing, the parent process should search the other half. After both finish processing their half, the parent computes and presents how many times **n** was found in the entire array (assume that no number is repeated more than 255 times).

```
1  #define ARRAY_SIZE 1000
2  int main ()
3  {
4    int numbers[ARRAY_SIZE];    /* array to lookup */
5    int n;                      /* the number to find */
6    time_t t;                   /* needed to init. the random number generator (RNG)
   */
7    int i;
8
9    /* intializes RNG (srand():stdlib.h; time(): time.h) */
10   srand ((unsigned) time (&t));
11
12   /* initialize array with random numbers (rand(): stdlib.h) */
13   for (i = 0; i < ARRAY_SIZE; i++)
14     numbers[i] = rand () % 10000;
15
16   /* initialize n */
17   n = rand () % 10000;
18 }
```

http://codepad.org/I3JfIuuc

Creating Processes with `fork()` — SCOMP-PL1

8. Compile and execute the code below, sending the main process to the background (in Linux, you tell the shell to execute a process in the background by adding "&" at the end). Observe that the parent process will execute a loop forever.

```
1  int main()
2  {
3   pid_t p;
4
5   if (fork() == 0) {
6      printf("PID = %d\n", getpid()); exit(0);
7   }
8
9   if ((p=fork()) == 0) {
10     printf("PID = %d\n", getpid()); exit(0);
11  }
12
13  printf("Parent PID = %d\n", getpid());
14
15  printf("Waiting... (for PID=%d)\n",p);
16  waitpid(p, NULL, 0);
17
18  printf("Enter Loop...\n");
19  while (1); /* Infinite loop */
20 }
```

http://codepad.org/BzRKc3Oo

   a) In the shell, list the running processes with **ps** (look for the PIDs printed). Are all parent and child processes listed? Why?
   b) Is there anything particular about the child process? Explain.
   c) Kill the parent process: `kill -KILL <PID>`

9. Write a program that creates 10 child processes. Each child process should print 100 numbers according to its creation order. That is the first child process created writes numbers 1..100, the second child process 101..200, the third 201..300, and so on. The parent process should wait for all child processes to finish.
   a) Is the output sorted? Can you guarantee it will be always sorted? Why?

10. Write a program that initializes an array of 2 000 random integers. Then, the program should create 10 child processes that will find the first occurrence of a number n and return the relative index where it was found within the section of the array it computed (that is, the index returned is a value in the range [0,200[). If no number is found, the child returns 255. The parent should wait for the child processes to finish and output the valid indexes returned.

11. Write a program that initializes an array **`numbers`** with 1000 random integers in the range [0,255]. The program should:
   a) create 5 child processes that will concurrently find the maximum value of 1/5 of the array;
   b) after computing the maximum value of the entire array, the parent process should create one child process to perform the following calculation `result[i]=((int) numbers[i]/max_value)*100` on half of the array and print the result;
   c) the parent process should perform the same calculation on the other half of the array;
   d) both child and parent process must perform the computation concurrently, but the output must be sorted by the array indexes.

12. Write a function **`spawn_childs(int n)`** which generates **`n`** child processes.
    a) Ensure that the function returns to the child process an index (1 for the first child process created, 2 for the second, 3 for the third and so on) and that to the parent process returns 0.
    b) Write a program that uses the function **`spawn_childs()`** to create 6 child processes. Each child process should exit returning its index value * 2. The parent should wait for all processes created.