Problem Set 01 - Processes

Reminders

The system call `fork ()` creates a child process that differs from its parent only by its `pid` and `ppid` values. `fork ()` returns 0 to the child process and the child's pid to the parent process. Right after the `fork()`, both processes have identical copies of their data and stack segments. But there is no sharing: each process has its own copy in a separate namespace.
The system call `wait()` blocks a process waiting for the end of the execution of any child it created. `wait()` returns the pid of the child that just ended. `wait()` takes a parameter of type `int *` that retrieves information about the termination of the child.
The family of `exec` system calls launch an executable program. If the call is successful, the code of the caller process gets replaced by that of the called program: there is no return from an `exec` unless the call fails (and only in this case).

Question 1. Execution of a simple fork

Consider the following C program

```
int main () {
      int pid;
      printf ( "Begin\n" );
      pid = fork ();
      if (pid == 0) {
            printf ( "execution 1 \n");
      } else {
            printf ( "execution 2 \n");
      }
      printf ( "End \n");
      return EXIT_SUCCESS;
}
```

Q1.1 Give the displays produced by the parent process and by the child process.

Child: "execution 1"
"End"

Parent: "Begin"
"execution 1"
"End"

Question 2. Execution of a nested fork

Consider the following program:

```
1   int main (int argc, char * argv []) {
2       int a, e,
3       a = 10;
4       if (fork () == 0) {
5           a = a * 2;
6           if (fork () == 0) {
7               a = a + 1;
8               exit (2);
9           }
10          printf ( "%d \n", a );
11          exit (1);
12      }
13      wait (&e);
14      printf ( "a:%d; e:%d \n", a, WEXITSTATUS (e));
15      return 0;
16  }
```

Q2.1 Give the number of processes created by this program and the displays produced by each process.

3 processes: parent (P), child (C), and grandchild (G)

GC: No display

C: "20"

P: "a:10; e:1"

Consider a new version of the same program where line 8 has been deleted.

Q2.2 Repeat Q2.1 with the new version.

3 processes: parent (P), child (C), and grandchild (G)

GC: "21"

C: "20"

P: "a:10; e:1"

Q2.3 Change the initial program to create a zombie process for 30 seconds.

Add `sleep(30)` between lines 12 & 13

Question 3. Creating chains of processes

We want a program that creates a chain of processes such that the initial process (the one started by the bash) creates a process, which in turn creates a second process, and so on until the creation of N process (in addition to the initial process).

Q3.1 Write this program. Represent the family tree of processes when N = 3.

```
#define N 3
int main (int argc, char* argv []) {
    int i;
    for (i = 0; ((i < N) && (fork() == 0)); i++)
        /* empty instruction */;
    return EXIT_SUCCESS;
}
```

P => C => GC => GGC

Q3.2 Modify the program so that the initial process waits until the end of its child.

```
int main (int argc, char* argv []) {
    int i;
    for (i = 0; ((i < N) && (fork() == 0)); i++)
        /* empty instruction */;
    if (i == 0)
        wait(NULL);
    return EXIT_SUCCESS;
}
```

Q3.3 Modify the program so that the initial process waits until the end of all the processes in the chain.

```
int main (int argc, char* argv []) {
    int i;
    for (i = 0; ((i < N) && (fork() == 0)); i++)
        /* empty instruction */;
    if (i < N)
        wait(NULL);
    return EXIT_SUCCESS;
}
```

Question 4. Code replacement

Consider the following program:

```
int main () {
	int p;
	p = fork ();
	if (p == 0) {
		execl("/bin/echo", "echo", "In", "the", "child", NULL);
	}
	wait (NULL);
	printf ( "In the parent\n");
	return EXIT_SUCCESS;
}
```

Q4.1 Assuming that the execl call is successful, give the result produced by running this program.

Child displays "In the child", then parent displays "In the parent", always in that order.

Now consider the program "nemaxe", where "prog" is a program that does not create any other process:

```
main(int argc, char*argv[]) {
	int rcode;
	printf("%s\n", argv[0]);
	switch (fork()) {
		case -1:
			perror("fork1");
			exit(1);
		case 0:
			switch (fork()) {
				case -1:
					perror("fork2");
					exit(1);
				case 0:
					if (execl("./prog","prog",0) == -1) {
						perror("execl");
						exit(1);
					}
					break;
				default:
					exit(0);
			}
		default:
			wait(&rcode);
	}
}
```

Q4.2 Represent the family tree of processes obtained by running this program.

P => C => GC

Q4.3 What are the possible sequences of process terminations?

P waits for C so 3 possible sequences: {GC; C; P}, {C; GC; P}, {C; P; GC}

Now we change the name of the executable from "nemaxe" to "prog" (the very same name as the argument in the execl call).

Q4.4 Represent the family tree of processes obtained by running this new program. What is wrong?

P => C => GC
(P) => C' => GC'
(P) => C'' …

The replacing code calls itself without any stopping condition, so it creates an infinite recursion.

The name of the executable is still "prog"

Q4.5 Suggest a code modification so that only one recursive call occurs.

Change the code of GC as follows:

```
if (argc == 1) {
      if (execl("./prog","prog", "1", NULL) == -1) {
            perror("execl");
            exit(1);
      }
}
break;
```