Lab #7: Processes

Overview

The purpose of this lab is to gain experience with `fork` and `exec`.

Limiting Processes

We are now working with `fork` so there is a real risk that a simple bug in your program will result in many, many processes being spawned (a "fork bomb"). Here are two techniques that you need to use to limit the impact of such a mistake.

- Limit in the shell: Use `ulimit -Su 10` to limit the number of processes that you can spawn (you can choose a number other than 10, but choose something relatively small). You must type this each time you start a new shell.
- Limit in your program: Since you may forget the above when you open a new shell, you might prefer to add the following function to your program (this is linked on the course website as separate header and course files) and call it as the first action in `main`.

```
void limit_fork(rlim_t max_procs)
{
   struct rlimit rl;

   if (getrlimit(RLIMIT_NPROC, &rl))
   {
      perror("getrlimit");
      exit(-1);
   }

   rl.rlim_cur = max_procs;

   if (setrlimit(RLIMIT_NPROC, &rl))
   {
      perror("setrlimit");
      exit(-1);
   }
}

int main(void)
{
   limit_fork(10);

   /* continue with program logic here */

   return 0;
}
```

Part 1: `fork`

Write a program named `f_test`. This program must take a single integer, `N`, as a command-line argument (look at `atoi` or, even better, `strtol` to convert a string into an integer). This program will first fork a child process. Then, the child must print the odd numbers from 1 to `N` (inclusive) while the parent prints the even numbers from 1 to `N` (inclusive). The parent process should properly `wait` for the child process to terminate.

For the odd numbers, use "%d\n" as the format string for printf. For the even numbers, use "\t%d\n" as the format string for printf.

Part 2: `odds` and `evens`

Write two C programs. They must each take a single integer, `N`, as a command-line argument. `odds` prints the odd numbers from 1 to `N` (inclusive). `evens` prints the even numbers from 1 to `N` (inclusive).

Part 3: `exec`

Write a program named `fe_test`. This program will behave similarly to the program from the first part, but will use `exec` to execute the programs written in the second part. The parent should fork two child processes. One child process will `exec` the `odds` program. The other will `exec` the `evens` program. The parent process should properly `wait` for both child processes to terminate but allow them to execute concurrently (i.e., `wait` after both children have been created).

Part 4: Redirection

Write a program named `to_file`. This program will take two command-line arguments. The first is the name of another program and the second is the name of a file (which may not yet exist).

Your program is going to use `exec` to run the specified program. Before doing so, however, your program will need to open the specified file and take the appropriate steps to redirect standard output to the specified file. This setup is done so that the `exec`'d program will write its output to the file.

Demonstration

Demonstrate your programs to the instructor.