SLP-assignment #8: mish

(22 points, in groups of two)

Design and implement a shell `mish` (**mi**ni **sh**ell) that can execute programs (hereafter referred to as commands). Use the file name `mish.c` for your implementation. The assignment is divided into three parts. Additionally, there are templates made available for reference.

Hint: Since parts b) and c) require the additional use of signals, it is recommended to start with part a) and to test this implementation thoroughly.

Part a): Basic Functionality

In this part, the basic functionality for interacting with the `mish` has to be implemented.

- Your program has to print the string

  ```
  mish>
  ```

  as a prompt symbol and then wait for the input of a command.
- The input line has to be divided into command name and additional arguments. Spaces and tabulators thereby serve as delimiters (`fgets(3)`, `strtok(3)`).
- Then, the command shall be executed in a newly created process (`fork(2)`) with the correct arguments (`execvp(3)`).
- The `mish` shall wait for the process to terminate (`waitpid(2)`) and print the exit status on the standard error stream. For the output of the status, you should differentiate whether the process terminated itself (`WIFEXITED`, `WEXITSTATUS`) or due to a signal (`WIFSIGNALED`, `WTERMSIG`). Furthermore, the `PID` of the process has to be printed:

  1. Case: the process terminates itself (here with exit status 0):

     ```
     mish> ls -l
     ...
     Exit status [2110] = 0
     ```

  2. Case: the process was terminated by a signal (here by an interrupt signal (`SIGINT=2`) triggered by pressing `CRTL-C`):

     ```
     mish> sleep 10
     Signal [1302] = 2
     ```

     You can test the output of the exit status with the programs `spic-wait` and `spic-exit`.
- After the exit status has been printed, the `mish` has to accept a new input. The `mish` shall terminate when it reads a end of file (`CRTL-D`) from the standard input channel.

Part b): Signals

In this part, the `mish` has to be extended to handle the signals `SIGINT` and `SIGCHLD`.

- **Ignoring the signal** `SIGINT`

  When you trigger a signal in a terminal window, e.g. `SIGINT` by pressing `CRTL-C`, this signal is then delivered to all running processes in the terminal, which in particular includes the `mish` and all currently running child processes.
  Extend the `mish` in a way, such that the signal `SIGINT` is ignored (`sigaction(2)`) only by the `mish` (and not the created child processes!). Now, you can interrupt running child processes by pressing `CRTL-C`, without terminating the `mish`.
- **Handling of the signal** `SIGCHLD`

  In preparation of part c), only non-blocking calls to `waitpid(2)` have to be used and the use of `wait(2)` has to be omitted. When a child process terminates, the `mish` receives a signal `SIGCHLD`. To wait for the termination of a child process, you can use `sigsuspend(2)` to wait for the occurrence of a signal `SIGCHLD`.

  - Using `sigaction(2)`, register a custom signal handling function for `SIGCHLD`. An occurrence of `SIGCHLD` has to be noted inside the corresponding handler by the use of an event variable.
  - Before a new prompt is printed, you have to wait for the termination of the child process. It can be necessary to wait (possibly multiple times) with `sigsuspend(2)` for the occurrence of `SIGCHLD`. Make sure to correctly synchronize your program by using `sigprocmask(2)`.
  - If `SIGCHLD` occurred, the terminated background process has to be collected with `waitpid(2)`. Like before, the exit status has to be printed out.

Part c): Background Processes

Extend your implementation of `mish` to support background processes that are indicated by the use of the symbol '`&`' at the end of the command line. When a command ends with '`&`', it has to be executed in the background. In this case, the `mish` does not have to wait for the termination of the child process and, instead, only print the process ID of the background process and immediately print a new prompt for accepting new commands. Background processes also have to ignore `SIGINT` and shall **not** be terminated by pressing `CRTL-C`.

The execution of a background process in the `mish` could look like this:

```
mish> sleep 10 &
Started [2110]
mish> [...]
[...]
Exit status [2110] = 0
```

To support both, foreground as well as background processes, the handling of zombie processes has to be extended. As before, zombie processes have to be collected right before printing a new prompt, but from now on, the following steps must be taken:

- If no foreground process is active, before printing the prompt, you only have to check if a `SIGCHLD` has occurred since the last time zombie processes have been collected. If this is the case, all accumulated zombie processes have to be collected with `waitpid(2)` and their exit status has to be printed out.
- Pay attention to the fact that during the collection of already terminated zombie processes, new background processes could terminate and therefore would need to be collected as well.
- If a foreground process is active, before printing the prompt, you have to wait for the termination of the foreground process. While waiting, the exit status of intermediately terminating background processes has to be printed out (`waitpid(2)`). Therefore it can be useful to (possibly multiple times) wait for the occurrence of `SIGCHLD` with `sigsuspend(2)`, before the foreground process finally terminates. Here, like before, for each collected child process the status has to be printed out.

Refer remarks on the next page.

Hints:

- To simplify the assignment, you can assume that a command line consists of at most 1023 characters. All other cases can be addressed with an appropriate error message.
- The program `/proj/i4spic/<login>/pub/aufgabe8/spic-wait` is well suited for testing the signal handlers. After starting it, it prints its process ID and waits for the delivery of a signal. You can send an arbitrary signal to the process by executing the command `kill(1)`. Alternatively, you can pass a signal number to `spic-wait`, which is then used by the program to terminate itself.
- The program `/proj/i4spic/<login>/pub/aufgabe8/spic-exit` is well suited for testing the output of exit status. The program terminates itself with the passed parameter as its exit code.
- Inside the directory `/proj/i4spic/<login>/pub/aufgabe8/` you can find the template (`mish_vorlage.c`) that can be used for the basic structure and some function signatures but has to be completed by you. The template might help you structure the program.
- In the directory `/proj/i4spic/<login>/pub/aufgabe8/` you can find the files `mish_teil_a`, `mish_teil_b` and `mish_teil_c`, which contain a reference implementation of the corresponding parts of the assignment.
- Always give a reason why you use the `volatile` keyword. If the same reasoning holds for multiple variables, you can justify them together.
- In the directory `/proj/i4spic/<login>/pub/aufgabe8/` you will find the file `mish` that contains a reference implementation.
- Always make sure to give out meaningful error messages on the standard error stream. (`fprintf(stderr,...)(3)` / `perror(3)`)
- You can test your program with `valgrind`. This may help when searching for errors. *Suppressed errors* can be ignored. More error messages can be obtained by using `valgrind` with the flags `--leak-check=full --show-reachable=yes` and building the binary program with debug symbols.
- Your program has to compile with the following flags:
  ```
  gcc -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3 -o mish mish.c
  ```
  This configuration is also used for grading.
- Functions of the `libc` that do not require error handling in SLP can be seen online in the Linux `libc`-Doku.
- You are free to write a `makefile` that includes instructions on how to build the program with the tool `make`. To do this, you can create a file called `Makefile` inside the submission directory (`aufgabe8/`). In the first line write
  ```
  CFLAGS = -std=c11 -pedantic -D_XOPEN_SOURCE=700 -Wall -Werror -O3
  ```
  Then you can build the file from the terminal by calling `make mish` or with the `make` button inside the SPiC-IDE.

Deadline

T01	14.07.2024	18:00:00
T02	14.07.2024	18:00:00
T03	15.07.2024	18:00:00
T04	16.07.2024	18:00:00
T05	16.07.2024	18:00:00
T06	17.07.2024	18:00:00
T07	17.07.2024	18:00:00
T08	18.07.2024	18:00:00
T09	15.07.2024	18:00:00