Operating systems

Programming Assignments 1

The students are encouraged to convey additional experiments related to exercised topics and share the results with rest of the group. A student should know how to use following tools `ps`, `kill`, `lsof`, `strace` and `ltrace`. If a question occurs in contents of an assignment, you have to provide answer as code, whenever possible.

Reading of chapters 7 through 12 of “Advanced Programming in the UNIX Environment” (2nd edition) is highly recommended for ambitious students.

Each assignment is worth 2 points. An assignment marked with plus sign are worth also 2 points, but they’re not counted into sum of points the list is worth. In other words there’s a possibility to get for example 120% points for a list.

Assignment 1

Write a program, that creates a zombie process. In parent process run `ps` command in order to prove, that child process has entered zombie state.

Assignment 2

Write a program, that registers at least one call-back function through `atexit` call. Check if the function is called when program terminates by following means: call to `abort` or `exit` functions and as a result of signal reception. Does the child process created by call to `fork` inherits the function?

Assignment 3

Simulate a call to `system` function ie. read command line arguments given to your program and forward them to `execve` call in child process. Wait until the offspring terminates and obtain its exit code. If the child process was terminated as a result of signal reception, then print its number (with description if possible), otherwise print the exit code.

Assignment 4

Print process’ environment using `getenv` and `getcwd` functions. Create a child process. If you modify environment or current working directory in the parent, will that change be visible to the offspring? Open a file in the ancestor using `open` function. Is the result of following actions: `read` and `close` visible to the other process (eg. does a file descriptor close action affect both processes, similarly does a file read action move file pointer in both processes) ?

Assignment 5

Construct a program that intercepts `SIGTERM` signal (ie. process termination request) and ignores `SIGINT` signal (ie. process interruption from keyboard). Signal handler routines should print out relevant messages. Embed previous program into a function and run it as a subprocess (using `fork` call), then make the parent process send following signals `SIGTERM`, `SIGINT` and `SIGKILL` sequentially to the child. Can you intercept `SIGKILL` signal? After you intercepted `SIGINT` is it possible to restore old signal handler for that signal?

Assignment 5+

Intercept `SIGSEGV` signal (eg. memory protection violation) using `sigaction` call, then provide interpretation for data associated with the signal (ie. look up `siginfo_t` structure). Print your interpretation as a message on `stderr` (ie. standard output for error messages). After you establish signal handler, refer to a memory location that is obviously out of reach (eg. to some element of an array to which pointer was initialized to `NULL` value).

Assignment 6

Create a handful of POSIX threads using `pthread_create` function. Make them print their identifiers and wait for a few seconds (ie. `sleep` call) before they terminate. In the meantime let the main thread show output from `ps` command (run it through `system` function) - the printout should show threads you created in the process. Make sure that the main threads does not terminate before child threads finish (use `pthread_wait` function). Why does POSIX threads library distinguish between detached and attached threads?

Assignment 6+

Modify solution for previous assignment in such a way, that one of the threads calls `fork` function. What happens to threads during and after process has been cloned? Are you able to give arguments in favour of such design?