Operating Systems
Constructor University
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2024-09-05
Due: 2024-09-13

Problem Sheet #1

Problem 1.1: *library and system calls* (1+1 = 2 points)

Answer the following questions by using `strace` and `ltrace` on a Linux system. Provide enough context information to make it clear how the results were obtained.

a) How many system calls and how many library calls does executing `/bin/date` produce?

b) What are the most frequent (top three) library and system calls and what do these calls do?

Problem 1.2: *system call errors* (1+1 = 2 points)

System call errors are usually indicated by returning a special value (usually -1 for system calls that return an int) and by indicating the details in the global variable `int errno`, declared in `errno.h`.

a) For each of the following system calls, describe a condition that causes it to fail (i.e., a condition that causes -1 to be returned and that sets `errno` to a distinct value).

- `int open(const char *path, int oflag, ...)`
- `int close(int fildes)`

b) What is the value of `errno` after a system call completed without an error?

Problem 1.3: *execute a command in a modified environment or print the environment* (6 points)

On Unix systems, processes have access to environment variables that can influence the behavior of programs. The global variable `environ`, declared as

```
extern char **environ;
```

points to an array of pointers to strings. The last pointer has the value `NULL`. By convention, the strings have the form "name=value" and the names are often written using uppercase characters. Examples of environment variables are `USER` (the name of the current user), `HOME` (the current user's home directory), or `PATH` (the colon-separated list of directories where the system searches for executables).

Write a program `env` that implements some of the functionality of the standard `env` program. The syntax of the command line arguments is the following:

```
env [OPTION]... [NAME=VALUE]... [COMMAND [ARG]...]
```

a) If called without any arguments, `env` prints the current environment to the standard output.

b) If called with a sequence of "name=value" pairs and no further arguments, the program adds the "name=value" pairs to the environment and then prints the environment to the standard output.

c) If called with a command and optional arguments, `env` executes the command with the given arguments.

d) If called with a sequence of "name=value" pairs followed by a command and optional arguments, the program adds the "name=value" pairs to the environment and executes the command with the given arguments in the modified environment.

e) If called with the option `-v`, the program writes a trace of what it is doing to the standard error.

f) If called with the option `-u name`, the program removes the variable `name` from the environment.

Here are some example invocations:

```
$ env                          # print the current environment
$ env foo=bar                  # add foo=bar and print the environment
$ env -u foo                   # remove foo and print the environment
$ env date                     # execute the program date
$ env TZ=GMT date              # add TZ=GMT and execute the program date
$ env -u TZ date               # remove TZ and execute the program date
$ env -u x a=b b=c date        # remove x, add a and b, execute date
```

Hand in the source code of your `env` program. Make sure that your program handles *all* error situations appropriately. Use the `getopt()` function of the C library for parsing command line options. Furthermore, use one of the `exec` system calls like `execvp()` to execute a command. (Using `system()` can be made to work but it is somewhat difficult to get right since concatenating strings using space characters may lead to surprises if the strings themselves contain space characters; to do this correctly, you have to quote the strings such that the shell called by the `system()` library function tokenizes the string properly again. Naive concatenation usually leads to a security weakness, it is often better to avoid the `system()` library function. See also the Caveats section in the Linux manual page describing the `system()` library function.)