

OS 2022 Problem Sheet #1

Problem 1.1: *freshie crash*

(2 points)

A freshmen is learning the C programming language. He wrote the following program but it keeps crashing or producing unexpected outputs. Explain why the program crashes or produces unexpected outputs.

```
1  #include <string.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  char* strdup(const char *s)
6  {
7      size_t len = strlen(s);
8      char d[len+1];
9      return strncpy(d, s, len+1);
10 }
11
12 int main(int argc, char *argv[])
13 {
14     int i;
15
16     for (i = 1; i < argc; i++) {
17         (void) puts(strdup(argv[i]));
18     }
19
20     return EXIT_SUCCESS;
21 }
```

Problem 1.2: *memory segments*

(2 points)

Look at the following program and write down what is stored in the text segment, the data segment, the heap segment, and the stack segment.

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4
5  char *strdup(const char *s)
6  {
7      char *p = NULL;
8      size_t len;
9
10     if (s) {
11         len = strlen(s);
12         p = (char *) malloc(len+1);
13         if (p) {
14             strcpy(p, s);
15         }
16     }
17     return p;
18 }
19
20 int main()
```

```

21 {
22     static char m[] = "Hello World!";
23     char *p = strdup(m);
24     if (!p) {
25         perror("strdup");
26         return EXIT_FAILURE;
27     }
28     if (puts(p) == EOF) {
29         perror("puts");
30         return EXIT_FAILURE;
31     }
32     if (fflush(stdout) == EOF) {
33         perror("fflush");
34         return EXIT_FAILURE;
35     }
36     return EXIT_SUCCESS;
37 }

```

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 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:

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

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.)