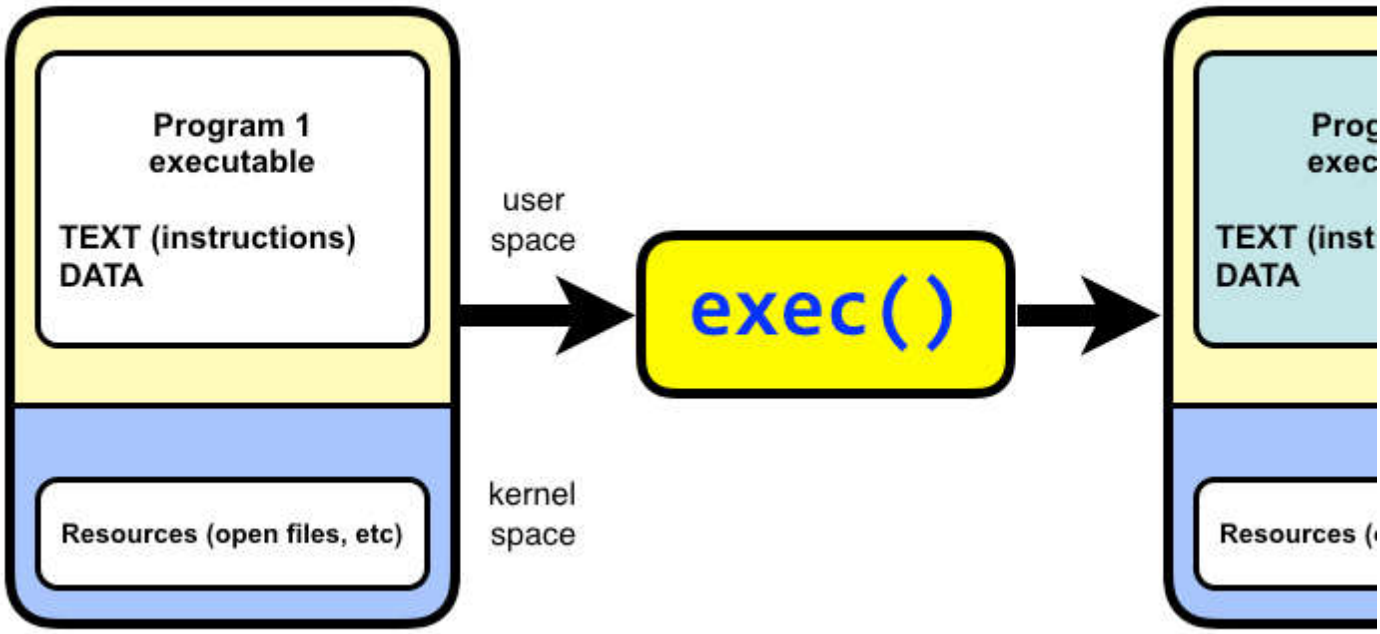
Read - The Exec Family

The exec family of system calls

The exec family of system calls replaces the program executed by a process. When a process calls exec, all code (text) and data in the process is lost and replaced with the executable of the new program. Although all data is replaced, all open file descriptors remains open after calling `exec` unless explicitly set to close-on-exec. In the below diagram a process is executing Program 1. The program calls `exec` to replace the program executed by the process to Program 2.



execlp

The `execlp` system call duplicates the actions of the shell in searching for an executable file if the specified file name does not contain a slash (/) character. The search path is the path specified in the environment by the `PATH` variable. If this variable isn't specified, the default path `":/bin:/usr/bin"` is used.

The `execlp` system call can be used when the number of arguments to the new program is known at compile time. If the number of arguments is not known at compile time, use execvp.

```
#include <unistd.h>

int execlp(const char *file, const char *arg, ...);
```

file

Name of the program to execute.

Remaining arguments

The `const char *arg` and subsequent ellipses can be thought of as `arg0, arg1, ..., argn`. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. The first argument, by convention, should point to the filename associated with the file being executed. The list of arguments must be terminated by a NULL pointer.

Example

In `module-2/examples/src/execlp_ls.c` (https://canvas.instructure.com/courses/1793037/files/86164726/download?wrap=1) you find the following example program demonstrating how `execv` can be used.

```
#include <unistd.h> // execlp()
#include <stdio.h>  // perror()
#include <stdlib.h> // EXIT_SUCCESS, EXIT_FAILURE

int main(void) {
  execlp("ls", "ls", "-l", NULL);
  perror("Return from execlp() not expected");
  exit(EXIT_FAILURE);
}
```

The program uses `execlp` to search the PATH for an executable file named `ls` and passing `-l` as argument to the new program. The new program is the same program used by the shell command `ls` to list files in a directory.

Use `make` to compile:

```
$ make
```

Run the program.

```
$ ./bin/execlp_ls
```

You should see something similar to this in the terminal.

```
-rw-r--r--@  1 lucas staff  410 Jan 27 21:16 Makefile
drwxr-xr-x  17 lucas staff  578 Jan 28 22:08 bin
drwxr-xr-x   3 lucas staff  102 Dec  1  2016 data
drwxr-xr-x   2 lucas staff   68 Jan 28 22:08 obj
drwxr-xr-x  17 lucas staff  578 Jan 28 22:08 src
```

path
The path (https://en.wikipedia.org/wiki/Path_(computing)) to the new program executable.
file
The name of the program executable
path

execvp

The `execvp` system call will duplicate the actions of the shell in searching for an executable file if the specified file name does not contain a slash (/) character. The search path is the path specified in the environment by the `PATH` variable. If this variable isn't specified, the default path `":/bin:/usr/bin"` is used. In addition, certain errors are treated specially.

```
#include <unistd.h>

int execvp(const char *file, char *const argv[]);
```

file
Name of the program to execute.
argv
Argument vector. An array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the filename associated with the file being executed. The array of pointers must be terminated by a NULL pointer.

Example

In `module-2/examples/src/execvp_ls.c` you find the following example program demonstrating how `execvp` can be used.

```
#include <unistd.h> // execvp()
#include <stdio.h>  // perror()
#include <stdlib.h> // EXIT_SUCCESS, EXIT_FAILURE

int main(void) {
  char *const cmd[] = {"ls", "-l", NULL};
  execvp(cmd[0], cmd);
  perror("Return from execvp() not expected");
  exit(EXIT_FAILURE);
}
```

The program uses `execvp` to search the PATH for an executable file named `ls` and passing `-l` as argument to the new program. The new program is the same program used by the shell command `ls` to list files in a directory. In comparison to using `execv` we don't have to provide the full path to `ls` when using `execvp`, only the name of the executable.

Use `make` to compile:

```
$ make
```

Run the program.

```
$ ./bin/execvp_ls
```

You should see something similar to this in the terminal.

```
total 8
-rw-r--r--@ 1 abcd1234  staff  410 Jan 27 21:16 Makefile
drwxr-xr-x  5 abcd1234  staff  170 Jan 27 21:17 bin
drwxr-xr-x  2 abcd1234  staff   68 Jan 27 21:17 obj
drwxr-xr-x  5 abcd1234  staff  170 Jan 27 21:16 src
```

execv

In comparison to `execvp` the `execv` system call doesn't search the PATH. Instead, the full path to the new executable must be specified. .

```
#include <unistd.h>

int execv(const char *path, char *const argv[]);
```

path

The path (https://en.wikipedia.org/wiki/Path_(computing)) to the new program executable.

argv

Argument vector. The `argv` argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in `argv[0]` should point to the filename of the executable for the new program.

Example

In `module-2/examples/src/execv_ls.c` you find the following example program demonstrating how `execv` can be used.

```
#include <unistd.h> // execv()
#include <stdio.h>  // perror()
#include <stdlib.h> // EXIT_SUCCESS, EXIT_FAILURE

int main() {
  char *const argv[] = {"/bin/ls", "-l", NULL};
  execv(argv[0], argv);
  perror("Return from execv() not expected");
  exit(EXIT_FAILURE);
}
```

The program uses `execv` to replace itself with the `/bin/ls` program passing `-l` as argument to the new program. The new program is the same program used by the shell command `ls` to list files in a directory.

Use `make` to compile:

```
$ make
```

Run the program.

```
$ ./bin/execv_ls
```

You should see something similar to this in the terminal.

```
total 8
-rw-r--r--@ 1 abcd1234  staff  410 Jan 27 21:16 Makefile
drwxr-xr-x  5 abcd1234  staff  170 Jan 27 21:17 bin
drwxr-xr-x  2 abcd1234  staff   68 Jan 27 21:17 obj
drwxr-xr-x  5 abcd1234  staff  170 Jan 27 21:16 src
```