EE3233 Systems Programming for Engrs

Reference: M. Kerrisk, The Linux Programming Interface

Lecture 12

Program Execution

ELECTRICAL & COMPUTER ENGINEERING

Executing a New Program: execve()

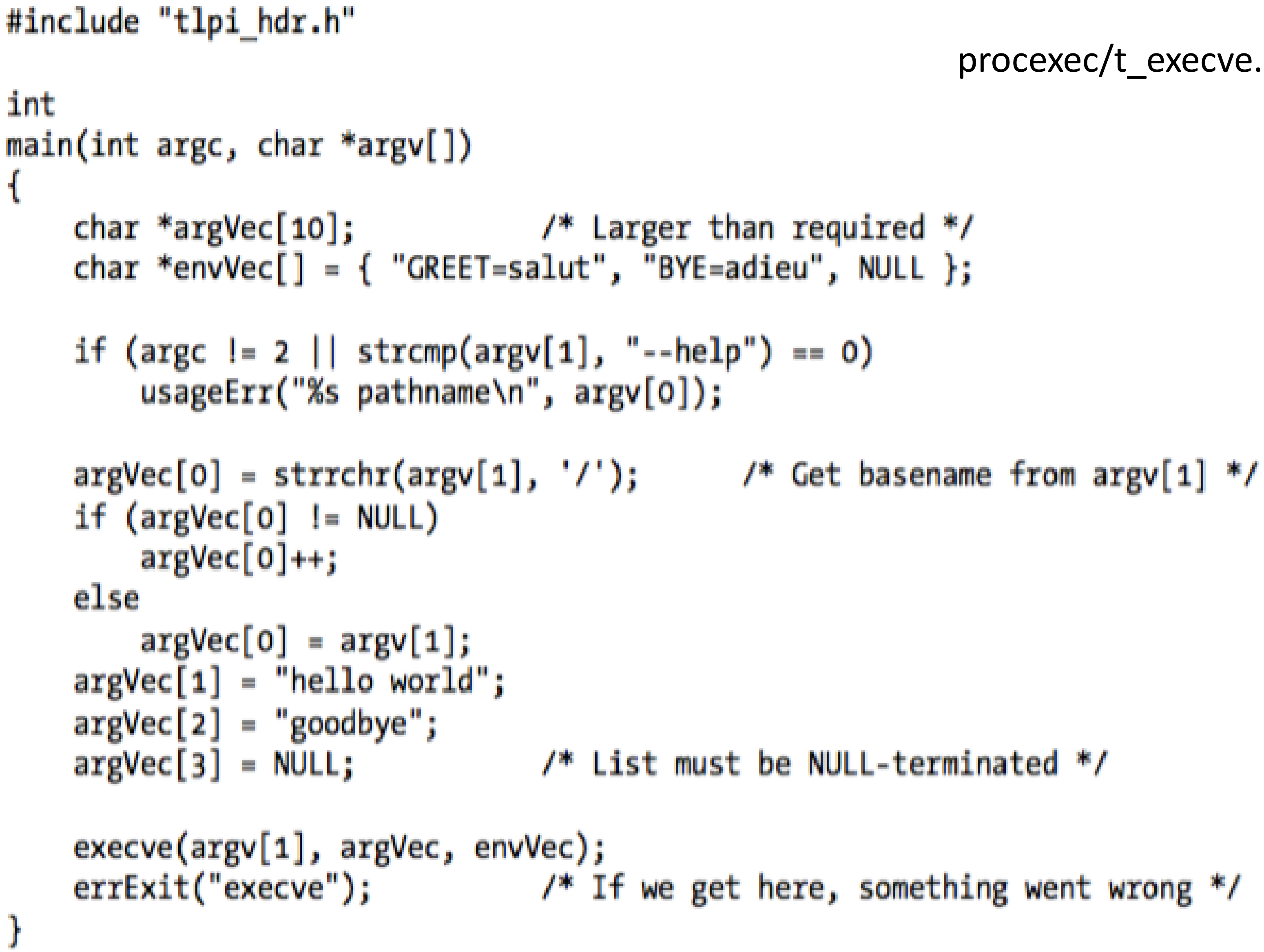
- *execve()* system calls loads a new program into a process's memory
  - The old program is discarded, and the process's stack, data, and heap are replaced by new program
  - New program commences execution at its *main()* function
- The most frequent use of *execve()* is in the child produced by a *fork()*
- Various library functions (beginning with exec) are layered on top of the *execve()* system call

Executing a New Program: execve()

```
#include <unistd.h>

int execve (const char *pathname, char *const argv[], char *const envp[]);
                                    Never returns on success; return -1 on error
```

- *pathname*: pathname of the new program
- *argv*: command line arguments to be passed to the new program
- *envp*: environment list (name=value)
- After execve(), the process ID remains the same

Executing a New Program: execve()

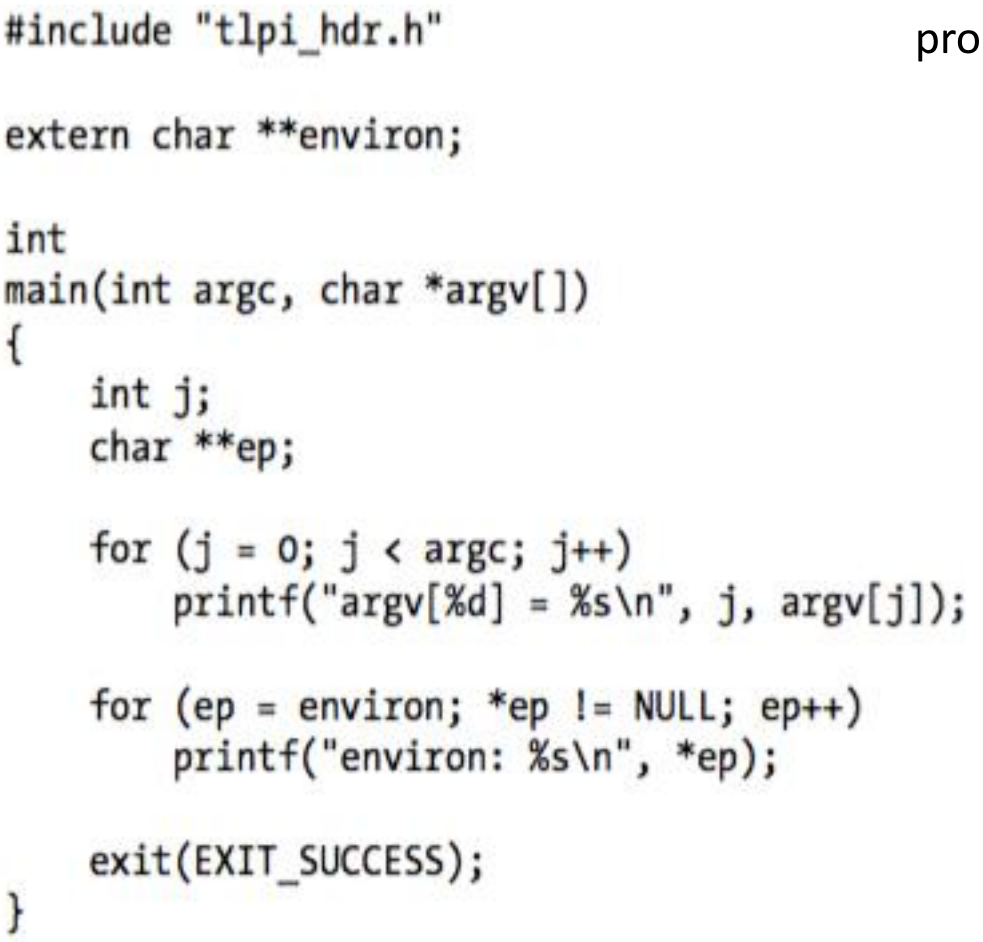


procexec/t_execve.c

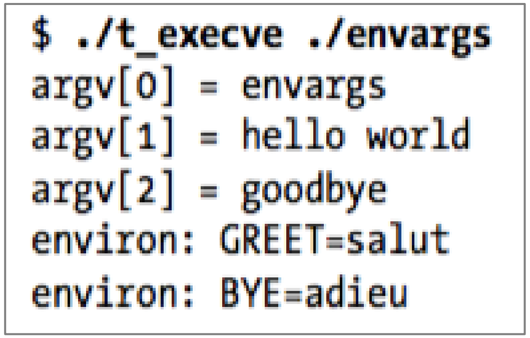
```
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    char *argVec[10];           /* Larger than required */
    char *envVec[] = { "GREET=salut", "BYE=adieu", NULL };

    if (argc != 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s pathname\n", argv[0]);

    argVec[0] = strrchr(argv[1], '/');      /* Get basename from argv[1] */
    if (argVec[0] != NULL)
        argVec[0]++;
    else
        argVec[0] = argv[1];
    argVec[1] = "hello world";
    argVec[2] = "goodbye";
    argVec[3] = NULL;           /* List must be NULL-terminated */

    execve(argv[1], argVec, envVec);
    errExit("execve");          /* If we get here, something went wrong */
}
```

Executing a New Program: execve()

procexec/envargs.c

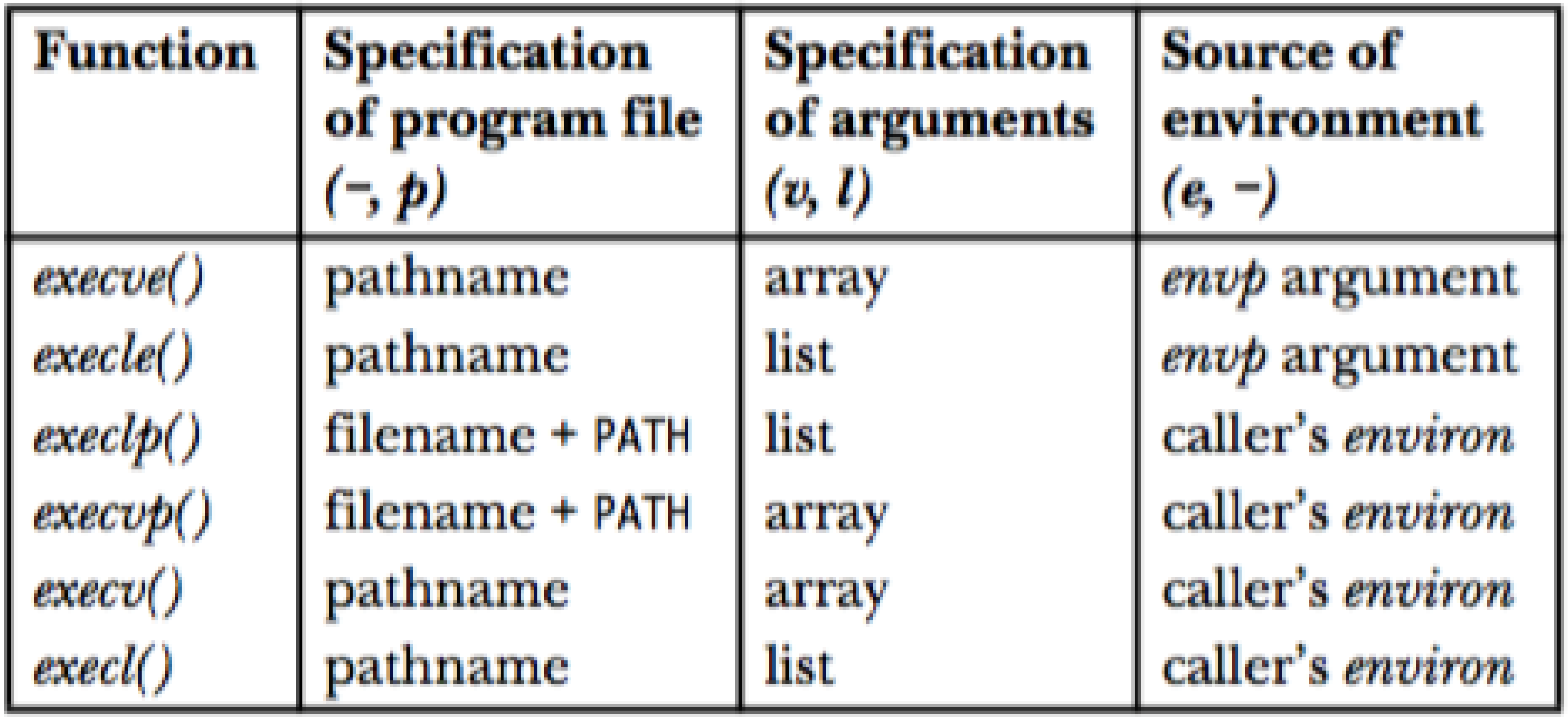


```
#include "tlpi_hdr.h"

extern char **environ;

int
main(int argc, char *argv[])
{
    int j;
    char **ep;

    for (j = 0; j < argc; j++)
        printf("argv[%d] = %s\n", j, argv[j]);

    for (ep = environ; *ep != NULL; ep++)
        printf("environ: %s\n", *ep);

    exit(EXIT_SUCCESS);
}
```



```
$ ./t_execve ./envargs
argv[0] = envargs
argv[1] = hello world
argv[2] = goodbye
environ: GREET=salut
environ: BYE=adieu
```

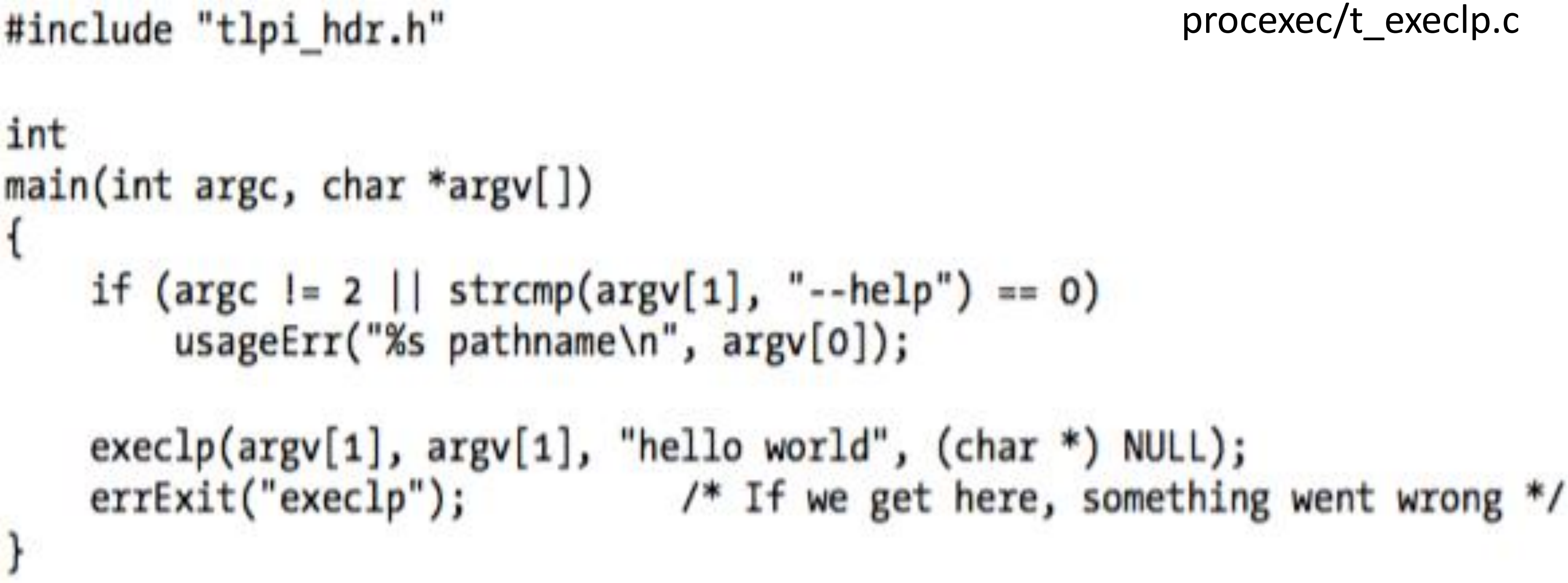
exec()



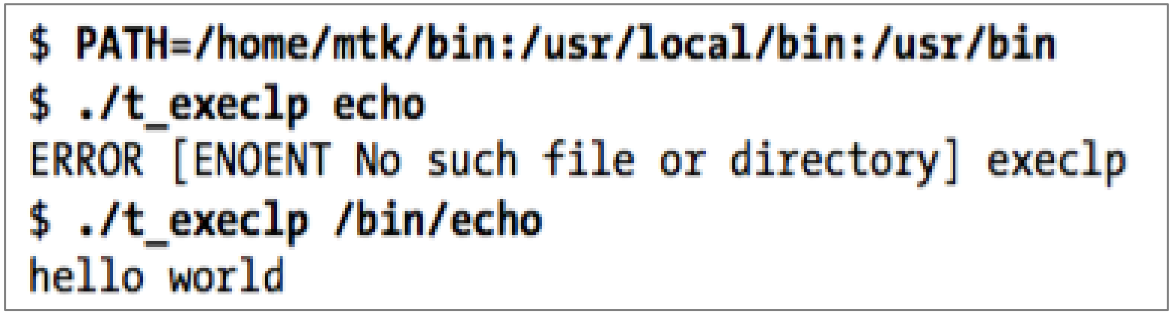
Function	**Specification of program file (–, *p*)**	**Specification of arguments (*v*, *l*)**	**Source of environment (*e*, –)**
execve()	pathname	array	*envp* argument
execle()	pathname	list	*envp* argument
execlp()	filename + PATH	list	caller's *environ*
execvp()	filename + PATH	array	caller's *environ*
execv()	pathname	array	caller's *environ*
execl()	pathname	list	caller's *environ*

execlp()

procexec/t_execlp.c



```c
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    if (argc != 2 || strcmp(argv[1], "--help") == 0)
        usageErr("%s pathname\n", argv[0]);

    execlp(argv[1], argv[1], "hello world", (char *) NULL);
    errExit("execlp");          /* If we get here, something went wrong */
}
```



```
$ PATH=/home/mtk/bin:/usr/local/bin:/usr/bin
$ ./t_execlp echo
ERROR [ENOENT No such file or directory] execlp
$ ./t_execlp /bin/echo
hello world
```

Interpreter Scripts

- An interpreter is a program that reads commands in text form and executes them
  - *awk, sed, perl, python, ruby*
- UNIX kernels allow interpreter scripts to be run in the same way as a binary program file if
  - execute permission is enabled (chmod +x <file>)
  - file contains an initial line that specifies the pathname of the interpreter to be used to run the script

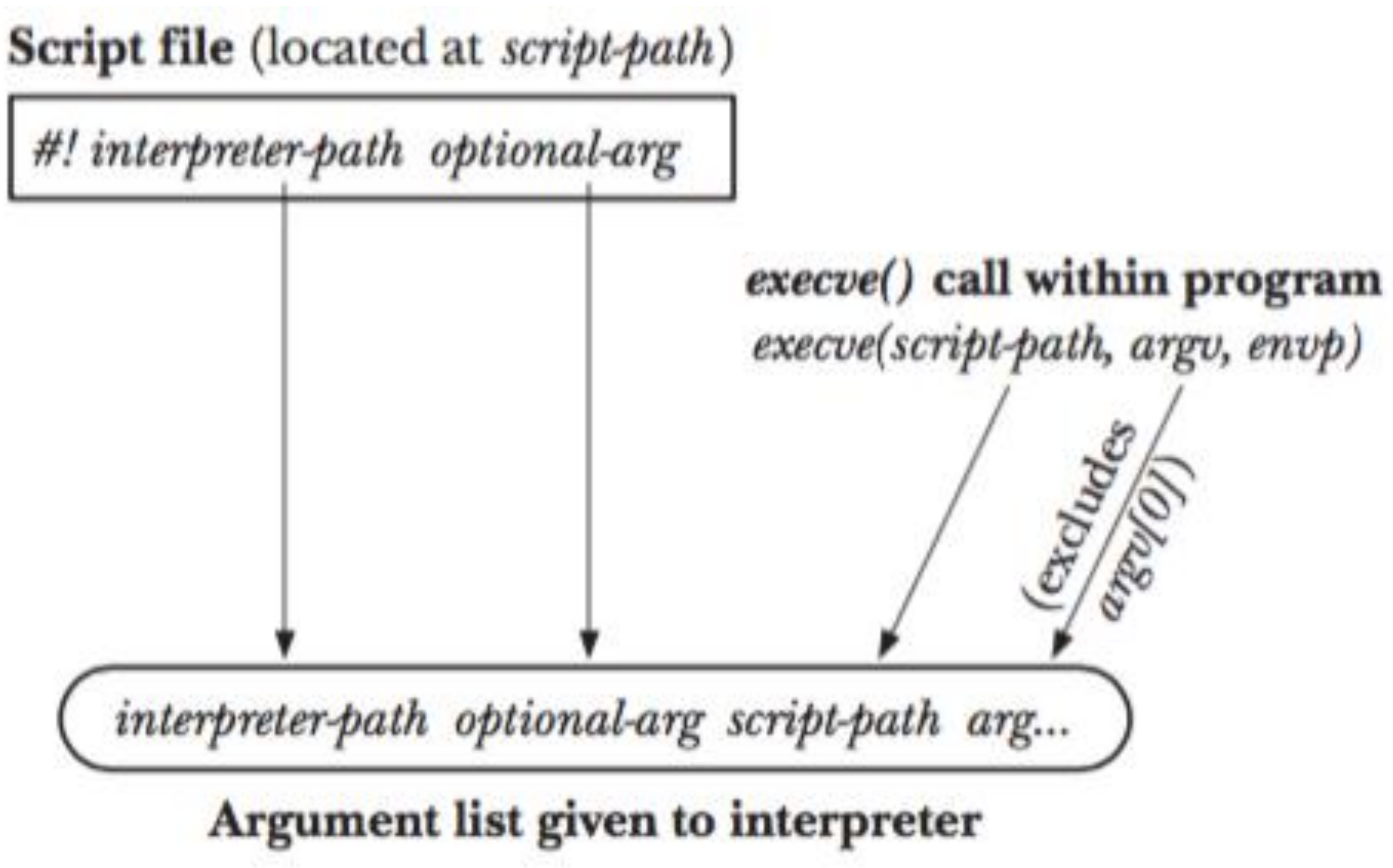
    #! interpreter-path [optional-arg]

Execution of Interpreter Scripts

- When *execve()* is used to run the script, *execve()* detects that the file has 2-byte sequence **#!**, then it extracts the remainder of the line (*pathname + argument*), and execs the interpreter file with following list of arguments

interpreter-path [optional-arg] script-path arg …

Argument list given to interpreter



Script file (located at *script-path*)

#! interpreter-path optional-arg

***execve()* call within program**

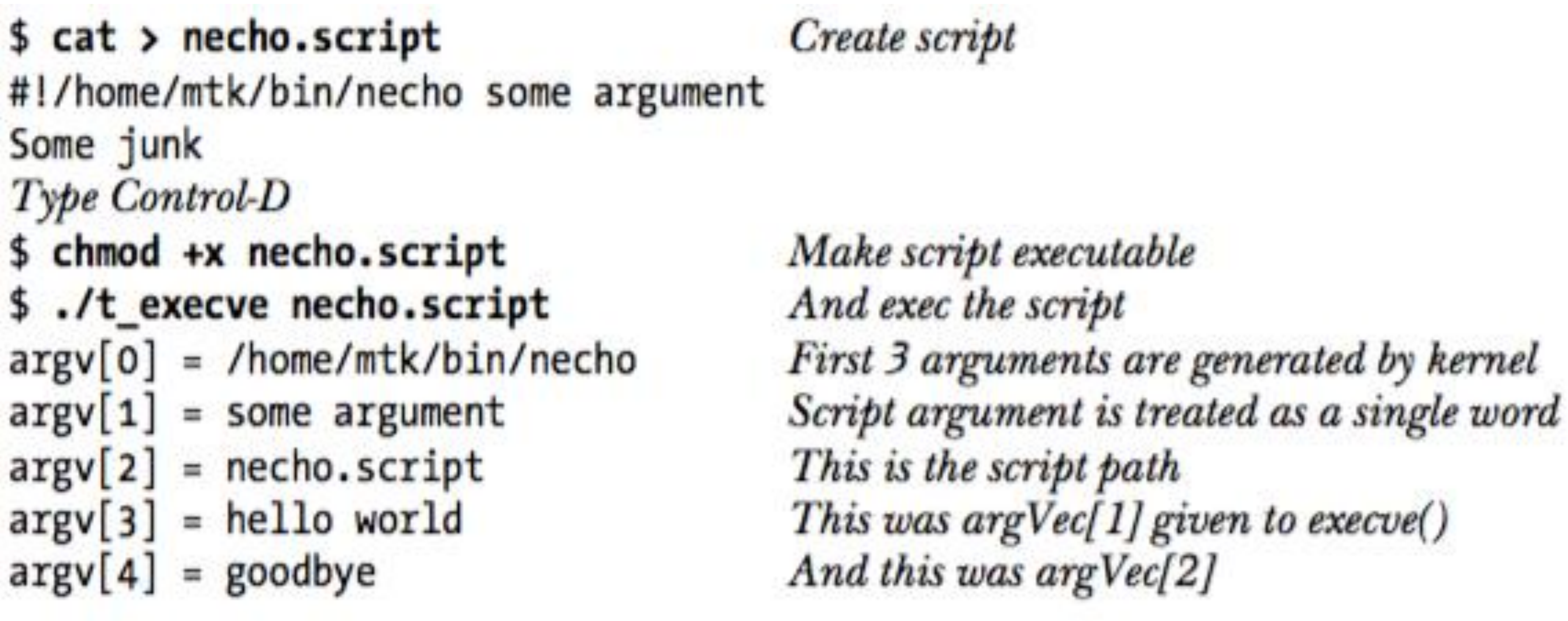
execve(script-path, argv, envp)

(excludes *argv[0]*)

interpreter-path optional-arg script-path arg...

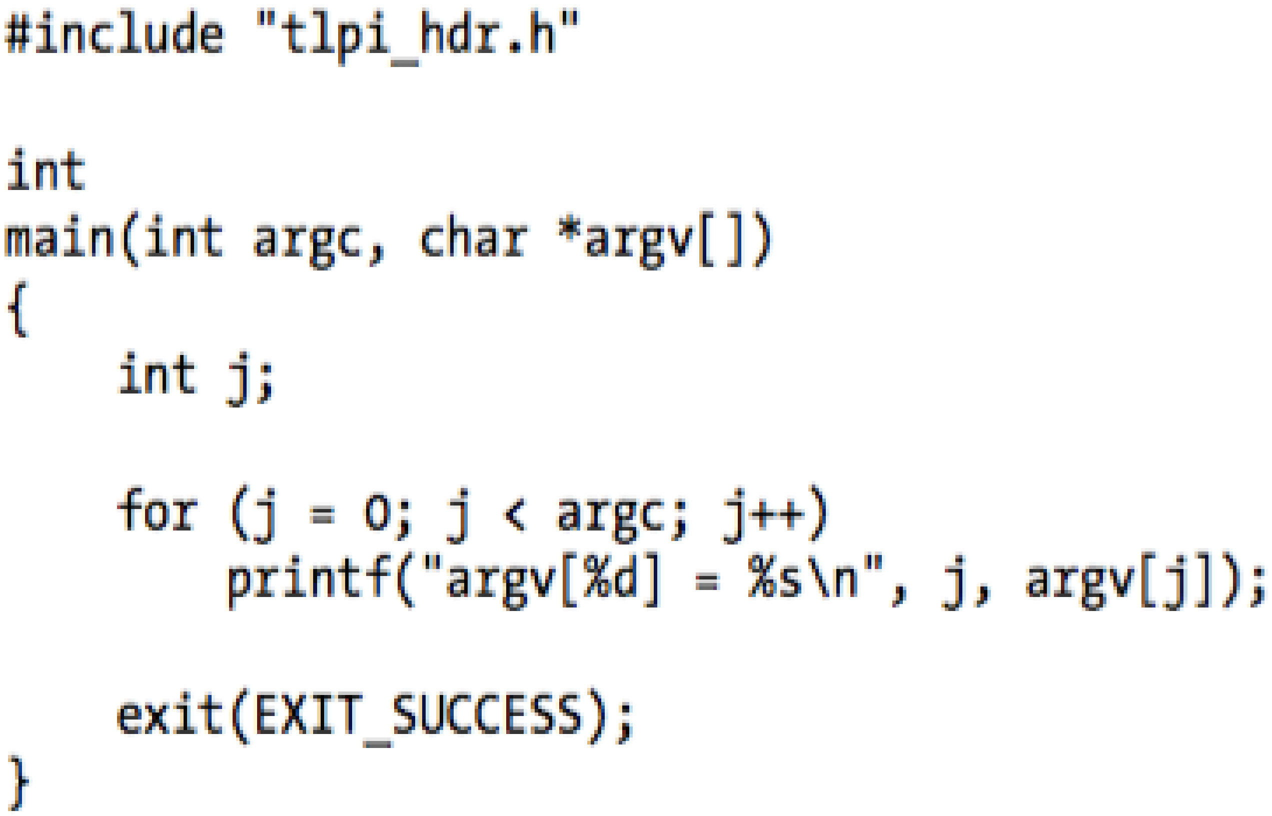
Argument list given to interpreter

Argument list given to interpreter



```
$ cat > necho.script                      Create script
#!/home/mtk/bin/necho some argument
Some junk
Type Control-D
$ chmod +x necho.script                   Make script executable
$ ./t_execve necho.script                 And exec the script
argv[0] = /home/mtk/bin/necho             First 3 arguments are generated by kernel
argv[1] = some argument                   Script argument is treated as a single word
argv[2] = necho.script                    This is the script path
argv[3] = hello world                     This was argVec[1] given to execve()
argv[4] = goodbye                         And this was argVec[2]
```

/proc/necho.c



```c
#include "tlpi_hdr.h"

int
main(int argc, char *argv[])
{
    int j;

    for (j = 0; j < argc; j++)
        printf("argv[%d] = %s\n", j, argv[j]);

    exit(EXIT_SUCCESS);
}
```

Be careful!!!

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
enum { BUFFERSIZE = 512};
void func(const char *input){
    char cmdbuf[BUFFERSIZE];
    int len_wanted = snprintf(cmdbuf,BUFFERSIZE,
                    "any_cmd '%s'", input);
    if (len_wanted >= BUFFERSIZE) {
        /* Handle error */
    } else if (len_wanted < 0) {
        /* Handle error */
    } else if (system(cmdbuf) == -1) {
        /* Handle error */
    }
}
```

https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152177