Lab 9

The first system calls we will need for our mini-shell

Command Line arguments

Needed for the first phase of the mini-shell

Command line arguments - Format

- *Command line arguments* are extra options that a user can pass to a program that is started from the command line.

  Format: ***`exec_filename`*** **`arg1 arg2 arg3`** …

  - Arguments are listed after the executable name.
  - Arguments are separated with whitespace.

Command line arguments - Declaration

- By modifying the main() function header, a C program can access command line arguments.

Change this:

```
int main (void)
```

To this:

```
int main (int argc, char * argv[])
```

Command line arguments – argc/argv parameters

main() function parameters:

- **`argc`** holds the total number of command line arguments (implicit+explicit).
- **`argv`** is an array of pointers to strings. Each argument value is stored as a string.

argv[0] contains the name of the executable.

Subsequent argv elements contain the explicit arguments (if any), in the order they appeared on the command line.

The final element in argv is always a NULL pointer.

A simple example will clarify this

argc/argv example 1. Code & Execution.

```
int main (int argc, char* argv[]){
        printf(“%s %d %s \n”, “You entered”, argc, “arguments”);
        printf(“%s: %s\n”, “The arg[0] is the program name”, argv[0]);
        printf(“%s: %s\n”, “The argument[1] is: ”, argv[1]);
        printf(“%s: %s\n”, “The argument[2] is: “, argv[2]);
}
```

```
$ gcc argv_example.c –o  argv_example
$ argv_example hello world
   You entered 3 arguments
   The argument[0] is the program name: argv_example
   The argument[1] is:  hello
   The argument[2] is:  world
```

Example 2 – necho.c (from LPI, page 123)

```
/* necho.c
  A simple version of echo(1): echo our command-line arguments.
*/
#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);
}
```

$necho hello world

argv[]

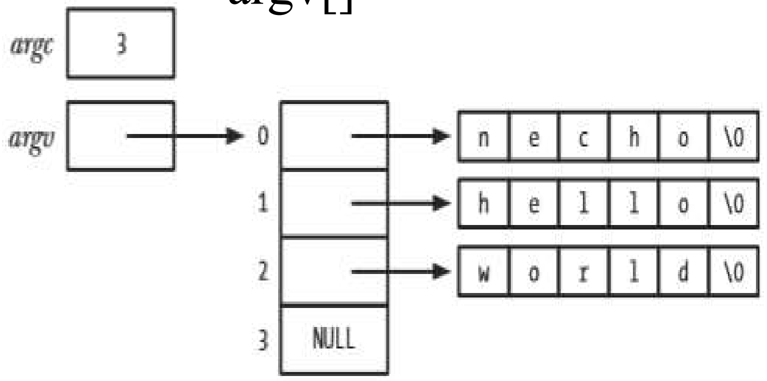

Figure 6-4: Values of *argc* and *argv* for the command *necho hello world*

Example 1 of cmdline

We type in the mini-shell:

```
csc60shell> pwd
```

Results:

```
argc = 1
argv[0] = “pwd”
```

Example 2 of cmdline

We type in the mini-shell:

```
csc60shell> cd ..
```

Results:

```
argc = 2
argv[0] = “cd”
argv[1] = “..”
```

Example 3 of cmdline

We type in the mini-shell:

csc60shell> **ls > lsout**

Results:

argc = 3

argv[0] = “ls”

argv[1] = “>”

argv[2] = “lsout”

System Calls for Lab 9

Exit the mini-shell- **exit**

Call:

```
#include <stdlib.h>

void exit (int status);   /* function prototype */
```

Examples:

```
exit (EXIT_SUCCESS);

exit (EXIT_FAILURE);
```

FROM: /usr/include/stdlib.h

```
132 /* We define these the same for all machines.
133    Changes from this to the outside world should be done in `_exit'.  */
134 #define EXIT_FAILURE    1       /* Failing exit status.  */
135 #define EXIT_SUCCESS    0       /* Successful exit status.  */
```

Get the current working path- **getcwd**

Call:

```
#include <stdlib.h>                                p.363

char *getcwd (char *cwdbuf, size_t size);

                              /* function prototype */
         Returns cwdbuf on success, or NULL on error
```

Example:

```
char buf [PATH_MAX];
getcwd(buf, PATH_MAX);
```

Change directory - **chdir**

Call:

```
#include <unistd.h>                                    p.364

int  chdir (const char * pathname);

                                      /* function prototype */
                             Returns 0 on success, or -1 on error.
```

Examples:

```
getcwd(buf, PATH_MAX);   //Remember where we are
chdir(somepath);         // Go somewhere else
chdir(buf);              // Return to original directory
```

Get an environment value – GETENV

The *getenv()* function retrieves individual values from the process environment.

Call:

```
#include <stdlib.h>                                    p.127

char getenv(const char *name);

                                        /* function prototype */
                              Returns pointer to (value) string,
                              or NULL if no such variable.
```

Examples:

```
char *myshell, *value;

myshell = getenv("SHELL");
value = getenv("HOME");
```

HOME

This is the initial directory into which the user is placed after logging in.

It is saved in process environment.

Dealing with Errors

This choice is good for non-system-call errors that the programmer’s code is checking.

- *Use a fprintf.*
  - *Example:*
  - fprintf(stderr, “More than one < is not allowed.\n");

This choice is good for reporting an error if a system call fails.

- *Use perror function.*
  - *Example:*
  - perror("Error executing xxx command");

Examples of System Call Errors

```
/* A first method using an error-variable*/
char *value;
int ret;

ret = chdir(value);
if (ret < 0) {
    fprintf(stderr, “Problem with chdir.\n”);
    _exit(EXIT_FAILURE);
}
```

Examples of System Call Errors

```
/* A second method **eliminating** the use of an
    error-variable. Common in textbook */
char *value;
int ret;

if (chdir(value) < 0) {
    fprintf(stderr, “Problem with chdir.\n”);
    _exit(EXIT_FAILURE);
}
```

Lab 9

System Calls for Mini-Shell
The End