# Note: The following example is taken from pages 99–100 of the text by Haviland et al. It is a bare bones implementation of the library function `system` provided by Unix. (The `system` library function takes one parameter of type string. It treats the parameter as a shell command and executes the command.)

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int do_command(char *); /* Prototype. */

/* The following main function merely calls the do_command function */
/* twice with different command strings. */

int main(void) {
    do_command("ls -l"); do_command("wc -l do_com.c");
    return 0;
} /* End of main. */

int do_command(char *command) {
    pid_t child; /* Child pid returned by fork. */
    pid_t c; /* Pid of child to be returned by wait. */

    if ((child = fork()) == 0) {
        /* Child process, which should execute the shell program. The "-c" */
        /* option to the shell asks the shell to take the command from the */
        /* next string rather than from stdin. */
        execlp("/bin/sh", "sh", ",-c", command, NULL);

        /* If this point is reached, then execlp must have failed. */
        fprintf(stderr, "Child process could not do execlp.\n"); exit(1);
    }
    else { /* Parent process. */
        if (child == (pid_t)(-1)) {
            fprintf(stderr, "Fork failed.\n"); exit(1);
        }
        else {
            c = wait(NULL); /* Wait for child to complete; ignore child’s exit status. */
            return 0;
        }
    }
} /* End of do_command. */
```