#include "apue.h"

#include <sys/wait.h>

int

main(void)

{

char buf[MAXLINE]; /\* from apue.h \*/

pid\_t pid;

int status;

printf("%% "); /\* print prompt (printf requires %% to print %) \*/

while (fgets(buf, MAXLINE, stdin) != NULL) {

if (buf[strlen(buf) - 1] == ’\n’)

buf[strlen(buf) - 1] = 0; /\* replace newline with null \*/

if ((pid = fork()) < 0) {

err\_sys("fork error");

} else if (pid == 0) { /\* child \*/

execlp(buf, buf, (char \*)0);

err\_ret("couldn’t execute: %s", buf);

exit(127);

}

/\* parent \*/

if ((pid = waitpid(pid, &status, 0)) < 0)

err\_sys("waitpid error");

printf("%% ");

}

exit(0);

}

This program uses standard I/O function fgets to read one line at a time from the standard input. When we type the end-of-file character (which is often Control-D) as the first character of a line, fgets returns a null pointer, the loop stops, and the process terminates.

Use standard C function strlen to calculate the length of the string and replace the new line with a null byte as the execlp function wants a null-terminated argument, not a newline-terminated argument.

We call fork to create a new process, which is a copy of the caller. We say that the caller is the parent and that the newly created process is the child. Then fork returns the non-negative process ID of the new child process to the parent and returns 0 to the child. Because fork creates a new process, we say that it is called once—by the parent — but returns twice—in the parent and in the child.

In the child, we call execlp to execute the command that was read from the standard input. This replaces the child process with the new program file. The combination of fork followed by exec is called spawning a new process on some operating systems. In the UNIX System, the two parts are separated into individual functions. We’ll say a lot more about these functions in Chapter 8.

Because the child calls execlp to execute the new program file, the parent wants to wait for the child to terminate. This is done by calling waitpid, specifying which process to wait for: the pid argument, which is the process ID of the child. The waitpid function also returns the termination status of the child — the status variable — but in this simple program, we don’t do anything with this value. We could examine it to determine how the child terminated.