System Programming in Linux

A Hands-On Introduction

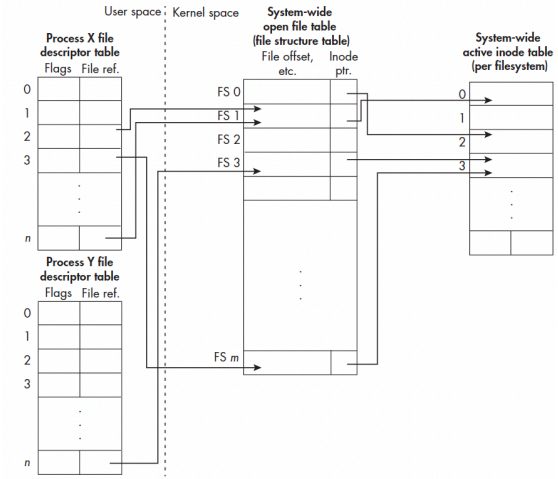
by Stewart N. Weiss

Errata updated to print 1

Page	Error	Correction	Print corrected

2	Author's note: The original version of K&R's program did not have a return type for `main()`. This change is for historical purposes.
	`#include`
	`void main()`
	`{`
		`#include`
		`main()  /* NOTE: Original main() had no return type. */`
		`{`
			Pending

3, 16, 27, 28, 64, 65, 317	Author's note: `void main()` generates warnings. `void main()`	`int main()`	Pending
58	Linux kernels from 2.6 and later use **the `sysenter` instruction**, and the GNU C Library *glibc 2.3.2* and later use **`sysenter`**.	Linux kernels from 2.6 and later use **both the `syscall` and `sysenter` instructions**, and the GNU C Library *glibc 2.3.2* and later use **both `syscall` and `sysenter`**.	Pending
125	`    char    ch;                 /* For option handling          */`	`    int     ch;                 /* For option handling          */`	Pending

Page	Error	Correction	Print corrected
156	**In Figure 4-1, FS3 is missing an arrow.**	 *Figure 4-1: The tables used to manage files opened by processes*	Pending
239	`BOOL found = FALSE; /* Flag to indicate match found */` `char ch;`	`BOOL found = FALSE; /* Flag to indicate match found */` `int ch;`	Pending

282	For example, if `fd` is a valid file descriptor for the file ***/var/loglastlog***, then
	`statx(fd, "", AT_EMPTY_PATH, STATX_ALL, &statxbuf);`
	refers to the file */var/log/lastlog*.
		For example, if `fd` is a valid file descriptor for the file ***/var/log/lastlog***, then
		`statx(fd, "", AT_EMPTY_PATH, STATX_ALL, &statxbuf);`
		refers to the file */var/log/lastlog*.
			Pending

286	`int i;` `char ch;`	`int i;` `int ch;`	Pending

307	`$ ./spl_statfs /media/guest/USB_stick/`
	`File: "/media/guest/USB_stick/"`
	`ID: 840 0 Namelen: 1530 Type: msdos`
		`$ ./spl_statfs /media/guest/USB_stick/`
		`File: "/media/guest/USB_stick/"`
		`ID: 84000000000 Namelen: 1530 Type: msdos`
			Pending

330	`double f(int (*funcp)(int, int), int*);`	`double f(int (*funcp)(int, int), int);`	Pending

Page	Error	Correction	Print corrected
335	In Linux and most Unix systems, the hierarchy has no cycles if it has no symbolic links.	In Linux and most Unix systems, the hierarchy **usually** has no cycles if it has no symbolic links.	Pending
377	For each name that we want to prepend, we get the string length of the name `len = strlen{dirname};` and subtract it from `front` so that `front` points to where this name will start in the buffer.	For each name that we want to prepend, we get the string length of the name `len = strlen(dirname);` and subtract it from `front` so that `front` points to where this name will start in the buffer.	Pending

378	`while ( (direntp = readdir dir_ptr)) != NULL ) {`
	`errno = 0;`
	`get_dev_ino(direntp->d_name, &this_entry);`
		`while ( (direntp = readdir(dir_ptr) ) != NULL ) {`
		`errno = 0;`
		`get_dev_ino(direntp->d_name, &this_entry);`
			Pending

391	**`SIGTRAP`** Used to implement debuggers and tracing programs such as `strace` and `ptrace`.	**`SIGTRAP`** Used to implement debuggers **such as `gdb`** and tracing programs such as `strace` and `ptrace`.	Pending

404	`printf("\ncleaning up in progress...\ndone\n");`
	`raise(SIGTSTP);`
	`printf("raise() called to stop process.\n");`
		`printf("\ncleaning up in progress...\ndone\n");` **`/* UNSAFE */`**
		`raise(SIGTSTP);`
		`printf("raise() called to stop process.\n");` **`/* UNSAFE */`**
			Pending

411	If we enter CTRL-C 6 times while it is blocked, when it becomes unblocked, the handler will run just once, not **10** times.	If we enter CTRL-C 6 times while it is blocked, when it becomes unblocked, the handler will run just once, not **6** times.	Pending

<table>
<tr><th>Page</th><th>Error</th><th>Correction</th><th>Print corrected</th></tr>
<tr><td>422</td><td>

```
#include <fenv.h>

void fpe_handler(int signo, siginfo_t *info, void *context)
{   /* These calls to printf() are all UNSAFE. */
    printf("Signal: %s\n", strsignal(info->si_signo));
    switch ( info->si_code ) {
    case FPE_INTDIV:
        printf("Code: FPE_INTDIV (Integer divide by zero)\n"); break;
    case FPE_FLTDIV:
        printf("Code: FPE_FLTDIV (Floating-point divide by zero)\n"); break;
    case FPE_FLTOVF:
        printf("Code: FPE_FLTOVF (Floating-point overflow)\n"); break;
        --snip--
    }

    raise(SIGTERM);
}
```

</td><td>

```
#include <fenv.h>

BOOL noreturn = FALSE;
void fpe_handler(int signo, siginfo_t *info, void *context)
{   /* These calls to printf() are all UNSAFE. */
    printf("Signal: %s\n", strsignal(info->si_signo));
    switch ( info->si_code ) {
    case FPE_INTDIV:
        printf("Code: FPE_INTDIV (Integer divide by zero)\n"); break;
    case FPE_FLTDIV:
        printf("Code: FPE_FLTDIV (Floating-point divide by zero)\n"); break;
    case FPE_FLTOVF:
        printf("Code: FPE_FLTOVF (Floating-point overflow)\n"); break;
        --snip--
    }

    if ( !noreturn )
        raise(SIGTERM); /* To force program to quit */
}
```

</td><td>Pending</td></tr>
<tr><td>442</td><td>The <code>timespec_diff()</code> function introduced on page 442 and used in several programs will not produce a correct time difference when the first time argument represents an older time than the second one. It is intended to be used only when the first is either equal to or later than the second. This is how it is used in all programs that call it. The book's source code repository has another version, <code>timespec_diff2()</code>, that handles this case correctly. The assert.h header file should be included in time_utils.c where the function is defined.</td><td>

```
/** timespec_diff(ts1, ts2, *diff) computes the difference ts1 - ts2,
storing it in *diff provided that ts2 is no later than ts1. */
```

```
void timespec_diff(struct timespec ts1, struct timespec ts2,
                   struct timespec *diff)
{
   long temp;
   diff->tv_sec = ts1.tv_sec - ts2.tv_sec;
   temp = ts1.tv_nsec - ts2.tv_nsec;
   assert ( diff->tv_sec >= 0 );
❶  if ( temp < 0 ) {
       /* Because temp < 0, we need to borrow 1 sec from tv_sec and
          add it to tv_nsec as 1000000000 nanoseconds. */
       diff->tv_sec--;
       diff->tv_nsec = 1000000000 + temp;
   }
   else
       diff->tv_nsec = temp;
}
```

</td><td>Pending</td></tr>
</table>

Page	Error	Correction	Print corrected

461	Author's note: There is an extremely tiny probability of a race condition due to the call to `draw_initial_bar()` preceding the signal handler setup in Listing 9-6. It should be placed after the two calls to `sigaction()`.	`int main(int argc, char *argv[])`
		`{`
		`  struct sigaction  act;`
		`  const struct timespec slight_pause = {2, 0}; /* 2-second interval */`
		`  struct timespec  remaining_sleep;`
		~~`  draw_initial_bar();`~~
		`  sigemptyset(&(act.sa_mask));`
		`  act.sa_flags = 0;`
		`  act.sa_handler = sig_handler;`
		`  if ( sigaction(SIGINT, &act, NULL) == -1 )`
		`  fatal_error(errno, "sigaction");`
		`  if ( sigaction(SIGQUIT, &act, NULL) == -1 )`
		`  fatal_error(errno, "sigaction");`
		`  draw_initial_bar();`
		
		`  act.sa_handler = refresh_progressbar;`
			Pending

472	`bigfile  0923331`	`bigfile  923331`	Pending
486	`  struct sigaction  sa;` `  char  c;`	`  struct sigaction  sa;` `  int  c;`	Pending

509	`for ( i = 0; i < elf_header64->e_phnum; i++ ) {`
	`  print_progheader64(fd, &(phdr64[i]);`
	`}`
		`for ( i = 0; i < elf_header64->e_phnum; i++ ) {`
		`  print_progheader64(fd, &(phdr64[i]));`
		`}`
			Pending

514	~~`#include <unistd.h>`~~ `#include <unistd.h>`	`#include <unistd.h>`	Pending
543	In modern Linux systems, the physical pages of the parent's memory image aren't actually copied until **the child process** attempts to modify them.	In modern Linux systems, the physical pages of the parent's memory image aren't actually copied until **one of the processes** attempts to modify them.	Pending

Page	Error	Correction	Print corrected

548	`int main(int argc, char *argv[])`
	`{`
	`  pid_t newpid;`
	`  int  num_children = 4;`
		`int main(int argc, char *argv[])`
		`{`
		`  pid_t newpid;`
		`  int  num_children = 3;`
			Pending

581	**`$ ./waitpid_demo`**
	`Producer PID = 37837`
	`Producer is about to start producing data.`
	`Consumer PID = 37838`
	`Data from producer, converted to uppercase:`
	`PPPPP`
	`Process 37837 stopped by signal 19  # Issued kill -19 37837 from 2nd terminal`
	`PPPP`
	`Process 37837 continued  # Issued kill -18 37837 from 2nd terminal`
	`Producer finished; waiting for consumer to finish.`
	`Process 37838 stopped by signal 19  # Issued kill -19 37838 from 2nd terminal`
	`Process 37838 continued  # Issued kill -18 37838 from 2nd terminal`
	`PPPPPPPPP`
	`Process 37838 terminated by signal 8 # Issued kill -8 37838 from 2nd terminal`
	`$`
		`$ ./waitpid_demo`
		`Producer PID = 37837`
		`Producer is about to start producing data.`
		`Consumer PID = 37838`
		`Data from producer, converted to uppercase:`
		`PPPPP`
		`Process 37837 stopped by signal 19  # Issued kill -19 37837 from 2nd terminal`
		`PPPP`
		`Process 37837 continued  # Issued kill -18 37837 from 2nd terminal`
		`Producer (PID=37837) finished producing data; exiting`
		
		`Process 37837 exited with status 63`
		
		`Producer finished; waiting for consumer to finish.`
		`Process 37838 stopped by signal 19  # Issued kill -19 37838 from 2nd terminal`
		`Process 37838 continued  # Issued kill -18 37838 from 2nd terminal`
		`PPPPPPPPP`
		`Process 37838 terminated by signal 8 # Issued kill -8 37838 from 2nd terminal`
		`$`
			Pending

582	An entirely different approach is to utilize the `SIGCHLD` signal. Normally, when a process terminates by calling an exit function or is killed or stopped by a signal, the kernel generates a `SIGCHlD` signal and sends it to that process's parent, provided that it's still running. There are exceptions to this rule that I'll explain shortly, but for now let's assume that the `SIGCHlD` signal is always sent to the parent process in these circumstances. In this alternative method, we put the call to `wait()` or `waitpid()` inside the `SIGCHlD` handler.	An entirely different approach is to utilize the `SIGCHLD` signal. Normally, when a process terminates by calling an exit function or is killed or stopped by a signal, the kernel generates a `SIGCHLD` signal and sends it to that process's parent, provided that it's still running. There are exceptions to this rule that I'll explain shortly, but for now let's assume that the `SIGCHLD` signal is always sent to the parent process in these circumstances. In this alternative method, we put the call to `wait()` or `waitpid()` inside the `SIGCHLD` handler.	Pending
614	`t4: P1  executes mov @counter, register1  {counter  == 5 }`	`t4: P1  executes mov @counter, register1  {counter  == 6 }`	Pending
623	`data = sharedbuf->buf[sharedbuf->rear];` `sharedbuf->rear = (sharedbuf->rear + 1) % BUF_SIZE;`	`data = sharedbuf->buf[sharedbuf->rear];` `sharedbuf->front = (sharedbuf->front + 1) % BUF_SIZE;`	Pending
635	1. Configure a `sigevent` structure **`sigev`** as follows:	1. Configure a `sigevent` structure **`sev`** as follows:	Pending

Page	Error	Correction	Print corrected
636	`struct sigaction sa;`	`struct sigaction sigact;`	Pending
639	`unsigned short int rows, cols;  /* Size of terminal window  */`	`unsigned short int rows, cols;  /* Size of terminal window  */` `msgtype  newmsg;  /* Message struct for message  */`	Pending
658	If such a function existed, we could copy the pipe's write end descriptor into the **descriptors** for standard **input and** output.	If such a function existed, we could copy the pipe's write end descriptor into the **descriptor** for standard output.	Pending
707	2. Modify ***calc_client.c*** so that if it has no command line argument, it reads the expression from standard input. 3. Modify *spl_calc.h*, ***calc_client.c***, and ***calc_server.c*** so that if the user supplies a `-l` command line option to the client, the server will request `bc` to load its standard math library, as described on its man page. 4. Modify both ***calc_client.c*** and ***calc_server.c*** so that if the user supplies an `-s value` command line option, the number of digits to the right of the decimal point for all answers will be the supplied `value`. 5. Write a version of ***calc_server.c*** that still relies upon the `bc` command but does not call `popen()` to run it. 6. Add a `-d` option to ***calc_server.c*** that, when present, turns it into a daemon, and when not, requires the user to run it in the background.	2. Modify ***spl_calc_client.c*** so that if it has no command line argument, it reads the expression from standard input. 3. Modify *spl_calc.h*, ***spl_calc_client.c***, and ***spl_calc_server.c*** so that if the user supplies a `-l` command line option to the client, the server will request `bc` to load its standard math library, as described on its man page. 4. Modify both ***spl_calc_client.c*** and ***spl_calc_server.c*** so that if the user supplies an `-s value` command line option, the number of digits to the right of the decimal point for all answers will be the supplied `value`. 5. Write a version of ***spl_calc_server.c*** that still relies upon the `bc` command but does not call `popen()` to run it. 6. Add a `-d` option to ***spl_calc_server.c*** that, when present, turns it into a daemon, and when not, requires the user to run it in the background.	Pending
722	`thread_data[t].task_id = t;`	`thread_data[t].id = t;`	Pending
785	`BOOL  reader_preference = FALSE;` `char  ch;`	`BOOL  reader_preference = FALSE;` `int  ch;`	Pending

<table>
<tr><th>Page</th><th>Error</th><th>Correction</th><th>Print corrected</th></tr>
<tr><td>814</td><td>

```
while ( more_data_to_read ) {          /* While source file has more data  */
    // if ( aio_read() completed ) {
    num_read = aio_return(&aio_block); /* Get num bytes read.              */
    writebuf = aio_block.aio_buf;      /* write() from buffer just filled  */
    i = i^1;                           /* Same as i = (i == 0) ? 1 : 0     */
    aio_block.aio_buf = buffer[i];     /* Use other buffer for next read.  */
    aio_block.aio_offset += num_bytes_read;     /* Adjust offset for read. */
    aio_read(&aiocb));                 /* Request next read.               */
    write(target_fd, writebuf, num_bytes_read); /* Start write().          */
    }
}
```

</td><td>

```
while ( more_data_to_read ) {          /* While source file has more data  */
    if ( aio_read() completed ) {
        num_read = aio_return(&aio_block); /* Get num bytes read.              */
        writebuf = aio_block.aio_buf;      /* write() from buffer just filled  */
        i = i^1;                           /* Same as i = (i == 0) ? 1 : 0     */
        aio_block.aio_buf = buffer[i];     /* Use other buffer for next read.  */
        aio_block.aio_offset += num_bytes_read;     /* Adjust offset for read. */
        aio_read(&aiocb));                 /* Request next read.               */
        write(target_fd, writebuf, num_bytes_read); /* Start write().          */
    }
}
```

</td><td>Pending</td></tr>
<tr><td>825</td><td>

```
    for ( int i = 0; i < NSENDERS; i++ ) {
        if ( pipe(pipefd[i]) == -1 ) /* Create pipe[pipefd[i]. */
```

</td><td>

```
    for ( int i = 0; i < NSENDERS; i++ ) {
        if ( pipe(pipefd[i]) == -1 ) /* Create pipe[pipefd[i]]. */
```

</td><td>Pending</td></tr>
<tr><td>837</td><td>

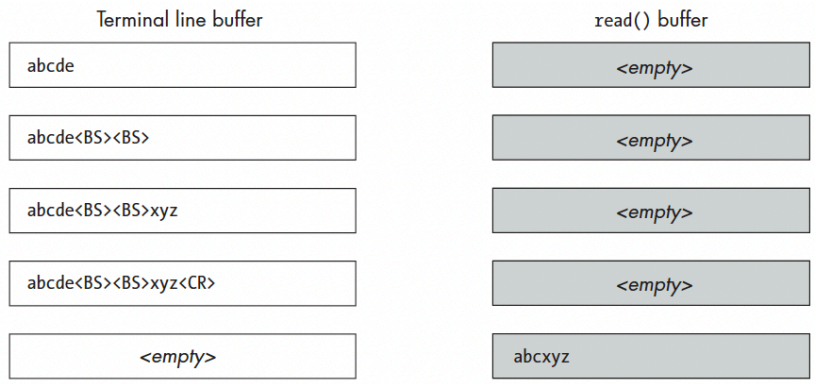


Figure 18-1: A schematic representation of the processing of an edited input line

</td><td>

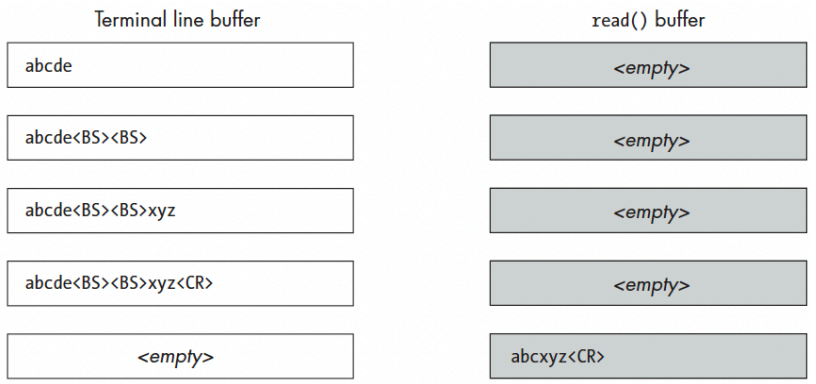


Figure 18-1: A schematic representation of the processing of an edited input line

</td><td>Pending</td></tr>
<tr><td>850</td><td>

Mask name	Description
NOFLSH	Disable flushing the input and output queues when generating signals for the **INT**, QUIT, and SUSP characters.

</td><td>

Mask name	Description
NOFLSH	Disable flushing the input and output queues when generating signals for the **INTR**, QUIT, and SUSP characters.

</td><td>Pending</td></tr>
</table>

Page	Error	Correction	Print corrected
851	`$ bash --no-editing`	`$ bash --noediting`	Pending
855	**Handling Excess Input in the Terminal** **When** `read()` reads from a terminal, it doesn't receive characters until a newline is entered. Assuming that `ttyfd` refers to a terminal, if a program calls `read(ttyfd, &buffer, 32)` but the user enters 40 characters and then a newline, `read()` will receive the first 32 characters and store them into `buffer`, but there will be 8 characters and a newline remaining in the input queue. A second call to `read(ttyfd, &buffer, 32)` will be satisfied immediately because **there's a newline in the input queue**. The remaining 8 characters and the newline will be stored into `buffer`. If a program doesn't want these characters to be read into `buffer`, it can call `tcflush(ttyfd, TCIFLUSH)` to remove them from the input queue before reading again.	**Handling Excess Input in the Terminal** **In canonical mode, when** `read()` reads from a terminal, it doesn't receive characters until a newline is entered **(or the user terminates input by entering the end-of-file character)**. Assuming that `ttyfd` refers to a terminal, if a program calls `read(ttyfd, &buffer, 32)` but the user enters 40 characters and then a newline, `read()` will receive the first 32 characters and store them into `buffer`, but there will be 8 characters and a newline remaining in the **terminal's** input queue. A second call to `read(ttyfd, &buffer, 32)` will be satisfied immediately because **the input queue is not empty**. The remaining 8 characters and the newline will be stored into `buffer`. If a program doesn't want these characters to be read into `buffer`, it can call `tcflush(ttyfd, TCIFLUSH)` to remove them from the input queue before reading again.	Pending
874	*fillq**e**ueue.c*	*fillqueue.c*	Pending
875	In the second terminal, turn off canonical mode and run `fillqueue`**:**	In the second terminal, turn off canonical mode and run `fillqueue` **(you may have to set `cap_sys_admin` on `fillqueue` as you did for `fakeinput_demo`):**	Pending

933	`    }`
	`    wrefresh(win);`
	`}`
		`    }`
		`    wrefresh(win);`
		`}`
			Pending

936	`if ( *proclist != NULL )`	`if ( *proclist != NULL ) {`	Pending

Additional Notes

Page 197

An alternative to the command

```
cpp -v /dev/null -o /dev/null
```

is

```
cpp -v /dev/null -P
```

which generates the same output.

Page 411

The code following the `while` loop in *sigprocmask_demo1.c* should never be reached. It is there to show how to use `SIG_SETMASK` to set the blocked signals to a previous signal set. It should have a preceding comment noting that it's unreachable as long as the `while` loop cannot be exited.

Page 423

Although it is not incorrect, the extra parentheses in this line

```
sigemptyset(&(action.sa_mask));
```

are unnecessary, because the struct `.` operator has higher precedence than the address-of operator, `&`. It can also be:

```
sigemptyset(&action.sa_mask);
```

Page 485

The *posix_timer_demo1.c* program relies on the assumption that the timer IDs returned by successive calls to `timer_create()` start at 0 and increase by 1. POSIX doesn't require this. In the versions of Linux on which I have tested this, they do behave this way. An actual application should not be based on this assumption.

Page 611

The two lines

```
shmp->ptr1 = (char*)
malloc(20);
```

are correct, but it is a single assignment that should be:

```
shmp->ptr1 = (char*) malloc(20);
```

Page 612

Although it is not incorrect, the line

```
printf("%s\n", (char*) (shmp->ptr2));
```

should be

```
printf("%s\n", shmp->ptr2);
```

since `shmp->ptr2` is a string pointer (`char*`).

Page 618

In *unnamedsem_demo.c*, although this assignment is correct:

```
shmp = mmap(NULL,
    sizeof(sharedmem), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

for consistency, since I prefer explicitly casting the `void` pointers, it should be:

```
shmp = (sharedmem*) mmap(NULL,
    sizeof(sharedmem), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
```

Page 633

Though it is not incorrect, the code after the `while` loop cannot be executed given the current form of the loop. The call to `fatal_error()` could be preceded by a call to `mq_close()` and the code after the loop could be deleted.

Page 639

The macro definition in *ulogger.c*

```
#define SIGUPDATE SIGRTMIN+1
```

is correct as is, but in general, it's best to put parentheses around macro values that contain operators, i.e., `(SIGRTMIN+1)`.

Page 642

The call to `cleanup()` at the end of *ulogger.c* is never reached and unnecessary.

Page 663

The POSIX specification says that writes are atomic if the number of bytes is less than or equal to `PIPE_BUF`.
Writes are atomic **as long as** the number of bytes is smaller than `PIPE_BUF`.
should read:
Writes are atomic **if** the number of bytes is smaller than `PIPE_BUF`.

Page 700

It is not an error, but it is inconsistent that I use `O_NDELAY` in this program instead of `O_NONBLOCK`. They are interchangeable:

```
while ( ((rawtext_fd = open(msg.raw_text_fifo,
      O_WRONLY | O_NDELAY)) == -1) && (tries < MAXTRIES) ) {
```

Page 720

At the bottom of this page, in the short listing that suggests two ways to pass data to threads, the declarations `int counter = 0;` and `int *on_heap = (int*) malloc(sizeof int);` are file-scoped. This code is really a straw man that leads to the better solution, and cannot be compiled because `on_heap` is assigned the return value of a function call.

The `on_heap` variable should be initialized in the `main` program before any threads are created:

```
int *on_heap; /* Initialize in main() with on_heap = (int*) malloc(sizeof int); */
```

Page 731

Not all of the output of `ps -L -eopid,ppid,tid,cmd | grep pthread_signal_demo` is shown in the listing.

Page 793

The text should note that the program in Listing 17-1 doesn't reset standard input to blocking mode just before it terminates because the shell does this automatically. This is an important observation—it means that you cannot stop and restart this program or any program that enables non-blocking input unless you write explicit handlers for the `SIGSTOP`, `SIGTSTP`, and `SIGCONT` signals that reenable non-blocking input when the program is resumed.

Page 835

Although the code is correct, the following changes make it more consistent:

```
    if ( -1 == write(STDOUT_FILENO, prompt, strlen(prompt)) )
        fatal_error(errno, "write");
    while ( read(STDIN_FILENO, &inbuf, 1) > 0 ) {
        inbuf = toupper(inbuf);
        if ( -1 == write(STDOUT_FILENO, &inbuf, 1) )
            fatal_error(errno, "write");
    }
```

Page 873

On Linux kernels since Linux 6.2, this program may not work unless you set the `CAP_SYS_ADMIN` capability on the binary. You'll need superuser privilege. After building the executable, add the line

```
$ sudo setcap cap_sys_admin=eip fakeinput_demo
```

before running:

```
$ ./fakeinput_demo
$ /usr/bin/echo 'This is a dangerous thing to do!'
This is a dangerous thing to do!
$
```