附录B　其他源代码

B.1　头文件

正文中的大多数程序都包含头文件 ourhdr.h，这示于程序B-1中。其中定义了常数（例如MAXLINE）和我们自编函数的原型。

因为大多数程序序包含下列头文件：<stdio.h>、<stdlib.h>（其中有exit函数原型），以及<unistd.h>（其中包含所有标准UNIX函数的原型），所以ourhdr.h包含了这些系统头文件，同时还包含了<string.h>。这样就减少了本书正文中所有程序的长度。

程序B-1　头文件ourhdr.h

```
/* Our own header, to be included *after* all standard system headers */

#ifndef __ourhdr_h
#define __ourhdr_h

#include    <sys/types.h>   /* required for some of our prototypes */
#include    <stdio.h>       /* for convenience */
#include    <stdlib.h>      /* for convenience */
#include    <string.h>      /* for convenience */
#include    <unistd.h>      /* for convenience */

#define MAXLINE 4096            /* max line length */

#define FILE_MODE    (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)
                     /* default file access permissions for new files */
#define DIR_MODE     (FILE_MODE | S_IXUSR | S_IXGRP | S_IXOTH)
                     /* default permissions for new directories */

typedef void    Sigfunc(int);   /* for signal handlers */

                     /* 4.3BSD Reno <signal.h> doesn't define SIG_ERR */
#if defined(SIG_IGN) && !defined(SIG_ERR)
#define SIG_ERR ((Sigfunc *)-1)
#endif

#define min(a,b)     ((a) < (b) ? (a) : (b))
#define max(a,b)     ((a) > (b) ? (a) : (b))

                     /* prototypes for our own functions */
char    *path_alloc(int *);          /* Program 2.2 */
int      open_max(void);             /* Program 2.3 */
void     clr_fl(int, int);           /* Program 3.5 */
void     set_fl(int, int);           /* Program 3.5 */
void     pr_exit(int);               /* Program 8.3 */
void     pr_mask(const char *);      /* Program 10.10 */
Sigfunc *signal_intr(int, Sigfunc *);/* Program 10.13 */

int      tty_cbreak(int);            /* Program 11.10 */
int      tty_raw(int);               /* Program 11.10 */
int      tty_reset(int);             /* Program 11.10 */
void     tty_atexit(void);           /* Program 11.10 */
#ifdef  ECHO    /* only if <termios.h> has been included */
struct termios  *tty_termios(void); /* Program 11.10 */
```

```
#endif

void     sleep_us(unsigned int);     /* Exercise 12.6 */
ssize_t  readn(int, void *, size_t);/* Program 12.13 */
ssize_t  writen(int, const void *, size_t);/* Program 12.12 */
int      daemon_init(void);          /* Program 13.1 */

int      s_pipe(int *);              /* Programs 15.2 and 15.3 */
int      recv_fd(int, ssize_t (*func)(int, const void *, size_t));
                                     /* Programs 15.6 and 15.8 */
int      send_fd(int, int);          /* Programs 15.5 and 15.7 */
int      send_err(int, int, const char *);/* Program 15.4 */
int      serv_listen(const char *); /* Programs 15.19 and 15.22 */
int      serv_accept(int, uid_t *); /* Programs 15.20 and 15.24 */
int      cli_conn(const char *);     /* Programs 15.21 and 15.23 */
int      buf_args(char *, int (*func)(int, char **));
                                     /* Program 15.17 */

int      ptym_open(char *);          /* Programs 19.1 and 19.2 */
int      ptys_open(int, char *);     /* Programs 19.1 and 19.2 */
#ifdef  TIOCGWINSZ
pid_t    pty_fork(int *, char *, const struct termios *,
                  const struct winsize *);  /* Program 19.3 */
#endif

int     lock_reg(int, int, int, off_t, int, off_t);
                                     /* Program 12.2 */
#define read_lock(fd, offset, whence, len) \
            lock_reg(fd, F_SETLK, F_RDLCK, offset, whence, len)
#define readw_lock(fd, offset, whence, len) \
            lock_reg(fd, F_SETLKW, F_RDLCK, offset, whence, len)
#define write_lock(fd, offset, whence, len) \
            lock_reg(fd, F_SETLK, F_WRLCK, offset, whence, len)
#define writew_lock(fd, offset, whence, len) \
            lock_reg(fd, F_SETLKW, F_WRLCK, offset, whence, len)
#define un_lock(fd, offset, whence, len) \
            lock_reg(fd, F_SETLK, F_UNLCK, offset, whence, len)

pid_t   lock_test(int, int, off_t, int, off_t);
                                     /* Program 12.3 */

#define is_readlock(fd, offset, whence, len) \
            lock_test(fd, F_RDLCK, offset, whence, len)
#define is_writelock(fd, offset, whence, len) \
            lock_test(fd, F_WRLCK, offset, whence, len)

void    err_dump(const char *, ...);    /* Appendix B */
void    err_msg(const char *, ...);
void    err_quit(const char *, ...);
void    err_ret(const char *, ...);
void    err_sys(const char *, ...);

void    log_msg(const char *, ...);     /* Appendix B */
void    log_open(const char *, int, int);
void    log_quit(const char *, ...);
void    log_ret(const char *, ...);
void    log_sys(const char *, ...);

void    TELL_WAIT(void);        /* parent/child from Section 8.8 */
void    TELL_PARENT(pid_t);
void    TELL_CHILD(pid_t);
void    WAIT_PARENT(void);
void    WAIT_CHILD(void);

#endif  /* __ourhdr_h */
```

程序中先包括一般系统头文件,然后再包括ourhdr.h，这样就能解决某些系统之间的差别（例如4.3BSD Reno中没有定义SIG_ERR），并且也可定义一些我们的函数原型，而这些仅当包括一般系统头文件之后才是需要的。当在原型中引用未定义的结构时，某些 ANSI C编译程序会认为不正常。

B.2 标准出错处理例程

我们提供了两个出错处理例程，它们可用于本书中大多数实例以处理各种出错情况。一个例程以err_开头，并向标准出错文件输出一条出错消息。另一个例程以 log_开头，用于精灵进程（见第13章），它们多半没有控制终端。

提供了这些出错处理函数后，只要在程序中写一行代码就可以进行出错处理，例如：

```
if (出错条件)
    err_dump(带任意参数的printf格式);
```

这样也就不再需要使用下列代码：

```
if (出错条件) {
    char   buff[200];
    sprintf(buff,带任意参数的printf格式);
    perror(buff);
    abort( );
}
```

我们的出错处理函数使用了ANSI C的变长参数表功能。其详细说明见Kernighan和Ritchie〔1998〕的7.3节。应当注意的是这一 ANSI C功能与早期系统（例如 SVR3和4.3BSD）提供的varargs功能不同。宏的名字相同，但更改了某些宏的参数。

表B-1列出了各个出错处理函数之间的区别。

表B-1 标准出错处理函数

函数	strerror(errno)?	终止?
Err_ret	是	return;
Err_sys	是	exit(1);
Err_dump	是	abort ( );
Err_msg	否	return;
Err_quit	否	exit (1);
Log_ret	是	return;
Log_sys	是	exit(2);
Log_msg	否	return;
Log_quit	否	exit(2);

程序B-2包括了输出至标准出错文件的各个出错处理函数。

程序B-2 输出至标准出错文件的出错处理函数

```
#include    <errno.h>       /* for definition of errno */
#include    <stdarg.h>      /* ANSI C header file */
#include    "ourhdr.h"

static void err_doit(int, const char *, va_list);
```

```
char    *pname = NULL;      /* caller can set this from argv[0] */

/* Nonfatal error related to a system call.
 * Print a message and return. */

void
err_ret(const char *fmt, ...)
{
    va_list     ap;
    va_start(ap, fmt);
    err_doit(1, fmt, ap);
    va_end(ap);
    return;
}

/* Fatal error related to a system call.
 * Print a message and terminate. */

void
err_sys(const char *fmt, ...)
{
    va_list     ap;
    va_start(ap, fmt);
    err_doit(1, fmt, ap);
    va_end(ap);
    exit(1);

}

/* Fatal error related to a system call.
 * Print a message, dump core, and terminate. */

void
err_dump(const char *fmt, ...)
{
    va_list     ap;

    va_start(ap, fmt);
    err_doit(1, fmt, ap);
    va_end(ap);
    abort();        /* dump core and terminate */
    exit(1);        /* shouldn't get here */
}

/* Nonfatal error unrelated to a system call.
 * Print a message and return. */

void
err_msg(const char *fmt, ...)
{
    va_list     ap;

    va_start(ap, fmt);
    err_doit(0, fmt, ap);
    va_end(ap);
    return;
}

/* Fatal error unrelated to a system call.
 * Print a message and terminate. */

void
err_quit(const char *fmt, ...)
{
    va_list     ap;
```

```
    va_start(ap, fmt);
    err_doit(0, fmt, ap);
    va_end(ap);
    exit(1);
}

/* Print a message and return to caller.
 * Caller specifies "errnoflag". */

static void
err_doit(int errnoflag, const char *fmt, va_list ap)
{
    int     errno_save;
    char    buf[MAXLINE];

    errno_save = errno;     /* value caller might want printed */
    vsprintf(buf, fmt, ap);
    if (errnoflag)
        sprintf(buf+strlen(buf), ": %s", strerror(errno_save));
    strcat(buf, "\n");
    fflush(stdout);     /* in case stdout and stderr are the same */
    fputs(buf, stderr);
    fflush(NULL);       /* flushes all stdio output streams */
    return;
}
```

程序B-3包括了各log_XXX出错处理函数。若进程不以精灵进程方式进行，那么调用者应当定义变量debug，并将其设置为非0值。在这种情况下，出错消息被送至标准出错文件。若debug标志为0，则使用syslog设施（见13.4.2节）。

程序B-3　用于精灵进程的处理函数

```
/* Error routines for programs that can run as a daemon. */

#include    <errno.h>       /* for definition of errno */
#include    <stdarg.h>      /* ANSI C header file */
#include    <syslog.h>
#include    "ourhdr.h"

static void log_doit(int, int, const char *, va_list ap);

extern int  debug;      /* caller must define and set this:
                           nonzero if interactive, zero if daemon */

/* Initialize syslog(), if running as daemon. */

void
log_open(const char *ident, int option, int facility)
{
    if (debug == 0)
        openlog(ident, option, facility);
}

/* Nonfatal error related to a system call.
 * Print a message with the system's errno value and return. */

void
log_ret(const char *fmt, ...)
{
    va_list     ap;

    va_start(ap, fmt);
    log_doit(1, LOG_ERR, fmt, ap);
    va_end(ap);
```

```
    return;
}

/* Fatal error related to a system call.
 * Print a message and terminate. */

void
log_sys(const char *fmt, ...)
{
    va_list     ap;

    va_start(ap, fmt);
    log_doit(1, LOG_ERR, fmt, ap);
    va_end(ap);
    exit(2);
}

/* Nonfatal error unrelated to a system call.
 * Print a message and return. */

void
log_msg(const char *fmt, ...)
{
    va_list     ap;

    va_start(ap, fmt);
    log_doit(0, LOG_ERR, fmt, ap);
    va_end(ap);
    return;
}

/* Fatal error unrelated to a system call.
 * Print a message and terminate. */

void
log_quit(const char *fmt, ...)
{
    va_list     ap;

    va_start(ap, fmt);
    log_doit(0, LOG_ERR, fmt, ap);
    va_end(ap);
    exit(2);
}

/* Print a message and return to caller.
 * Caller specifies "errnoflag" and "priority". */

static void
log_doit(int errnoflag, int priority, const char *fmt, va_list ap)
{
    int     errno_save;
    char    buf[MAXLINE];

    errno_save = errno;     /* value caller might want printed */
    vsprintf(buf, fmt, ap);
    if (errnoflag)
        sprintf(buf+strlen(buf), ": %s", strerror(errno_save));
    strcat(buf, "\n");
    if (debug) {
        fflush(stdout);
        fputs(buf, stderr);
        fflush(stderr);
    } else
        syslog(priority, buf);
    return;
}
```