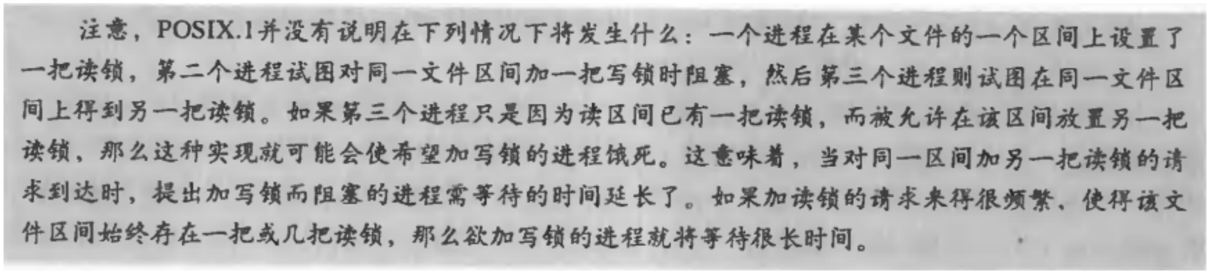
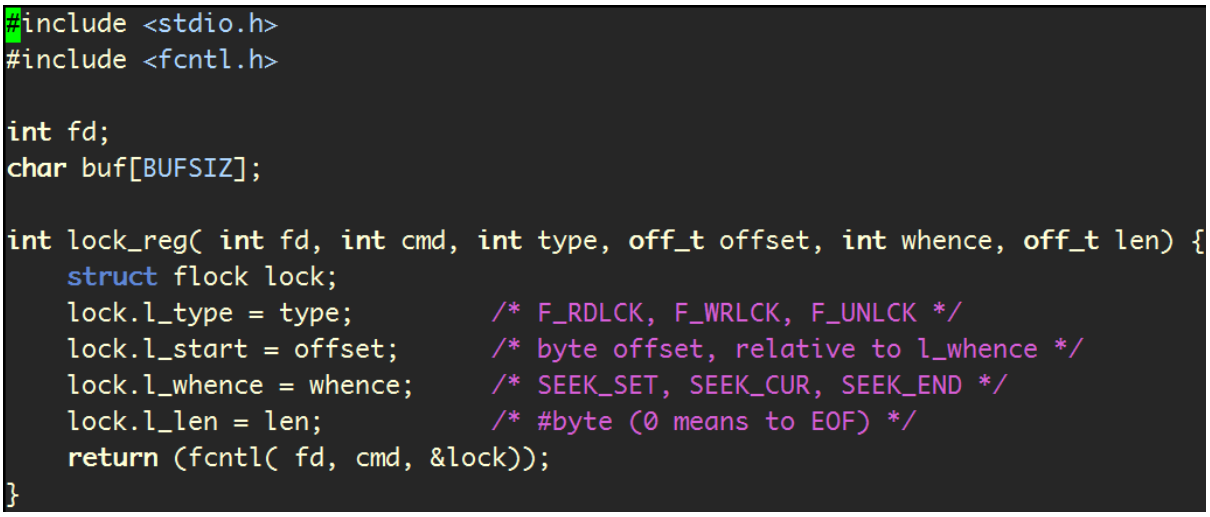
第十四章 高级I/O

习题：

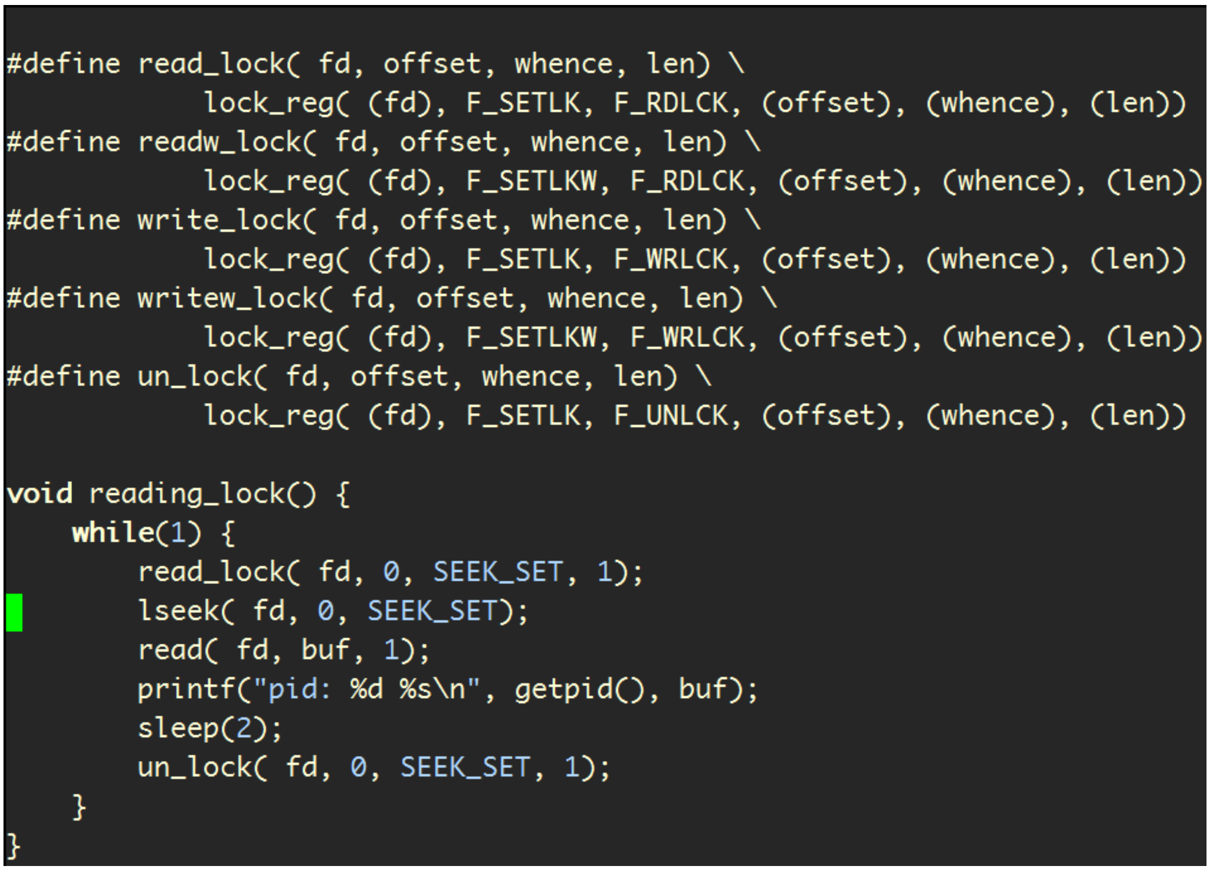
14-1、会被饿死，和下面书中截图描述相同；



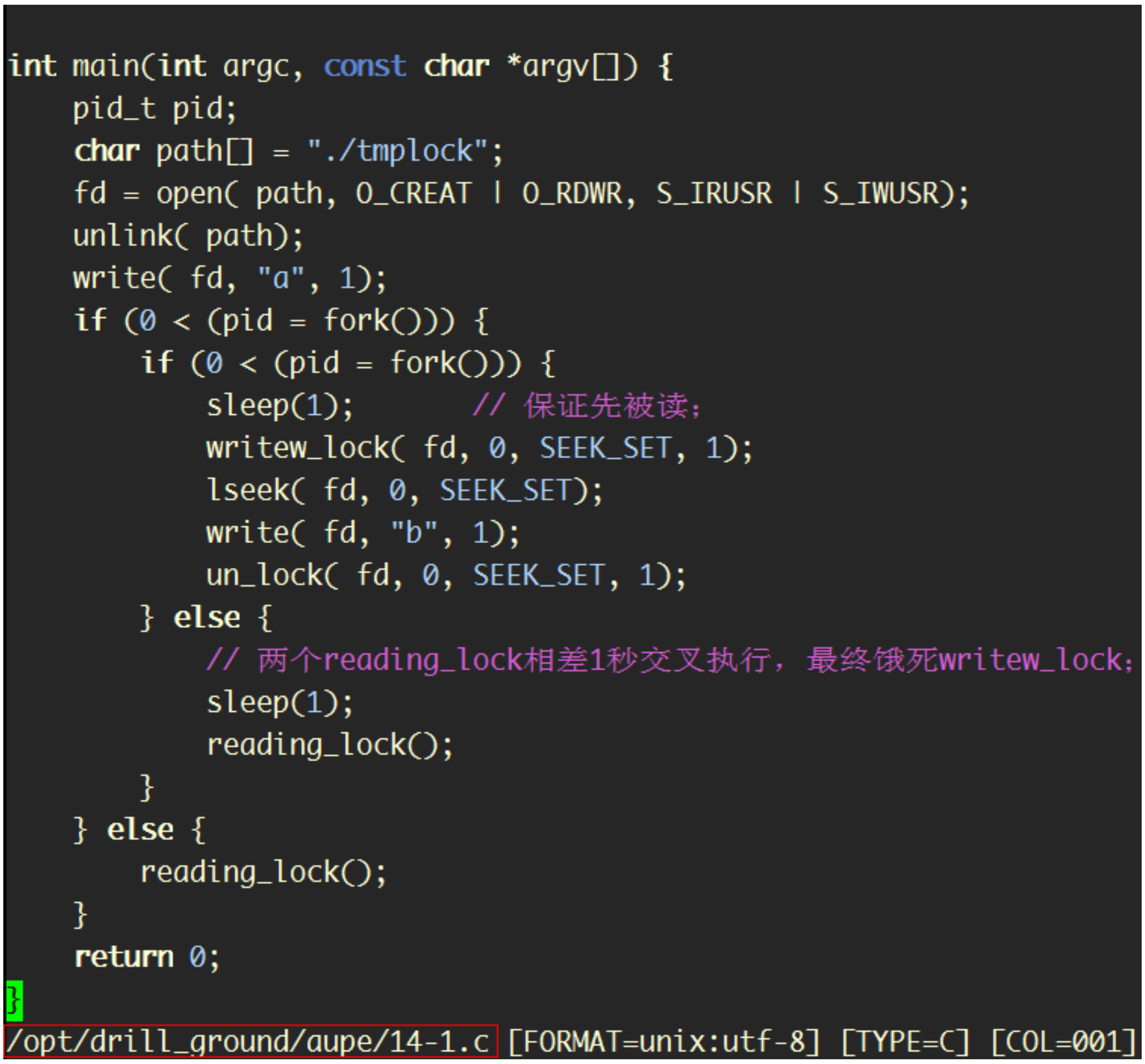
注意，POSIX.1并没有说明在下列情况下将发生什么：一个进程在某个文件的一个区间上设置了一把读锁，第二个进程试图对同一文件区间加一把写锁时阻塞，然后第三个进程则试图在同一文件区间上得到另一把读锁。如果第三个进程只是因为读区间已有一把读锁，而被允许在该区间放置另一把读锁，那么这种实现就可能会使希望加写锁的进程饿死。这意味着，当对同一区间加另一把读锁的请求到达时，提出加写锁而阻塞的进程需等待的时间延长了。如果加读锁的请求来得很频繁，使得该文件区间始终存在一把或几把读锁，那么欲加写锁的进程就将等待很长时间。



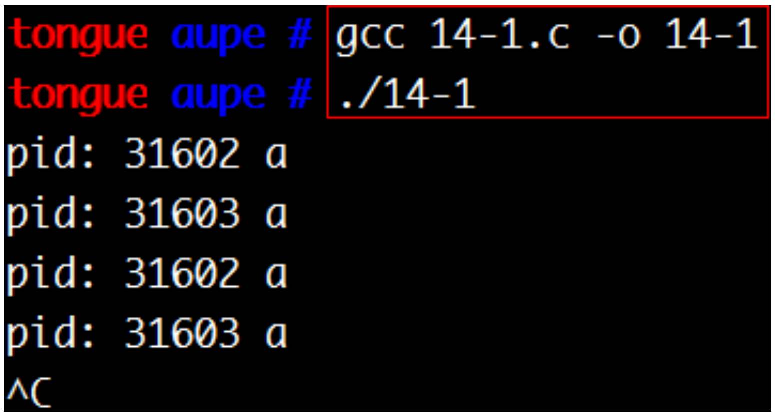
```c
#include <stdio.h>
#include <fcntl.h>

int fd;
char buf[BUFSIZ];

int lock_reg( int fd, int cmd, int type, off_t offset, int whence, off_t len) {
    struct flock lock;
    lock.l_type = type;         /* F_RDLCK, F_WRLCK, F_UNLCK */
    lock.l_start = offset;      /* byte offset, relative to l_whence */
    lock.l_whence = whence;     /* SEEK_SET, SEEK_CUR, SEEK_END */
    lock.l_len = len;           /* #byte (0 means to EOF) */
    return (fcntl( fd, cmd, &lock));
}
```



```c
#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))

void reading_lock() {
    while(1) {
        read_lock( fd, 0, SEEK_SET, 1);
        lseek( fd, 0, SEEK_SET);
        read( fd, buf, 1);
        printf("pid: %d %s\n", getpid(), buf);
        sleep(2);
        un_lock( fd, 0, SEEK_SET, 1);
    }
}
```

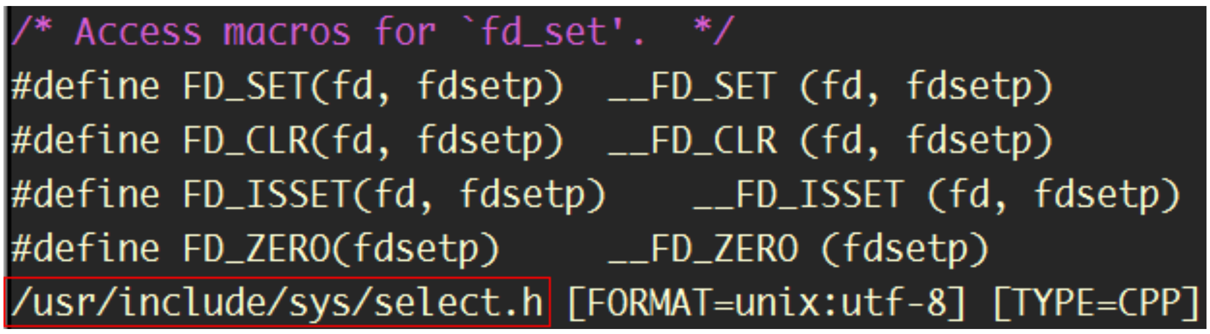


```c
int main(int argc, const char *argv[]) {
    pid_t pid;
    char path[] = "./tmplock";
    fd = open( path, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    unlink( path);
    write( fd, "a", 1);
    if (0 < (pid = fork())) {
        if (0 < (pid = fork())) {
            sleep(1);        // 保证先被读;
            writew_lock( fd, 0, SEEK_SET, 1);
            lseek( fd, 0, SEEK_SET);
            write( fd, "b", 1);
            un_lock( fd, 0, SEEK_SET, 1);
        } else {
            // 两个reading_lock相差1秒交叉执行，最终饿死writew_lock;
            sleep(1);
            reading_lock();
        }
    } else {
        reading_lock();
    }
    return 0;
}
/opt/drill_ground/aupe/14-1.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001]
```

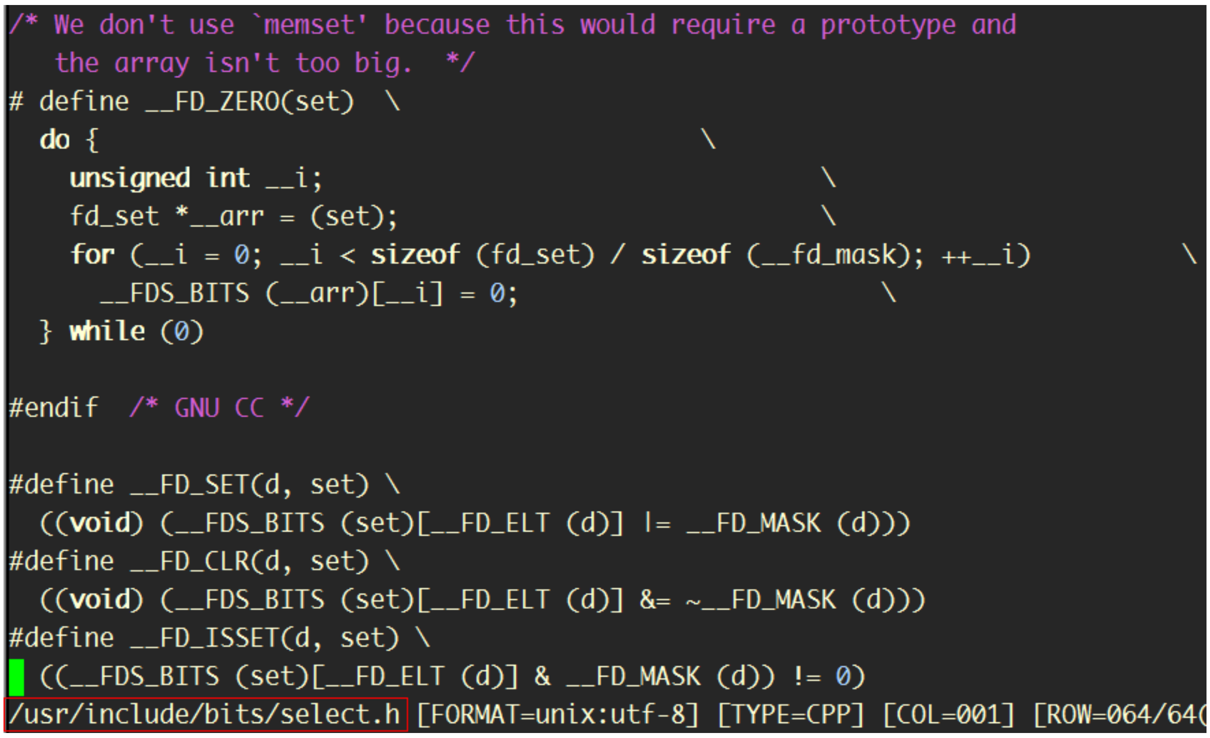


```
tongue aupe # gcc 14-1.c -o 14-1
tongue aupe # ./14-1
pid: 31602 a
pid: 31603 a
pid: 31602 a
pid: 31603 a
^C
```

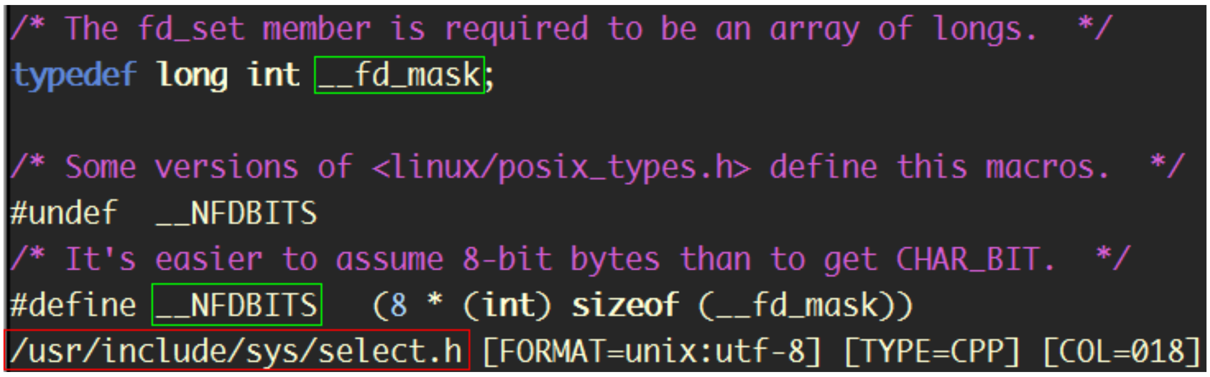
14-2、



```
/* Access macros for `fd_set'.  */
#define FD_SET(fd, fdsetp)  __FD_SET (fd, fdsetp)
#define FD_CLR(fd, fdsetp)  __FD_CLR (fd, fdsetp)
#define FD_ISSET(fd, fdsetp)    __FD_ISSET (fd, fdsetp)
#define FD_ZERO(fdsetp)     __FD_ZERO (fdsetp)
/usr/include/sys/select.h [FORMAT=unix:utf-8] [TYPE=CPP]
```

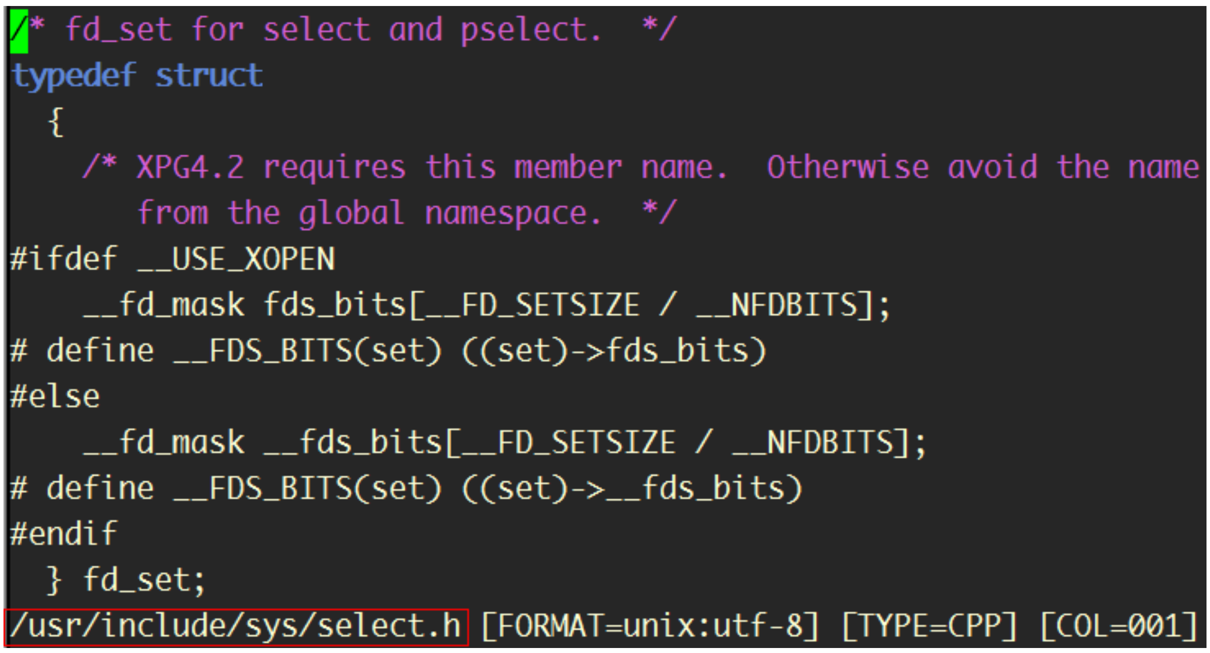


```
/* We don't use `memset' because this would require a prototype and
   the array isn't too big.  */
# define __FD_ZERO(set)  \
  do {                                    \
    unsigned int __i;                         \
    fd_set *__arr = (set);                    \
    for (__i = 0; __i < sizeof (fd_set) / sizeof (__fd_mask); ++__i)         \
      __FDS_BITS (__arr)[__i] = 0;                \
  } while (0)

#endif  /* GNU CC */

#define __FD_SET(d, set) \
  ((void) (__FDS_BITS (set)[__FD_ELT (d)] |= __FD_MASK (d)))
#define __FD_CLR(d, set) \
  ((void) (__FDS_BITS (set)[__FD_ELT (d)] &= ~__FD_MASK (d)))
#define __FD_ISSET(d, set) \
  ((__FDS_BITS (set)[__FD_ELT (d)] & __FD_MASK (d)) != 0)
/usr/include/bits/select.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=064/64
```

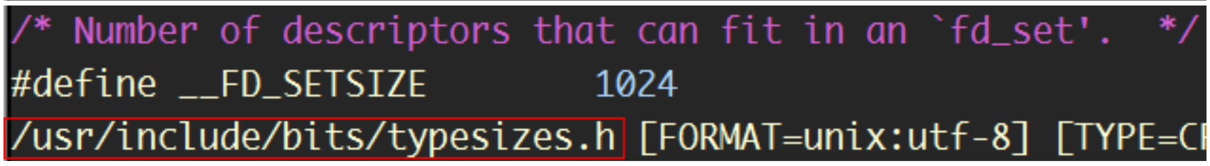


```
/* The fd_set member is required to be an array of longs.  */
typedef long int __fd_mask;

/* Some versions of <linux/posix_types.h> define this macros.  */
#undef  __NFDBITS
/* It's easier to assume 8-bit bytes than to get CHAR_BIT.  */
#define __NFDBITS   (8 * (int) sizeof (__fd_mask))
/usr/include/sys/select.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=018]
```

一位代表一个文件描述符，一个字节8位，一个长整形64位，1024除以64等于16，
所以需要16个长整形的空间才能完整描述1024位；
__NFDBITS = (8 * (8) sizeof(long int)) = 64
由上图发现__fd_mask是个长整形，__fds_bits[1024(__FD_SETSIZE) / (64)__NFDBITS]
等于__fds_bits[16]，即连续的16个long int空间，1024位；



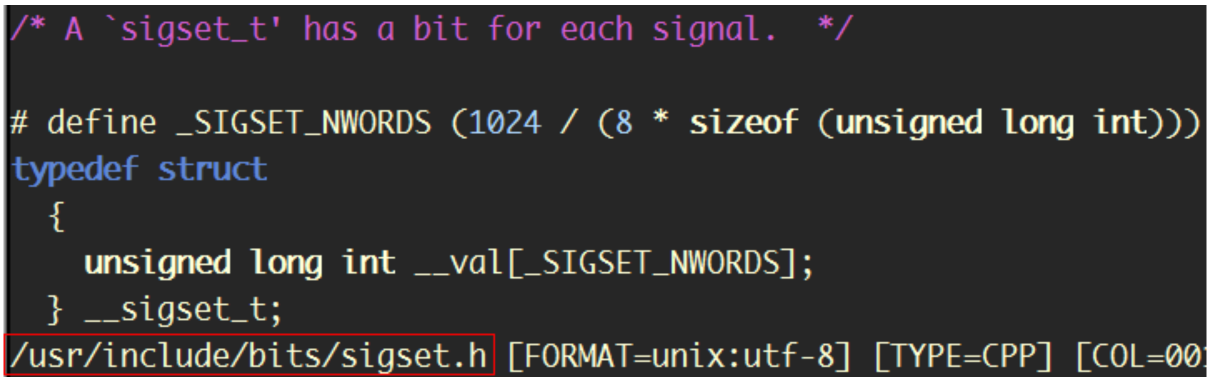
```
/* fd_set for select and pselect.  */
typedef struct
  {
    /* XPG4.2 requires this member name.  Otherwise avoid the name
       from the global namespace.  */
#ifdef __USE_XOPEN
    __fd_mask fds_bits[__FD_SETSIZE / __NFDBITS];
# define __FDS_BITS(set) ((set)->fds_bits)
#else
    __fd_mask __fds_bits[__FD_SETSIZE / __NFDBITS];
# define __FDS_BITS(set) ((set)->__fds_bits)
#endif
  } fd_set;
/usr/include/sys/select.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001]
```



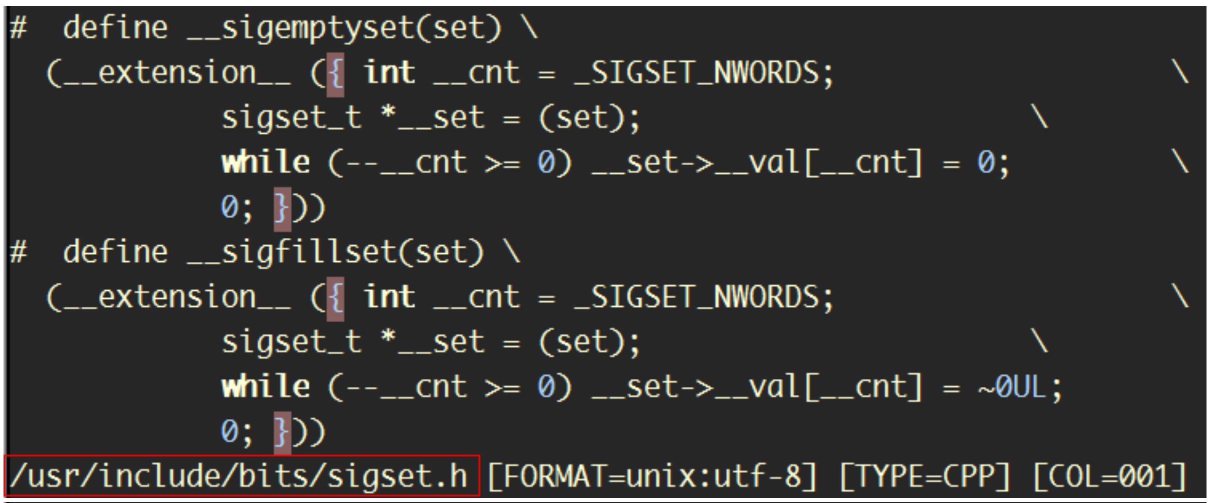
```
/* Number of descriptors that can fit in an `fd_set'.  */
#define __FD_SETSIZE        1024
/usr/include/bits/typesizes.h [FORMAT=unix:utf-8] [TYPE=CF
```

源码位置：glibc-2.17/sysdeps/unix/sysv/linux/pselect.c
glibc-2.17/ports/sysdeps/unix/sysv/linux/generic/select.c

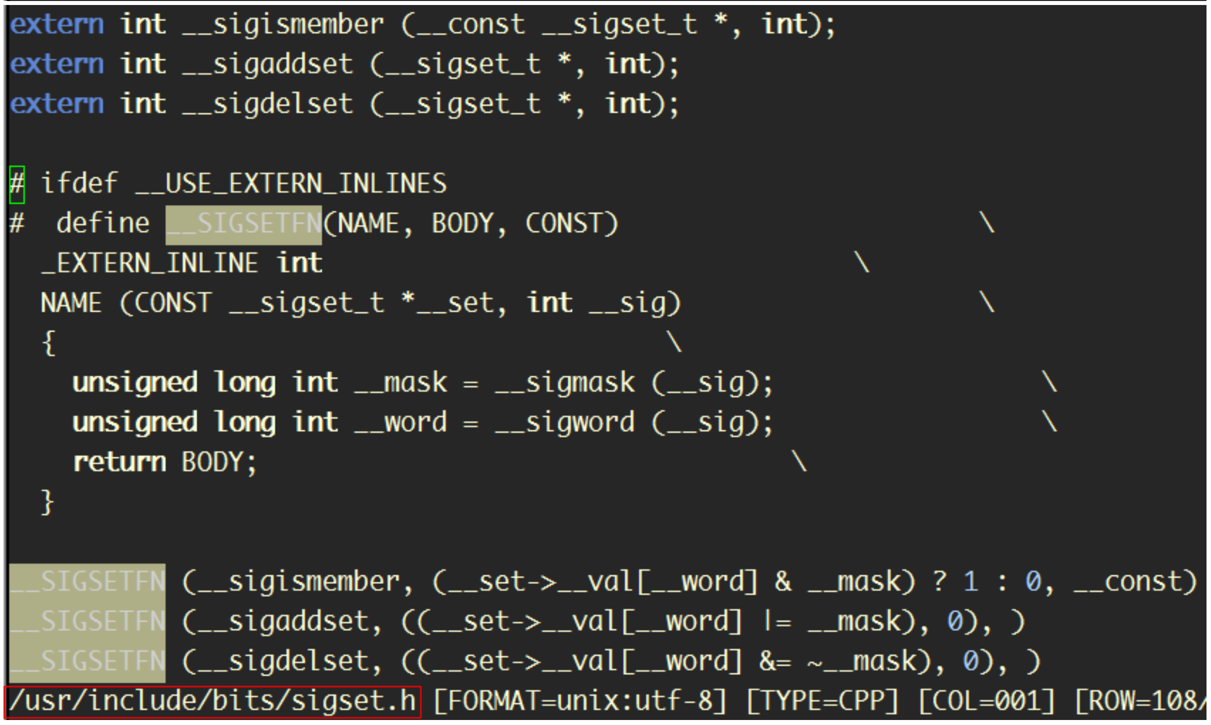
14-3、改"/usr/include/bits/typesizes.h"中__FD_SETSIZE的值为2048即可；
14-4、如下，他们实现思路基本相同；
信号集函数实现：



```
/* A `sigset_t' has a bit for each signal.  */

# define _SIGSET_NWORDS (1024 / (8 * sizeof (unsigned long int)))
typedef struct
  {
    unsigned long int __val[_SIGSET_NWORDS];
  } __sigset_t;
/usr/include/bits/sigset.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=00
```



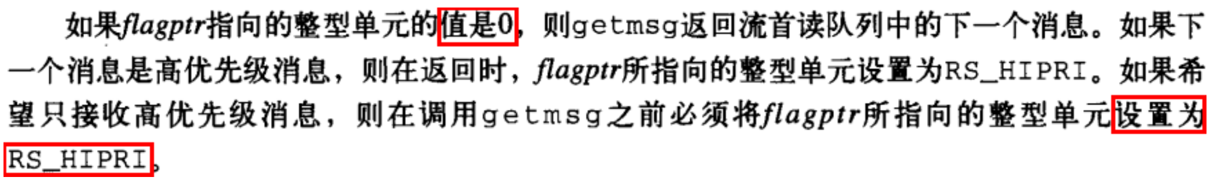
```
#  define __sigemptyset(set) \
  (__extension__ ({ int __cnt = _SIGSET_NWORDS;                \
                    sigset_t *__set = (set);               \
                    while (--__cnt >= 0) __set->__val[__cnt] = 0;     \
                    0; }))
#  define __sigfillset(set) \
  (__extension__ ({ int __cnt = _SIGSET_NWORDS;                \
                    sigset_t *__set = (set);               \
                    while (--__cnt >= 0) __set->__val[__cnt] = ~0UL;
                    0; }))
/usr/include/bits/sigset.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001]
```



```
extern int __sigismember (__const __sigset_t *, int);
extern int __sigaddset (__sigset_t *, int);
extern int __sigdelset (__sigset_t *, int);

# ifdef __USE_EXTERN_INLINES
#  define __SIGSETFN(NAME, BODY, CONST)                    \
  _EXTERN_INLINE int                                \
  NAME (CONST __sigset_t *__set, int __sig)                \
  {                                                 \
    unsigned long int __mask = __sigmask (__sig);              \
    unsigned long int __word = __sigword (__sig);              \
    return BODY;                                   \
  }

__SIGSETFN (__sigismember, (__set->__val[__word] & __mask) ? 1 : 0, __const)
__SIGSETFN (__sigaddset, ((__set->__val[__word] |= __mask), 0), )
__SIGSETFN (__sigdelset, ((__set->__val[__word] &= ~__mask), 0), )
/usr/include/bits/sigset.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=108/
```

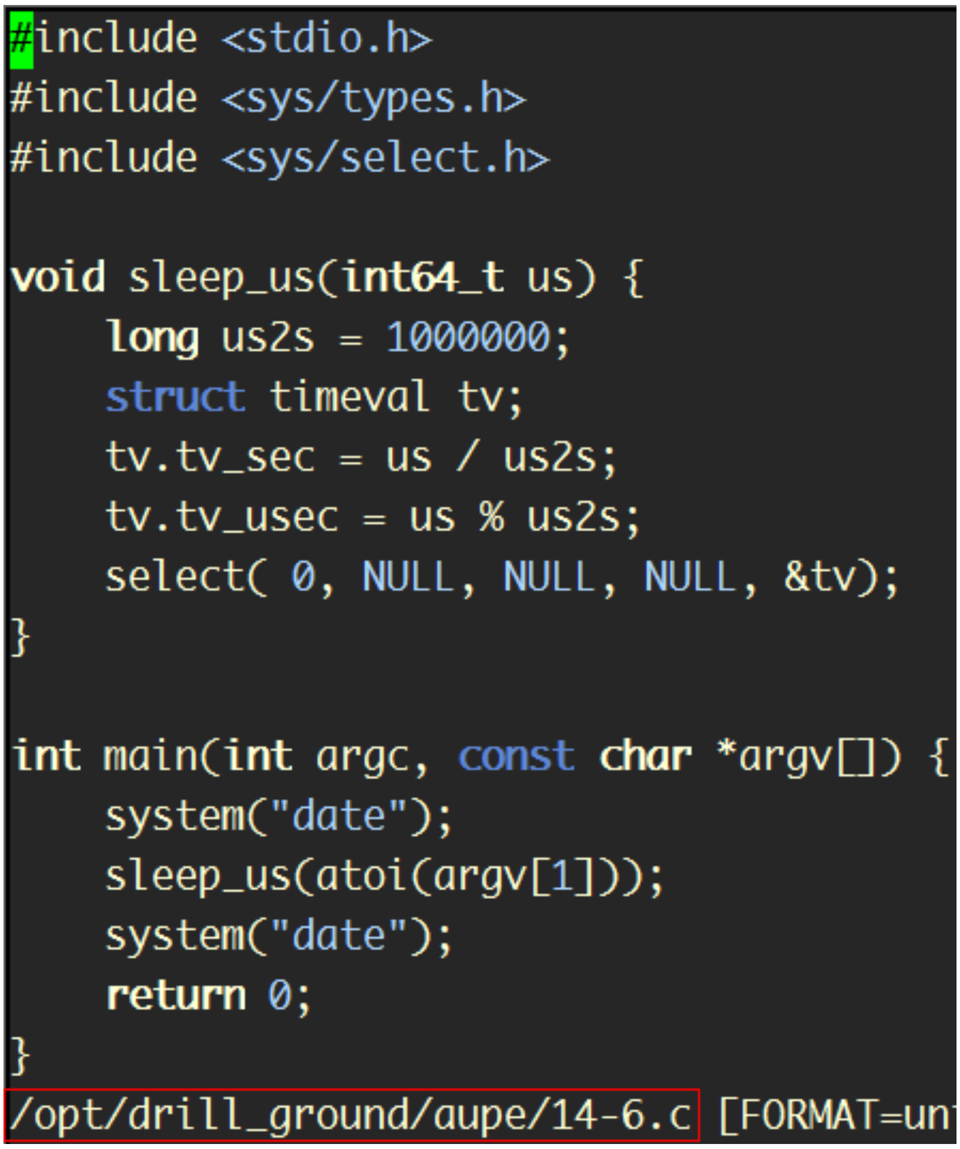
select函数实现：
见上面14-2题。

14-5、不考虑getpmsg，仅getmsg可返回两种，一种是任意消息，另一种是高优先级消息；

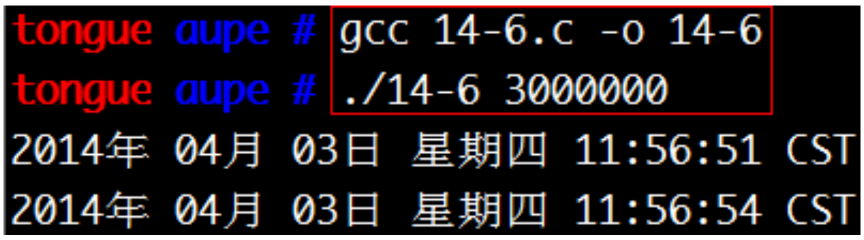


如果*flagptr*指向的整型单元的值是0，则getmsg返回流首读队列中的下一个消息。如果下一个消息是高优先级消息，则在返回时，*flagptr*所指向的整型单元设置为RS_HIPRI。如果希望只接收高优先级消息，则在调用getmsg之前必须将*flagptr*所指向的整型单元设置为RS_HIPRI。

14-6、如下：



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

void sleep_us(int64_t us) {
    long us2s = 1000000;
    struct timeval tv;
    tv.tv_sec = us / us2s;
    tv.tv_usec = us % us2s;
    select( 0, NULL, NULL, NULL, &tv);
}

int main(int argc, const char *argv[]) {
    system("date");
    sleep_us(atoi(argv[1]));
    system("date");
    return 0;
}
/opt/drill_ground/aupe/14-6.c [FORMAT=un
```



```
tongue aupe # gcc 14-6.c -o 14-6
tongue aupe # ./14-6 3000000
2014年 04月 03日 星期四 11:56:51 CST
2014年 04月 03日 星期四 11:56:54 CST
```

14-7、实现中，通过记录锁让lseek和read或lseek和write用起来像pread及pwrite，即在该程序中，如原子操作一样。当然也可以把程序中的lseek、read、write替换成pread、pwrite，那样的话就不需要记录锁了。

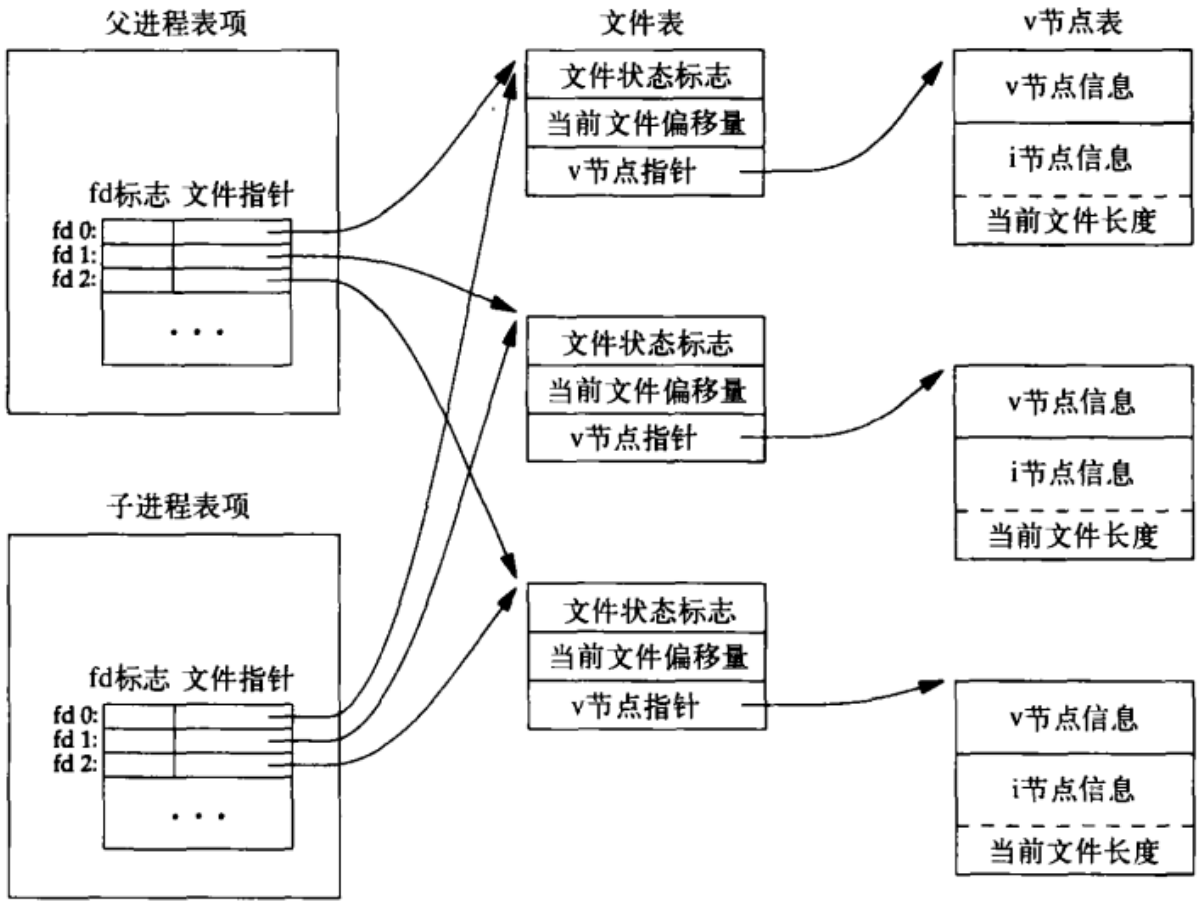
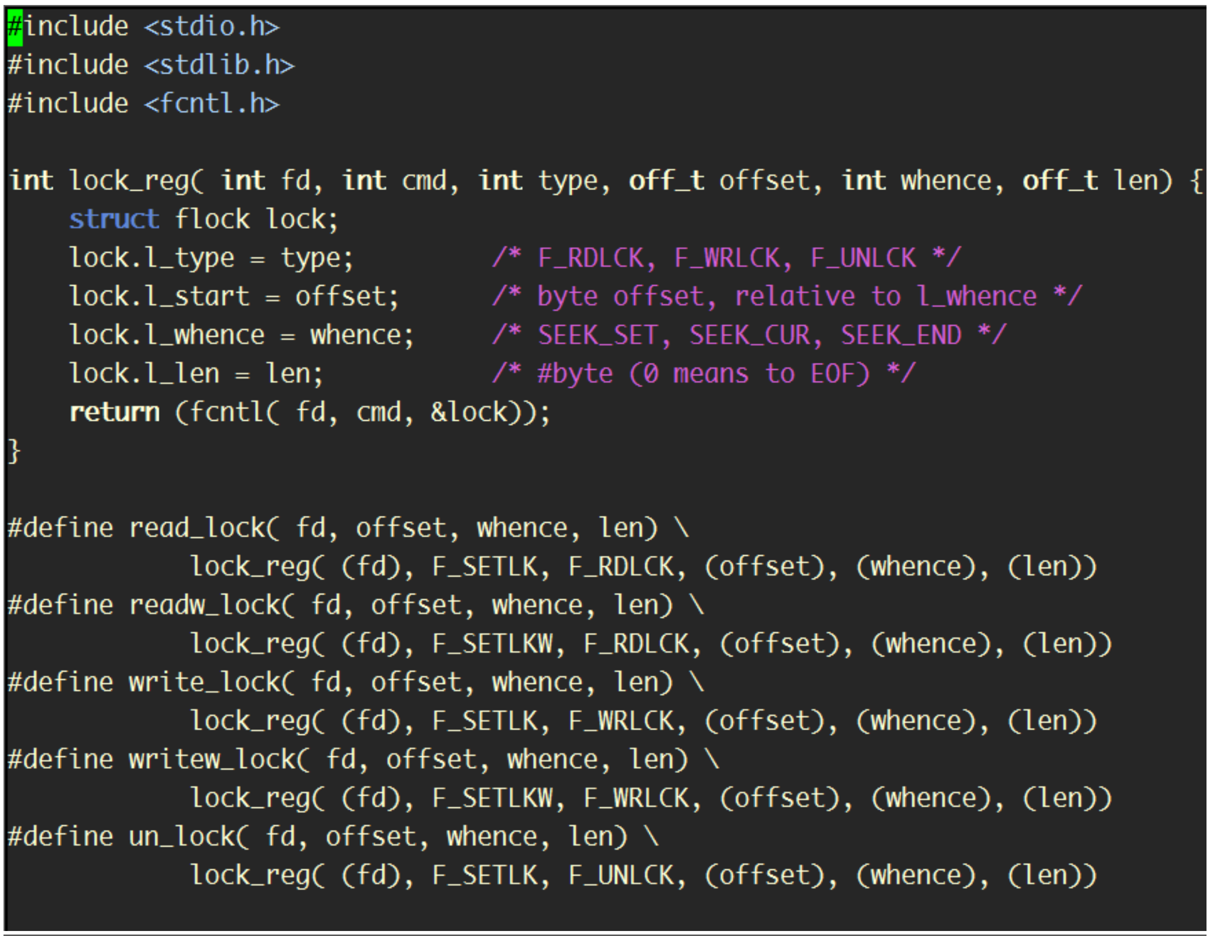
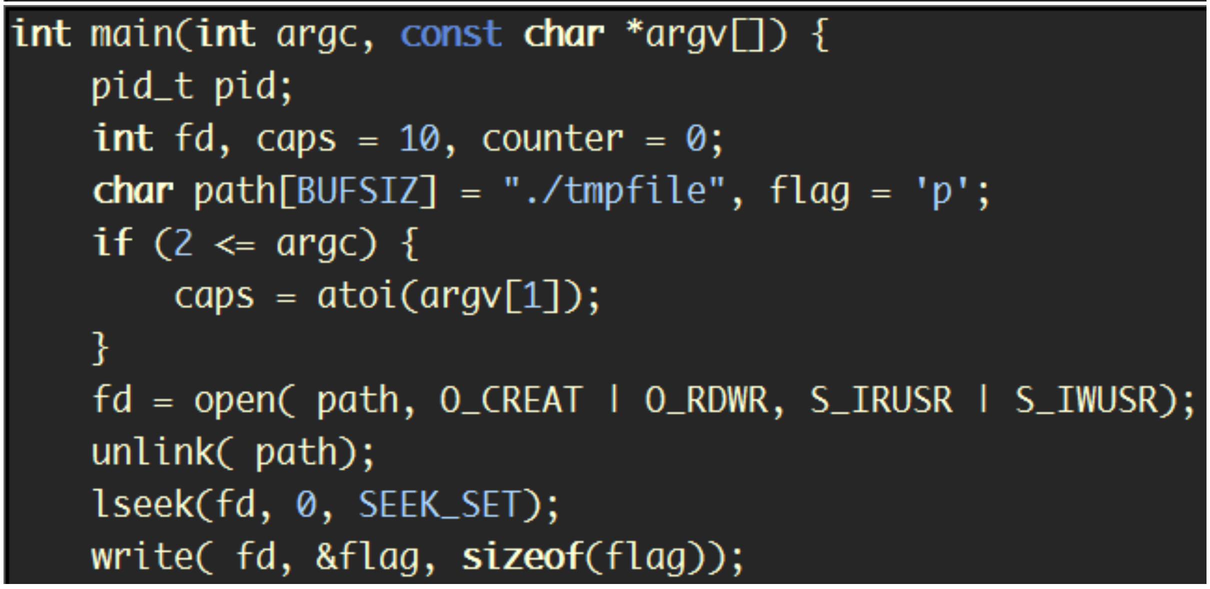


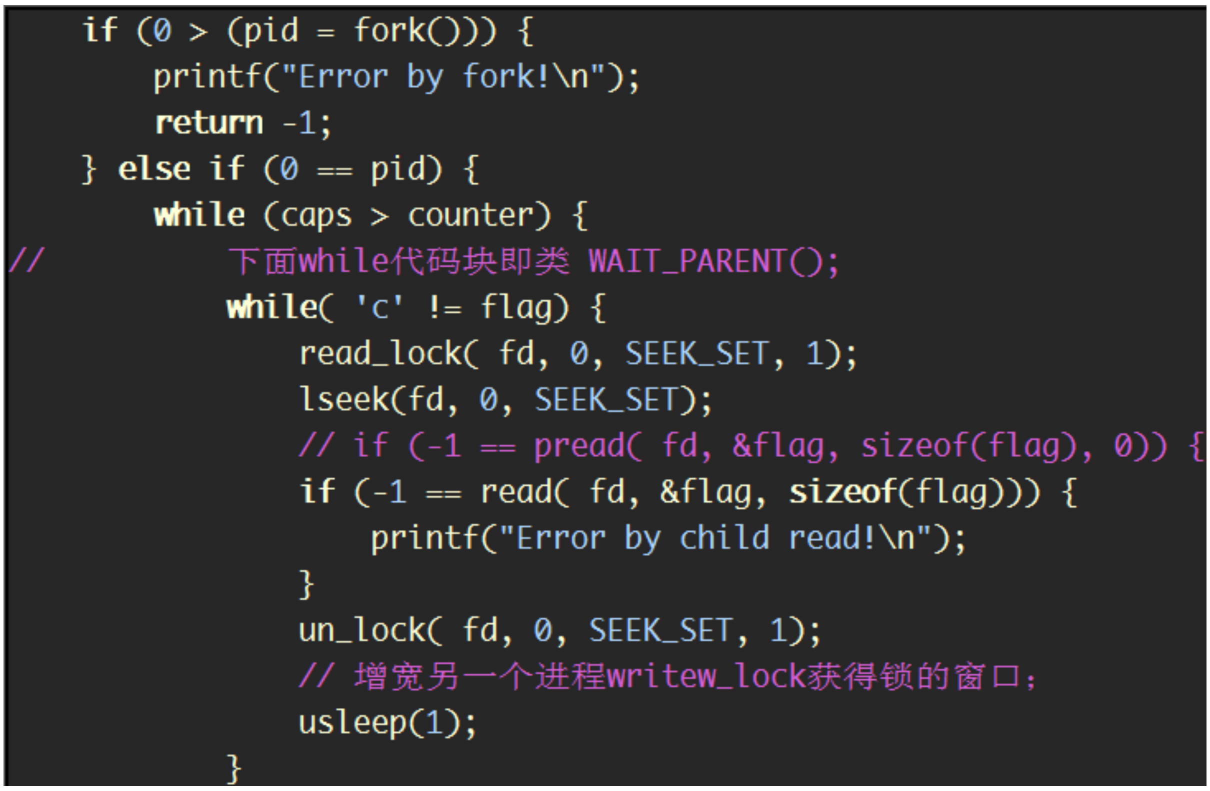
图8-1 调用fork之后父、子进程之间对打开文件的共享



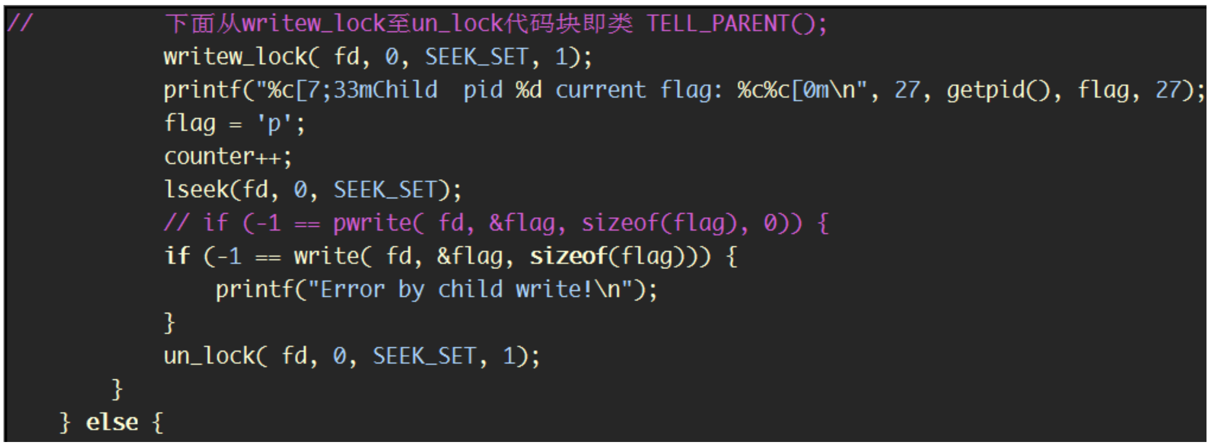
```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

int lock_reg( int fd, int cmd, int type, off_t offset, int whence, off_t len) {
    struct flock lock;
    lock.l_type = type;         /* F_RDLCK, F_WRLCK, F_UNLCK */
    lock.l_start = offset;      /* byte offset, relative to l_whence */
    lock.l_whence = whence;     /* SEEK_SET, SEEK_CUR, SEEK_END */
    lock.l_len = len;           /* #byte (0 means to EOF) */
    return (fcntl( fd, cmd, &lock));
}

#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))
```



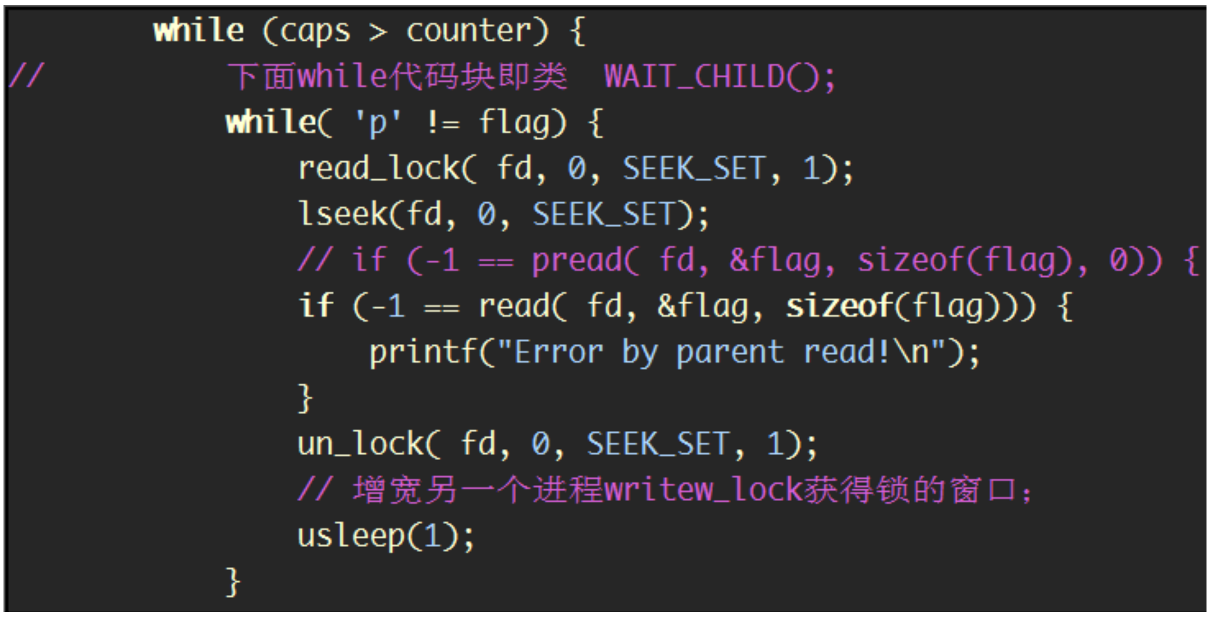
```c
int main(int argc, const char *argv[]) {
    pid_t pid;
    int fd, caps = 10, counter = 0;
    char path[BUFSIZ] = "./tmpfile", flag = 'p';
    if (2 <= argc) {
        caps = atoi(argv[1]);
    }
    fd = open( path, O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
    unlink( path);
    lseek(fd, 0, SEEK_SET);
    write( fd, &flag, sizeof(flag));
```



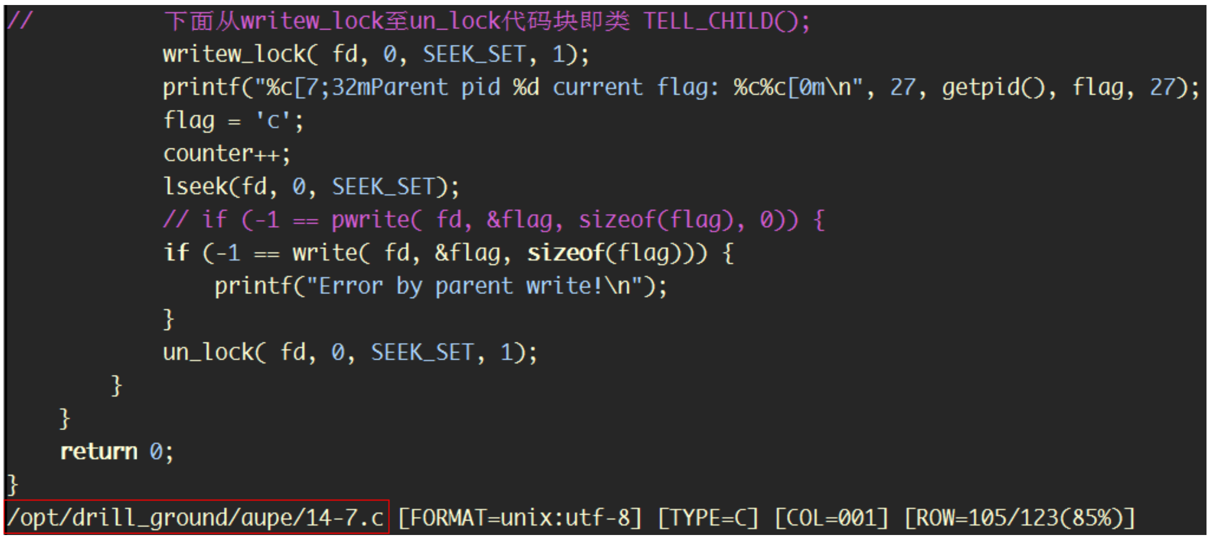
```
    if (0 > (pid = fork())) {
        printf("Error by fork!\n");
        return -1;
    } else if (0 == pid) {
        while (caps > counter) {
//          下面while代码块即类 WAIT_PARENT();
            while( 'c' != flag) {
                read_lock( fd, 0, SEEK_SET, 1);
                lseek(fd, 0, SEEK_SET);
                // if (-1 == pread( fd, &flag, sizeof(flag), 0)) {
                if (-1 == read( fd, &flag, sizeof(flag))) {
                    printf("Error by child read!\n");
                }
                un_lock( fd, 0, SEEK_SET, 1);
                // 增宽另一个进程writew_lock获得锁的窗口;
                usleep(1);
            }
```



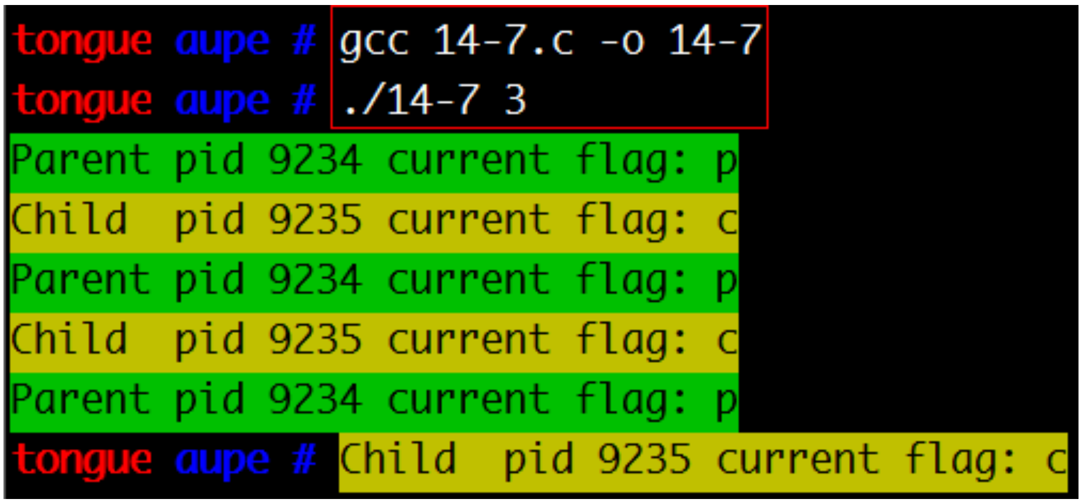
```
//          下面从writew_lock至un_lock代码块即类 TELL_PARENT();
            writew_lock( fd, 0, SEEK_SET, 1);
            printf("%c[7;33mChild  pid %d current flag: %c%c[0m\n", 27, getpid(), flag, 27);
            flag = 'p';
            counter++;
            lseek(fd, 0, SEEK_SET);
            // if (-1 == pwrite( fd, &flag, sizeof(flag), 0)) {
            if (-1 == write( fd, &flag, sizeof(flag))) {
                printf("Error by child write!\n");
            }
            un_lock( fd, 0, SEEK_SET, 1);
        }
    } else {
```



```
    while (caps > counter) {
//          下面while代码块即类  WAIT_CHILD();
      while( 'p' != flag) {
        read_lock( fd, 0, SEEK_SET, 1);
        lseek(fd, 0, SEEK_SET);
        // if (-1 == pread( fd, &flag, sizeof(flag), 0)) {
        if (-1 == read( fd, &flag, sizeof(flag))) {
          printf("Error by parent read!\n");
        }
        un_lock( fd, 0, SEEK_SET, 1);
        // 增宽另一个进程writew_lock获得锁的窗口;
        usleep(1);
      }
```

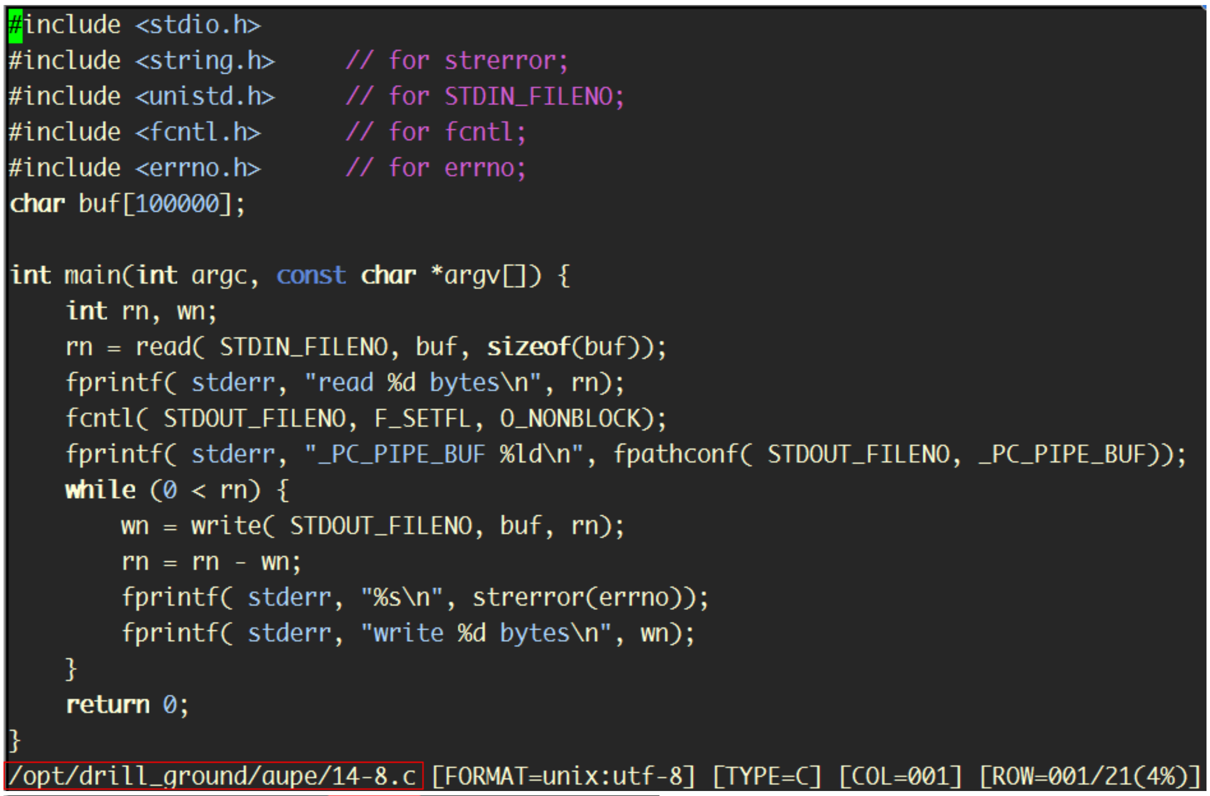


```
//          下面从writew_lock至un_lock代码块即类 TELL_CHILD();
      writew_lock( fd, 0, SEEK_SET, 1);
      printf("%c[7;32mParent pid %d current flag: %c%c[0m\n", 27, getpid(), flag, 27);
      flag = 'c';
      counter++;
      lseek(fd, 0, SEEK_SET);
      // if (-1 == pwrite( fd, &flag, sizeof(flag), 0)) {
      if (-1 == write( fd, &flag, sizeof(flag))) {
        printf("Error by parent write!\n");
      }
      un_lock( fd, 0, SEEK_SET, 1);
    }
  }
  return 0;
}
/opt/drill_ground/aupe/14-7.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=105/123(85%)]
```

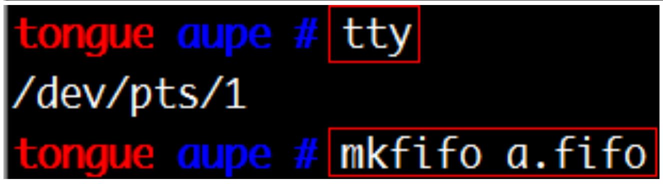


```
tongue aupe # gcc 14-7.c -o 14-7
tongue aupe # ./14-7 3
Parent pid 9234 current flag: p
Child  pid 9235 current flag: c
Parent pid 9234 current flag: p
Child  pid 9235 current flag: c
Parent pid 9234 current flag: p
tongue aupe # Child  pid 9235 current flag: c
```

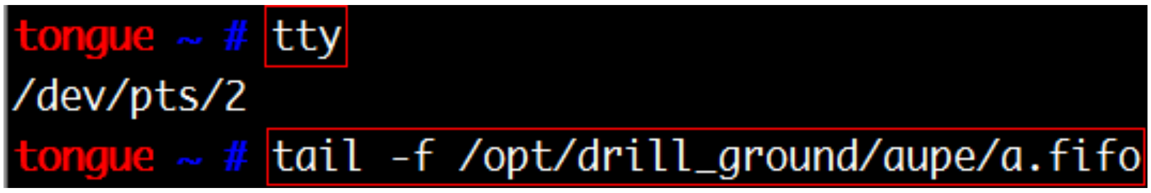
14-8、



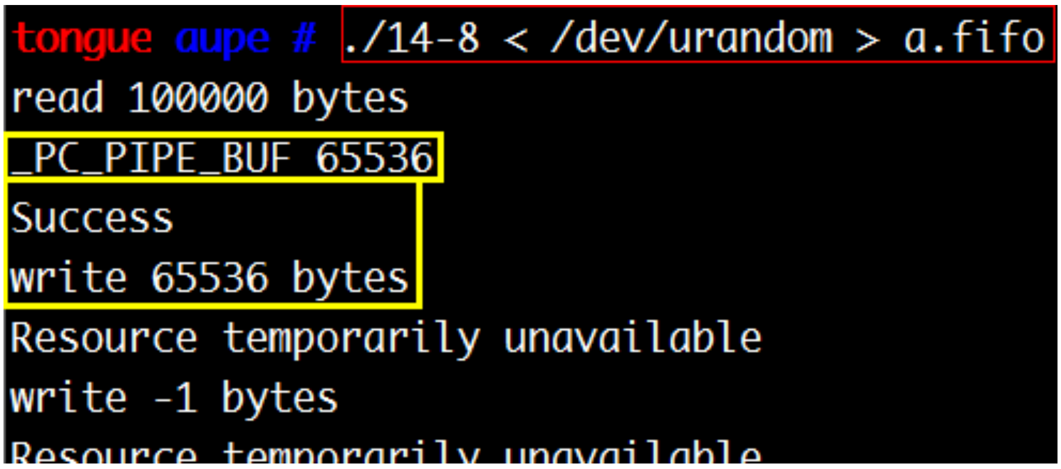
```
#include <stdio.h>
#include <string.h>     // for strerror;
#include <unistd.h>     // for STDIN_FILENO;
#include <fcntl.h>      // for fcntl;
#include <errno.h>      // for errno;
char buf[100000];

int main(int argc, const char *argv[]) {
    int rn, wn;
    rn = read( STDIN_FILENO, buf, sizeof(buf));
    fprintf( stderr, "read %d bytes\n", rn);
    fcntl( STDOUT_FILENO, F_SETFL, O_NONBLOCK);
    fprintf( stderr, "_PC_PIPE_BUF %ld\n", fpathconf( STDOUT_FILENO, _PC_PIPE_BUF));
    while (0 < rn) {
        wn = write( STDOUT_FILENO, buf, rn);
        rn = rn - wn;
        fprintf( stderr, "%s\n", strerror(errno));
        fprintf( stderr, "write %d bytes\n", wn);
    }
    return 0;
}
/opt/drill_ground/aupe/14-8.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=001/21(4%)]
```



```
tongue aupe # tty
/dev/pts/1
tongue aupe # mkfifo a.fifo
```

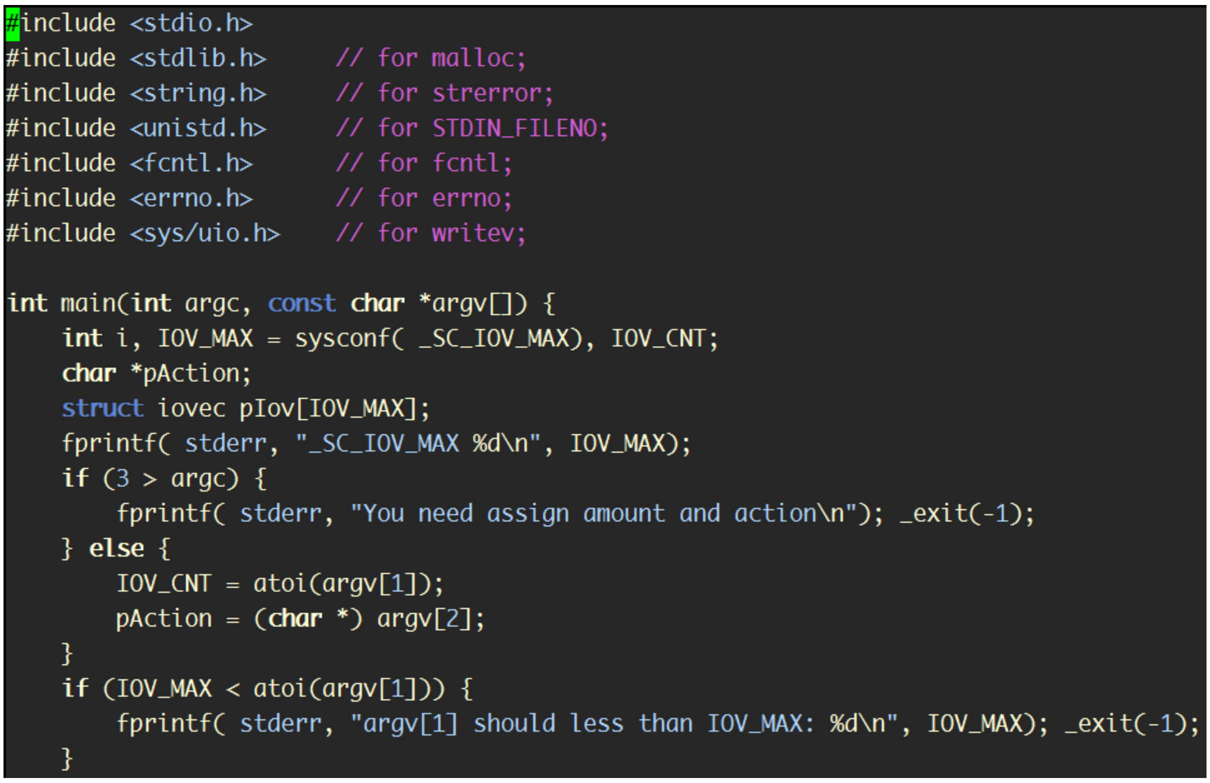


```
tongue ~ # tty
/dev/pts/2
tongue ~ # tail -f /opt/drill_ground/aupe/a.fifo
```

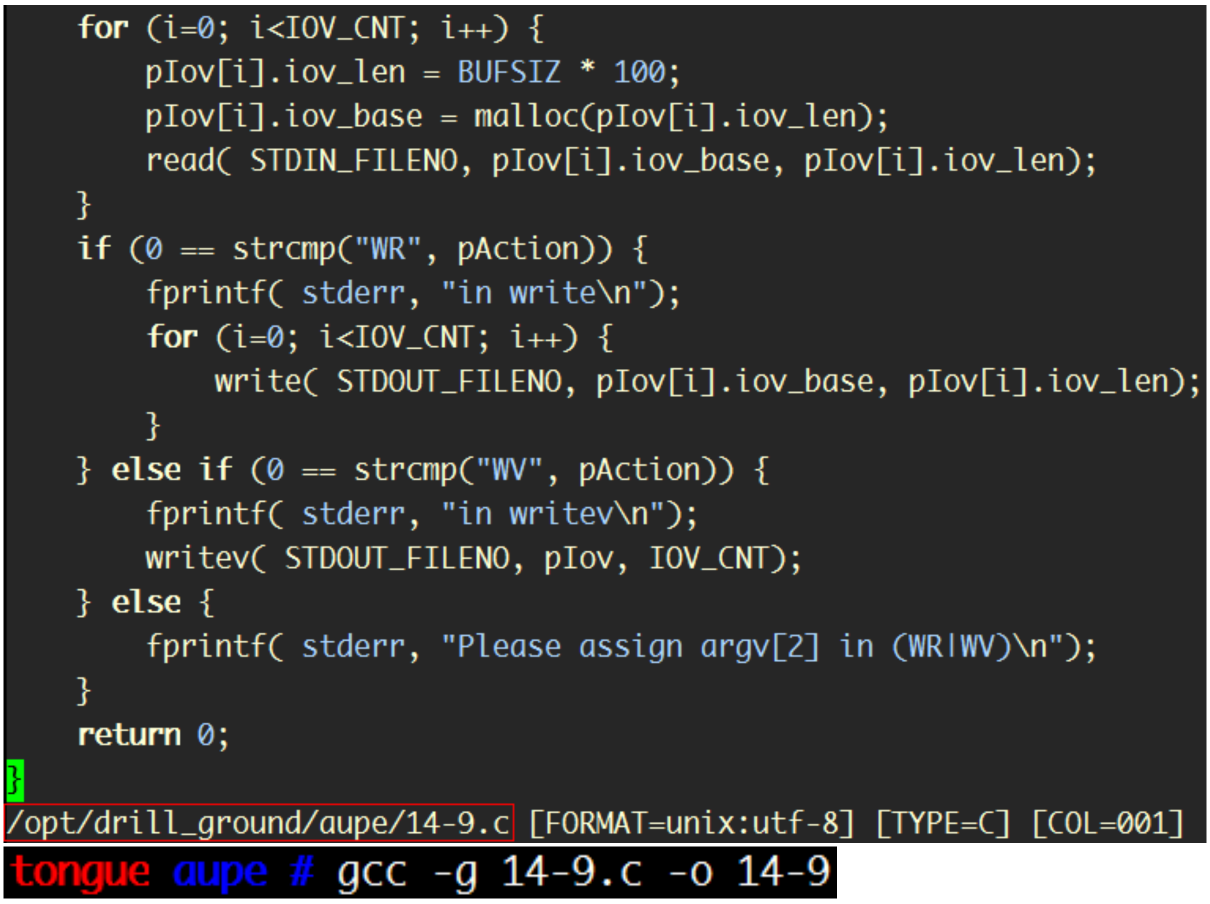


```
tongue aupe # ./14-8 < /dev/urandom > a.fifo
read 100000 bytes
_PC_PIPE_BUF 65536
Success
write 65536 bytes
Resource temporarily unavailable
write -1 bytes
Resource temporarily unavailable
```

14-9、基本差别不大，可能和linux对它的实现有关；

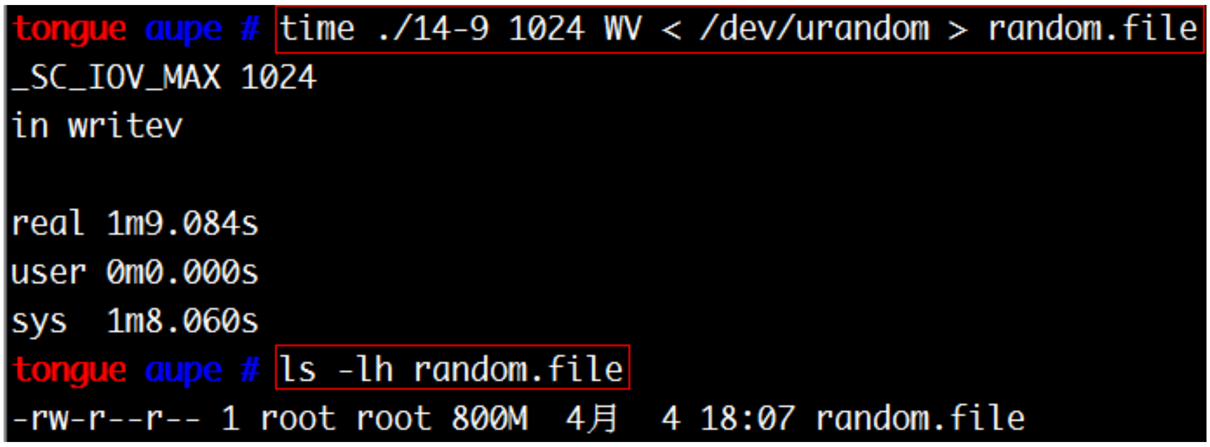


```c
#include <stdio.h>
#include <stdlib.h>     // for malloc;
#include <string.h>     // for strerror;
#include <unistd.h>     // for STDIN_FILENO;
#include <fcntl.h>      // for fcntl;
#include <errno.h>      // for errno;
#include <sys/uio.h>    // for writev;

int main(int argc, const char *argv[]) {
    int i, IOV_MAX = sysconf( _SC_IOV_MAX), IOV_CNT;
    char *pAction;
    struct iovec pIov[IOV_MAX];
    fprintf( stderr, "_SC_IOV_MAX %d\n", IOV_MAX);
    if (3 > argc) {
        fprintf( stderr, "You need assign amount and action\n"); _exit(-1);
    } else {
        IOV_CNT = atoi(argv[1]);
        pAction = (char *) argv[2];
    }
    if (IOV_MAX < atoi(argv[1])) {
        fprintf( stderr, "argv[1] should less than IOV_MAX: %d\n", IOV_MAX); _exit(-1);
    }
```

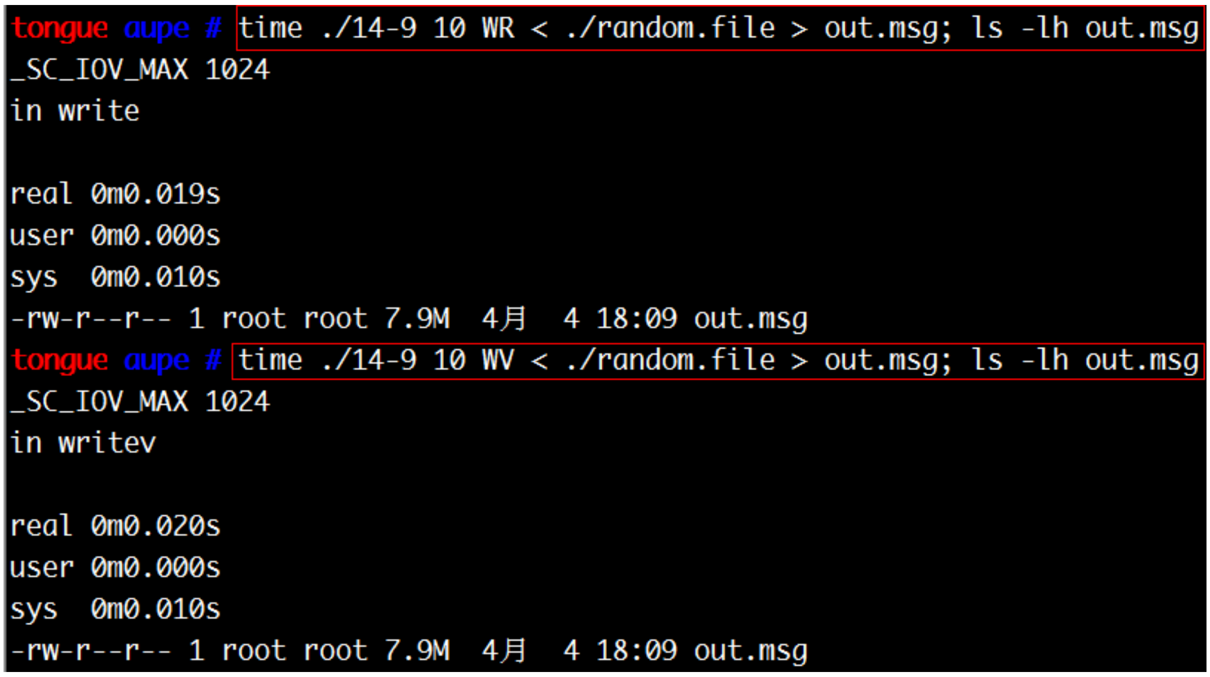


```c
    for (i=0; i<IOV_CNT; i++) {
        pIov[i].iov_len = BUFSIZ * 100;
        pIov[i].iov_base = malloc(pIov[i].iov_len);
        read( STDIN_FILENO, pIov[i].iov_base, pIov[i].iov_len);
    }
    if (0 == strcmp("WR", pAction)) {
        fprintf( stderr, "in write\n");
        for (i=0; i<IOV_CNT; i++) {
            write( STDOUT_FILENO, pIov[i].iov_base, pIov[i].iov_len);
        }
    } else if (0 == strcmp("WV", pAction)) {
        fprintf( stderr, "in writev\n");
        writev( STDOUT_FILENO, pIov, IOV_CNT);
    } else {
        fprintf( stderr, "Please assign argv[2] in (WR|WV)\n");
    }
    return 0;
}
/opt/drill_ground/aupe/14-9.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001]
```

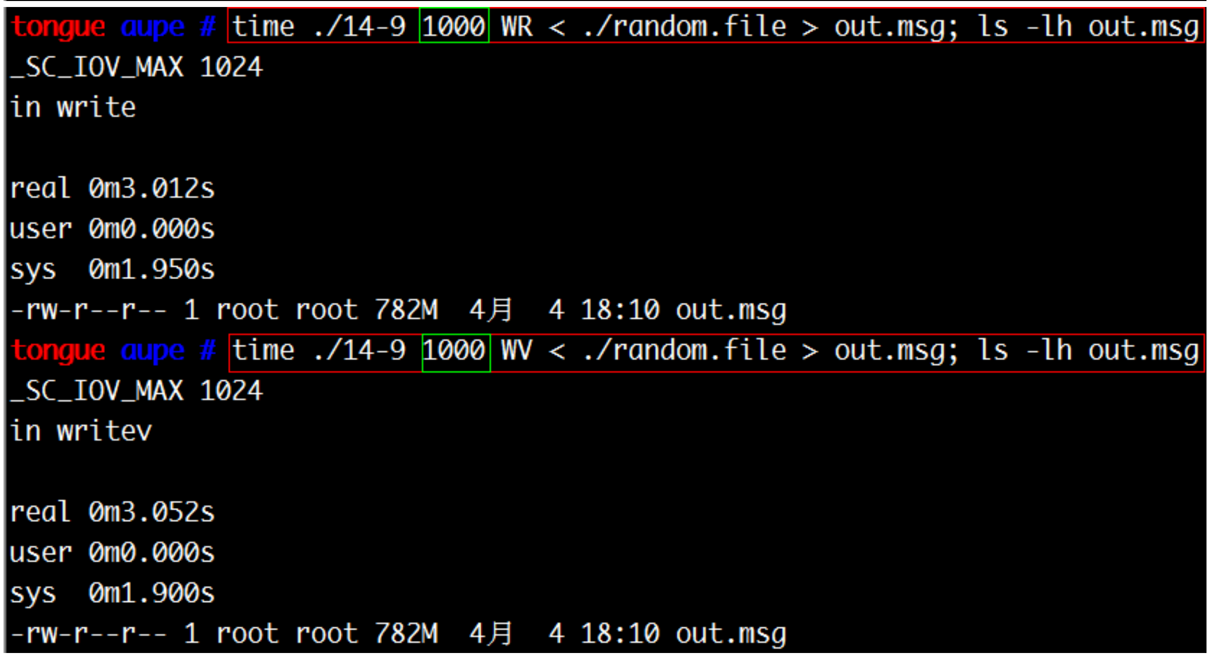
```
tongue aupe # gcc -g 14-9.c -o 14-9
```



```
tongue aupe # time ./14-9 1024 WV < /dev/urandom > random.file
_SC_IOV_MAX 1024
in writev

real 1m9.084s
user 0m0.000s
sys  1m8.060s
tongue aupe # ls -lh random.file
-rw-r--r-- 1 root root 800M  4月  4 18:07 random.file
```

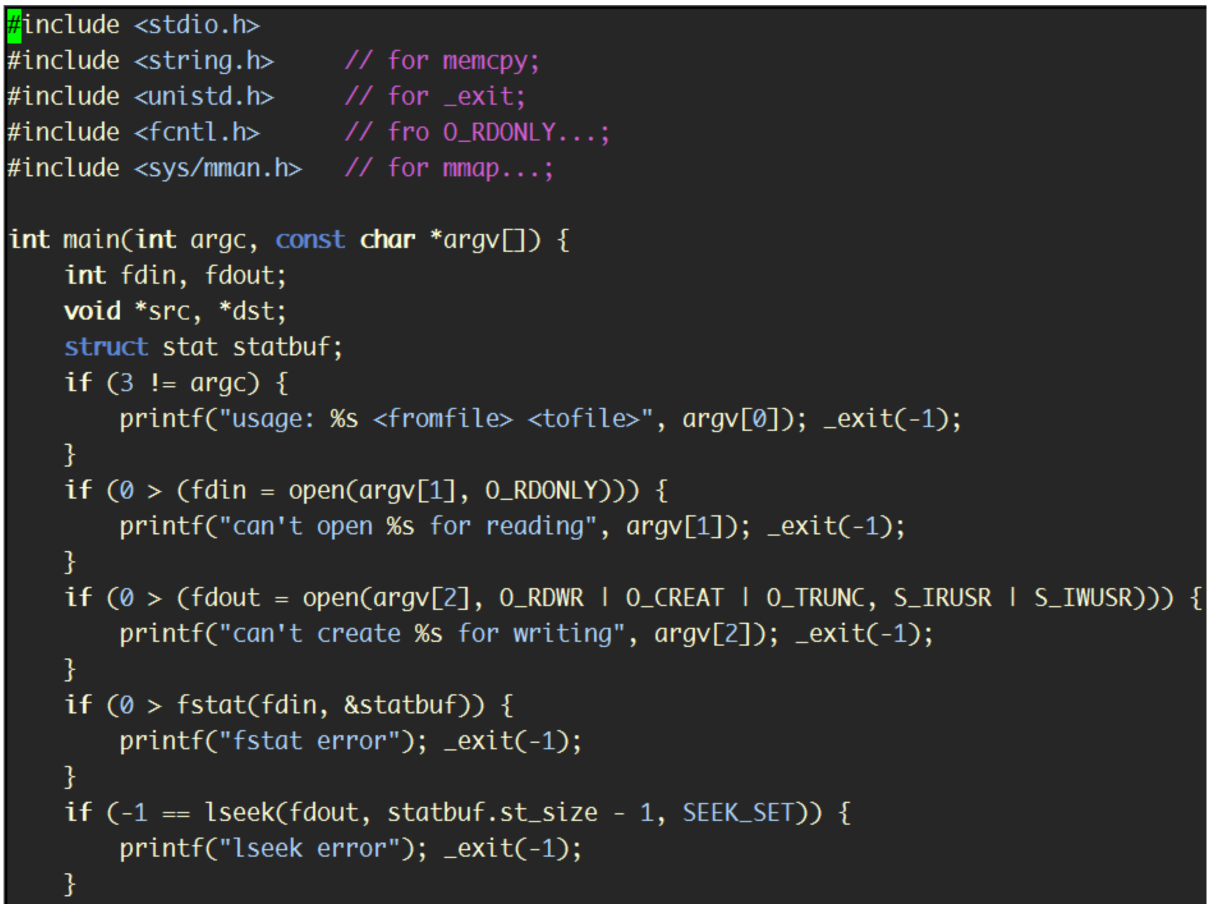


```
tongue aupe # time ./14-9 10 WR < ./random.file > out.msg; ls -lh out.msg
_SC_IOV_MAX 1024
in write

real 0m0.019s
user 0m0.000s
sys  0m0.010s
-rw-r--r-- 1 root root 7.9M  4月  4 18:09 out.msg
tongue aupe # time ./14-9 10 WV < ./random.file > out.msg; ls -lh out.msg
_SC_IOV_MAX 1024
in writev

real 0m0.020s
user 0m0.000s
sys  0m0.010s
-rw-r--r-- 1 root root 7.9M  4月  4 18:09 out.msg
```

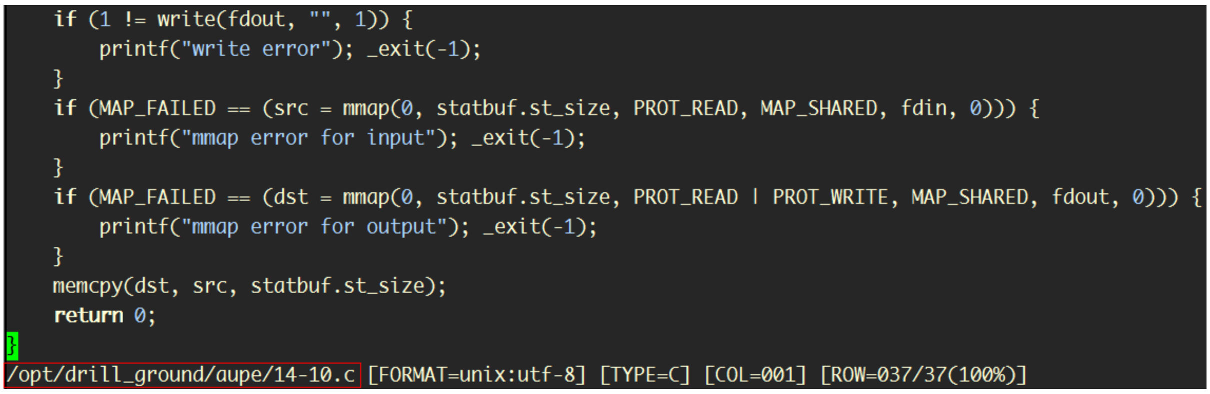


```
tongue aupe # time ./14-9 1000 WR < ./random.file > out.msg; ls -lh out.msg
_SC_IOV_MAX 1024
in write

real 0m3.012s
user 0m0.000s
sys  0m1.950s
-rw-r--r-- 1 root root 782M  4月  4 18:10 out.msg
tongue aupe # time ./14-9 1000 WV < ./random.file > out.msg; ls -lh out.msg
_SC_IOV_MAX 1024
in writev

real 0m3.052s
user 0m0.000s
sys  0m1.900s
-rw-r--r-- 1 root root 782M  4月  4 18:10 out.msg
```

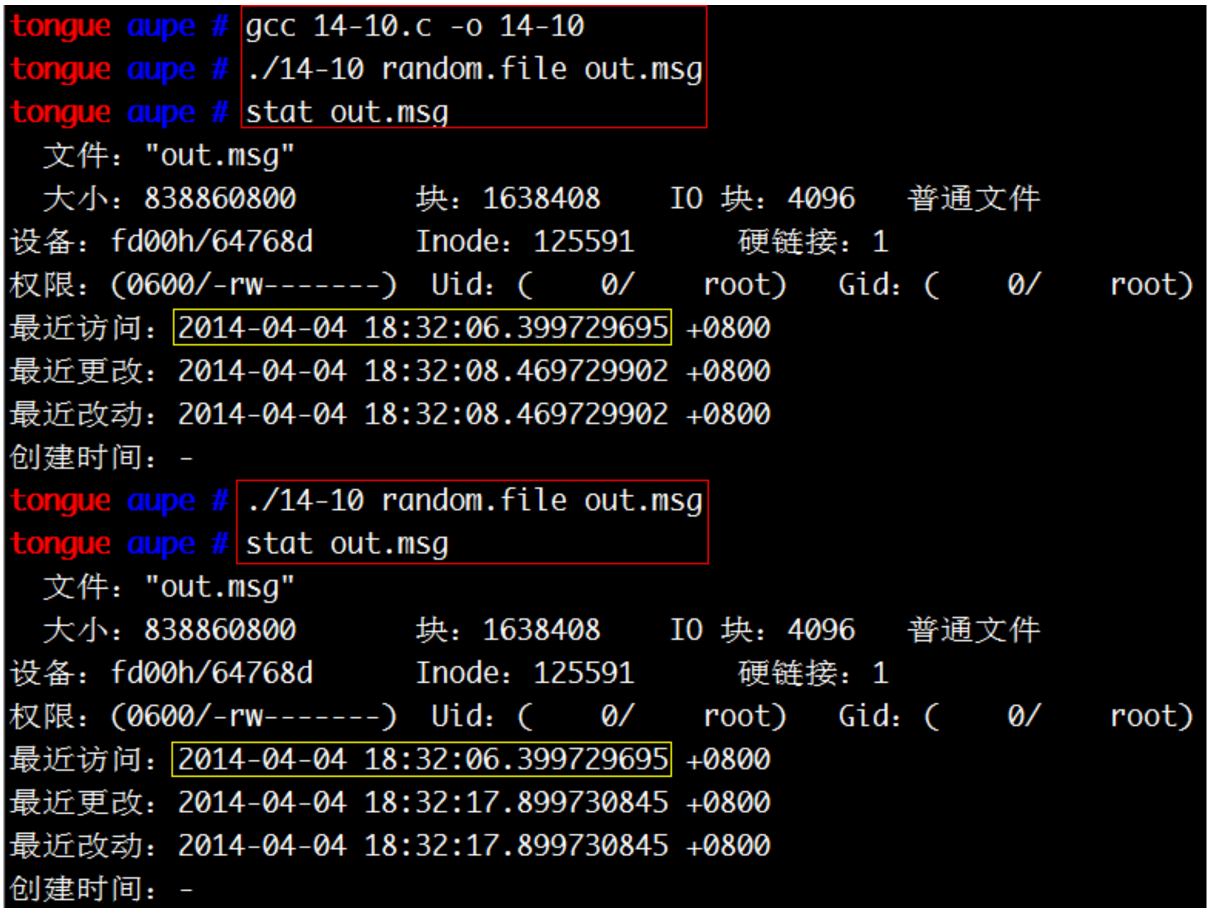
14-10、没有发生改变；



```c
#include <stdio.h>
#include <string.h>     // for memcpy;
#include <unistd.h>     // for _exit;
#include <fcntl.h>      // fro O_RDONLY...;
#include <sys/mman.h>   // for mmap...;

int main(int argc, const char *argv[]) {
    int fdin, fdout;
    void *src, *dst;
    struct stat statbuf;
    if (3 != argc) {
        printf("usage: %s <fromfile> <tofile>", argv[0]); _exit(-1);
    }
    if (0 > (fdin = open(argv[1], O_RDONLY))) {
        printf("can't open %s for reading", argv[1]); _exit(-1);
    }
    if (0 > (fdout = open(argv[2], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR))) {
        printf("can't create %s for writing", argv[2]); _exit(-1);
    }
    if (0 > fstat(fdin, &statbuf)) {
        printf("fstat error"); _exit(-1);
    }
    if (-1 == lseek(fdout, statbuf.st_size - 1, SEEK_SET)) {
        printf("lseek error"); _exit(-1);
    }
```

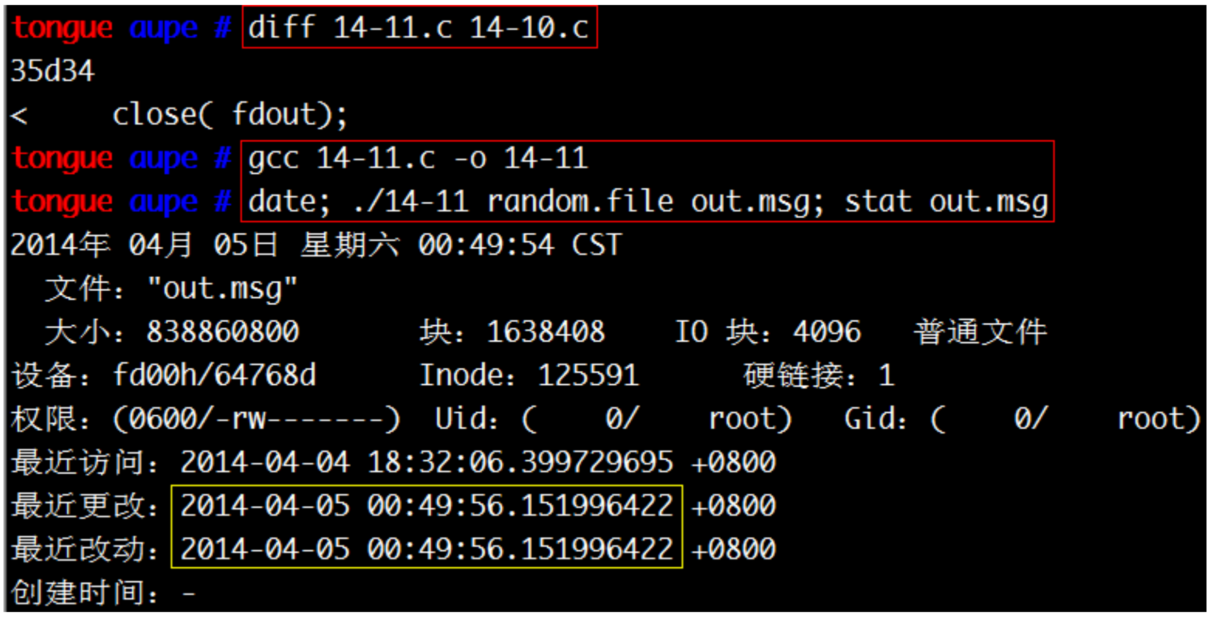


```c
    if (1 != write(fdout, "", 1)) {
        printf("write error"); _exit(-1);
    }
    if (MAP_FAILED == (src = mmap(0, statbuf.st_size, PROT_READ, MAP_SHARED, fdin, 0))) {
        printf("mmap error for input"); _exit(-1);
    }
    if (MAP_FAILED == (dst = mmap(0, statbuf.st_size, PROT_READ | PROT_WRITE, MAP_SHARED, fdout, 0))) {
        printf("mmap error for output"); _exit(-1);
    }
    memcpy(dst, src, statbuf.st_size);
    return 0;
}
/opt/drill_ground/aupe/14-10.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=037/37(100%)]
```



```
tongue aupe # gcc 14-10.c -o 14-10
tongue aupe # ./14-10 random.file out.msg
tongue aupe # stat out.msg
  文件："out.msg"
  大小：838860800       块：1638408    IO 块：4096   普通文件
设备：fd00h/64768d      Inode：125591       硬链接：1
权限：(0600/-rw-------)  Uid：(    0/    root)   Gid：(    0/    root)
最近访问：2014-04-04 18:32:06.399729695 +0800
最近更改：2014-04-04 18:32:08.469729902 +0800
最近改动：2014-04-04 18:32:08.469729902 +0800
创建时间：-
tongue aupe # ./14-10 random.file out.msg
tongue aupe # stat out.msg
  文件："out.msg"
  大小：838860800       块：1638408    IO 块：4096   普通文件
设备：fd00h/64768d      Inode：125591       硬链接：1
权限：(0600/-rw-------)  Uid：(    0/    root)   Gid：(    0/    root)
最近访问：2014-04-04 18:32:06.399729695 +0800
最近更改：2014-04-04 18:32:17.899730845 +0800
最近改动：2014-04-04 18:32:17.899730845 +0800
创建时间：-
```

14-11、对14-10.c稍作修改即得。



```
tongue aupe # diff 14-11.c 14-10.c
35d34
<       close( fdout);
tongue aupe # gcc 14-11.c -o 14-11
tongue aupe # date; ./14-11 random.file out.msg; stat out.msg
2014年 04月 05日 星期六 00:49:54 CST
  文件："out.msg"
  大小：838860800       块：1638408    IO 块：4096   普通文件
设备：fd00h/64768d      Inode：125591       硬链接：1
权限：(0600/-rw-------)  Uid：(    0/    root)   Gid：(    0/    root)
最近访问：2014-04-04 18:32:06.399729695 +0800
最近更改：2014-04-05 00:49:56.151996422 +0800
最近改动：2014-04-05 00:49:56.151996422 +0800
创建时间：-
```