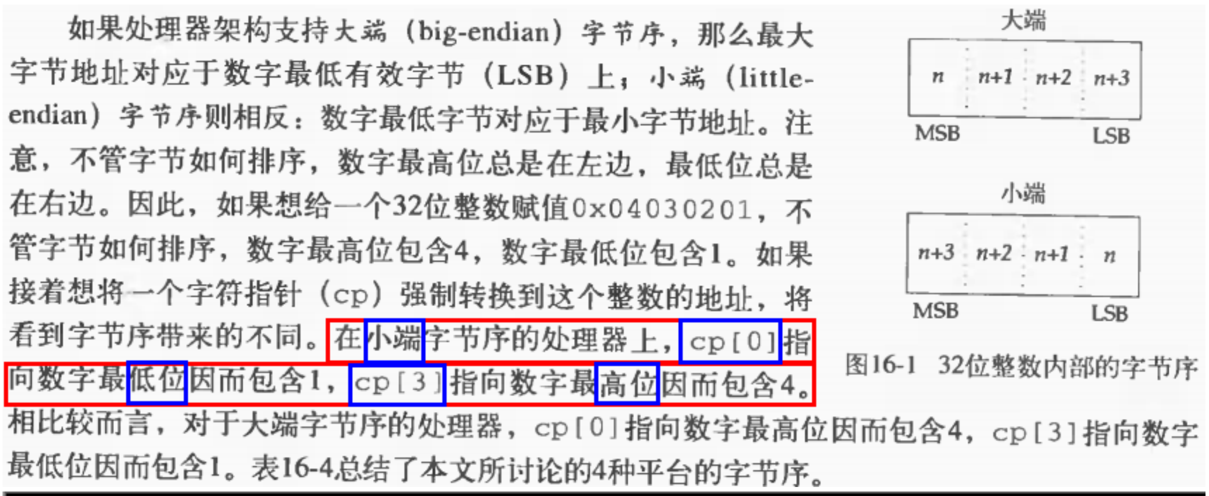
第十六章 网络IPC套接字

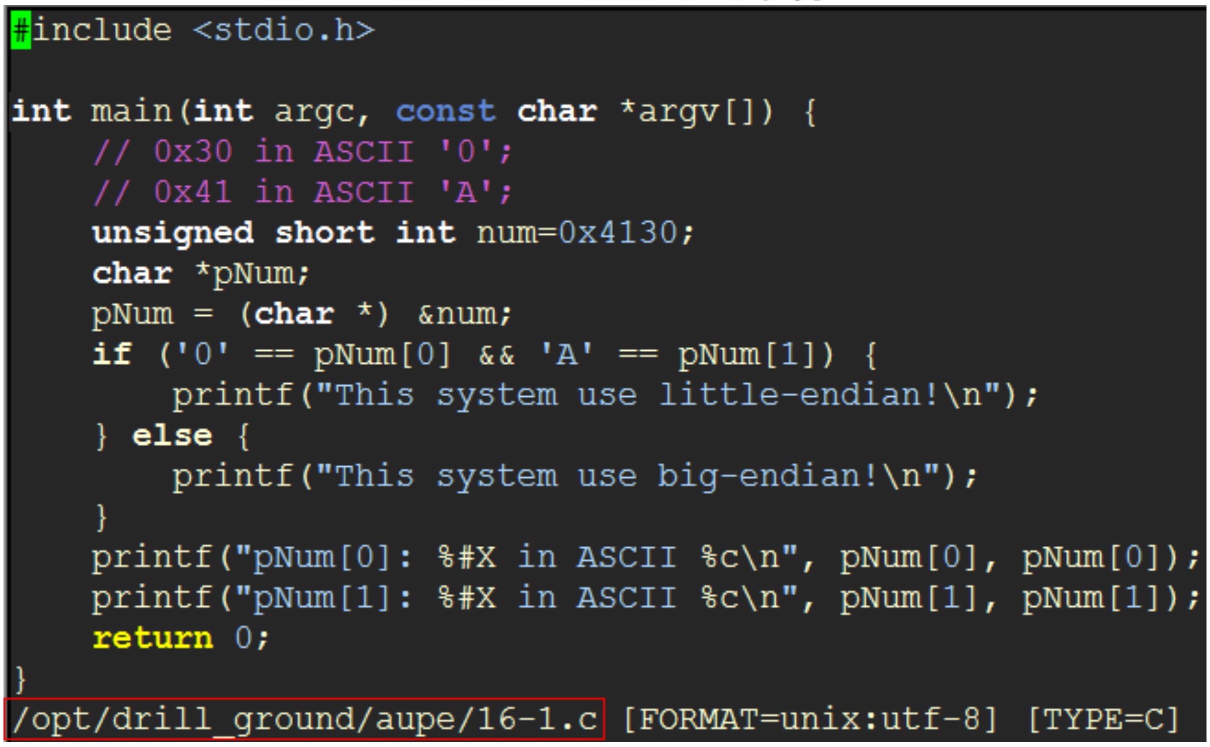
习题：

16-1、系统大小端指的是，存放数字的起始地址中，记录数字低位的为小端；记录数字高位的为大端；

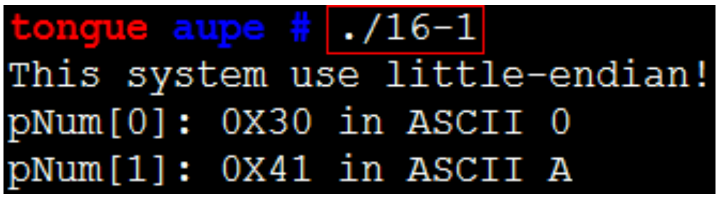


如果处理器架构支持大端（big-endian）字节序，那么最大字节地址对应于数字最低有效字节（LSB）上；小端（little-endian）字节序则相反：数字最低字节对应于最小字节地址。注意，不管字节如何排序，数字最高位总是在左边，最低位总是在右边。因此，如果想给一个32位整数赋值0x04030201，不管字节如何排序，数字最高位包含4，数字最低位包含1。如果接着想将一个字符指针（cp）强制转换到这个整数的地址，将看到字节序带来的不同。在小端字节序的处理器上，cp[0]指向数字最低位因而包含1，cp[3]指向数字最高位因而包含4。相比较而言，对于大端字节序的处理器，cp[0]指向数字最高位因而包含4，cp[3]指向数字最低位因而包含1。表16-4总结了本文所讨论的4种平台的字节序。

图16-1 32位整数内部的字节序

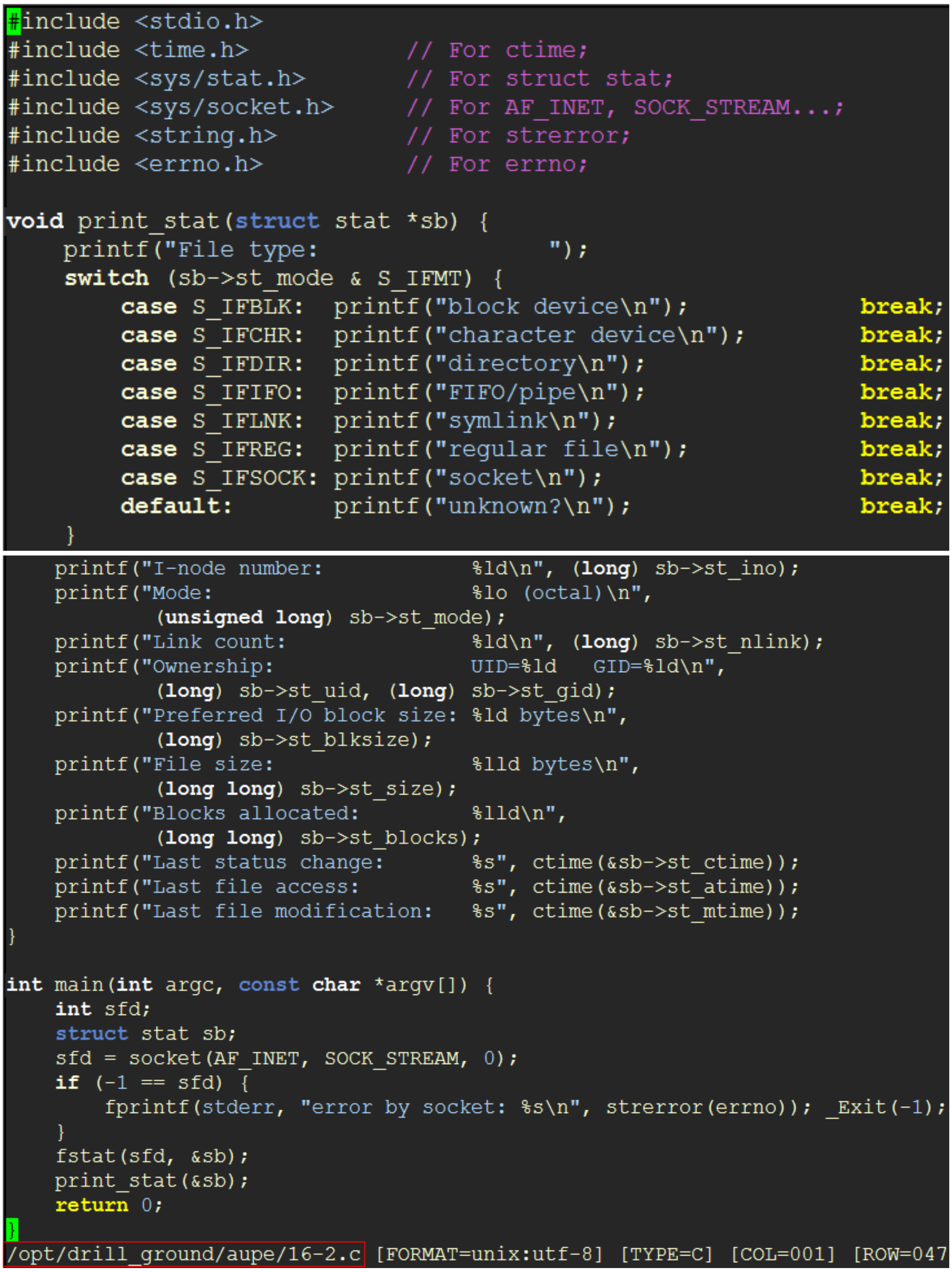


```c
#include <stdio.h>

int main(int argc, const char *argv[]) {
    // 0x30 in ASCII '0';
    // 0x41 in ASCII 'A';
    unsigned short int num=0x4130;
    char *pNum;
    pNum = (char *) &num;
    if ('0' == pNum[0] && 'A' == pNum[1]) {
        printf("This system use little-endian!\n");
    } else {
        printf("This system use big-endian!\n");
    }
    printf("pNum[0]: %#X in ASCII %c\n", pNum[0], pNum[0]);
    printf("pNum[1]: %#X in ASCII %c\n", pNum[1], pNum[1]);
    return 0;
}
/opt/drill_ground/aupe/16-1.c  [FORMAT=unix:utf-8]  [TYPE=C]
```



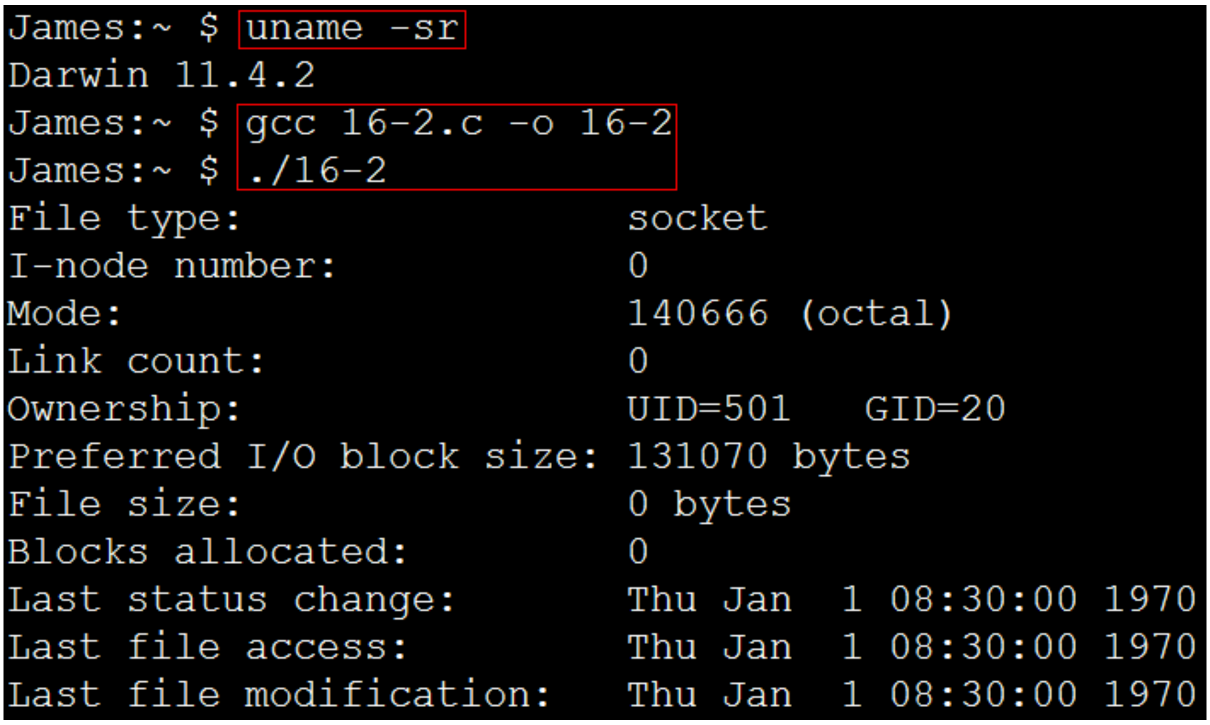
```
tongue aupe # ./16-1
This system use little-endian!
pNum[0]: 0X30 in ASCII 0
pNum[1]: 0X41 in ASCII A
```

16-2、较为明显之处是I-node number和Link count不一样；



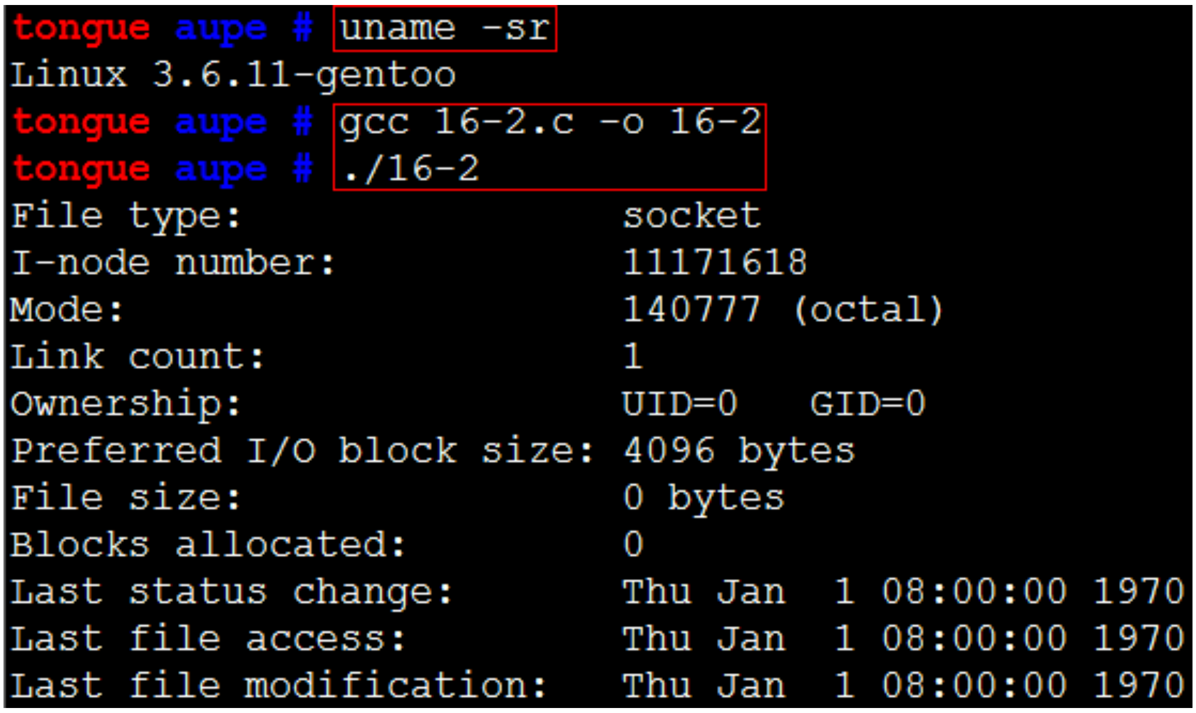
```c
#include <stdio.h>
#include <time.h>               // For ctime;
#include <sys/stat.h>           // For struct stat;
#include <sys/socket.h>         // For AF_INET, SOCK_STREAM...;
#include <string.h>             // For strerror;
#include <errno.h>              // For errno;

void print_stat(struct stat *sb) {
    printf("File type:                ");
    switch (sb->st_mode & S_IFMT) {
        case S_IFBLK:  printf("block device\n");            break;
        case S_IFCHR:  printf("character device\n");        break;
        case S_IFDIR:  printf("directory\n");               break;
        case S_IFIFO:  printf("FIFO/pipe\n");               break;
        case S_IFLNK:  printf("symlink\n");                 break;
        case S_IFREG:  printf("regular file\n");            break;
        case S_IFSOCK: printf("socket\n");                  break;
        default:       printf("unknown?\n");                break;
    }
    printf("I-node number:            %ld\n", (long) sb->st_ino);
    printf("Mode:                     %lo (octal)\n",
            (unsigned long) sb->st_mode);
    printf("Link count:               %ld\n", (long) sb->st_nlink);
    printf("Ownership:                UID=%ld   GID=%ld\n",
            (long) sb->st_uid, (long) sb->st_gid);
    printf("Preferred I/O block size: %ld bytes\n",
            (long) sb->st_blksize);
    printf("File size:                %lld bytes\n",
            (long long) sb->st_size);
    printf("Blocks allocated:         %lld\n",
            (long long) sb->st_blocks);
    printf("Last status change:       %s", ctime(&sb->st_ctime));
    printf("Last file access:         %s", ctime(&sb->st_atime));
    printf("Last file modification:   %s", ctime(&sb->st_mtime));
}

int main(int argc, const char *argv[]) {
    int sfd;
    struct stat sb;
    sfd = socket(AF_INET, SOCK_STREAM, 0);
    if (-1 == sfd) {
        fprintf(stderr, "error by socket: %s\n", strerror(errno)); _Exit(-1);
    }
    fstat(sfd, &sb);
    print_stat(&sb);
    return 0;
}
/opt/drill_ground/aupe/16-2.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=047
```

On MAC



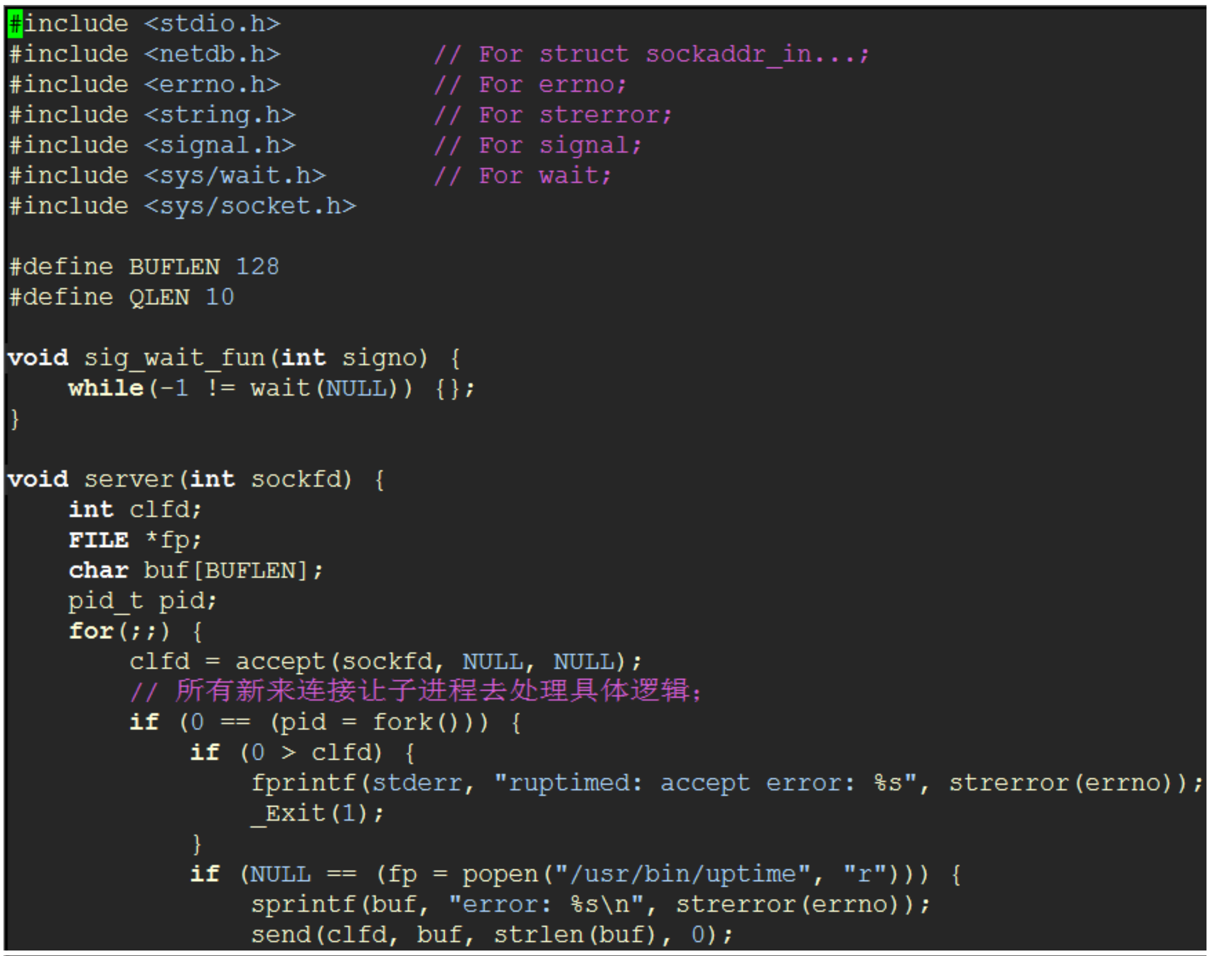
```
James:~ $ uname -sr
Darwin 11.4.2
James:~ $ gcc 16-2.c -o 16-2
James:~ $ ./16-2
File type:                   socket
I-node number:               0
Mode:                        140666 (octal)
Link count:                  0
Ownership:                   UID=501   GID=20
Preferred I/O block size:    131070 bytes
File size:                   0 bytes
Blocks allocated:            0
Last status change:          Thu Jan  1 08:30:00 1970
Last file access:            Thu Jan  1 08:30:00 1970
Last file modification:      Thu Jan  1 08:30:00 1970
```

On Linux

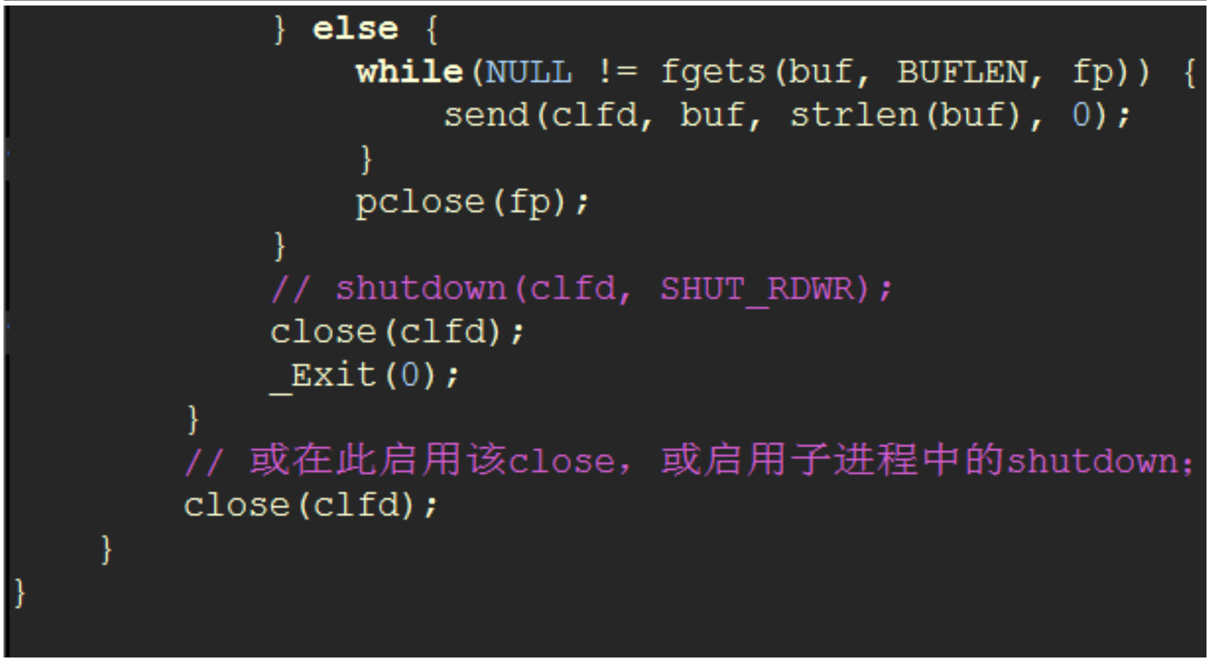


```
tongue aupe # uname -sr
Linux 3.6.11-gentoo
tongue aupe # gcc 16-2.c -o 16-2
tongue aupe # ./16-2
File type:                   socket
I-node number:               11171618
Mode:                        140777 (octal)
Link count:                  1
Ownership:                   UID=0   GID=0
Preferred I/O block size:    4096 bytes
File size:                   0 bytes
Blocks allocated:            0
Last status change:          Thu Jan  1 08:00:00 1970
Last file access:            Thu Jan  1 08:00:00 1970
Last file modification:      Thu Jan  1 08:00:00 1970
```

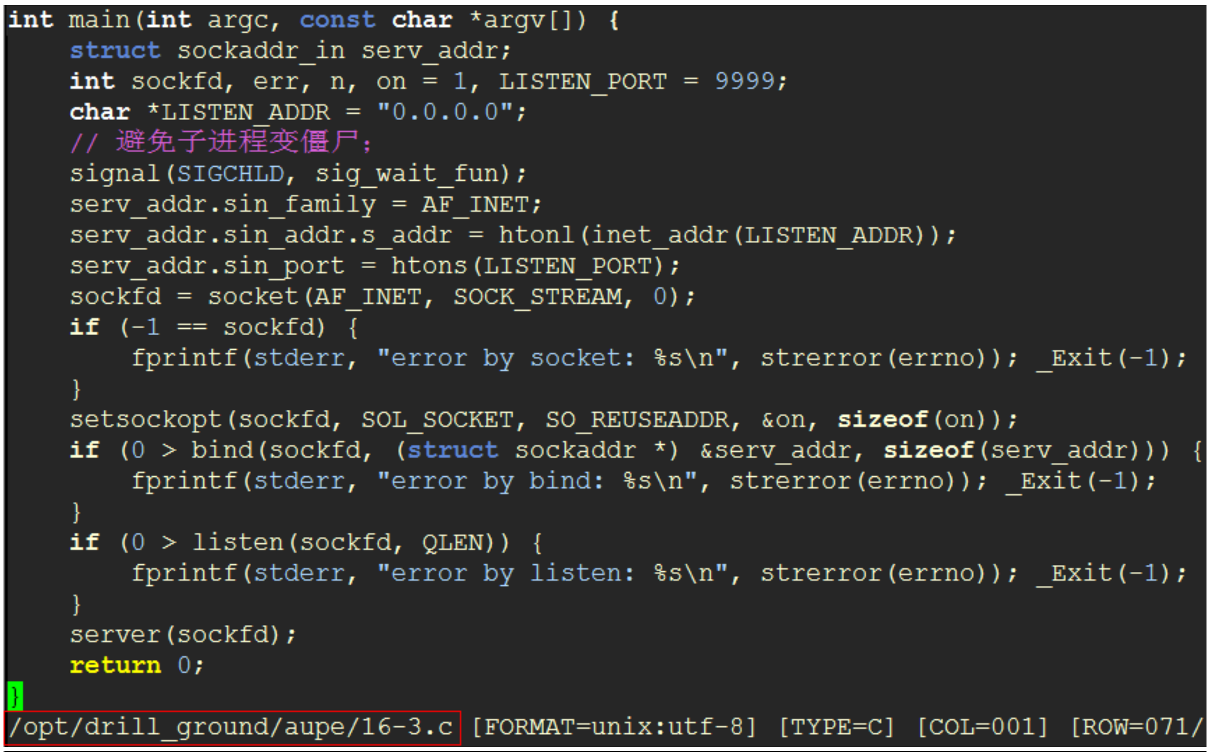
16-3、如下；续：可能之前解题有误，如果想实现类似nginx那样，让一个服务可以侦听多个ip或端口，由于一个socket只能绑定一个ip和端口的组合，想要侦听多个，那么可以通过I/O多路转接来实现。如select、poll、epoll等。



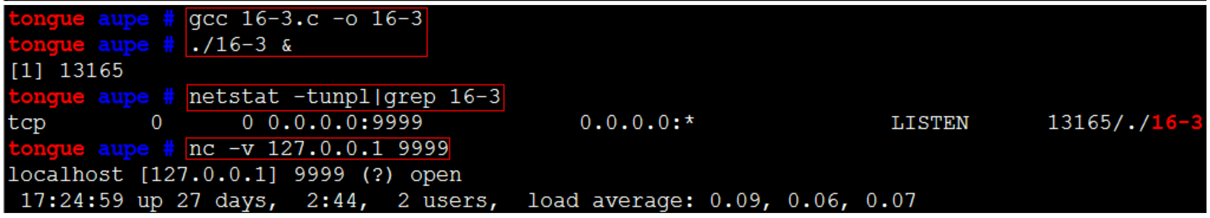
```c
#include <stdio.h>
#include <netdb.h>           // For struct sockaddr_in...;
#include <errno.h>           // For errno;
#include <string.h>          // For strerror;
#include <signal.h>          // For signal;
#include <sys/wait.h>        // For wait;
#include <sys/socket.h>

#define BUFLEN 128
#define QLEN 10

void sig_wait_fun(int signo) {
    while(-1 != wait(NULL)) {};
}

void server(int sockfd) {
    int clfd;
    FILE *fp;
    char buf[BUFLEN];
    pid_t pid;
    for(;;) {
        clfd = accept(sockfd, NULL, NULL);
        // 所有新来连接让子进程去处理具体逻辑;
        if (0 == (pid = fork())) {
            if (0 > clfd) {
                fprintf(stderr, "ruptimed: accept error: %s", strerror(errno));
                _Exit(1);
            }
            if (NULL == (fp = popen("/usr/bin/uptime", "r"))) {
                sprintf(buf, "error: %s\n", strerror(errno));
                send(clfd, buf, strlen(buf), 0);
```



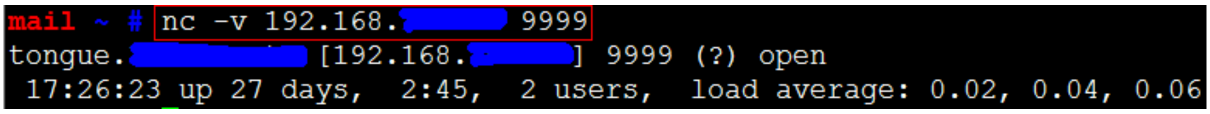
```c
            } else {
                while(NULL != fgets(buf, BUFLEN, fp)) {
                    send(clfd, buf, strlen(buf), 0);
                }
                pclose(fp);
            }
            // shutdown(clfd, SHUT_RDWR);
            close(clfd);
            _Exit(0);
        }
        // 或在此启用该close, 或启用子进程中的shutdown;
        close(clfd);
    }
}
```



```c
int main(int argc, const char *argv[]) {
    struct sockaddr_in serv_addr;
    int sockfd, err, n, on = 1, LISTEN_PORT = 9999;
    char *LISTEN_ADDR = "0.0.0.0";
    // 避免子进程变僵尸;
    signal(SIGCHLD, sig_wait_fun);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(inet_addr(LISTEN_ADDR));
    serv_addr.sin_port = htons(LISTEN_PORT);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (-1 == sockfd) {
        fprintf(stderr, "error by socket: %s\n", strerror(errno)); _Exit(-1);
    }
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
    if (0 > bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))) {
        fprintf(stderr, "error by bind: %s\n", strerror(errno)); _Exit(-1);
    }
    if (0 > listen(sockfd, QLEN)) {
        fprintf(stderr, "error by listen: %s\n", strerror(errno)); _Exit(-1);
    }
    server(sockfd);
    return 0;
}
/opt/drill_ground/aupe/16-3.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=071/
```

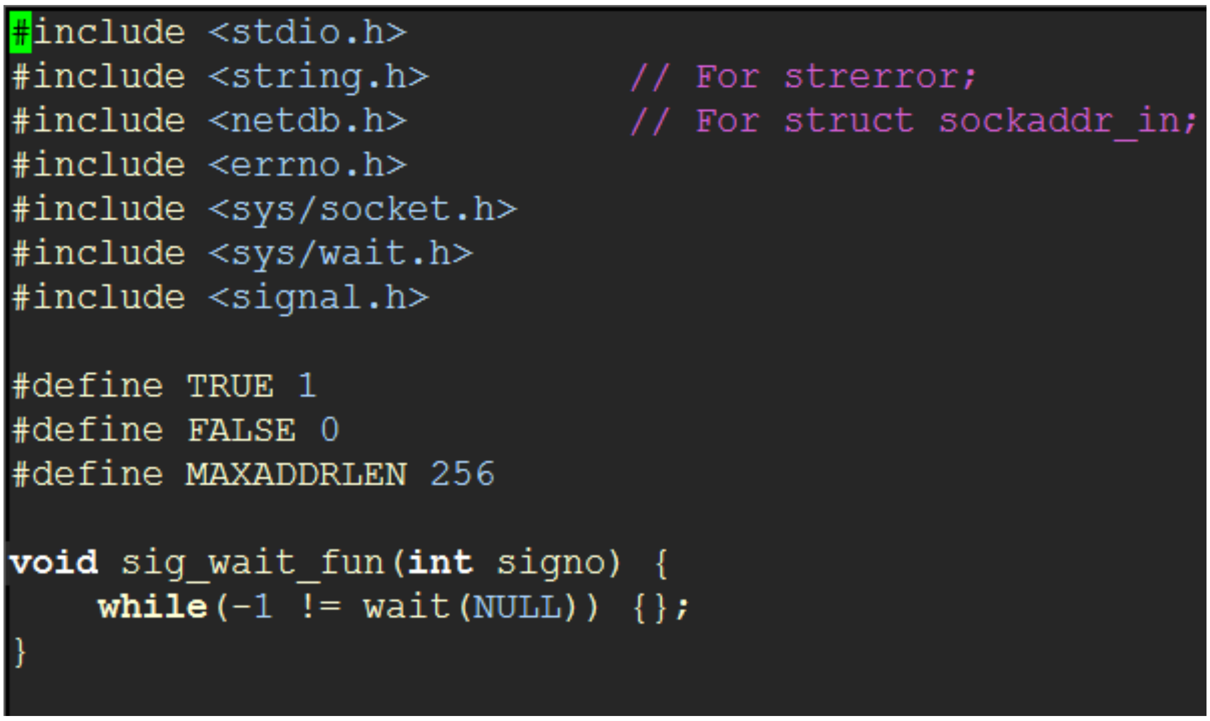


```
tongue aupe # gcc 16-3.c -o 16-3
tongue aupe # ./16-3 &
[1] 13165
tongue aupe # netstat -tunpl|grep 16-3
tcp        0      0 0.0.0.0:9999            0.0.0.0:*               LISTEN      13165/./16-3
tongue aupe # nc -v 127.0.0.1 9999
localhost [127.0.0.1] 9999 (?) open
 17:24:59 up 27 days,  2:44,  2 users,  load average: 0.09, 0.06, 0.07
```

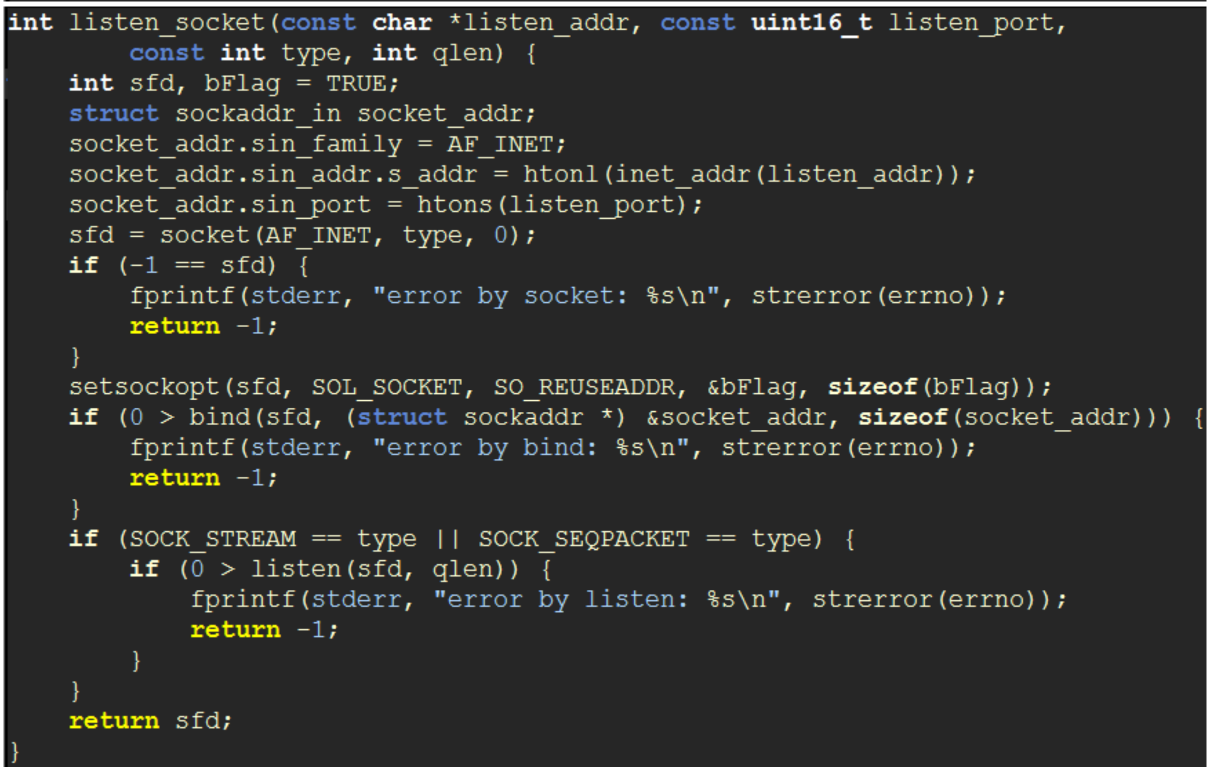


```
mail ~ # nc -v 192.168.[redacted] 9999
tongue.[redacted] [192.168.[redacted]] 9999 (?) open
 17:26:23 up 27 days,  2:45,  2 users,  load average: 0.02, 0.04, 0.06
```

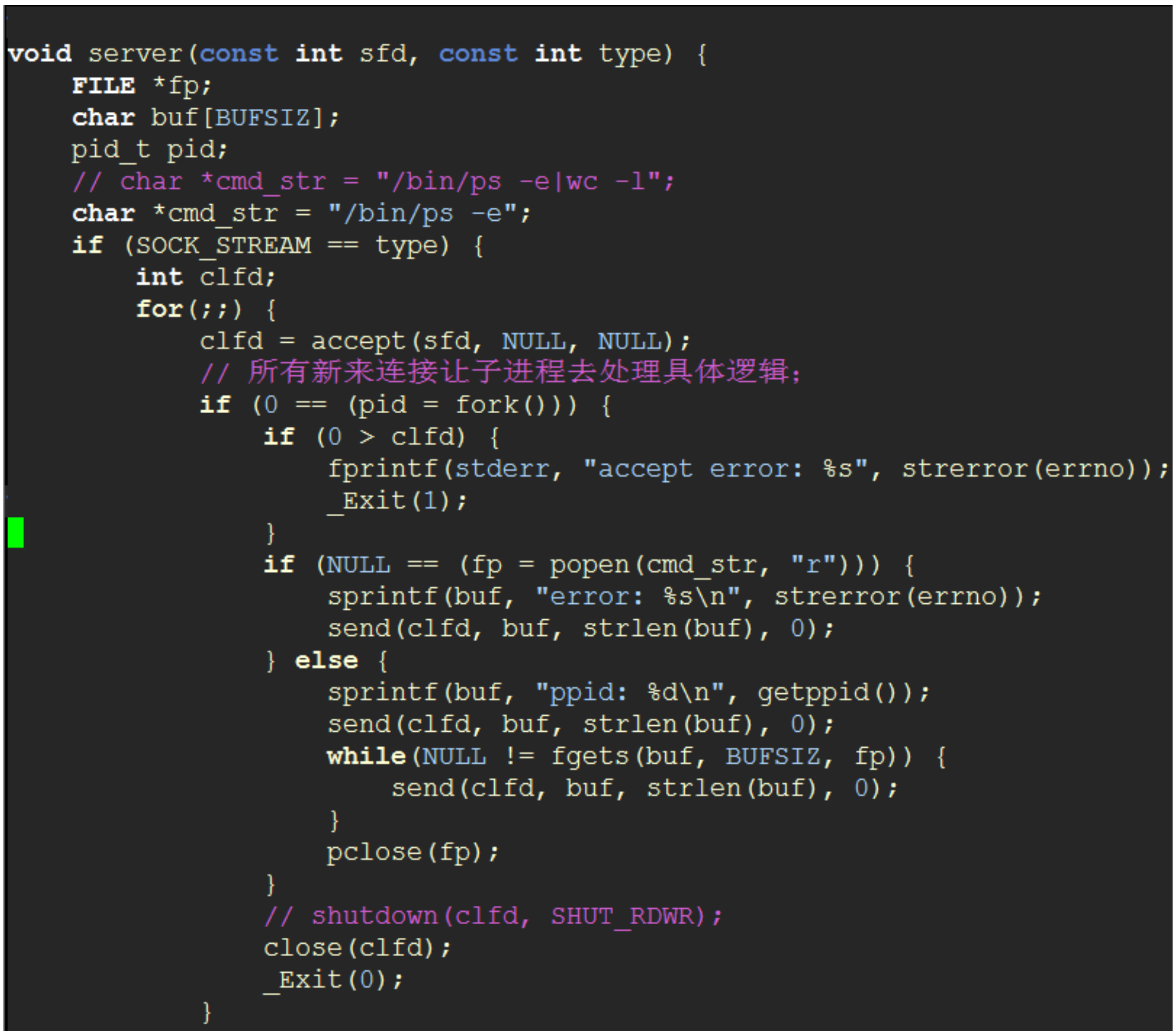
16-4、
Server:

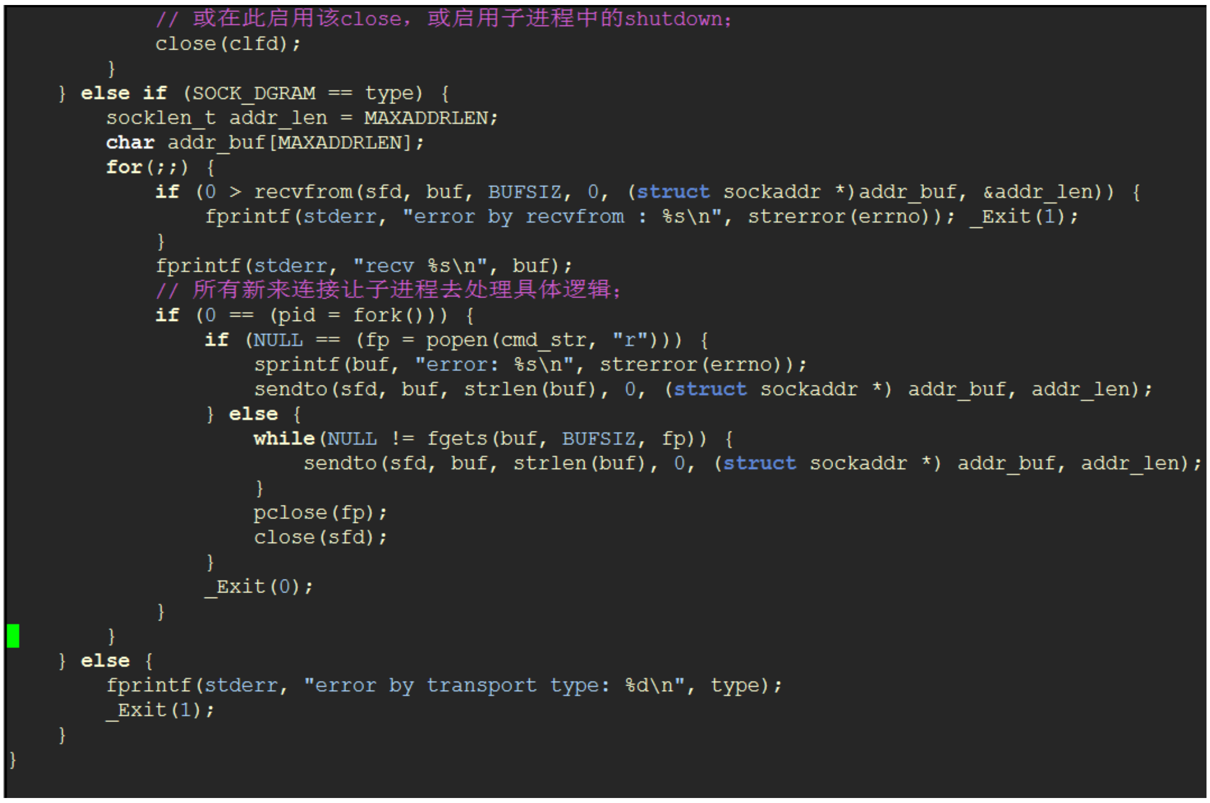


```c
#include <stdio.h>
#include <string.h>         // For strerror;
#include <netdb.h>          // For struct sockaddr_in;
#include <errno.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <signal.h>

#define TRUE 1
#define FALSE 0
#define MAXADDRLEN 256

void sig_wait_fun(int signo) {
    while(-1 != wait(NULL)) {};
}
```

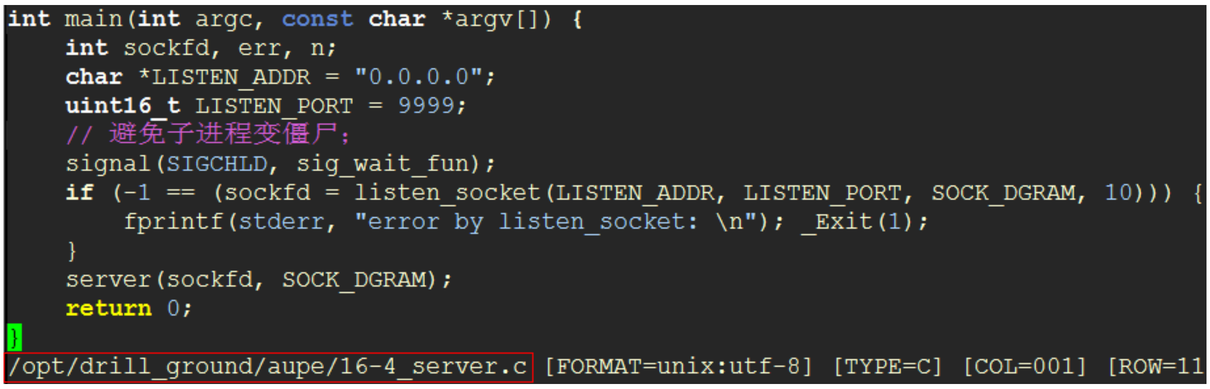


```c
int listen_socket(const char *listen_addr, const uint16_t listen_port,
        const int type, int qlen) {
    int sfd, bFlag = TRUE;
    struct sockaddr_in socket_addr;
    socket_addr.sin_family = AF_INET;
    socket_addr.sin_addr.s_addr = htonl(inet_addr(listen_addr));
    socket_addr.sin_port = htons(listen_port);
    sfd = socket(AF_INET, type, 0);
    if (-1 == sfd) {
        fprintf(stderr, "error by socket: %s\n", strerror(errno));
        return -1;
    }
    setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &bFlag, sizeof(bFlag));
    if (0 > bind(sfd, (struct sockaddr *) &socket_addr, sizeof(socket_addr))) {
        fprintf(stderr, "error by bind: %s\n", strerror(errno));
        return -1;
    }
    if (SOCK_STREAM == type || SOCK_SEQPACKET == type) {
        if (0 > listen(sfd, qlen)) {
            fprintf(stderr, "error by listen: %s\n", strerror(errno));
            return -1;
        }
    }
    return sfd;
}
```


```
void server(const int sfd, const int type) {
    FILE *fp;
    char buf[BUFSIZ];
    pid_t pid;
    // char *cmd_str = "/bin/ps -e|wc -l";
    char *cmd_str = "/bin/ps -e";
    if (SOCK_STREAM == type) {
        int clfd;
        for(;;) {
            clfd = accept(sfd, NULL, NULL);
            // 所有新来连接让子进程去处理具体逻辑;
            if (0 == (pid = fork())) {
                if (0 > clfd) {
                    fprintf(stderr, "accept error: %s", strerror(errno));
                    _Exit(1);
                }
                if (NULL == (fp = popen(cmd_str, "r"))) {
                    sprintf(buf, "error: %s\n", strerror(errno));
                    send(clfd, buf, strlen(buf), 0);
                } else {
                    sprintf(buf, "ppid: %d\n", getppid());
                    send(clfd, buf, strlen(buf), 0);
                    while(NULL != fgets(buf, BUFSIZ, fp)) {
                        send(clfd, buf, strlen(buf), 0);
                    }
                    pclose(fp);
                }
                // shutdown(clfd, SHUT_RDWR);
                close(clfd);
                _Exit(0);
            }
```

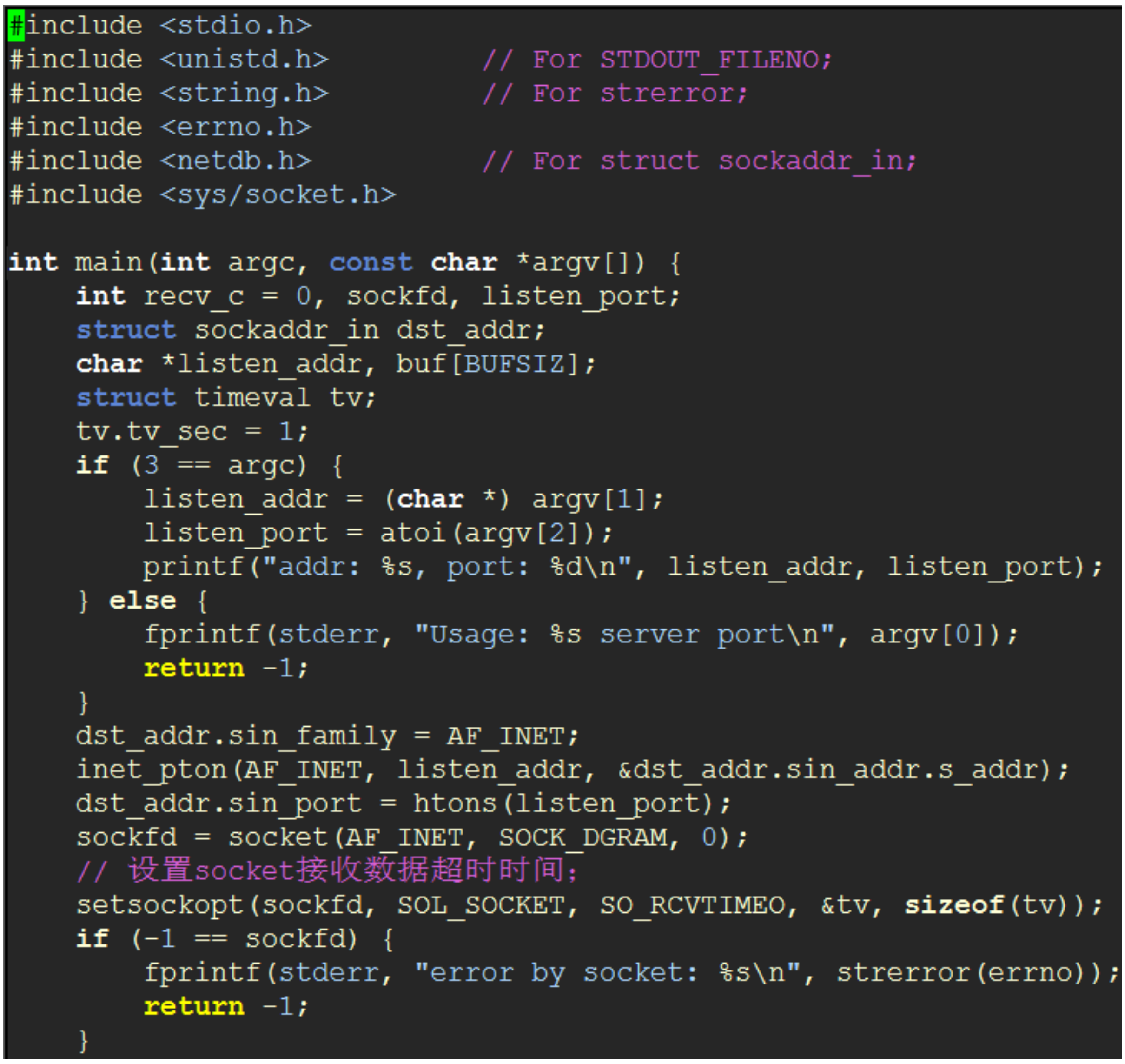


```
            // 或在此启用该close，或启用子进程中的shutdown;
            close(clfd);
        }
    } else if (SOCK_DGRAM == type) {
        socklen_t addr_len = MAXADDRLEN;
        char addr_buf[MAXADDRLEN];
        for(;;) {
            if (0 > recvfrom(sfd, buf, BUFSIZ, 0, (struct sockaddr *)addr_buf, &addr_len)) {
                fprintf(stderr, "error by recvfrom : %s\n", strerror(errno)); _Exit(1);
            }
            fprintf(stderr, "recv %s\n", buf);
            // 所有新来连接让子进程去处理具体逻辑;
            if (0 == (pid = fork())) {
                if (NULL == (fp = popen(cmd_str, "r"))) {
                    sprintf(buf, "error: %s\n", strerror(errno));
                    sendto(sfd, buf, strlen(buf), 0, (struct sockaddr *) addr_buf, addr_len);
                } else {
                    while(NULL != fgets(buf, BUFSIZ, fp)) {
                        sendto(sfd, buf, strlen(buf), 0, (struct sockaddr *) addr_buf, addr_len);
                    }
                    pclose(fp);
                    close(sfd);
                }
                _Exit(0);
            }
        }
    } else {
        fprintf(stderr, "error by transport type: %d\n", type);
        _Exit(1);
    }
}
```

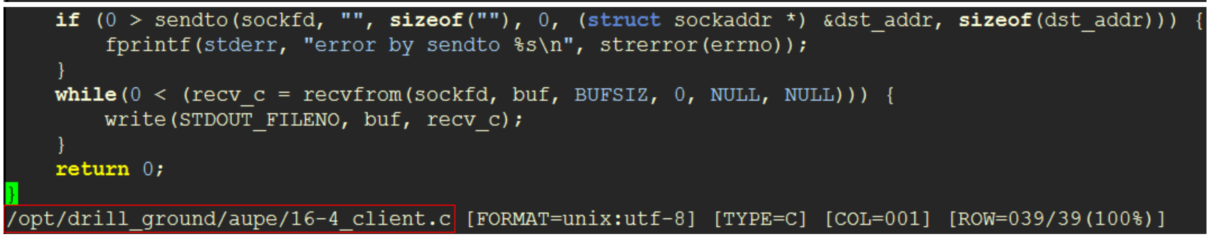


```
int main(int argc, const char *argv[]) {
    int sockfd, err, n;
    char *LISTEN_ADDR = "0.0.0.0";
    uint16_t LISTEN_PORT = 9999;
    // 避免子进程变僵尸;
    signal(SIGCHLD, sig_wait_fun);
    if (-1 == (sockfd = listen_socket(LISTEN_ADDR, LISTEN_PORT, SOCK_DGRAM, 10))) {
        fprintf(stderr, "error by listen_socket: \n"); _Exit(1);
    }
    server(sockfd, SOCK_DGRAM);
    return 0;
}
/opt/drill_ground/aupe/16-4_server.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=11
```

Client:

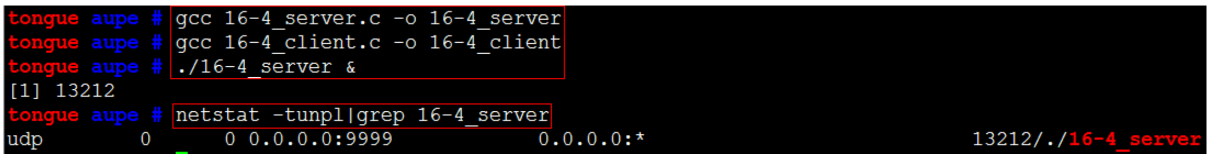


```c
#include <stdio.h>
#include <unistd.h>          // For STDOUT_FILENO;
#include <string.h>          // For strerror;
#include <errno.h>
#include <netdb.h>           // For struct sockaddr_in;
#include <sys/socket.h>

int main(int argc, const char *argv[]) {
    int recv_c = 0, sockfd, listen_port;
    struct sockaddr_in dst_addr;
    char *listen_addr, buf[BUFSIZ];
    struct timeval tv;
    tv.tv_sec = 1;
    if (3 == argc) {
        listen_addr = (char *) argv[1];
        listen_port = atoi(argv[2]);
        printf("addr: %s, port: %d\n", listen_addr, listen_port);
    } else {
        fprintf(stderr, "Usage: %s server port\n", argv[0]);
        return -1;
    }
    dst_addr.sin_family = AF_INET;
    inet_pton(AF_INET, listen_addr, &dst_addr.sin_addr.s_addr);
    dst_addr.sin_port = htons(listen_port);
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    // 设置socket接收数据超时时间;
    setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
    if (-1 == sockfd) {
        fprintf(stderr, "error by socket: %s\n", strerror(errno));
        return -1;
    }
```

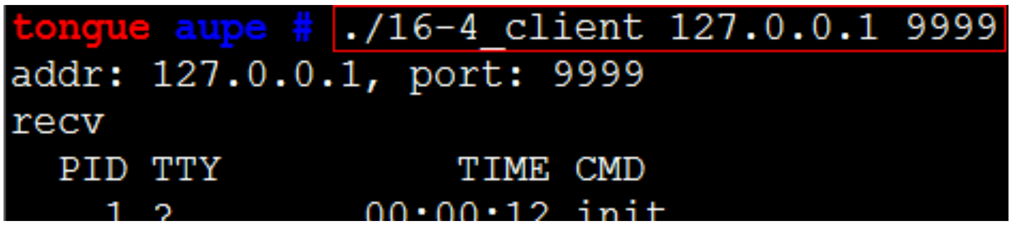


```c
    if (0 > sendto(sockfd, "", sizeof(""), 0, (struct sockaddr *) &dst_addr, sizeof(dst_addr))) {
        fprintf(stderr, "error by sendto %s\n", strerror(errno));
    }
    while(0 < (recv_c = recvfrom(sockfd, buf, BUFSIZ, 0, NULL, NULL))) {
        write(STDOUT_FILENO, buf, recv_c);
    }
    return 0;
}
/opt/drill_ground/aupe/16-4_client.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=039/39(100%)]
```

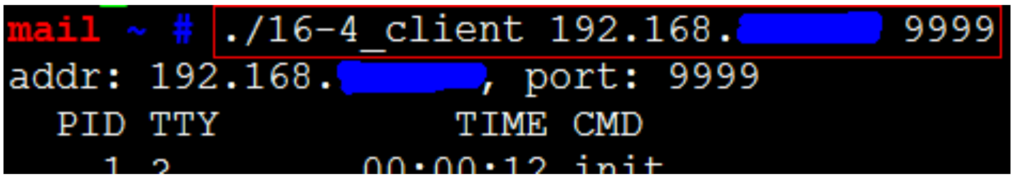
演示：



```
tongue aupe # gcc 16-4_server.c -o 16-4_server
tongue aupe # gcc 16-4_client.c -o 16-4_client
tongue aupe # ./16-4_server &
[1] 13212
tongue aupe # netstat -tunpl|grep 16-4_server
udp        0      0 0.0.0.0:9999            0.0.0.0:*                           13212/./16-4_server
```



```
tongue aupe # ./16-4_client 127.0.0.1 9999
addr: 127.0.0.1, port: 9999
recv
  PID TTY          TIME CMD
    1 ?        00:00:12 init
```



```
mail ~ # ./16-4_client 192.168.[redacted] 9999
addr: 192.168.[redacted], port: 9999
  PID TTY          TIME CMD
    1 ?        00:00:12 init
```

16-5、16-3已包含实现，既对子进程退出，资源的释放，使用异步信号通知的方式实现；
16-6、没兴趣。