<https://www.cnblogs.com/kex1n/p/7414181.html>

# [linux socket编程示例](http://www.cnblogs.com/kex1n/p/7414181.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <signal.h>  #include <unistd.h>  #include <stdlib.h>  #include <assert.h>  #include <stdio.h>  #include <string.h>    static bool stop = false;  static void handle\_term( int sig ) // kill pid;  in another tty will triggle this signal  {      stop = true;      printf("signal SIGTERM catched...\n");  }    static void handle\_int(int sig)  // ctrl+c; will triggle this signal  {      printf("signal SIGINT catched...\n");      stop = true;  }      //./listen 127.0.0.1 8888 100    int main( int argc, char\* argv[] )  {      signal( SIGTERM, handle\_term );      signal(SIGINT, handle\_int);        if( argc <= 3 )      {          printf( "usage: %s ip\_address port\_number backlog\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );      int backlog = atoi( argv[3] );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        printf("after bind...\n");        //backlog is the max number of waitting connect in wait queue      ret = listen( sock, backlog );   //listen is a none-block function      assert( ret != -1 );      printf("after listen...\n");        while ( ! stop )      {          sleep( 1 );      }        close( sock );      return 0;  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int reuse = 1;      setsockopt( sock, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof( reuse ) );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );      printf("AFTER bind...\n");        ret = listen( sock, 5 );      assert( ret != -1 );      printf("AFTER listen...\n");        //the returned client is client's address      struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength ); //accept is a block function      printf("AFTER accept...\n");        if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          //#define INET\_ADDRSTRLEN 16 , IPV4 address char array length, <netinet/in.h>          char remote[INET\_ADDRSTRLEN ];          printf( "connected with ip: %s and port: %d\n",              inet\_ntop( AF\_INET, &client.sin\_addr, remote, INET\_ADDRSTRLEN ), ntohs( client.sin\_port ) );          close( connfd );      }        close( sock );      return 0;  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>    #define BUF\_SIZE 1024    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( sock, 5 );      assert( ret != -1 );      printf("after listen...\n");        struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength );      printf("after accept...\n");      if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          char buffer[ BUF\_SIZE ];            memset( buffer, '\0', BUF\_SIZE );          ret = recv( connfd, buffer, BUF\_SIZE-1, 0 );          printf( "got %d bytes of normal data '%s'\n", ret, buffer );            memset( buffer, '\0', BUF\_SIZE );          //MSG\_OOB: support recv out-of-band data          //Only TCP support oob data          //TCP only 1 byte oob data          //use MSG\_OOB flag when call send to send oob data          ret = recv( connfd, buffer, BUF\_SIZE-1, MSG\_OOB );          printf( "got %d bytes of oob data '%s'\n", ret, buffer );            memset( buffer, '\0', BUF\_SIZE );          ret = recv( connfd, buffer, BUF\_SIZE-1, 0 );          printf( "got %d bytes of normal data '%s'\n", ret, buffer );            close( connfd );      }        close( sock );      return 0;  } |

connect

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | #include <sys/socket.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <string.h>  #include <stdlib.h>    #define BUFFER\_SIZE 512    int main( int argc, char\* argv[] )  {      if( argc <= 3 )      {          printf( "usage: %s ip\_address port\_number send\_bufer\_size\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in server\_address;      bzero( &server\_address, sizeof( server\_address ) );      server\_address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &server\_address.sin\_addr );      server\_address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int sendbuf = atoi( argv[3] );      int len = sizeof( sendbuf );      setsockopt( sock, SOL\_SOCKET, SO\_SNDBUF, &sendbuf, sizeof( sendbuf ) );      getsockopt( sock, SOL\_SOCKET, SO\_SNDBUF, &sendbuf, ( socklen\_t\* )&len );      printf( "the tcp send buffer size after setting is %d\n", sendbuf );        if ( connect( sock, ( struct sockaddr\* )&server\_address, sizeof( server\_address ) ) != -1 )      {          //          printf("call getsockname ...\n");          struct sockaddr\_in local\_address;          socklen\_t length;          int ret = getpeername(sock, ( struct sockaddr\* )&local\_address, &length);          assert(ret == 0);          char local[INET\_ADDRSTRLEN ];          printf( "local with ip: %s and port: %d\n",              inet\_ntop( AF\_INET, &local\_address.sin\_addr, local, INET\_ADDRSTRLEN ), ntohs( local\_address.sin\_port ) );          //            char buffer[ BUFFER\_SIZE ];          memset( buffer, 'a', BUFFER\_SIZE );          send( sock, buffer, BUFFER\_SIZE, 0 );      }        close( sock );      return 0;  } |

accept:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int reuse = 1;      setsockopt( sock, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof( reuse ) );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );      printf("AFTER bind...\n");        ret = listen( sock, 5 );      assert( ret != -1 );      printf("AFTER listen...\n");        //the returned client is client's address      struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength ); //accept is a block function      printf("AFTER accept...\n");        if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          //#define INET\_ADDRSTRLEN 16 , IPV4 address char array length, <netinet/in.h>          char remote[INET\_ADDRSTRLEN ];          printf( "connected with ip: %s and port: %d\n",              inet\_ntop( AF\_INET, &client.sin\_addr, remote, INET\_ADDRSTRLEN ), ntohs( client.sin\_port ) );            //          printf("call getsockname ...\n");          struct sockaddr\_in local\_address;          socklen\_t length;          int ret = getsockname(connfd, ( struct sockaddr\* )&local\_address, &length);          if (ret == 0)          {              char local[INET\_ADDRSTRLEN ];              printf( "local connfd ip: %s and port: %d\n", inet\_ntop( AF\_INET, &local\_address.sin\_addr, local, INET\_ADDRSTRLEN ), ntohs( local\_address.sin\_port ) );          }          else              printf("getsockname on connfd fail...\n");            bzero( &local\_address, sizeof( local\_address ) );          ret = getpeername(connfd, ( struct sockaddr\* )&local\_address, &length);          if (ret == 0)          {              char local1[INET\_ADDRSTRLEN ];              printf( "remote ip: %s and port: %d\n", inet\_ntop( AF\_INET, &local\_address.sin\_addr, local1, INET\_ADDRSTRLEN ), ntohs( local\_address.sin\_port ) );            }          else              printf("getpeername on connfd fail...\n");            close( connfd );      }        close( sock );      return 0;  } |

setsendbuffer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | #include <sys/socket.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <string.h>  #include <stdlib.h>    #define BUFFER\_SIZE 512    int main( int argc, char\* argv[] )  {      if( argc <= 3 )      {          printf( "usage: %s ip\_address port\_number send\_bufer\_size\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in server\_address;      bzero( &server\_address, sizeof( server\_address ) );      server\_address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &server\_address.sin\_addr );      server\_address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int sendbuf = atoi( argv[3] );      int len = sizeof( sendbuf );      setsockopt( sock, SOL\_SOCKET, SO\_SNDBUF, &sendbuf, sizeof( sendbuf ) );      getsockopt( sock, SOL\_SOCKET, SO\_SNDBUF, &sendbuf, ( socklen\_t\* )&len );      printf( "the tcp send buffer size after setting is %d\n", sendbuf );        if ( connect( sock, ( struct sockaddr\* )&server\_address, sizeof( server\_address ) ) != -1 )      {          char buffer[ BUFFER\_SIZE ];          memset( buffer, 'a', BUFFER\_SIZE );          send( sock, buffer, BUFFER\_SIZE, 0 );      }        close( sock );      return 0;  } |

setrecvbuffer:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>    #define BUFFER\_SIZE 1024    int main( int argc, char\* argv[] )  {      if( argc <= 3 )      {          printf( "usage: %s ip\_address port\_number receive\_buffer\_size\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );      int recvbuf = atoi( argv[3] );      printf("recvbuf is %d\n", recvbuf); /////!!!      int len = sizeof( recvbuf );      printf("len is %d\n", len); /////!!!      setsockopt( sock, SOL\_SOCKET, SO\_RCVBUF, &recvbuf, sizeof( recvbuf ) );      getsockopt( sock, SOL\_SOCKET, SO\_RCVBUF, &recvbuf, ( socklen\_t\* )&len );      printf( "the receive buffer size after settting is %d\n", recvbuf );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( sock, 5 );      assert( ret != -1 );        struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength );      if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          char buffer[ BUFFER\_SIZE ];          memset( buffer, '\0', BUFFER\_SIZE );          while( recv( connfd, buffer, BUFFER\_SIZE-1, 0 ) > 0 )          {              printf("%s\n", buffer);          }          close( connfd );      }        close( sock );      return 0;  } |

daytime:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <netdb.h>  #include <stdio.h>  #include <unistd.h>  #include <assert.h>  #include <errno.h>    int main( int argc, char \*argv[] )  {      //assert( argc == 2 );      //char \*host = argv[1];        struct hostent \*host;   //存放主机信息      //char addr\_p[NET\_ADDR\_STR\_LEN]; //用于存放点分十进制IP地址的字符串      if((host = gethostent()) == NULL)      {          perror("fail to get host's information\n");          return -1;      }      printf("hostName: %s\n" , host->h\_name);        //用域名或主机名获取IP地址      struct hostent\* hostinfo = gethostbyname( host->h\_name );      assert( hostinfo );      struct servent\* servinfo = getservbyname( "daytime", "tcp" );      assert( servinfo );      printf( "daytime port is %d\n", ntohs( servinfo->s\_port ) );        struct sockaddr\_in address;      address.sin\_family = AF\_INET;      address.sin\_port = servinfo->s\_port;      address.sin\_addr = \*( struct in\_addr\* )\*hostinfo->h\_addr\_list;        char remote[INET\_ADDRSTRLEN ];      printf( "connected with ip: %s and port: %d\n",              inet\_ntop( AF\_INET, &address.sin\_addr, remote, INET\_ADDRSTRLEN ), ntohs( address.sin\_port ) );        int sockfd = socket( AF\_INET, SOCK\_STREAM, 0 );      //int reuse = 1;      //setsockopt( sockfd, SOL\_SOCKET, SO\_REUSEADDR, &reuse, sizeof( reuse ) );      int result = connect( sockfd, (struct sockaddr\* )&address, sizeof( address ) );      printf( "errno is: %d\n", errno );      perror("connect error:");      assert( result != -1 );        char buffer[128];      result = read( sockfd, buffer, sizeof( buffer ) );      assert( result > 0 );      buffer[ result ] = '\0';      printf( "the day tiem is: %s", buffer );      close( sockfd );      return 0;  } |

testdup:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( sock, 5 );      assert( ret != -1 );        struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength );      if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          close( STDOUT\_FILENO );          dup( connfd );          printf( "abcd\n" ); //the same as write/send data in connfd          close( connfd );      }        close( sock );      return 0;  } |

writev:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>  #include <sys/stat.h>  #include <sys/types.h>  #include <fcntl.h>    #define BUFFER\_SIZE 1024  static const char\* status\_line[2] = { "200 OK", "500 Internal server error" };    int main( int argc, char\* argv[] )  {      if( argc <= 3 )      {          printf( "usage: %s ip\_address port\_number filename\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );      const char\* file\_name = argv[3];        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( sock, 5 );      assert( ret != -1 );        struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength );      if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          char header\_buf[ BUFFER\_SIZE ];          memset( header\_buf, '\0', BUFFER\_SIZE );          char\* file\_buf;          struct stat file\_stat;          bool valid = true;          int len = 0;          if( stat( file\_name, &file\_stat ) < 0 )          {              valid = false;          }          else          {              if( S\_ISDIR( file\_stat.st\_mode ) )              {                  valid = false;              }              else if( file\_stat.st\_mode & S\_IROTH )              {                  int fd = open( file\_name, O\_RDONLY );                  file\_buf = new char[ file\_stat.st\_size + 1 ];                  memset( file\_buf, '\0', file\_stat.st\_size + 1 );                  if ( read( fd, file\_buf, file\_stat.st\_size ) < 0 )                  {                      valid = false;                  }              }              else              {                  valid = false;              }          }            if( valid )          {              ret = snprintf( header\_buf, BUFFER\_SIZE-1, "%s %s\r\n", "HTTP/1.1", status\_line[0] );              len += ret;              ret = snprintf( header\_buf + len, BUFFER\_SIZE-1-len,                               "Content-Length: %d\r\n", (int)file\_stat.st\_size );              len += ret;              ret = snprintf( header\_buf + len, BUFFER\_SIZE-1-len, "%s", "\r\n" );              struct iovec iv[2];              iv[ 0 ].iov\_base = header\_buf;              iv[ 0 ].iov\_len = strlen( header\_buf );              iv[ 1 ].iov\_base = file\_buf;              iv[ 1 ].iov\_len = file\_stat.st\_size;              ret = writev( connfd, iv, 2 );          }          else          {              ret = snprintf( header\_buf, BUFFER\_SIZE-1, "%s %s\r\n", "HTTP/1.1", status\_line[1] );              len += ret;              ret = snprintf( header\_buf + len, BUFFER\_SIZE-1-len, "%s", "\r\n" );              send( connfd, header\_buf, strlen( header\_buf ), 0 );          }          close( connfd );          delete [] file\_buf;      }        close( sock );      return 0;  } |

 sendfile:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <sys/sendfile.h>    int main( int argc, char\* argv[] )  {      if( argc <= 3 )      {          printf( "usage: %s ip\_address port\_number filename\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );      const char\* file\_name = argv[3];        int filefd = open( file\_name, O\_RDONLY );      assert( filefd > 0 );      struct stat stat\_buf;      fstat( filefd, &stat\_buf );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( sock, 5 );      assert( ret != -1 );        struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength );      if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          //linux, not GNU. high performance send a file, 'zero copy'          sendfile( connfd, filefd, NULL, stat\_buf.st\_size );          close( connfd );      }        close( sock );      return 0;  } |

 splice:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <errno.h>  #include <string.h>  #include <fcntl.h>    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int sock = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( sock >= 0 );        int ret = bind( sock, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( sock, 5 );      assert( ret != -1 );        struct sockaddr\_in client;      socklen\_t client\_addrlength = sizeof( client );      int connfd = accept( sock, ( struct sockaddr\* )&client, &client\_addrlength );      if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );      }      else      {          int pipefd[2];          assert( ret != -1 );          ret = pipe( pipefd ); //pipe, pipefd[0] is opened for read, pipefd[1] is opened for write.          //splice用于在两个文件描述符之间移动数据， 也是零拷贝。使用splice时， fd\_in和fd\_out中必须至少有一个是管道文件描述符。          //sendfile只适用于将数据从文件拷贝到套接字上，限定了它的使用范围。Linux在2.6.17版本引入splice系统调用，          //用于在两个文件描述符中移动数据.            //an ECHO implement use splice.          //copy data from connfd(recv from client) to pipefd[1]          ret = splice( connfd, NULL, pipefd[1], NULL, 32768, SPLICE\_F\_MORE | SPLICE\_F\_MOVE );          assert( ret != -1 );          //copy data from pipefd[1] -->pipefd[0] --> connfd, send to client.          ret = splice( pipefd[0], NULL, connfd, NULL, 32768, SPLICE\_F\_MORE | SPLICE\_F\_MOVE );          assert( ret != -1 );          close( connfd );      }        close( sock );      return 0;  } |

tee:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <fcntl.h>    int main( int argc, char\* argv[] )  {      if ( argc != 2 )      {          printf( "usage: %s <file>\n", argv[0] );          return 1;      }      int filefd = open( argv[1], O\_CREAT | O\_WRONLY | O\_TRUNC, 0666 );      assert( filefd > 0 );        int pipefd\_stdout[2];      int ret = pipe( pipefd\_stdout );      assert( ret != -1 );        int pipefd\_file[2];      ret = pipe( pipefd\_file );      assert( ret != -1 );        //close( STDIN\_FILENO );      // dup2( pipefd\_stdout[1], STDIN\_FILENO );      //write( pipefd\_stdout[1], "abc\n", 4 );        //copy data: stdin --> pipefd\_stdout[1]      ret = splice( STDIN\_FILENO, NULL, pipefd\_stdout[1], NULL, 32768, SPLICE\_F\_MORE | SPLICE\_F\_MOVE );      assert( ret != -1 );        //tee在两个管道文件描述符之间复制数据，同是零拷贝。但它不消耗数据，数据被操作之后，仍然可以用于后续操作。      //copy data: pipefd\_stdout[0] --> pipefd\_file[1]      ret = tee( pipefd\_stdout[0], pipefd\_file[1], 32768, SPLICE\_F\_NONBLOCK );      assert( ret != -1 );        //copy data: pipefd\_file[0] --> filefd      ret = splice( pipefd\_file[0], NULL, filefd, NULL, 32768, SPLICE\_F\_MORE | SPLICE\_F\_MOVE );      assert( ret != -1 );        //copy data: pipefd\_stdout[0] --> stdout      ret = splice( pipefd\_stdout[0], NULL, STDOUT\_FILENO, NULL, 32768, SPLICE\_F\_MORE | SPLICE\_F\_MOVE );      assert( ret != -1 );        close( filefd );      close( pipefd\_stdout[0] );      close( pipefd\_stdout[1] );      close( pipefd\_file[0] );      close( pipefd\_file[1] );      return 0;  } |

 select：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100 | #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <fcntl.h>  #include <stdlib.h>    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );      printf( "ip is %s and port is %d\n", ip, port );        int ret = 0;      struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int listenfd = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( listenfd >= 0 );        ret = bind( listenfd, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( listenfd, 5 );      assert( ret != -1 );        struct sockaddr\_in client\_address;      socklen\_t client\_addrlength = sizeof( client\_address );      int connfd = accept( listenfd, ( struct sockaddr\* )&client\_address, &client\_addrlength );      if ( connfd < 0 )      {          printf( "errno is: %d\n", errno );          close( listenfd );      }        char remote\_addr[INET\_ADDRSTRLEN];      printf( "connected with ip: %s and port: %d\n", inet\_ntop( AF\_INET, &client\_address.sin\_addr, remote\_addr, INET\_ADDRSTRLEN ), ntohs( client\_address.sin\_port ) );        char buf[1024];      fd\_set read\_fds;      fd\_set exception\_fds;        FD\_ZERO( &read\_fds );      FD\_ZERO( &exception\_fds );        // option:SO\_OOBINLINE, value:1 --- put out of band data into normal data.      int nReuseAddr = 1;      setsockopt( connfd, SOL\_SOCKET, SO\_OOBINLINE, &nReuseAddr, sizeof( nReuseAddr ) );      while( 1 )      {          memset( buf, '\0', sizeof( buf ) );          FD\_SET( connfd, &read\_fds );          FD\_SET( connfd, &exception\_fds );            ret = select( connfd + 1, &read\_fds, NULL, &exception\_fds, NULL );          printf( "select one\n" );          if ( ret < 0 )          {              printf( "selection failure\n" );              break;          }            if ( FD\_ISSET( connfd, &read\_fds ) )          {              ret = recv( connfd, buf, sizeof( buf )-1, 0 );              if( ret <= 0 )              {                  break;              }              printf( "get %d bytes of normal data: %s\n", ret, buf );          }          else if( FD\_ISSET( connfd, &exception\_fds ) )          {              ret = recv( connfd, buf, sizeof( buf )-1, MSG\_OOB );              if( ret <= 0 )              {                  break;              }              printf( "get %d bytes of oob data: %s\n", ret, buf );              }        }        close( connfd );      close( listenfd );      return 0;  } |

epoll:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168 | #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <fcntl.h>  #include <stdlib.h>  #include <sys/epoll.h>  #include <pthread.h>    #define MAX\_EVENT\_NUMBER 1024  #define BUFFER\_SIZE 10    int setnonblocking( int fd )  {      int old\_option = fcntl( fd, F\_GETFL );      int new\_option = old\_option | O\_NONBLOCK;      fcntl( fd, F\_SETFL, new\_option );      return old\_option;  }    void addfd( int epollfd, int fd, bool enable\_et )  {      epoll\_event event;      event.data.fd = fd;      event.events = EPOLLIN;  //表示对应的文件描述符可以读（包括对端SOCKET正常关闭）      if( enable\_et )      {          //EPOLLET： 将EPOLL设为边缘触发(Edge Triggered)模式，这是相对于水平触发(Level Triggered)来说的。          event.events |= EPOLLET;      }      epoll\_ctl( epollfd, EPOLL\_CTL\_ADD, fd, &event ); //register to epoll      setnonblocking( fd );  }    void lt( epoll\_event\* events, int number, int epollfd, int listenfd )  {      char buf[ BUFFER\_SIZE ];      for ( int i = 0; i < number; i++ )      {          int sockfd = events[i].data.fd;          if ( sockfd == listenfd )          {              struct sockaddr\_in client\_address;              socklen\_t client\_addrlength = sizeof( client\_address );              int connfd = accept( listenfd, ( struct sockaddr\* )&client\_address, &client\_addrlength );              addfd( epollfd, connfd, false );          }          else if ( events[i].events & EPOLLIN )          {              printf( "event trigger once\n" );              memset( buf, '\0', BUFFER\_SIZE );              int ret = recv( sockfd, buf, BUFFER\_SIZE-1, 0 );              if( ret <= 0 )              {                  close( sockfd );                  continue;              }              printf( "get %d bytes of content: %s\n", ret, buf );          }          else          {              printf( "something else happened \n" );          }      }  }    void et( epoll\_event\* events, int number, int epollfd, int listenfd )  {      char buf[ BUFFER\_SIZE ];      for ( int i = 0; i < number; i++ )      {          int sockfd = events[i].data.fd;          if ( sockfd == listenfd )          {              struct sockaddr\_in client\_address;              socklen\_t client\_addrlength = sizeof( client\_address );              int connfd = accept( listenfd, ( struct sockaddr\* )&client\_address, &client\_addrlength );              addfd( epollfd, connfd, true );          }          else if ( events[i].events & EPOLLIN )          {              printf( "event trigger once\n" );              while( 1 )              {                  memset( buf, '\0', BUFFER\_SIZE );                  int ret = recv( sockfd, buf, BUFFER\_SIZE-1, 0 );                  if( ret < 0 )                  {                      if( ( errno == EAGAIN ) || ( errno == EWOULDBLOCK ) )                      {                          printf( "read later\n" );                          break;                      }                      close( sockfd );                      break;                  }                  else if( ret == 0 )                  {                      close( sockfd );                  }                  else                  {                      printf( "get %d bytes of content: %s\n", ret, buf );                  }              }          }          else          {              printf( "something else happened \n" );          }      }  }    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        int ret = 0;      struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int listenfd = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( listenfd >= 0 );        ret = bind( listenfd, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( listenfd, 5 );      assert( ret != -1 );        epoll\_event events[ MAX\_EVENT\_NUMBER ];      int epollfd = epoll\_create( 5 ); //tell epoll the listen number is 5      assert( epollfd != -1 );      addfd( epollfd, listenfd, true );        while( 1 )      {          //参数events用来从内核得到事件的集合，maxevents告之内核这个events有多大，          //这个maxevents的值不能大于创建epoll\_create()时的size，参数timeout是超时时间          //该函数返回需要处理的事件数目，如返回0表示已超时。          int ret = epoll\_wait( epollfd, events, MAX\_EVENT\_NUMBER, -1 );          if ( ret < 0 )          {              printf( "epoll failure\n" );              break;          }            lt( events, ret, epollfd, listenfd );          //et( events, ret, epollfd, listenfd );      }        close( listenfd );      return 0;  } |

 oneshot:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162 | #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <assert.h>  #include <stdio.h>  #include <unistd.h>  #include <errno.h>  #include <string.h>  #include <fcntl.h>  #include <stdlib.h>  #include <sys/epoll.h>  #include <pthread.h>    #define MAX\_EVENT\_NUMBER 1024  #define BUFFER\_SIZE 1024    struct fds  {     int epollfd;     int sockfd;  };    int setnonblocking( int fd )  {      int old\_option = fcntl( fd, F\_GETFL );      int new\_option = old\_option | O\_NONBLOCK;      fcntl( fd, F\_SETFL, new\_option );      return old\_option;  }    void addfd( int epollfd, int fd, bool oneshot )  {      epoll\_event event;      event.data.fd = fd;      //EPOLLIN表示对应的文件描述符可以读（包括对端SOCKET正常关闭）；      //EPOLLET将EPOLL设为边缘触发(Edge Triggered)模式，这是相对于水平触发(Level Triggered)来说的。      event.events = EPOLLIN | EPOLLET; //      if( oneshot )      {          //EPOLLONESHOT：只监听一次事件，当监听完这次事件之后，如果还需要继续监听这个socket的话，          //需要再次把这个socket加入到EPOLL队列里          event.events |= EPOLLONESHOT;      }      //EPOLL\_CTL\_ADD：注册新的fd到epfd中；      epoll\_ctl( epollfd, EPOLL\_CTL\_ADD, fd, &event );      setnonblocking( fd );  }    void reset\_oneshot( int epollfd, int fd )  {      epoll\_event event;      event.data.fd = fd;      event.events = EPOLLIN | EPOLLET | EPOLLONESHOT;      //EPOLL\_CTL\_MOD：修改已经注册的fd的监听事件；      epoll\_ctl( epollfd, EPOLL\_CTL\_MOD, fd, &event );  }    void\* worker( void\* arg )  {      int sockfd = ( (fds\*)arg )->sockfd;      int epollfd = ( (fds\*)arg )->epollfd;      printf( "start new thread to receive data on fd: %d\n", sockfd );      char buf[ BUFFER\_SIZE ];      memset( buf, '\0', BUFFER\_SIZE );      while( 1 )      {          int ret = recv( sockfd, buf, BUFFER\_SIZE-1, 0 );          //ret=0 -- 这里表示对端的socket已正常关闭.          if( ret == 0 )          {              close( sockfd );              printf( "foreiner closed the connection\n" );              break;          }          else if( ret < 0 )          {              //sockfd is NONBLOCK, EAGAIN -- no more data to read, so reset sockfd again              if( errno == EAGAIN )              {                  reset\_oneshot( epollfd, sockfd );                  printf( "read later\n" );                  break;              }          }          else          {              printf( "get content: %s\n", buf );              sleep( 5 );          }      }      printf( "end thread receiving data on fd: %d\n", sockfd );  }    int main( int argc, char\* argv[] )  {      if( argc <= 2 )      {          printf( "usage: %s ip\_address port\_number\n", basename( argv[0] ) );          return 1;      }      const char\* ip = argv[1];      int port = atoi( argv[2] );        int ret = 0;      struct sockaddr\_in address;      bzero( &address, sizeof( address ) );      address.sin\_family = AF\_INET;      inet\_pton( AF\_INET, ip, &address.sin\_addr );      address.sin\_port = htons( port );        int listenfd = socket( PF\_INET, SOCK\_STREAM, 0 );      assert( listenfd >= 0 );        ret = bind( listenfd, ( struct sockaddr\* )&address, sizeof( address ) );      assert( ret != -1 );        ret = listen( listenfd, 5 );      assert( ret != -1 );        epoll\_event events[ MAX\_EVENT\_NUMBER ];      int epollfd = epoll\_create( 5 );      assert( epollfd != -1 );      addfd( epollfd, listenfd, false );        while( 1 )      {          int ret = epoll\_wait( epollfd, events, MAX\_EVENT\_NUMBER, -1 );          if ( ret < 0 )          {              printf( "epoll failure\n" );              break;          }            for ( int i = 0; i < ret; i++ )          {              int sockfd = events[i].data.fd;              if ( sockfd == listenfd )              {                  struct sockaddr\_in client\_address;                  socklen\_t client\_addrlength = sizeof( client\_address );                  int connfd = accept( listenfd, ( struct sockaddr\* )&client\_address, &client\_addrlength );                  addfd( epollfd, connfd, true );              }              else if ( events[i].events & EPOLLIN )              {                  pthread\_t thread;                  fds fds\_for\_new\_worker;                  fds\_for\_new\_worker.epollfd = epollfd;                  fds\_for\_new\_worker.sockfd = sockfd;                  pthread\_create( &thread, NULL, worker, ( void\* )&fds\_for\_new\_worker );              }              else              {                  printf( "something else happened \n" );              }          }      }        close( listenfd );      return 0;  } |