| Winsock & .NET | Winsock | < Socket & Client-server Design Considerations | Linux Socket Index | TCP, UDP Program Examples > |

NETWORK PROGRAMMING LINUX SOCKET PART 7 - CODE SNIPPET EXAMPLES

Working program examples if any compiled using gcc, tested using the public IPs, rur with several times of update, as normal user. The Fedora machine used for the testin Execute" disabled and the SELinux set to default configuration.

Iterative, Connectionless Servers (UDP)

Creating a Passive UDP Socket

- The following is a sample codes for a passive UDP socket.

```
int passiveUDP(const char *service)
{
    return passivesock(service, "udp", 0);
}
u_short portbase = 0;

int passivesock(const char *service, const char *transport,
{
    struct servent  *pse;
    struct protoent *ppe;
    struct sockaddr_in sin;
    int     s, type;
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_addr.s_addr = INADDR_ANY;
    /* Map service name to port number */
    if(pse = getservbyname(service, transport))
        sin.sin_port = htons(ntohs((u_short)pse->s_port) +
    else if((sin.sin_port = htons((u_short)atoi(service)))
        errexit("can't get \"%s\" service entry\n", service
```

Socket Example 6
Socket Example 7
Advanced TCP/IP 1
Advanced TCP/IP 2
Advanced TCP/IP 3
Advanced TCP/IP 4
Advanced TCP/IP 5

```
        /* Map protocol name to protocol number */
        if((ppe = getprotobyname(transport)) == 0)
            errexit("can't get \"%s\" protocol entry\n", transp
        /* Use protocol to choose a socket type */
        if(strcmp(transport, "udp") == 0)
            type = SOCK_DGRAM;
        else
            type = SOCK_STREAM;
        /* Allocate a socket */
        s = socket(PF_INET, type, ppe->p_proto);
        if(s < 0)
            errexit("can't create socket: %s\n", strerror(errnc
        /* Bind the socket */
        if(bind(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
            errexit("can't bind to %s port: %s\n", service, str
        if(type == SOCK_STREAM && listen(s, qlen) < 0)
            errexit("can't listen on %s port: %s\n", service, s
        return s;
    }
```

A TIME Server

- The following is a sample codes for Time server.

```
/* main() - Iterative UDP server for TIME service */
int main(int argc, char *argv[ ])
{
    struct sockaddr_in fsin;
    char *service = "time";
    char buf[1];
    int sock;
    time_t now;
    int alen;
    sock = passiveUDP(service);
 while (1)
{
     alen = sizeof(fsin);
     if(recvfrom(sock, buf, sizeof(buf), 0, (struct sockaddr *)&fs
        errexit("recvfrom: %s\n", strerror(errno));
     time(&now);
     now = htonl((u_long)now);
     sendto(sock, (char *)&now, sizeof(now), 0, (struct sockaddr *
 }
}
```

Iterative, Connection-Oriented Servers (TCP)

A DAYTIME Server

- The following is a sample codes for Daytime server.

```
int passiveTCP(const char *service, int qlen)
{
    return passivesock(service, "tcp", qlen);
}
int main(int argc, char *argv[ ])
{
    struct sockaddr_in fsin;
    char *service = "daytime";
    int msock, ssock;
    int alen;
    msock = passiveTCP(service, 5);
    while (1) {
        ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
        if(ssock < 0)
            errexit("accept failed: %s\n", strerror(errno));
        TCPdaytimed(ssock);
        close(ssock);
  }
}
void TCPdaytimed(int fd)
{
    char *pts;
    time_t now;
    char *ctime();
    time(&now);
    pts = ctime(&now);
    write(fd, pts, strlen(pts));
    return;
}
```

- Close call requests a graceful shutdown.
- Data in transit is reliably delivered.
- Close requires messages and time.
- If the server closes you may be safe.
- If the client must close, the client may not cooperate.
- In our simple server, a client can make rapid calls and use resources associated with TCP shutdown timeout.

Concurrent, Connection-Oriented Servers (TCP)

The Value of Concurrency

- An iterative server may block for excessive time periods.
- An example is an echo server. A client could send many megabytes blocking other clients for substantial periods.
- A concurrent echo server could handle multiple clients simultaneously. Abusive clients would not affect polite clients as much.

A Concurrent Echo Server Using fork()

- The following is a sample codes for concurrent Echo server using fork().

```
int main(int argc, char *argv[ ])
{
    char   *service = "echo";  /* service name or port number */
    struct sockaddr_in fsin;   /* the address of a client */
    int alen;                  /* length of client's address */
    int msock;                 /* master server socket  */
    int ssock;                 /* slave server socket */
    msock = passiveTCP(service, QLEN);
    signal(SIGCHLD, reaper);
 while (1)
 {
     alen = sizeof(fsin);
     ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
     if(ssock < 0) {
        if(errno == EINTR)
            continue;
       errexit("accept: %s\n", strerror(errno));
  }
  switch (fork())
 {
     /* child */
     case 0:
       close(msock);
     exit(TCPechod(ssock));
     /* parent */
     default:
       close(ssock);
       break;
     case -1:
        errexit("fork: %s\n", strerror(errno));
   }
   }
}
int TCPechod(int fd)
{
    char  buf[BUFSIZ];
    int   cc;
 while (cc = read(fd, buf, sizeof buf))
 {
        if(cc < 0)
            errexit("echo read: %s\n", strerror(errno));
        if(write(fd, buf, cc) < 0)
            errexit("echo write: %s\n", strerror(errno));
  }
    return 0;
}
void reaper(int sig)
{
    int status;
    while (wait3(&status, WNOHANG, (struct rusage *)0) >= 0)
```

```
        /* empty ;*/
    }
```

Single-Process, Concurrent Servers (TCP)

Data-driven Processing

- Arrival of data triggers processing.
- A message is typically a request.
- Server replies and awaits additional requests.
- If processing time is small, the requests may be possible to handle sequentially.
- Timesharing would be necessary only when the processing load is too high for sequential processing.
- Timesharing with multiple slaves is easier.

Using Select for Data-driven Processing

- A process calls select to wait for one (or more) of a collection of open files (or sockets) to be ready for I/O.

```
int select(int n, fd_set *readfds, fd_set *writefds, fd_set *except
FD_CLR(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ZERO(fd_set *set);
```

- select() returns the number of fd's ready for I/O.
- FD_ISSET is used to determine which fd's are ready.
- select() returns 0 if the timer expires.
- select() returns -1 if there is an error.

An ECHO Server using a Single Process

- The following is a sample codes for Echo server using a single process.

```
int main(int argc, char *argv[ ])
{
    char   *service = "echo";
    struct sockaddr_in fsin;
    int    msock;
    fd_set rfds;
    fd_set afds;
    int    alen;
    int    fd, nfds;
    msock = passiveTCP(service, QLEN);

    nfds = getdtablesize();
    FD_ZERO(&afds);
    FD_SET(msock, &afds);
    while (1) {
```

```
            memcpy(&rfds, &afds, sizeof(rfds));
          if(select(nfds, &rfds, (fd_set *)0, (fd_set *)0, (struct time
               errexit("select: %s\n", strerror(errno));
            if(FD_ISSET(msock, &rfds))
 {
    int ssock;
    alen = sizeof(fsin);
    ssock = accept(msock, (struct sockaddr *)&fsin, &alen);
    if(ssock < 0)
       errexit("accept: %s\n", strerror(errno));
     FD_SET(ssock, &afds);
   }
   for(fd=0; fd < nfds; ++fd)
     if(fd != msock && FD_ISSET(fd, &rfds))
        if(echo(fd) == 0)
          {
              (void) close(fd);
              FD_CLR(fd, &afds);
           }
    }
}
int echo(int fd)
{
    char buf[BUFSIZ];
    int  cc;
    cc = read(fd, buf, sizeof buf);
    if(cc < 0)
        errexit("echo read: %s\n", strerror(errno));
    if(cc && write(fd, buf, cc) < 0)
        errexit("echo write: %s\n", strerror(errno));
    return cc;
}
```

Multiprotocol Servers

Why use multiple protocols in a server?

- Using separate UDP and TCP servers gives the system administrator more flexibility.
- Using separate servers result in 2 moderately simple servers.
- Using one server eliminates duplicate code simplifying software maintenance.
- Using one server reduces the number of active processes.

A Multiprotocol DAYTIME Server

- The following is a sample codes for Multiprotocol Daytime server.

```
int main(int argc, char *argv[])
{
    char   *service = "daytime"; /* service name or port number */
    char   buf[LINELEN+1];       /* buffer for one line of text */
```

```
    struct sockaddr_in fsin;     /* the request from address */
    int    alen;                 /* from-address length  */
    int    tsock;                /* TCP master socket  */
    int    usock;                /* UDP socket  */
    int    nfds;
    fd_set rfds;                 /* readable file descriptors */
    tsock = passiveTCP(service, QLEN);
    usock = passiveUDP(service);
    /* bit number of max fd */
    nfds = MAX(tsock, usock) + 1;
    FD_ZERO(&rfds);
    while (1) {
        FD_SET(tsock, &rfds);
        FD_SET(usock, &rfds);
        if(select(nfds, &rfds, (fd_set *)0, (fd_set *)0, (struct t
            errexit("select error: %s\n", strerror(errno));
        if(FD_ISSET(tsock, &rfds))
       {
            /* TCP slave socket */
            int ssock;
            alen = sizeof(fsin);
            ssock = accept(tsock, (struct sockaddr *)&fsin, &alen)
            if(ssock < 0)
                errexit("accept failed: %s\n", strerror(errno));
            daytime(buf);
            (void) write(ssock, buf, strlen(buf));
            (void) close(ssock);
        }
  if(FD_ISSET(usock, &rfds))
 {
   alen = sizeof(fsin);
   if(recvfrom(usock, buf, sizeof(buf), 0, (struct sockaddr *)&fsi
      errexit("recvfrom: %s\n", strerror(errno));
   daytime(buf);
   (void) sendto(usock, buf, strlen(buf), 0, (struct sockaddr *)&f
   }
  }
}
int daytime(char buf[])
{
    char   *ctime();
    time_t now;

    (void) time(&now);
    sprintf(buf, "%s", ctime(&now));
}
```

Multiservice Servers

Why combine services into one server?

- Fewer processes.
- Less memory.
- Less code duplication.
- Server complexity is really a result of accepting connections and handling concurrency.
- Having one server means the complex code does not need to be replicated.

Iterative Connectionless Server Design

- Server opens multiple UDP sockets each bound to a different port.
- Server keeps an array of function pointers to associate each socket with a service functions.
- Server uses select to determine which socket (port) to service next and calls the proper service function.

Iterative Connection-Oriented Server Design

- Server opens multiple passive TCP sockets each bound to a different port.
- Server keeps an array of function pointers to associate each socket with a service functions.
- Server uses select to determine which socket (port) to service next.
- When a connection is ready, server calls accept to start handling a connection.
- Server calls the proper service function.

Concurrent Connection-Oriented Server Design

- Master uses select to wait for connections over a set of passive TCP sockets.
- Master forks after accept.
- Slave handles communication with the client.

Single-Process Server Design

- Master uses select to wait for connections over a set of passive TCP sockets.
- After each accept the new socket is added to the fd_set(s) as needed to handle client communication.
- Complex if the client protocols are not trivial.

Invoking Separate Programs from a Server

- Master uses select() to wait for connections over a set of passive TCP sockets.
- Master forks after accept.
- Child process uses execve to start a slave program to handle client communication.
- Different protocols are separated making it simpler to maintain.
- Changes to a slave program can be implemented without restarting the master.

Multiservice, Multiprotocol Servers

- Master uses select to wait for connections over a set of passive TCP sockets.
- In addition the fd_set includes a set of UDP sockets awaiting client messages.
- If a UDP message arrives, the master calls a handler function which formulates and

issues a reply.
- If a TCP connection is needed the master calls accept.
- For simpler TCP connections, the master can handle read and write requests iteratively.
- The master can also use select.
- Lastly the master can use fork and let the child handle the connection.

Super Server Code Example

- The following is a sample codes for super server.

```
struct service {
    char  *sv_name;
    char  sv_useTCP;
    int   sv_sock;
    int   (*sv_func)(int);
};
struct service svent[ ] = {
    { "echo", TCP_SERV, NOSOCK, TCPechod },
    { "chargen", TCP_SERV, NOSOCK, TCPchargend },
    { "daytime", TCP_SERV, NOSOCK, TCPdaytimed },
    { "time", TCP_SERV, NOSOCK, TCPtimed },
    { 0, 0, 0, 0 },
};
int main(int argc, char *argv[ ])
{
    struct service  *psv,    /* service table pointer */
            *fd2sv[NOFILE];  /* map fd to service pointer */
    int     fd, nfds;
    fd_set  afds, rfds;      /* readable file descriptors */
    nfds = 0;
    FD_ZERO(&afds);
    for(psv = &svent[0]; psv->sv_name; ++psv)
 {
        if(psv->sv_useTCP)
            psv->sv_sock = passiveTCP(psv->sv_name, QLEN);
        else
            psv->sv_sock = passiveUDP(psv->sv_name);
        fd2sv[psv->sv_sock] = psv;
```

```
            nfds = MAX(psv->sv_sock+1, nfds);
            FD_SET(psv->sv_sock, &afds);
        }

        (void) signal(SIGCHLD, reaper);
          while (1) {
            memcpy(&rfds, &afds, sizeof(rfds));
            if(select(nfds, &rfds, (fd_set *)0, (fd_set *)0, (struct time
             {
               if(errno == EINTR)
                 continue;
               errexit("select error: %s\n", strerror(errno));
             }
              for(fd=0; fd<nfds; ++fd)
              {
                  if(FD_ISSET(fd, &rfds))
                  {
                      psv = fd2sv[fd];
                      if(psv->sv_useTCP)
                         doTCP(psv);
                      else
                         psv->sv_func(psv->sv_sock);
                  }
                }
          }
    }
    /* doTCP() - handle a TCP service connection request */
    void doTCP(struct service *psv)
    {
        /* the request from address */
        struct  sockaddr_in fsin;
        /* from-address length */
        int alen;
        int fd, ssock;
        alen = sizeof(fsin);
        ssock = accept(psv->sv_sock, (struct sockaddr *)&fsin, &alen);
        if(ssock < 0)
            errexit("accept: %s\n", strerror(errno));
        switch (fork())
       {
        case 0:
            break;
        case -1:
            errexit("fork: %s\n", strerror(errno));
        default:
            (void) close(ssock);
            /* parent */
            return;
        }
        /* child */
```

```
        for(fd = NOFILE; fd >= 0; --fd)
            if(fd != ssock) (void) close(fd);
        exit(psv->sv_func(ssock));
    }
    /* reaper() - clean up zombie children */
    void reaper(int sig)
    {
        int status;
        while(wait3(&status, WNOHANG, (struct rusage *)0) >= 0)
        /* empty */;
    }
```

Continue on next Module…

Further reading and digging:

1. Check the best selling C / C++, Networking, Linux and Open Source books at Amazon.com.
2. GCC, GDB and other related tools.

| Winsock & .NET | Winsock | < Socket & Client-server Design Considerations | Linux Socket Index | TCP, UDP Program Examples > |