Sockets e *Threads* Usando C

Claudine Santos Badue
Bruno Augusto Vivas e Pôssas

Junho de 2004

Projeto e Análise de Algoritmos
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais

Sockets

- A socket is an end-point of communication to which a name may be bound.
- Two processes each create a socket, and connect to those two end-points to establish a reliable byte stream or unreliable datagram.
- Once connected, the descriptors for the socket can be read from or written to by user processes similar to regular file operations. The transparency of sockets allows the kernel to redirect the output of another process residing on another machine.

Sockets

- Sockets exist within communication domains that are abstractions which imply both an addressing structure and a set of protocols which implement various socket types with the domain.
- BSD socket supports the Unix domain, the Internet domain, and the XNS domain. In our environment, we are limited to use the Internet domain for communication over local networks.

Sockets

- In the Internet domain, stream sockets and datagram sockets use TCP/IP and UDP/IP as the underlying protocols.
  - A stream socket provides for the connection-based, bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries.
  - A datagram socket on the other hand supports connectionless, bidirectional flow of data that is not promised to be sequenced, reliable, or unduplicated.

socket - create an endpoint for communication

- SYNOPSIS
  int socket(int domain, int type, int protocol);
- DESCRIPTION
  - Socket creates an endpoint for communication and returns a descriptor.

socket

- SYNOPSIS

  int socket(int domain, int type, int protocol);

- DESCRIPTION
  - The domain parameter specifies a communication domain; this selects the protocol family which will be used for communication. The currently understood formats include:

Name	Purpose
PF_UNIX,PF_LOCAL	Local communication
PF_INET	**IPv4 Internet protocols**
PF_INET6	IPv6 Internet protocols
PF_IPX	IPX - Novell protocols
PF_NETLINK	Kernel user interface device
PF_X25	ITU-T X.25 / ISO-8208 protocol
PF_AX25	Amateur radio AX.25 protocol
PF_ATMPVC	Access to raw ATM PVCs
PF_APPLETALK	Appletalk
PF_PACKET	Low level packet interface

socket

- SYNOPSIS
  int socket(int domain, int type, int protocol);
- DESCRIPTION
  - The socket has the indicated type, which specifies the communication semantics. Basic defined types are:
    * SOCK_STREAM: Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
    * SOCK_DGRAM: Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
  - The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family.

connect - initiate a connection on a socket

- SYNOPSIS
  int connect(int sockfd, const struct sockaddr *serv_addr,
  socklen_t addrlen);
- DESCRIPTION
  - The file descriptor sockfd must refer to a socket.
  - If the socket is of type SOCK_DGRAM then the serv_addr address is the address to which datagrams are sent by default, and the only address from which datagrams are received.

connect

- SYNOPSIS
  int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
- DESCRIPTION
  - If the socket is of type SOCK_STREAM, this call attempts to make a connection to another socket. The other socket is specified by serv_addr, which is an address (of length addrlen) in the communications space of the socket. Each communications space interprets the serv_addr parameter in its own way.
  - Generally, connection-based protocol sockets may successfully connect only once; connectionless protocol sockets may use connect multiple times to change their association.

bind - bind a name to a socket

- SYNOPSIS
  int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
- DESCRIPTION
  - Bind gives the socket sockfd the local address my_addr. my_addr is addrlen bytes long. Traditionally, this is called "assigning a name to a socket."
  - When a socket is created with socket, it exists in a name space (address family) but has no name assigned.
  - It is normally necessary to assign a local address using bind before a SOCK_STREAM socket may receive connections.

listen - listen for connections on a socket

- SYNOPSIS
  int listen(int s, int backlog);
- DESCRIPTION
  - Listen specify a willingness to accept incoming connections and a queue limit for incoming connections to the socket s.
  - The backlog parameter defines the maximum length the queue of pending connections may grow to.
  - If a connection request arrives with the queue full the client may receive an error with an indication of ECONNREFUSED or, if the underlying protocol supports retransmission, the request may be ignored so that retries succeed.

accept - accept a connection on a socket

- SYNOPSIS
  int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
- DESCRIPTION
  - The accept function is used with connection-based socket types.
  - It extracts the first connection request on the queue of pending connections, creates a new connected socket with mostly the same properties as s, and allocates a new file descriptor for the socket, which is returned.
  - The newly created socket is no longer in the listening state.

accept

- SYNOPSIS
  int accept(int s, struct sockaddr *addr, socklen_t *addrlen);
- DESCRIPTION
  - The argument s is a socket that has been created with socket, bound to a local address with bind, and is listening for connections after a listen.
  - The argument addr is a pointer to a sockaddr structure. This structure is filled in with the address of the connecting entity, as known to the communications layer.
  - The addrlen argument is initialized with the size of the structure pointed to by addr and modified on return to indicate the actual length (in bytes) of the address returned.
  - If no pending connections are present on the queue, accept blocks the caller until a connection is present.

send, sendto - send a message to a socket

- SYNOPSIS
  int send(int s, const void *msg, size_t len, int flags);
  int sendto(int s, const void *msg, size_t len, int flags, const struct sockaddr *to, socklen_t tolen);
- DESCRIPTION
  - send and sendto are used to transmit a message to a socket.
  - The address of the target is given by to with tolen specifying its size. The length of the message is given by len.
  - The flags parameter is a flagword and can contain the following flags: MSG_NOSIGNAL, MSG_OOB, MSG_DONTROUTE, MSG_DONTWAIT, MSG4_CONFIRM.

recv, recvfrom - receive a message from a socket

- SYNOPSIS
  int recv(int s, void *buf, size_t len, int flags);
  int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);
- DESCRIPTION
  - recvfrom is used to receive messages from a socket.
  - If the socket is not connection-oriented, and from is not NULL, the source address of the message is filled in.
  - The argument fromlen is initialized to the size of the buffer and modified on return to indicate the actual size of the address stored there.

recv, recvfrom

- SYNOPSIS
  int recv(int s, void *buf, size_t len, int flags);
  int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);
- DESCRIPTION
  - The flags argument to a recv call is formed by OR'ing one or more of the following values: MSG_WAITALL, MSG_NOSIGNAL, MSG_OOB, MSG_PEEK, MSG_TRUNC, MSG_ERRQUEUE.

recv, recvfrom

- SYNOPSIS
  int recv(int s, void *buf, size_t len, int flags);
  int recvfrom(int s, void *buf, size_t len, int flags, struct sockaddr *from, socklen_t *fromlen);
- DESCRIPTION
  - recv is normally used only on a connected socket and is identical to recvfrom with a NULL from parameter.
  - If no messages are available at the socket, the receive calls wait for a message to arrive.
  - The receive calls normally return any data available, up to the requested amount, rather than waiting for receipt of the full amount requested.

Client/Server Demo

- http://www.dcc.ufmg.br/∼nivio/cursos/pa04/tp4/sockets/clientserver/client.c
- http://www.dcc.ufmg.br/∼nivio/cursos/pa04/tp4/sockets/clientserver/server.c

select: wait for a number of file descriptors to change status

- SYNOPSIS
  int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
- DESCRIPTION
  - Three independent sets of descriptors are watched. Those listed in readfds will be watched to see if characters become available for reading, (more precisely, to see if a read will not block), those in writefds will be watched to see if a write will not block, and those in exceptfds will be watched for exceptions.
  - On exit, the sets are modified in place to indicate which descriptors actually changed status.

select

- SYNOPSIS
  int select(int n, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
- DESCRIPTION
  - n is the highest-numbered descriptor in any of the three sets, plus 1.
  - timeout is an upper bound on the amount of time elapsed before select returns. It may be zero, causing select to return immediately. (This is useful for polling.) If timeout is NULL (no timeout), select can block indefinitely.

select

- SYNOPSIS

  FD_ZERO(fd_set *set);

  FD_SET(int fd, fd_set *set);

  FD_CLR(int fd, fd_set *set);

  FD_ISSET(int fd, fd_set *set);

- DESCRIPTION
  - Four macros are provided to manipulate the sets.
    - FD_ZERO will clear a set.
    - FD_SET and FD_CLR add or remove a given descriptor from a set.
    - FD_ISSET tests to see if a descriptor is part of the set; this is useful after select returns.

Master/Slave Demo

- http://www.dcc.ufmg.br/∼nivio/cursos/pa04/tp4/sockets/masterslave/master_sel.c
- http://www.dcc.ufmg.br/∼nivio/cursos/pa04/tp4/sockets/masterslave/slave.c

Threads

- Thread is a lightweight process.
- A thread is considered lightweight because it runs within the context of a program and takes advantage of the resources allocated for that program and the program's environment.
- Threads enhance performance and functionality of a program by allowing it to efficiently perform multiple tasks simultaneously.

pthread_create - create a new thread

- SYNOPSIS
  int pthread_create(pthread_t * thread, pthread_attr_t * attr, void * (*start_routine)(void *), void * arg);
- DESCRIPTION
  - pthread_create creates a new thread of control that executes concurrently with the calling thread.
  - The new thread calls the function start_routine, passing it arg as first argument.
  - The attr argument specifies thread attributes to be applied to the new thread. The attr argument can also be NULL, in which case default attributes are used: the created thread is joinable (not detached) and has an ordinary (not realtime) scheduling policy.
  - On success, the identifier of the newly created thread is stored in the location pointed by the thread argument.

pthread_join - wait for termination of another thread

- SYNOPSIS
  int pthread_join(pthread_t th, void **thread_return);
- DESCRIPTION
  - pthread_join suspends the execution of the calling thread until the thread identified by th terminates.
  - If thread_return is not NULL, the return value of th is stored in the location pointed to by thread_return.

Master/Slave Demo

- http://www.dcc.ufmg.br/∼nivio/cursos/pa04/tp4/sockets/masterslave/master.c
- http://www.dcc.ufmg.br/∼nivio/cursos/pa04/tp4/sockets/masterslave/slave.c