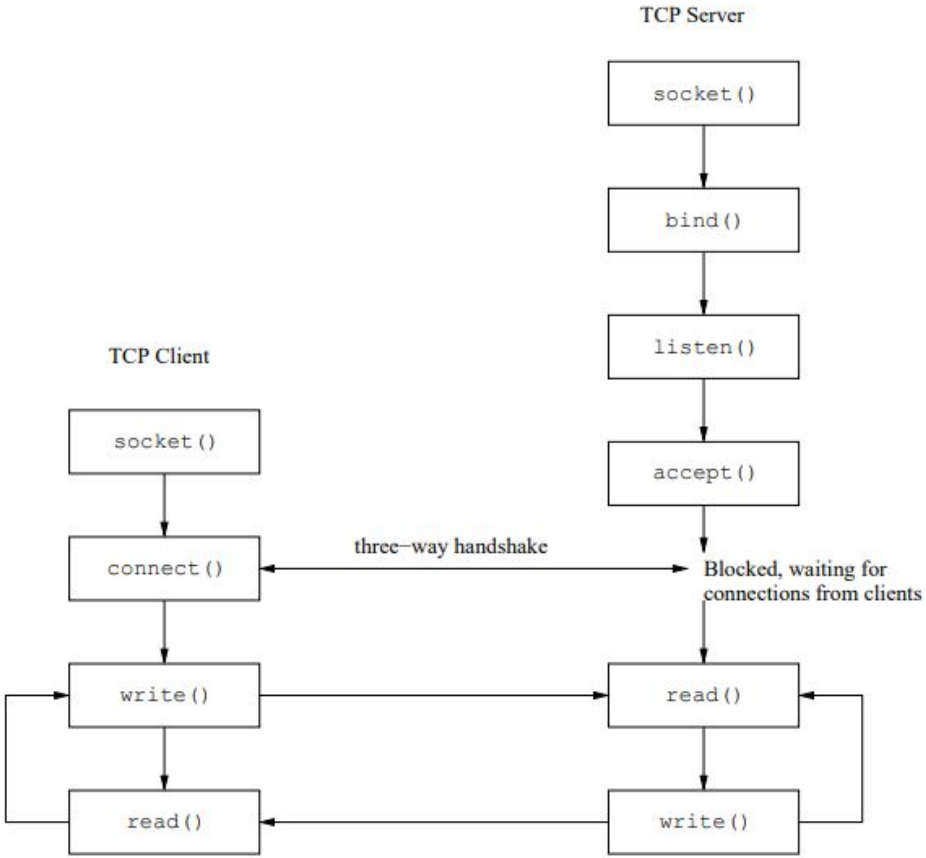
Network sockets

Sockets

The purpose - organization of both intra-machine and inter-machine method of interaction between processes

- Sockets support a large number of network protocols
- The most popular Protocol is TCP/IP

Scheme


TCP Server
socket()
bind()
listen()
accept()
TCP Client
socket()
connect()
three−way handshake
Blocked, waiting for connections from clients
write()
read()
read()
write()

Socket descriptor

The socket descriptor - analogue of file descriptor

You can use some files related functions **(read, write, ...)**

Creation of the descriptors

#include <sys/socket.h>

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

Returns the file descriptor (socket) in case of success, -1 in case of error

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

AF_INET	Internet Domain IPv4
AF_INET6	Internet Domain IPv6
AF_UNIX	Domain UNIX
AF_UNSPEC	Undefined domain

AF_ (от address family – address family)

AF_UNSPEC indicates an undefined domain that can represent any domain

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

SOCK_DGRAM	Not focused on creating a logical connection, messages fixed length, message delivery is not guaranteed
SOCK_RAW	The interface of the datagram on the IP Protocol
SOCK_STREAM	Focused on creating a logical connection, the order of data transmission, guaranteed message delivery, bidirectional byte flow
SOCK_SEQPACKET	Focused on the creation of a logical connection, the order of data transmission, fixed-length messages, guaranteed delivery of messages

Protocols without logical connection

SOCK_DGRAM

- Datagram - independent message
- Each message has a specified destination
- Delivery is not guaranteed

Typically, use UDP in applications where speed is more critical than reliability.

Protocols with logical connection

SOCK_STREAM, SOCK_SEQPACKET

The necessary installation connections

Messages do not contain a destination

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

Select the Protocol type. Default is set to **0** (the system chooses the Protocol based on the socket type)

AF_INET + SOCK_STREAM = **?**

AF_INET + SOCK_DGRAM = **?**

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

Select the Protocol type. Default is set to **0** (the system chooses the Protocol based on the socket type)

AF_INET + SOCK_STREAM = **TCP**
AF_INET + SOCK_DGRAM = **UDP**

Comparison with Filesystem

socket() similar to **open**()

Support:

read, write, close

Not support:

lseek, fsync, fdatasync

Sockets

To connect two processes, you need two sockets:

- The server socket
- The client socket

Server socket

socket	To create a socket
bind	Bind socket to address
listen	Start waiting for connections
accept	Waiting for connection. Getting the client socket

Server side

Each socket must have an address
(both client and server)

For sockets in the domain AF_INET (IPv4) address:

[ip_address] + [port]   8.8.8.8:53

A bind of socket address

```
#include <sys/socket.h>
```

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t len);
```

Returns 0 if successful, -1 if failed

Restrictions

- The address you specify must be a valid address for the machine on which the process is running – we cannot specify an address that belongs to another machine.

- The address format must match the format that is supported by the address family that you specified when you created the socket.

- The port number cannot be less than 1024 if the process does not have the appropriate privileges (for example, superuser privileges).

- Typically, each specific address can be associated with only one socket

Specify Address

Each domain has its own address format

AF_INET:

```
struct in_addr {
    in_addr_t s_addr; /* адрес IPv4 */
};
struct sockaddr_in {
    sa_family_t sin_family;  /* address family */
    in_port_t sin_port;          /* port number */
    struct in_addr sin_addr; /* address IPv4 */
};
```

Address translation

```
include <arpa/inet.h>
```

Convert a text address to a numeric address

```
int inet_pton(int af, const char *src, void *dst);
```

Inverse transformation

```
const char *inet_ntop(int af, const void *src, char *dst,
socklen_t size);
```

Example

inet_pton.c

Port conversion

```
#include <arpa/inet.h>
```

```
uint16_t htons(uint16_t hostint16);
```

Returns a 16-bit integer with network byte order

Inverse transformation

```
#include <arpa/inet.h>
```

```
uint16_t ntohs(uint16_t netint16);
```

Returns a 16-bit integer with a hardware byte order

Listen on all network interfaces

Special address: **INADDR_ANY**

Waiting for connections

```
#include <sys/socket.h>

int listen(int sockfd, int backlog);
```

Returns 0 if successful, -1 if failed

backlog - maximum number of pending requests

Receiving connections

```
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *restrict addr,
socklen_t *restrict len);
```

Returns a new socket handle on success, -1 on failure

The call is blocking!

int accept()

New socket - socket connected to the client

The **sockfd** socket remains free and ready to accept new connections

addr, len - buffer and its size where the client address will be put

Socket options

#include <sys/socket.h>

int **setsockopt**(int sockfd, int level, int option, const void *val, socklen_t len);

Returns 0 if successful, -1 if failed

int level

1. Generic parameters that are common to all socket types.

2. Parameters that are supported at the socket level but depend on the Protocol used.

3. Parameters unique to each individual Protocol.

For example:

SOL_SOCKET, **IPPROTO_TCP**, **IPPROTO_IP**

SOL_SOCKET

For example:

```
int reuse = 1;
setsockopt( fd, SOL_SOCKET, SO_REUSEADDR, &reuse, sizeof(int) )
```

Allow port reuse

Example

Creating a simple server

simple_server.c

As a client, use **telnet** or **netcat**

Example

The server listens to all network interfaces

simple_server_all.c

Data transmission

The **read** and **write** functions can be used if a connection has been established

Compatibility with programs for working with files is achieved

Additional function

#include <sys/socket.h>

ssize_t **send**(int sockfd, const void *buf, size_t nbytes, int flags);

Returns the number of bytes sent in case of success, -1 in case of error

Flags

MSG_DONTROUTE	Do not send the packet outside the local network
MSG_DONTWAIT	Allow non-blocking execution mode of operation (equivalent to the O_NONBLOCK flag)
MSG_EOR	Indicates the end of recording if supported by the Protocol
MSG_OOB	Indicates the transmission of emergency data, if supported by the Protocol

sendto (SOCK_DGRAM)

```
#include <sys/socket.h>

ssize_t sendto(int sockfd, const void *buf, size_t nbytes, int
flags, const struct sockaddr *destaddr, socklen_t destlen);
```

Returns the number of bytes sent
if successful, -1 in case of error

Receive data

```
#include <sys/socket.h>
```

```
ssize_t recv(int sockfd, void *buf, size_t nbytes, int flags);
```

Returns the length of the message in bytes, 0 if no messages are available and a socket write operation is prohibited on the remote end of the connection, -1 if an error occurs

Flags

MSG_OOB	Accept emergency data if supported by the Protocol
MSG_PEEK	Return the contents of the package, but do not remove it from the receiving queue
MSG_TRUNC	Request that the actual size of the package be returned, even if it has been truncated
MSG_WAITALL	Wait until all data is received (sock_stream only)

Receive data (SOCK_DGRAM)

#include <sys/socket.h>

ssize_t **recvfrom**(int sockfd, void *restrict buf, size_t len, int flags, struct sockaddr *restrict addr, socklen_t *restrict addrlen);

Allows you to save data about the sender socket.

Example

Simple client-server based communication
Protocol datagram

server_dg.c

```
netcat -u host port
```

Example

Simple client-server based communication Protocol (stream sockets)

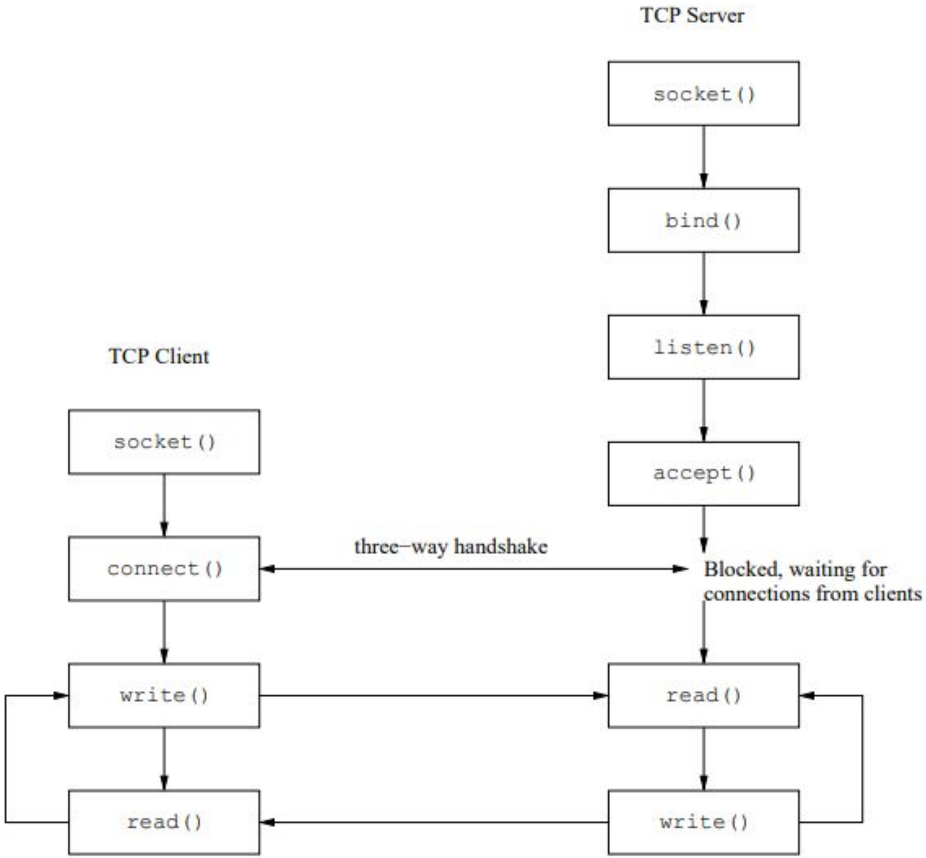
server_stream.c

Client side

The client socket must also have an address setup

Typically, the client address assignment is delegated to the operating system

Scheme


TCP Server
socket()
bind()
listen()
accept()
TCP Client
socket()
connect()
three−way handshake
Blocked, waiting for connections from clients
write()
read()
read()
write()

The connection

#include <sys/socket.h>

int **connect**(int sockfd, const struct sockaddr *addr, socklen_t len);

Returns 0 if successful, -1 if failed

Address

Address of the server to which you want to connect

Errors:

- Server unreachable
- Server queue is full

connect() и SOCK_DGRAM

The server address is used as the address that will be included in each message

Example

Connect the client to the server. Organization of communication.

client_stream.c

Exercise 10a

In the previous example, we have shown a sequential server.

- It serves requests sequentially, in order of arrival (FIFO)
- A client has to wait for all preceding requests and for its request to be served before getting the response

Problems:

- A short request by a client may have to wait for longer requests to be completed
- The server can be blocked on I/O while serving a request; this is inefficient!

What is solution?

Exercise 10a

Solution:

- Multi-process: one process per client (dynamically created, or “pre-forked”);
- Multi-thread: one thread per client (dynamically created, or pre-created).

Template: fork-server.c

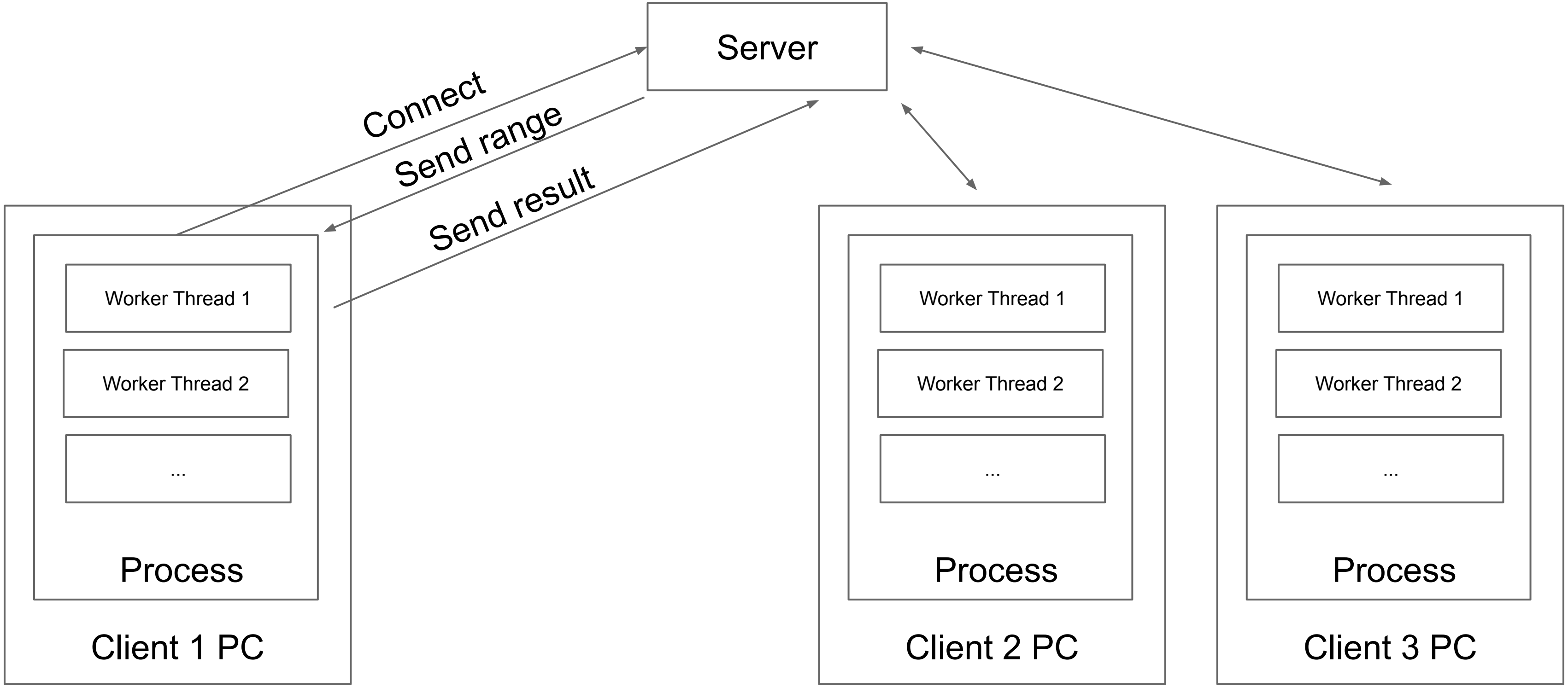
Exercise 10b

Create program for distributed Pi calculation.

Server: Wait for N clients. Send new range (for example, start position and count) to each client. When all clients finished output result to the screen.

Client: Compute range (you could use OpenMP to make client parallel) and send results to the server.

Exercise 10b



Server

Connect

Send range

Send result

Worker Thread 1

Worker Thread 2

...

Process

Client 1 PC

Worker Thread 1

Worker Thread 2

...

Process

Client 2 PC

Worker Thread 1

Worker Thread 2

...

Process

Client 3 PC