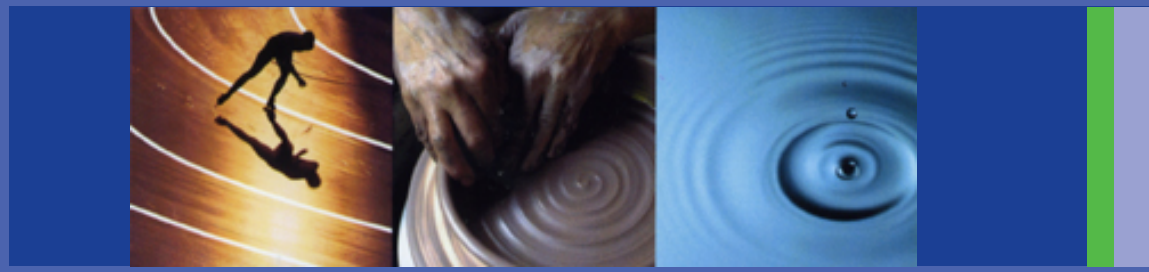


Мультиплексирование ввода и вывода при передаче данных по сети

Клочков Максим Никоолаевич, ИУ-7

9 ноября 2025 г.

Работа с сетью с помощью BSD sockets

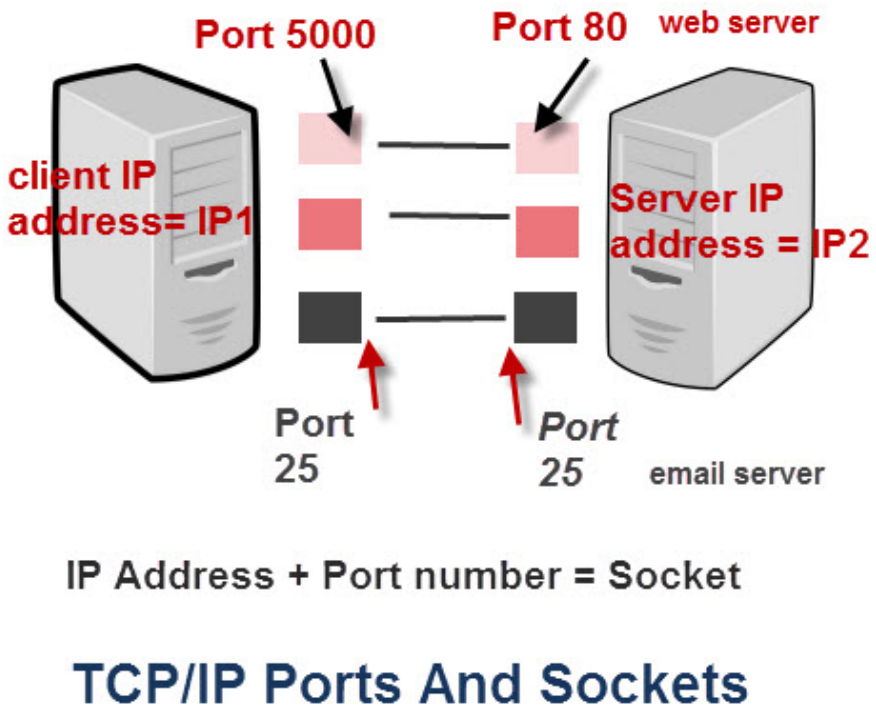
В большинстве современных операционных систем высокоуровневая передача данных по сети реализована с помощью API, известного как BSD sockets

Если сетевое соединение установлено, то с точки зрения ОС и исполняемого процесса сокет похож на обычный открытый файл, описывается дескриптором

Можно читать и писать обычными системными вызовами **read** и **write** (есть еще **send** и **recv**, часто представляют собой «обёртки» над read и write, или наоборот)

То, что записали (write) на одной стороне сетевого соединения, можно прочитать (read) на другой стороне сетевого соединения, и наоборот

Данные, передаваемые по сети, испытывают задержки и частично теряются — может быть решено ядром операционной системы, а именно реализацией протокола TCP



Проблемы и решения

Проблемы:

- Размер буферов TCP в ядре операционной системы ограничен — в сокет за один раз может быть записан конечный объем информации (несколько килобайт или десятков килобайт), причем заранее не известно, сколько именно
- Принимающий (читающий) процесс не знает заранее, когда ему будут присланы данные

Решение «в лоб»:

- Организовать на каждое сетевое соединение отдельный процесс или поток
- Читать или записывать данные в режиме активного ожидания

```
1  while (response_sent < http_response->body_length) {
2      bytes_sent = write(
3          sock_fd,
4          http_response->body + body_sent,
5          MIN(BUFFER_LEN, http_response->body_length - response_sent)
6      );
7      response_sent += bytes_sent;
8  }
```

Неэффективно: сервер «расходует» целый процесс или поток (thread) на *одно* сетевое подключение

Что предлагает операционная система

Системный вызов select и его вариации:

- Вызовы select, pselect, poll, epoll, kqueue — делают примерно одно и тоже, в некоторых ОС какие-то вызовы могут отсутствовать, какие-то могут быть библиотечными функциями-«обёртками»

- Осуществляют выборку событий, связанных с набором дескрипторов (любых — не обязательно дескрипторов сетевых сокетов)

Обрабатываются три типа событий:

- Изменение состояния файлового дескриптора (появление входящего запроса на соединение, появление данных, доступных на чтение, освобождение места в буфере для записи, исключительная ситуация)

- Истечение заданного тайм-аута (от момента обращения к системному вызову)

- Получение неигнорируемого сигнала

```
FD_ZERO(fd_set *set);
FD_CLR(int fd, fd_set *set);
FD_SET(int fd, fd_set *set);
FD_ISSET(int fd, fd_set *set);

int select(
    int descriptor_count,
    fd_set *read_fd_set,
    fd_set *write_fd_set,
    fd_set *except_fd_set,
    strict timeval *timeout,
    );
```

Особенности:

- Любой fd_set можно обнулить (заменить нулевым указателем), если обработка событий соответствующего типа не требуется
- Структура timeval изменяется после вызова select, при каждом запуске надо заново заполнить начальными значениями

select и его вариации не расходуют процессорное время (аналогично sleep/usleep/nanosleep)

Как пользоваться

1. Для каждого обрабатываемого соединения: завести переменную состояния и организовать собственный буфер
2. Перевести дескрипторы в неблокирующий режим
3. Организовать цикл обработки событий
4. В цикле после каждого вызова select анализировать состояние каждого дескриптора
5. Читать **один** фрагмент данных из тех, дескрипторов, которые готовы для чтения
6. Писать **один** фрагмент данных в те дескрипторы, которые готовы для записи
7. Вернуться к началу цикла

```
flags = fcntl(d, F_GETFL);
fcntl(d, F_SETFL, flags | O_NONBLOCK);


for(;;) {
    nfds = select(...);
    for(int i = 0; i < nfds; i++) {
      switch(request->state) {
        case STATE_READ:
          readRequest(request);
          break;
        case STATE_SEND:
          send_response(request);
          break;
      }
  }

void send_response(httpRequestResponsePtr request) {
  request->state = STATE_SEND;

  int len = write(
    request->socket,
    request->writeBuffer->data + request->writeBuffer->bytesWritten,
    request->writeBuffer->size
    );

  if(len > 0) {
    request->writeBuffer->bytesWritten += len;
    request->writeBuffer->size -= len;
    if(request->writeBuffer->size == 0) {
      request->state = STATE_COMPLETE;
    }
  }
}
```

Замечания

«Дескриптор готов для чтения» — это значит, что в случае неблокирующего режима вызов **read (или recv)** для этого дескриптора реально прочитает из него 1 или больше байт, и вернет число, большее нуля (число прочитанных байт)

«Дескриптор готов для записи» — это значит, что в случае неблокирующего режима вызов **write (или send)** для этого дескриптора реально запишет в него 1 или больше байт, и вернет число, большее нуля (число записанных байт)

Если вызвать **select** для набора дескрипторов, из которых вы реально не готовы читать (или если у вас нет для них данных, которые прямо сейчас надо записывать) — это возвращает вас к реализации активного ожидания (и приводит к деградации пропускной способности)

Вопросы?