

Лекция №2

Многопоточное программирование

Дмитрий Калугин-Балашов



Стивенс У.

UNIX. Разработка сетевых приложений.

W. Richard Stevens.
UNIX Network Programming

Сокеты Беркли



Сокеты Беркли



```
int s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

AF_INET

AF_UNIX

AF_INET
6

Сокеты Беркли



```
int s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

SOCK_STREAM
AM

SOCK_DGRAM
AM

SOCK_RAW

Сокеты Беркли



```
int s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

IPPROTO_T
CP

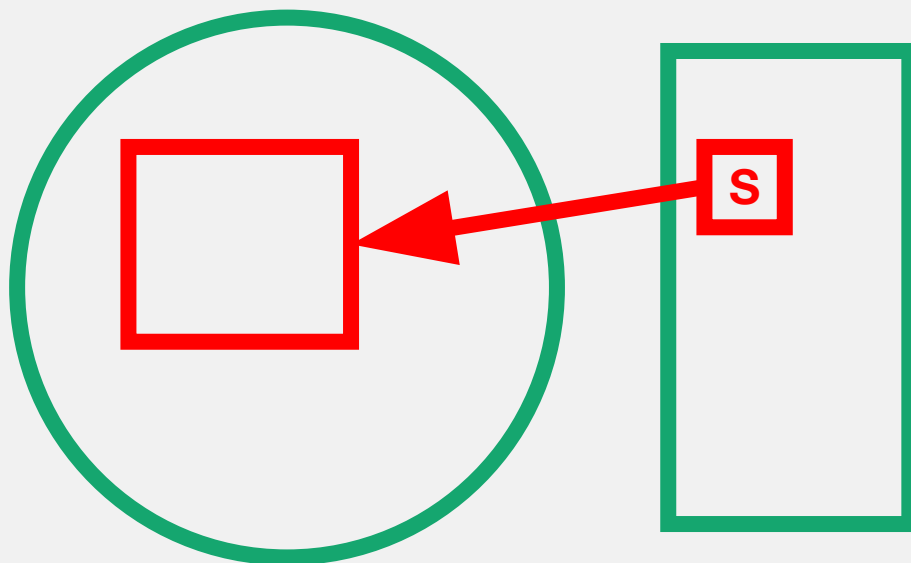
IPPROTO_U
DP

0

Сокеты Беркли



```
int s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```



Сокеты Беркли



Таблица открытых
файлов процесса
FDT

0
1
2
3
4
5
6
7

Таблица
файлов
SFT

счетчик Чтение
1
счетчик Чтение- запись
1
счетчик Запись
1

Таблица описателей
файлов vnode

счетчик (/etc/passwd)
2
счетчик (local)
1

Сокеты Беркли



Таблица открытых
файлов процесса
FD T

(процесс А)

0	
1	
2	
3	
4	
5	
6	
7	

(процесс В)

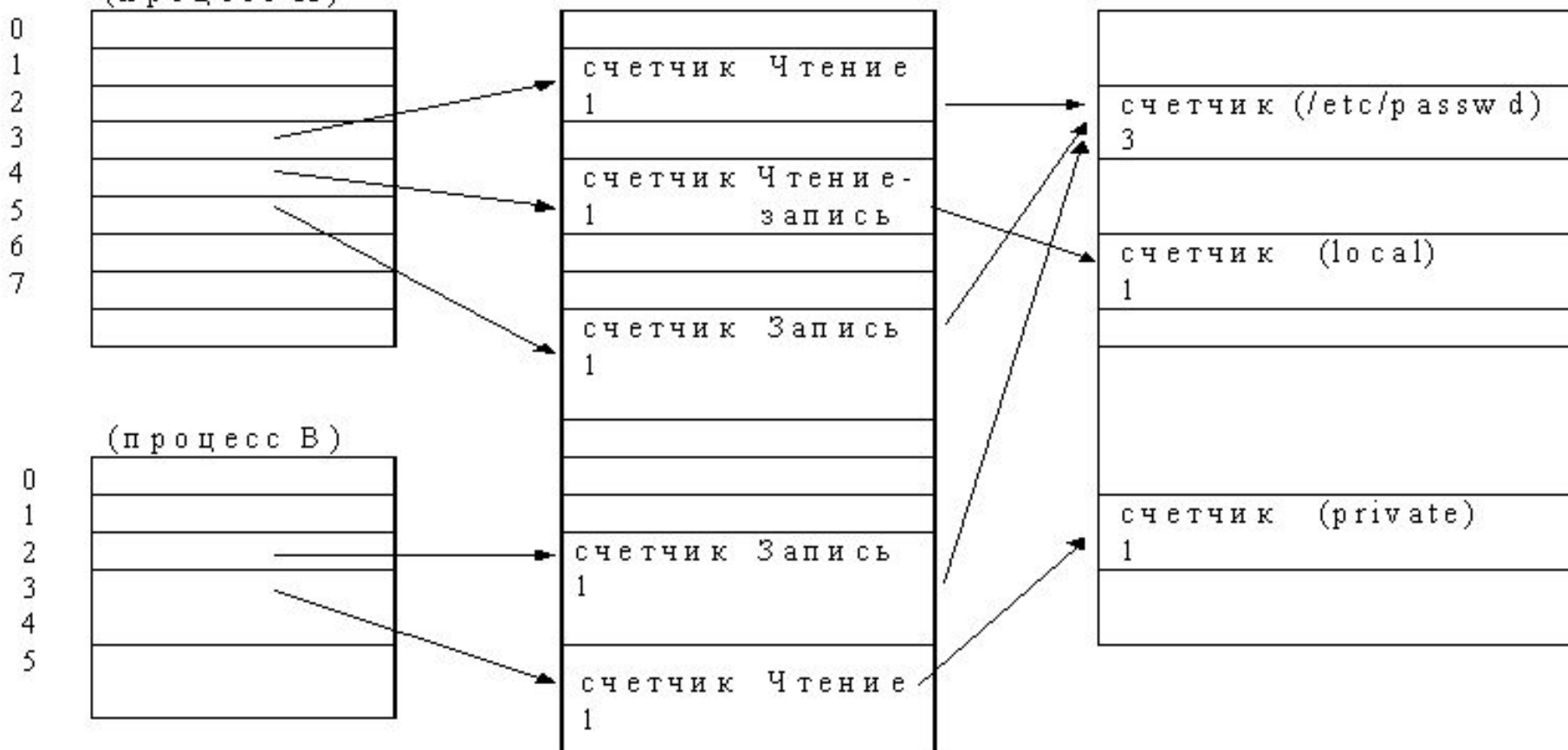
0	
1	
2	
3	
4	
5	

Таблица
файлов
SFT

счетчик Чтение 1
счетчик Чтение- 1 запись
счетчик Запись 1
счетчик Запись 1
счетчик Чтение 1

Таблица описателей
файлов vnode

счетчик (/etc/passwd) 3
счетчик (local) 1
счетчик (private) 1



Сокеты Беркли



Таблицы открытых
файлов процесса
FDT

(процесс А)

0	
1	
2	
3	
4	
5	

(процесс В)

0	
1	
2	
3	
4	

Таблица
файлов SFT

Счетчик 2	Чтение
счетчик 1	Чтение- запись
счетчик 1	Чтение- запись

Таблица
описателей
файлов vnode

Счетчик 2 (/etc/passwd)
Счетчик 2 (private)

Сокеты Беркли



```
bind(s, (struct sockaddr *)sa, sizeof(sa));
```

Сокеты Беркли



`bind(s, (struct sockaddr *)sa, sizeof(sa))`

Сокеты Беркли



`struct sockaddr`

Сокеты Беркли



unsigned short sa_family



struct sockaddr

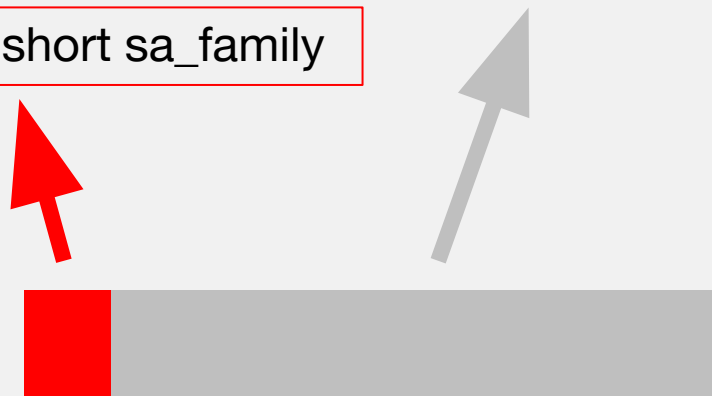
Сокеты Беркли



struct sockaddr

unsigned short sa_family

char sa_data[14]



Сокеты Беркли



struct sockaddr
struct sockaddr_in

unsigned short sa_family

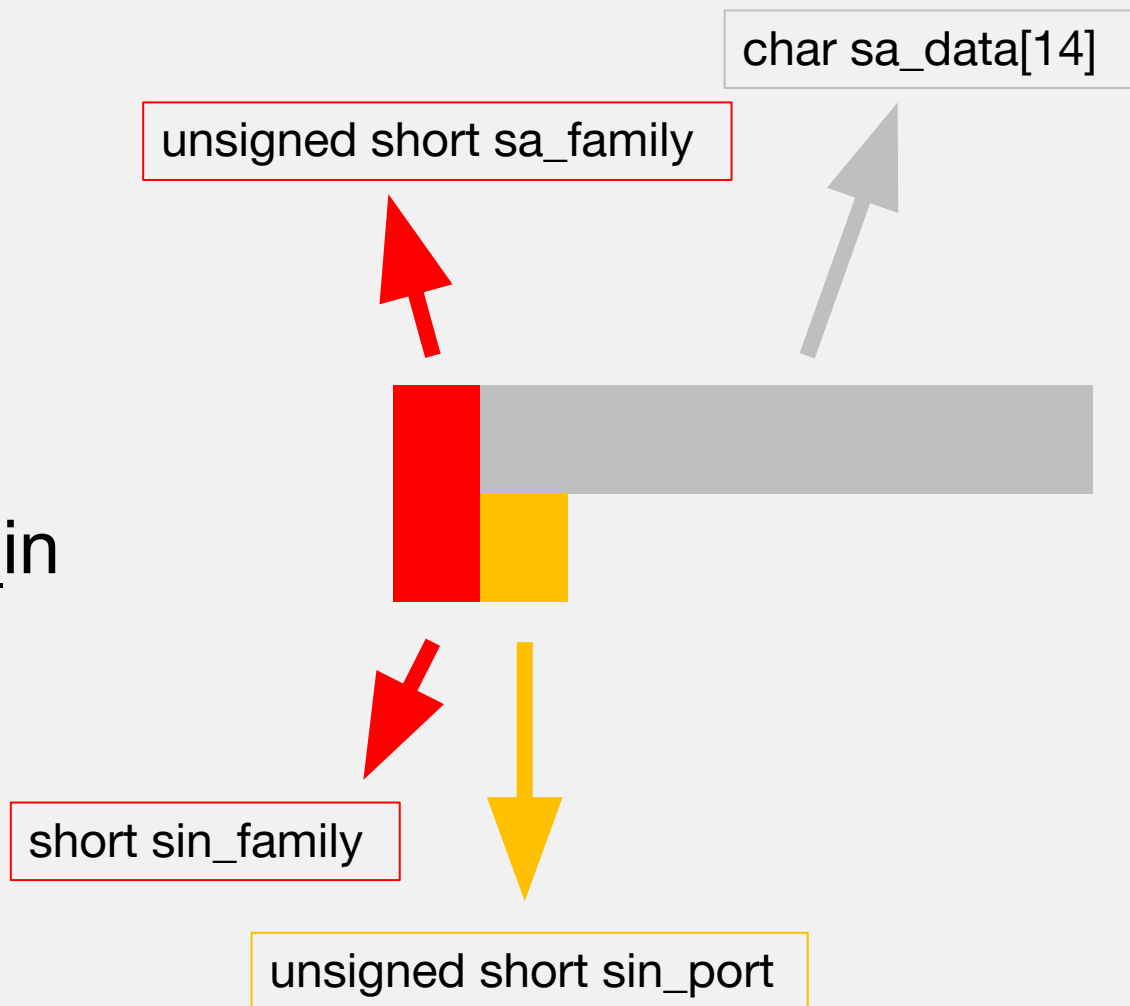
char sa_data[14]

short sin_family

Сокеты Беркли



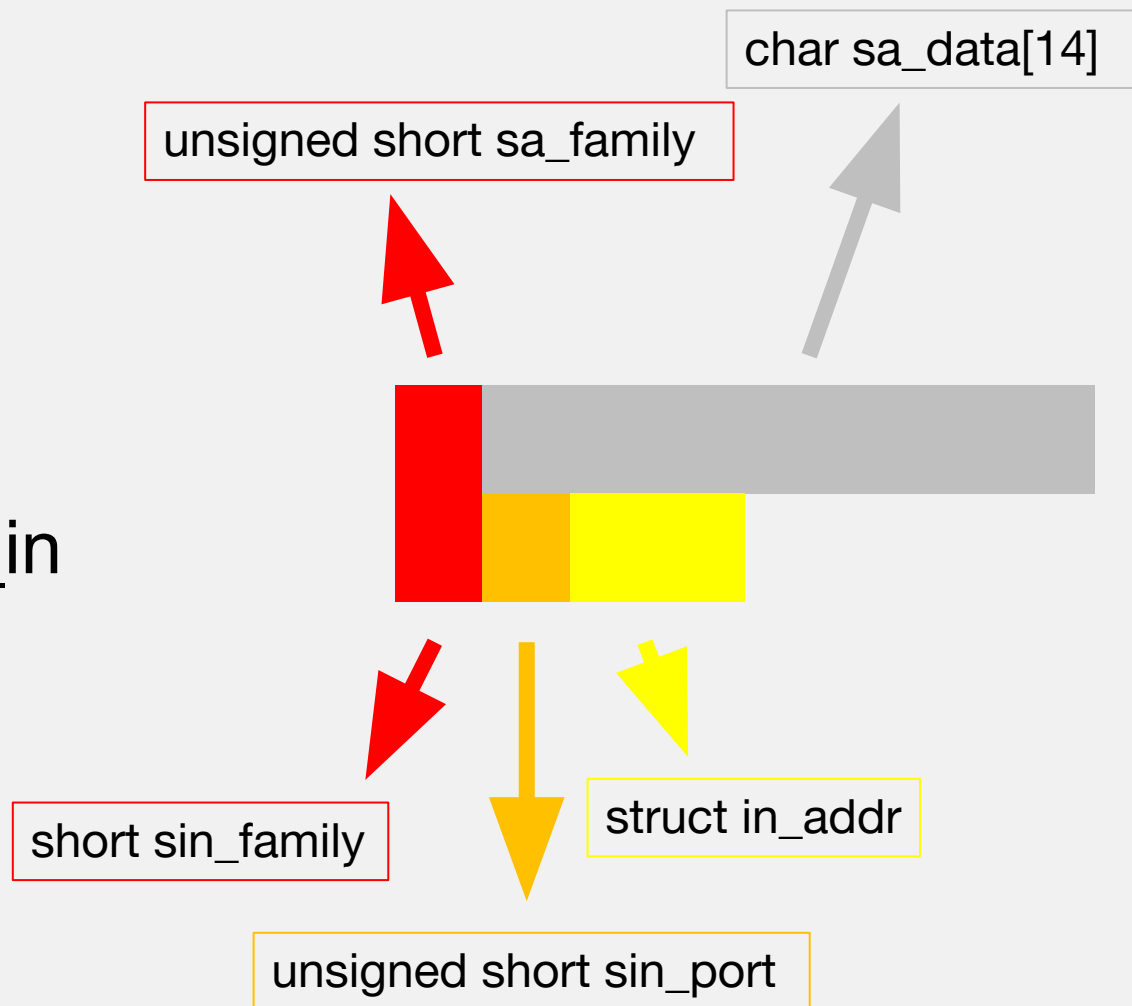
struct sockaddr
struct sockaddr_in



Сокеты Беркли



struct sockaddr
struct sockaddr_in



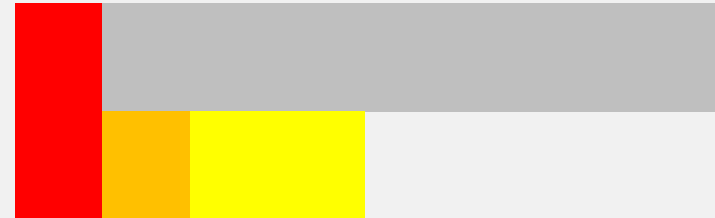
Сокеты Беркли



struct sockaddr
struct sockaddr_in

unsigned short sa_family

char sa_data[14]



short sin_family

struct in_addr

unsigned short sin_port

{ unsigned short s_addr; }

Сокеты Беркли



struct sockaddr
struct sockaddr_in

unsigned short sa_family

char sa_data[14]



char
sin_zero[8]

short sin_family

struct in_addr

unsigned short sin_port

{ unsigned short s_addr; }



Заполнение структуры sockaddr_in

```
1. struct sockaddr_in SockAddr;
```



Заполнение структуры sockaddr_in

1. **struct** sockaddr_in SockAddr;
2. `memset(&SockAddr, 0, sizeof(SockAddr));`



Заполнение структуры sockaddr_in

1. **struct** sockaddr_in SockAddr;
2. `// memset(&SockAddr, 0, sizeof(SockAddr));`
3. `bzero(&SockAddr, sizeof(SockAddr));`



Заполнение структуры sockaddr_in

```
1. struct sockaddr_in SockAddr;  
2. // memset(&SockAddr, 0, sizeof(SockAddr));  
3. bzero(&SockAddr, sizeof(SockAddr));  
4. SockAddr.sin_family = AF_INET;
```




Заполнение структуры sockaddr_in

```
1. struct sockaddr_in SockAddr;  
2. // memset(&SockAddr, 0, sizeof(SockAddr));  
3. bzero(&SockAddr, sizeof(SockAddr));  
4. SockAddr.sin_family = AF_INET;  
5. SockAddr.sin_port = htons(12345);
```



Заполнение структуры sockaddr_in

```
1. struct sockaddr_in SockAddr;  
2. // memset(&SockAddr, 0, sizeof(SockAddr));  
3. bzero(&SockAddr, sizeof(SockAddr));  
4. SockAddr.sin_family = AF_INET;  
5. SockAddr.sin_port = htons(12345);  
6. SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```



Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;  
2.  // memset(&SockAddr, 0, sizeof(SockAddr));  
3.  bzero(&SockAddr, sizeof(SockAddr));  
4.  SockAddr.sin_family = AF_INET;  
5.  SockAddr.sin_port = htons(12345);  
6.  // SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);  
7.  SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
```



Заполнение структуры sockaddr_in

```
1. struct sockaddr_in SockAddr;  
2. // memset(&SockAddr, 0, sizeof(SockAddr));  
3. bzero(&SockAddr, sizeof(SockAddr));  
4. SockAddr.sin_family = AF_INET;  
5. SockAddr.sin_port = htons(12345);  
6. // SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);  
7. // SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);  
8. SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");
```



Заполнение структуры sockaddr_in

```
1.  struct sockaddr_in SockAddr;
2.  // memset(&SockAddr, 0, sizeof(SockAddr));
3.  bzero(&SockAddr, sizeof(SockAddr));
4.  SockAddr.sin_family = AF_INET;
5.  SockAddr.sin_port = htons(12345);
6.  // SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
7.  // SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
8.  // SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");
9.  struct hostent * hp = gethostbyname("rvncerr.org");
10. bcopy(hp->h_addr, &(SockAddr.sin_addr.s_addr), hp->h_length);
```



Заполнение структуры sockaddr_in

```
1.  struct hostent
2.  {
3.      char *h_name;
4.      char **h_aliases;
5.      int h_addrtype;
6.      int h_length;
7.      char **h_addr_list;
8.  }
```



Заполнение структуры sockaddr_in

```
1.  struct hostent
2.  {
3.      char *h_name;
4.      char **h_aliases;
5.      int h_addrtype;
6.      int h_length;
7.      char **h_addr_list;
8.  }
9.  #define h_addr h_addr_list[0]
```

IPv6

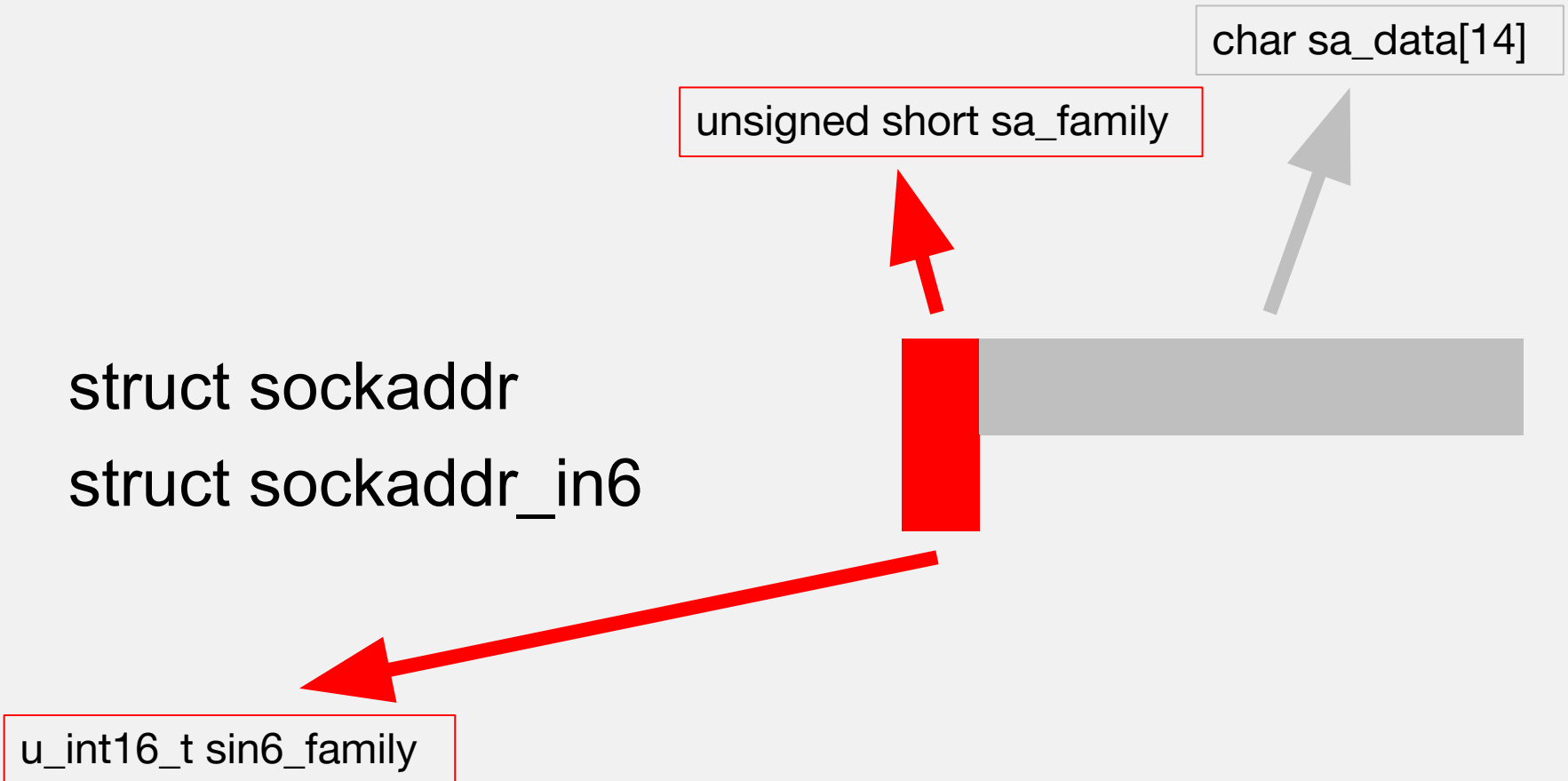


char sa_data[14]

unsigned short sa_family



struct sockaddr
struct sockaddr_in6



IPv6



struct sockaddr
struct sockaddr_in6

unsigned short sa_family

char sa_data[14]

u_int16_t sin6_family

u_int16_t sin6_port



struct sockaddr
struct sockaddr_in6

unsigned short sa_family

char sa_data[14]



u_int16_t sin6_family

u_int32_t sin6_flowinfo

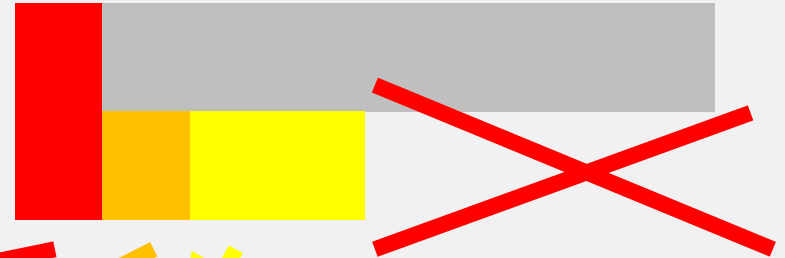
u_int16_t sin6_port



struct sockaddr
struct sockaddr_in6

unsigned short sa_family

char sa_data[14]



u_int16_t sin6_family

u_int32_t sin6_flowinfo

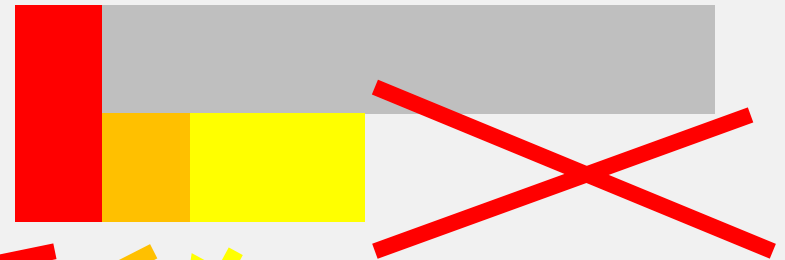
u_int16_t sin6_port



struct sockaddr
struct sockaddr_in6

unsigned short sa_family

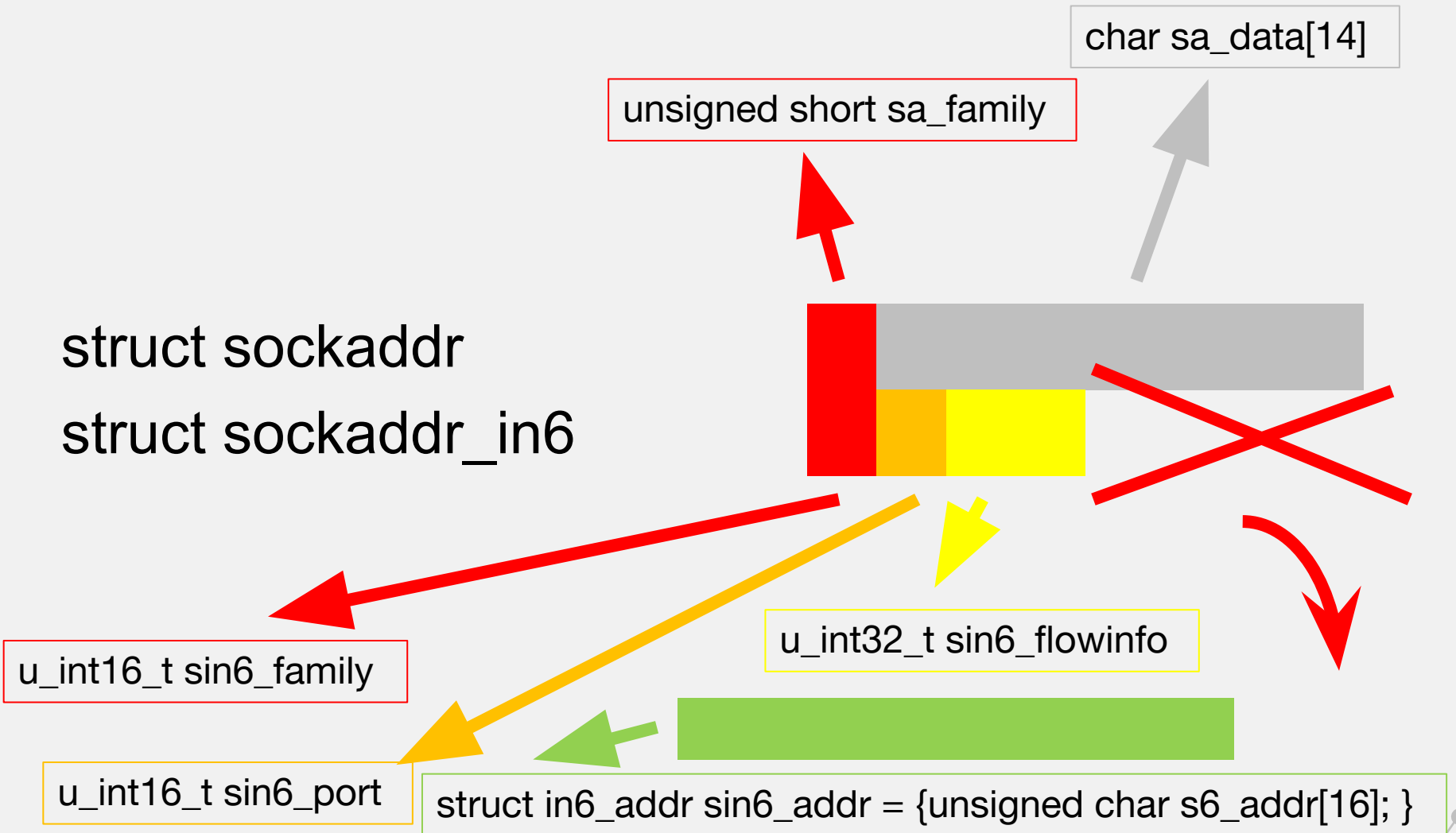
char sa_data[14]



u_int16_t sin6_family

u_int32_t sin6_flowinfo

u_int16_t sin6_port



IPv6

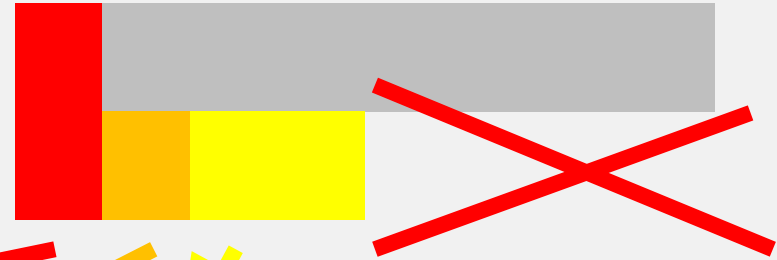


u_int32_t sin6_scope_id

char sa_data[14]

unsigned short sa_family

struct sockaddr
struct sockaddr_in6



u_int16_t sin6_family

u_int32_t sin6_flowinfo

u_int16_t sin6_port

struct in6_addr sin6_addr = {unsigned char s6_addr[16]; }



Заполнение структуры sockaddr_in6

```
1.   SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");
```




Заполнение структуры sockaddr_in6

1. `// SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");`
2. `inet_pton(AF_INET, "178.63.66.215", &(SockAddr.sin_addr));`



Заполнение структуры sockaddr_in6

1. `// SockAddr.sin_addr.s_addr = inet_addr("178.63.66.215");`
2. `inet_pton(AF_INET, "178.63.66.215", &(SockAddr.sin_addr));`
3. `inet_pton(AF_INET6, "2001:db8:8714:3a90::12",
&(SockAddr6.sin6_addr));`



Заполнение структуры sockaddr_un

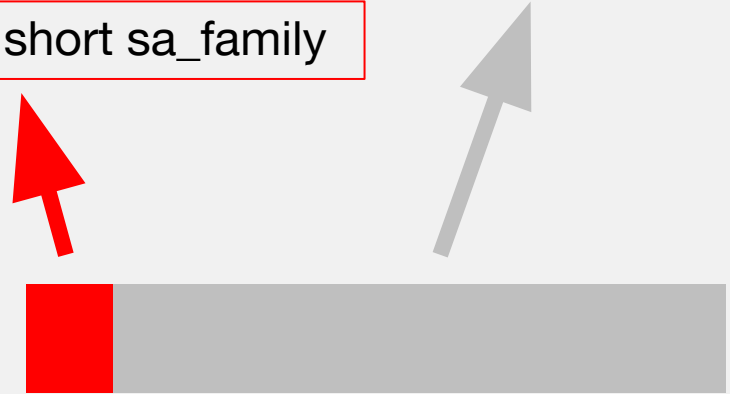
```
1. struct sockaddr_un addr;  
2. memset(&addr, 0, sizeof(addr));  
3. addr.sun_family = AF_UNIX;  
4. strncpy(addr.sun_path, "/tmp/server.sock",  
    sizeof(addr.sun_path)-1);
```



struct sockaddr
struct sockaddr_un

unsigned short sa_family

char sa_data[14]





struct sockaddr
struct sockaddr_un

unsigned short sa_family

char sa_data[14]

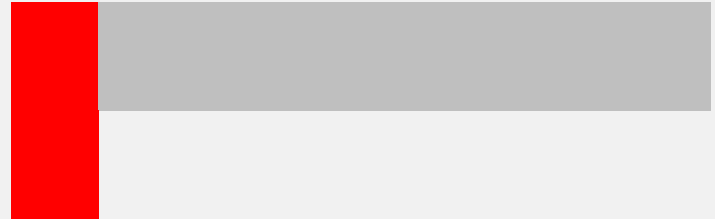
sa_family_t sun_family



struct sockaddr
struct sockaddr_un

unsigned short sa_family

char sa_data[14]



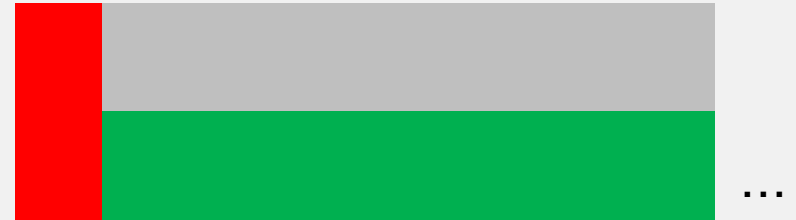
sa_family_t sun_family



struct sockaddr
struct sockaddr_un

unsigned short sa_family

char sa_data[14]



char sun_path[UNIX_PATH_MAX]

sa_family_t sun_family



struct sockaddr
struct sockaddr_un

unsigned short sa_family

char sa_data[14]



char sun_path[108]

sa_family_t sun_family

Сокеты Беркли



```
listen(s, SOMAXCONN /* 128 */);
```

Сокеты Беркли



```
int SlaveSocket = accept(MasterSocket, 0, 0);
```

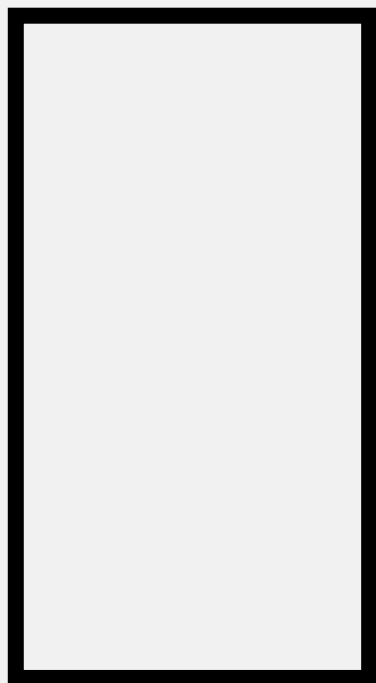
Сокеты Беркли



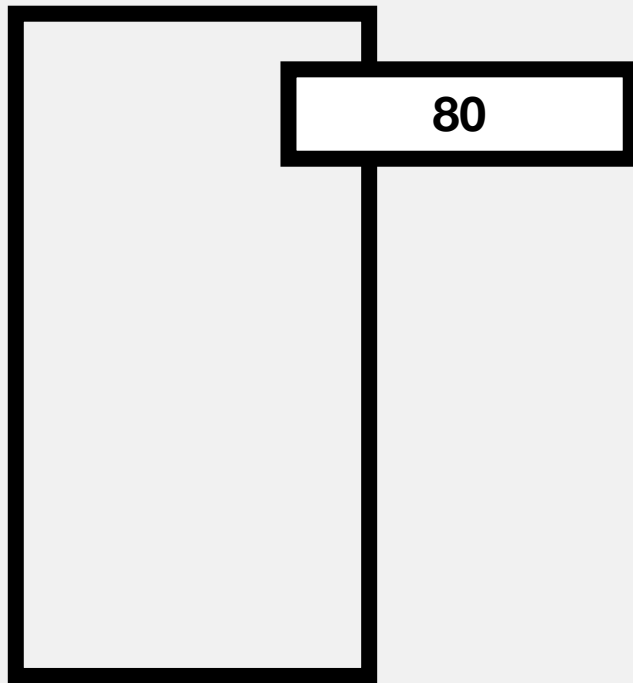
```
int SlaveSocket = accept(MasterSocket, 0, 0);
```

```
int SlaveSocket = accept(MasterSocket,  
    struct sockaddr *addr, socklen_t *addrlen);
```

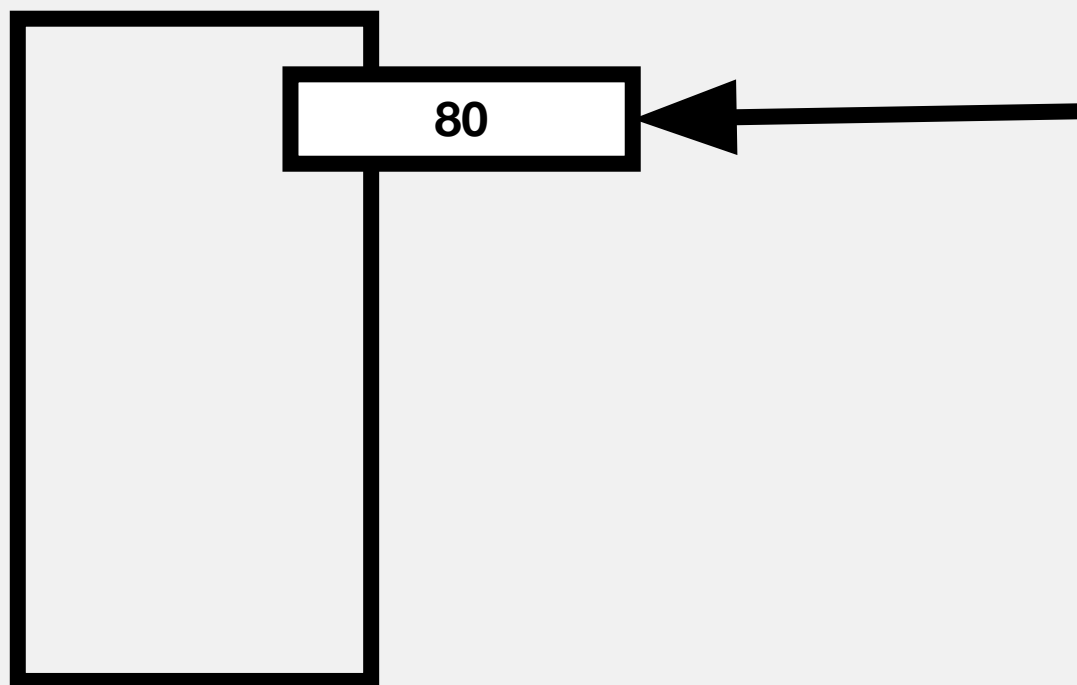
Сокеты Беркли



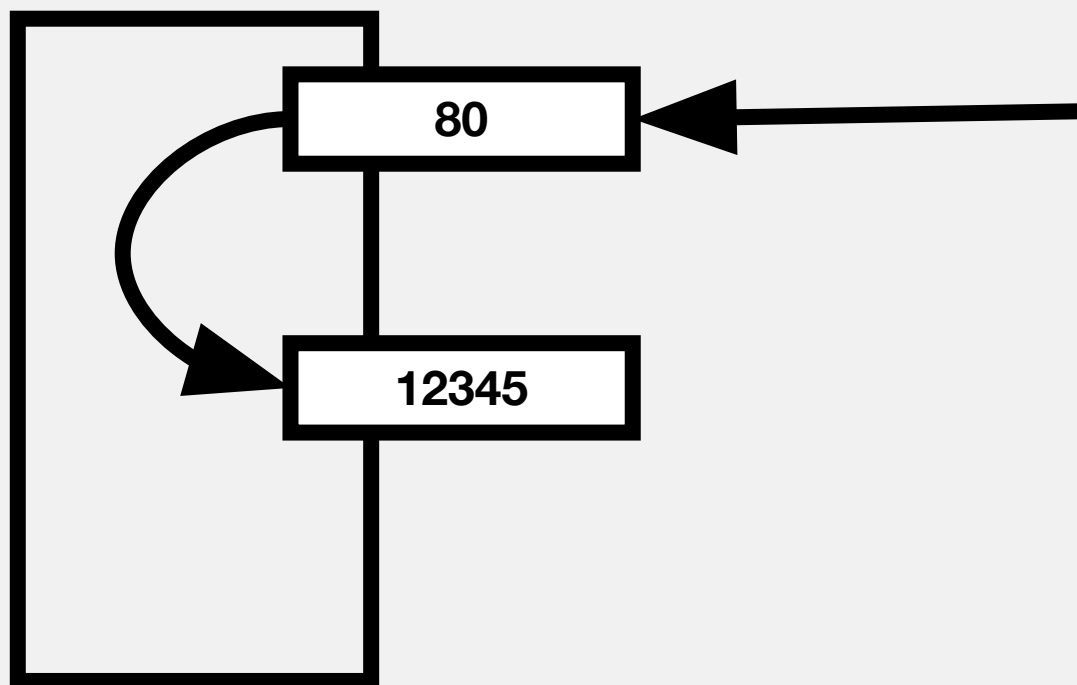
Сокеты Беркли



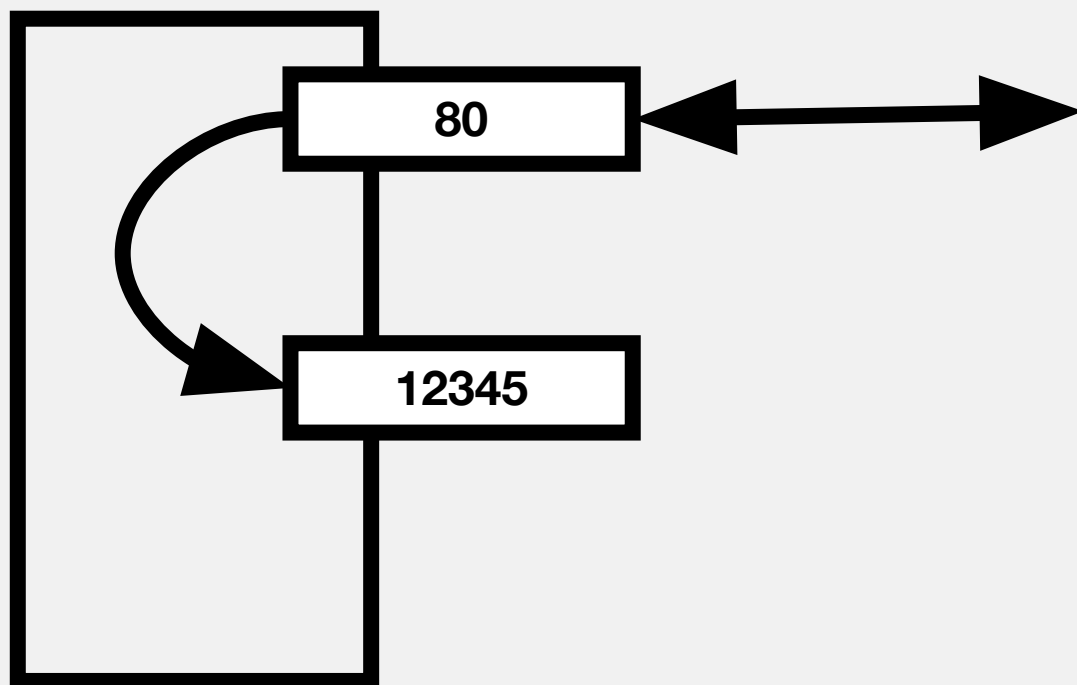
Сокеты Беркли



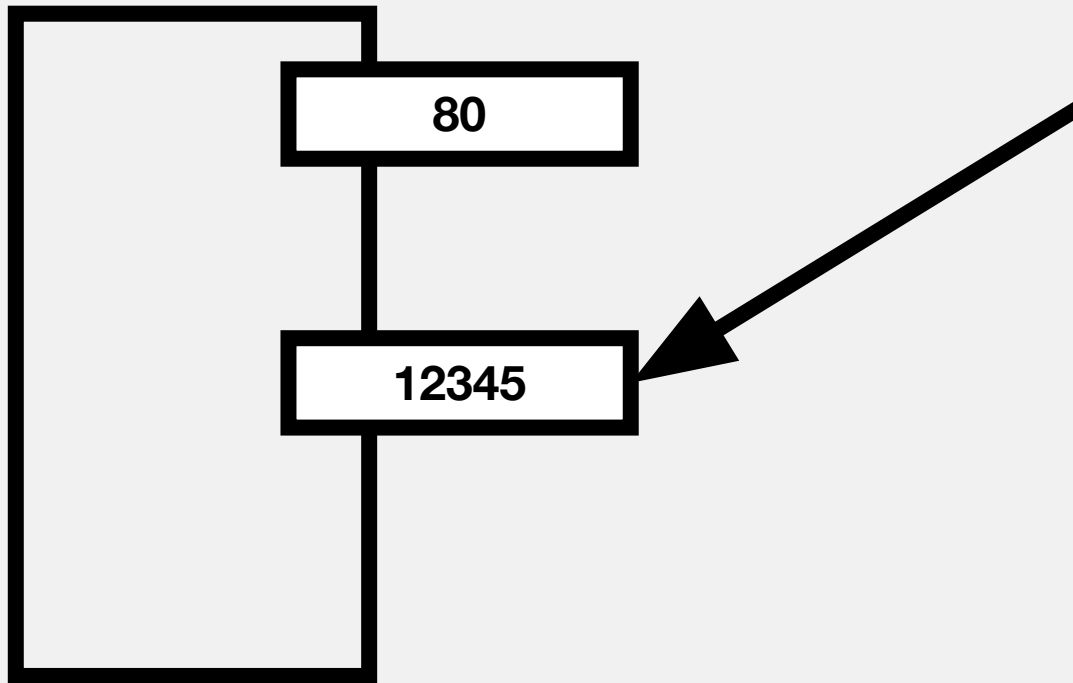
Сокеты Беркли



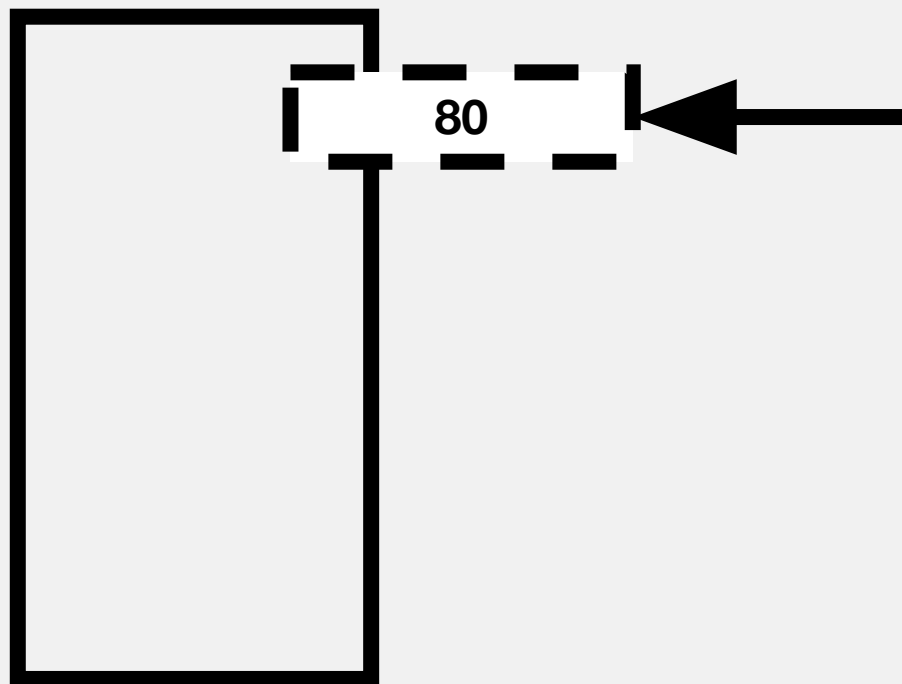
Сокеты Беркли



Сокеты Беркли



Сокеты Беркли





TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```



TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```



TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

6.  bind(MasterSocket, (struct sockaddr *)&SockAddr,
    sizeof(SockAddr));
```



TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

6.  bind(MasterSocket, (struct sockaddr *)&SockAddr,
    sizeof(SockAddr));

7.  listen(MasterSocket, SOMAXCONN);
```



TCP-сервер

```
1.  int MasterSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

6.  bind(MasterSocket, (struct sockaddr *)&SockAddr,
    sizeof(SockAddr));

7.  listen(MasterSocket, SOMAXCONN);

8.  while(true)
9.  {
10.     int SlaveSocket = accept(MasterSocket, 0, 0);
11.     // ...
12. }
```



TCP-клиент

```
1.  int ClientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

2.  struct sockaddr_in SockAddr;
3.  SockAddr.sin_family = AF_INET;
4.  SockAddr.sin_port = htons(12345);
5.  SockAddr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);

6.  connect(ClientSocket, (const void*) &SockAddr, sizeof
    (SockAddr));
```


Сокеты Беркли



```
shutdown(ClientSocket, SHUT_RDWR);  
shutdown(MasterSocket, SHUT_RDWR);
```

SHUT_RDWR

SHUT_RD

SHUT_WR

```
close(ClientSocket);  
close(MasterSocket);
```

Сокеты Беркли



```
ssize_t read(int fd, void *buf, size_t count);  
ssize_t write(int fd, const void *buf, size_t  
count);
```

Сокеты Беркли



~~ssize_t read(int fd, void *buf, size_t count);~~

~~ssize_t write(int fd, const void *buf, size_t count);~~

ssize_t recv(int s, void *buf, size_t len, int flags);

ssize_t send(int s, const void *buf, size_t len, int flags);

Сокеты Беркли



~~ssize_t read(int fd, void *buf, size_t count);~~

~~ssize_t write(int fd, const void *buf, size_t count);~~

ssize_t recv(int s, void *buf, size_t len, int flags);

ssize_t send(int s, const void *buf, size_t len, int flags);

MSG_NOSIGNAL



~~ssize_t read(int fd, void *buf, size_t count);~~

~~ssize_t write(int fd, const void *buf, size_t count);~~

ssize_t recv(int s, void *buf, size_t len, int flags);

ssize_t send(int s, const void *buf, size_t len, int flags);

**signal(SIGPIPE,
SIG_IGN);**

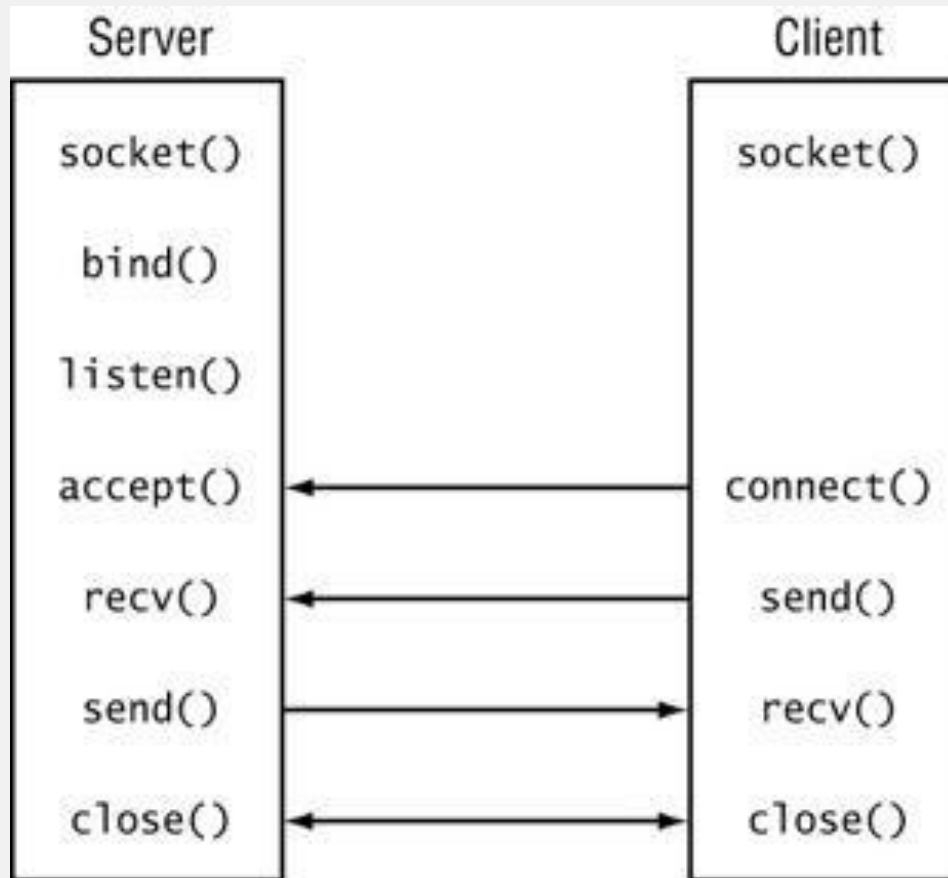
**MSG_NOSIGN
AL**



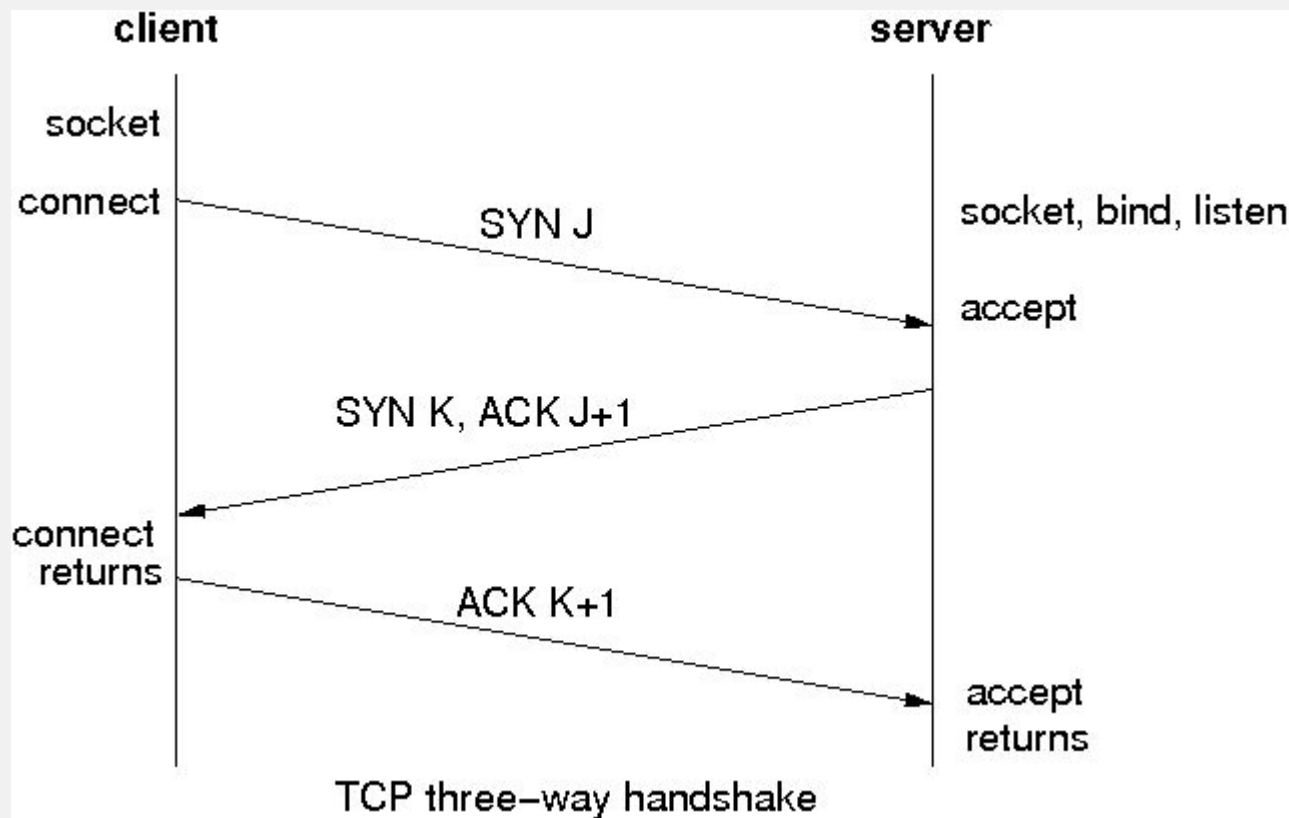
```
ssize_t sendto(int s, const void *buf, size_t len,  
int flags, const struct sockaddr *to, socklen_t  
tolen);
```

```
ssize_t recvfrom(int s, void *buf, size_t len, int  
flags, struct sockaddr *from, socklen_t *fromlen);
```

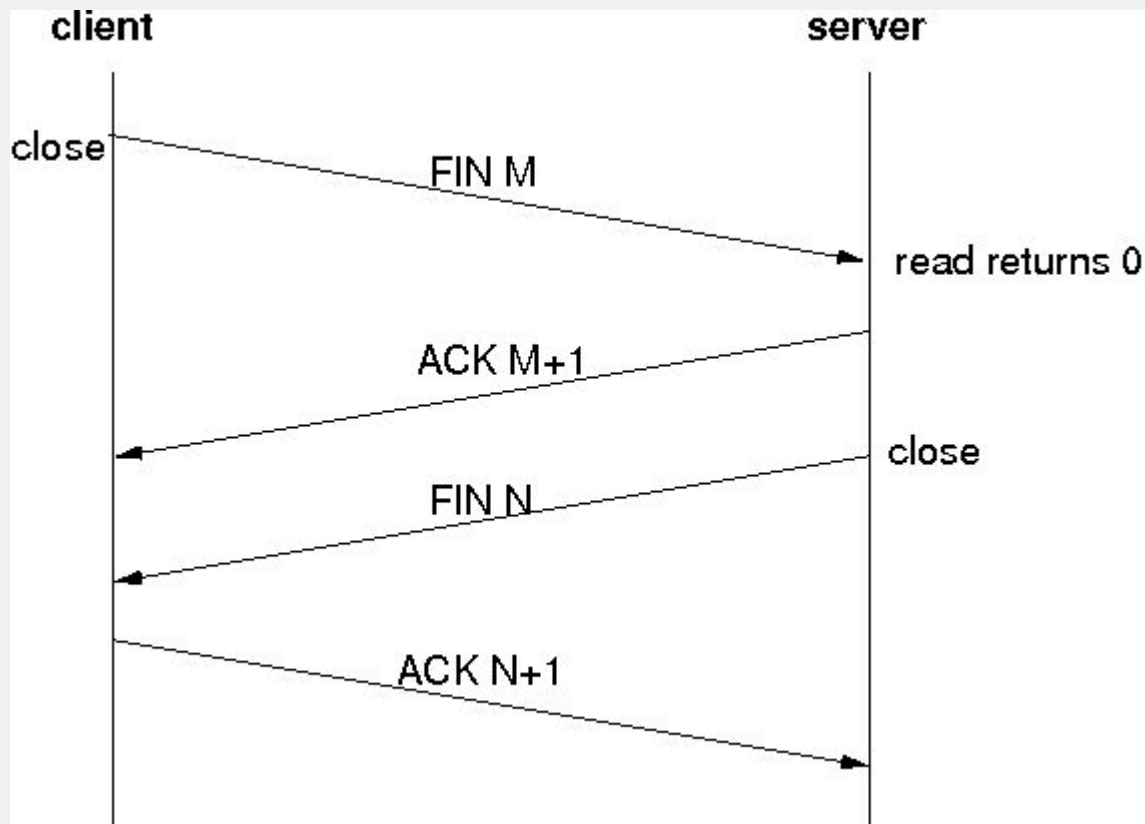
Сокеты Беркли



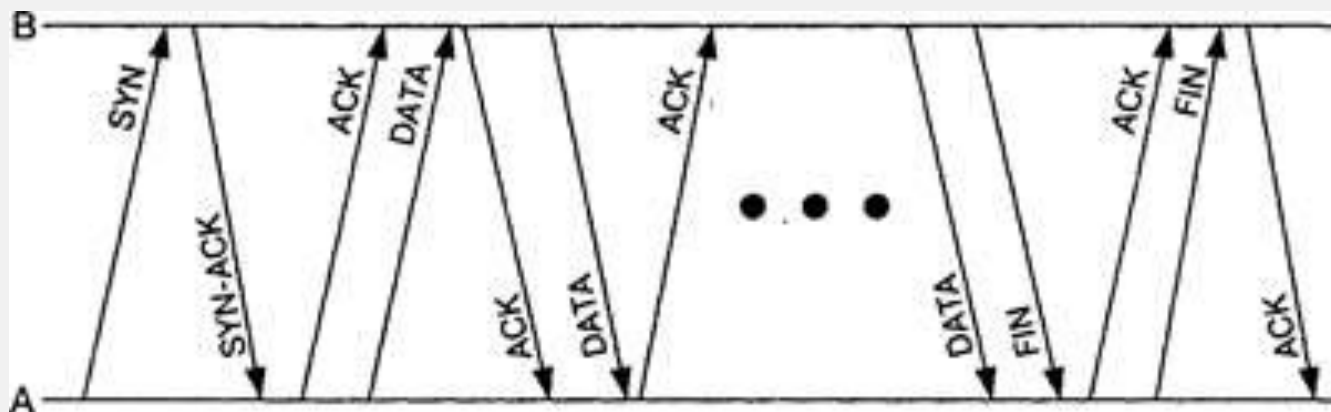
Сокеты Беркли

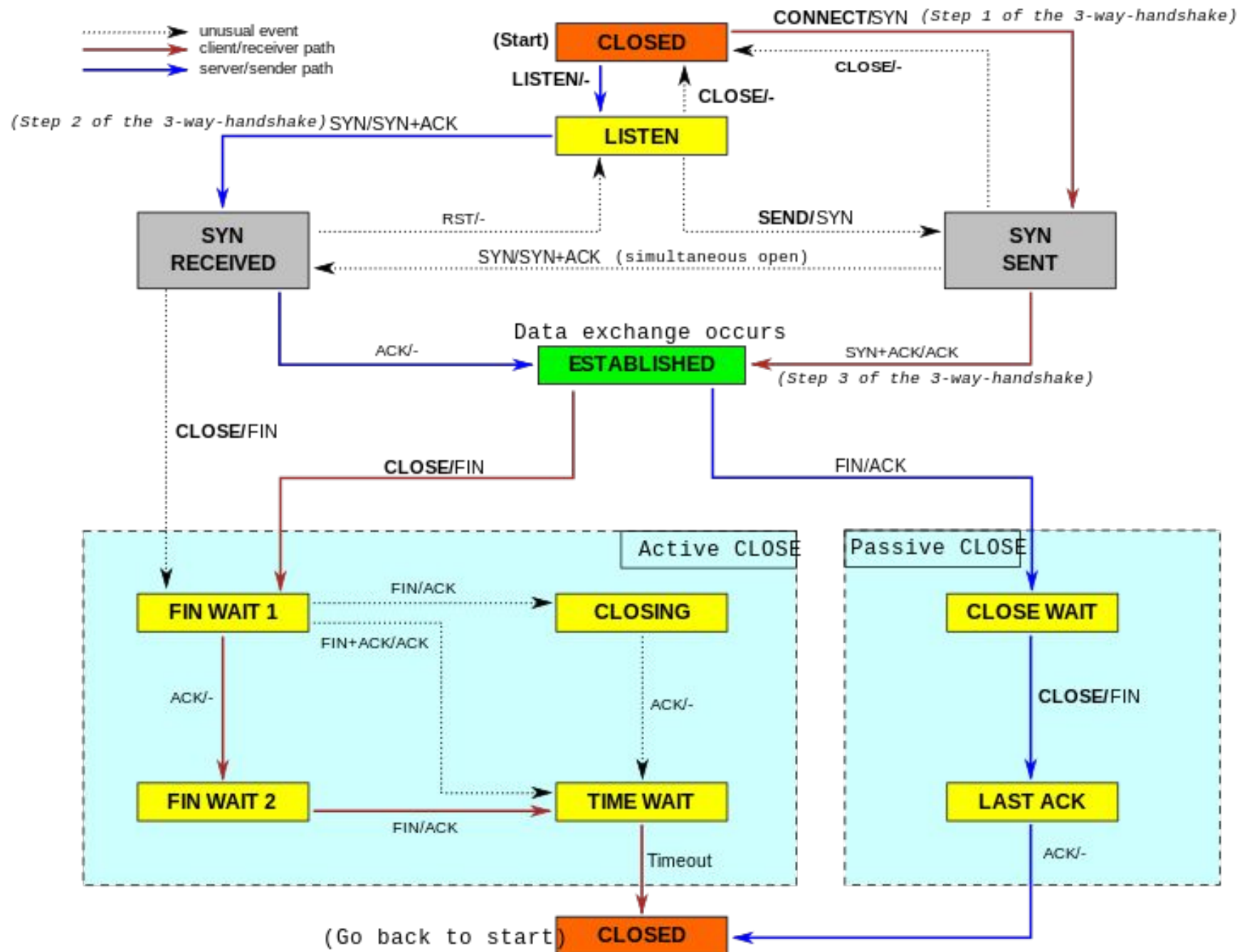


Сокеты Беркли



Сокеты Беркли







Неблокирующий сокет

```
1.  int set_nonblock(int fd)
2.  {
3.      int flags;
4.      #if defined(O_NONBLOCK)
5.      if (-1 == (flags = fcntl(fd, F_GETFL, 0))) flags = 0;
6.      return fcntl(fd, F_SETFL, flags | O_NONBLOCK);
7.      #else
8.      flags = 1;
9.      return ioctl(fd, FIOBIO, &flags);
10.     #endif
11. }
```



Использование setsockopt

```
1.  int optval = 1;
2.  setsockopt(MasterSocket, SOL_SOCKET, SO_REUSEADDR, &optval,
    sizeof(optval));

3.  struct timeval tv;
4.  tv.tv_sec = 16;
5.  tv.tv_usec = 0;
6.  setsockopt(SlaveSocket, SOL_SOCKET, SO_RCVTIMEO, (char*) &tv,
    sizeof(tv));
7.  setsockopt(SlaveSocket, SOL_SOCKET, SO_SNDTIMEO, (char*) &tv,
    sizeof(tv));
```

Мультиплексирование



Мультиплексирование



fd fd fd fd fd fd fd fd fd fd

Мультиплексирование



fd fd **fd fd** fd fd **fd** fd fd fd

Мультиплексирование



fd fd fd fd fd fd fd fd **fd** fd

Мультиплексирование



fd fd fd fd fd fd fd **fd** fd fd



Why?



Why?
CPU!



Работа с select

```
1. fd_set Set;
```



Работа с select

1. `fd_set Set;`
2. `FD_ZERO(&Set);`
3. `FD_SET(MasterSocket, &Set);`



Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }
```



Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  int Max = std::max(MasterSocket,
    *std::max_element(SlaveSockets.begin(), SlaveSockets.end()));
```




Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  int Max = std::max(MasterSocket,
    *std::max_element(SlaveSockets.begin(), SlaveSockets.end()));
9.  select(Max+1, &Set, NULL, NULL, NULL);
```



Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  int Max = std::max(MasterSocket,
    *std::max_element(SlaveSockets.begin(), SlaveSockets.end()));
9.  select(Max+1, &Set, NULL, NULL, NULL);

10. for(auto Iter = SlaveSockets.begin();
11.     Iter != SlaveSockets.end();
12.     Iter++)
13.     if(FD_ISSET(*Iter, &Set)) { /* ... */ }
```



Работа с select

```
1.  fd_set Set;
2.  FD_ZERO(&Set);
3.  FD_SET(MasterSocket, &Set);
4.  for(auto Iter = SlaveSockets.begin(); Iter !=
    SlaveSockets.end(); Iter++)
5.  {
6.      FD_SET(*Iter, &Set);
7.  }

8.  /* ... */

9.  if(FD_ISSET(MasterSocket, &Set))
10. {
11.     int SlaveSocket = accept(MasterSocket, 0, 0);
12.     SlaveSockets.insert(SlaveSocket);
13. }
```



Работа с poll

1. **struct** pollfd Set[POLL_SIZE];
2. Set[0].fd = MasterSocket;
3. Set[0].events = POLLIN;



Работа с poll

```
1.  struct pollfd Set[POLL_SIZE];
2.  Set[0].fd = MasterSocket;
3.  Set[0].events = POLLIN;

4.  unsigned int Index = 1;
5.  for(auto Iter = SlaveSockets.begin(); Iter !=
SlaveSockets.end(); Iter++)
6.  {
7.      Set[Index].fd = *Iter;
8.      Set[Index].events = POLLIN;
9.      Index++;
10. }
11. unsigned int SetSize = 1 + SlaveSockets.size();
```



Работа с poll

```
1.  struct pollfd Set[POLL_SIZE];
2.  Set[0].fd = MasterSocket;
3.  Set[0].events = POLLIN;

4.  unsigned int Index = 1;
5.  for(auto Iter = SlaveSockets.begin(); Iter !=
SlaveSockets.end(); Iter++)
6.  {
7.      Set[Index].fd = *Iter;
8.      Set[Index].events = POLLIN;
9.      Index++;
10. }
11. unsigned int SetSize = 1 + SlaveSockets.size();

12. poll(Set, SetSize, -1);
```



Работа с poll

```
1.  struct pollfd Set[POLL_SIZE];
2.  Set[0].fd = MasterSocket;
3.  Set[0].events = POLLIN;

4.  /* ... */

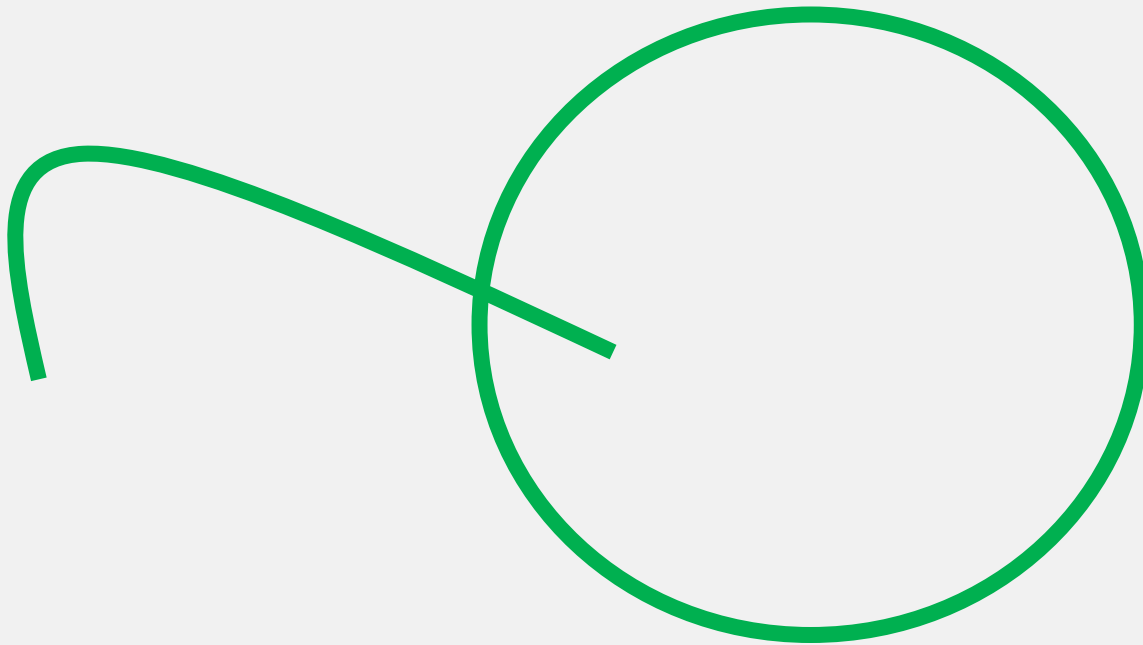
5.  poll(Set, SetSize, -1);

6.  for(unsigned int i = 0; i < SetSize; i++)
7.  {
8.      if(Set[i].revents & POLLIN)
9.      {
10.         /* ... */
11.      }
12. }
```

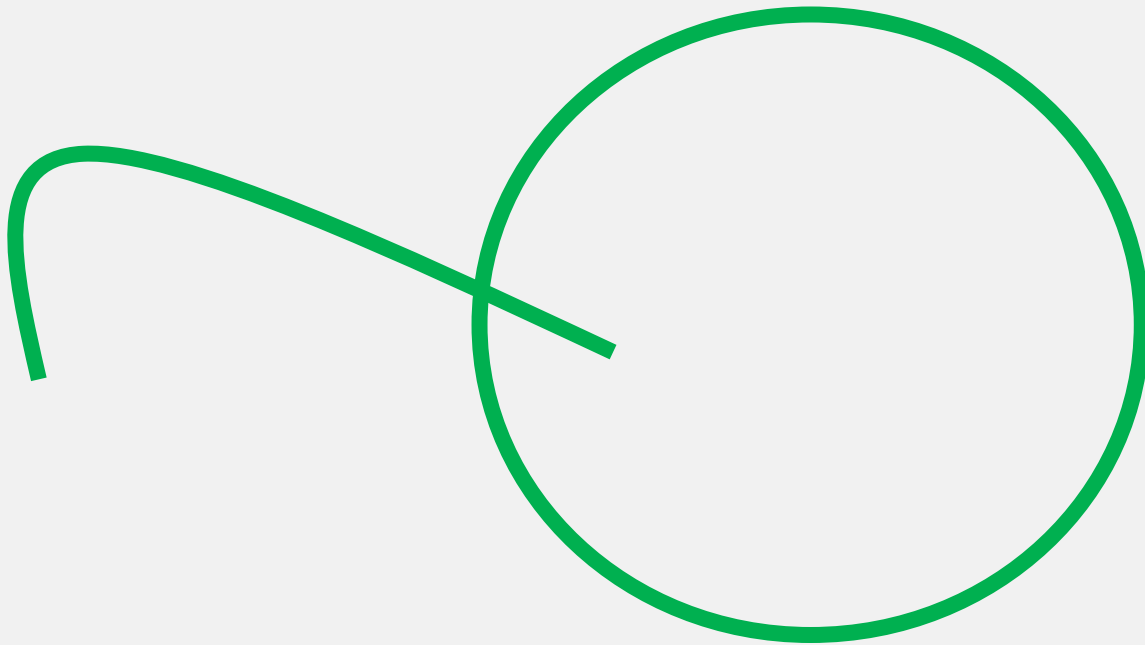
C10K Problem



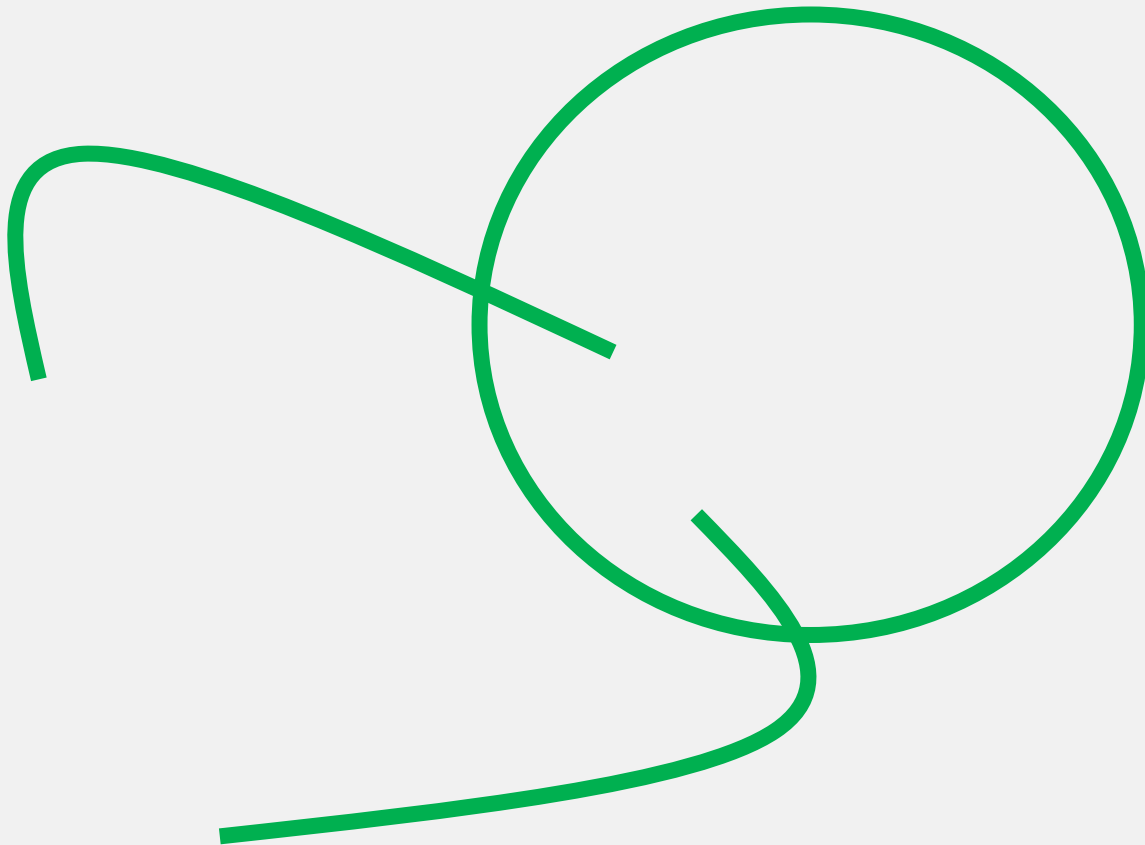
C10K Problem



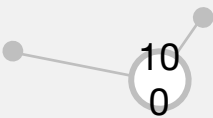
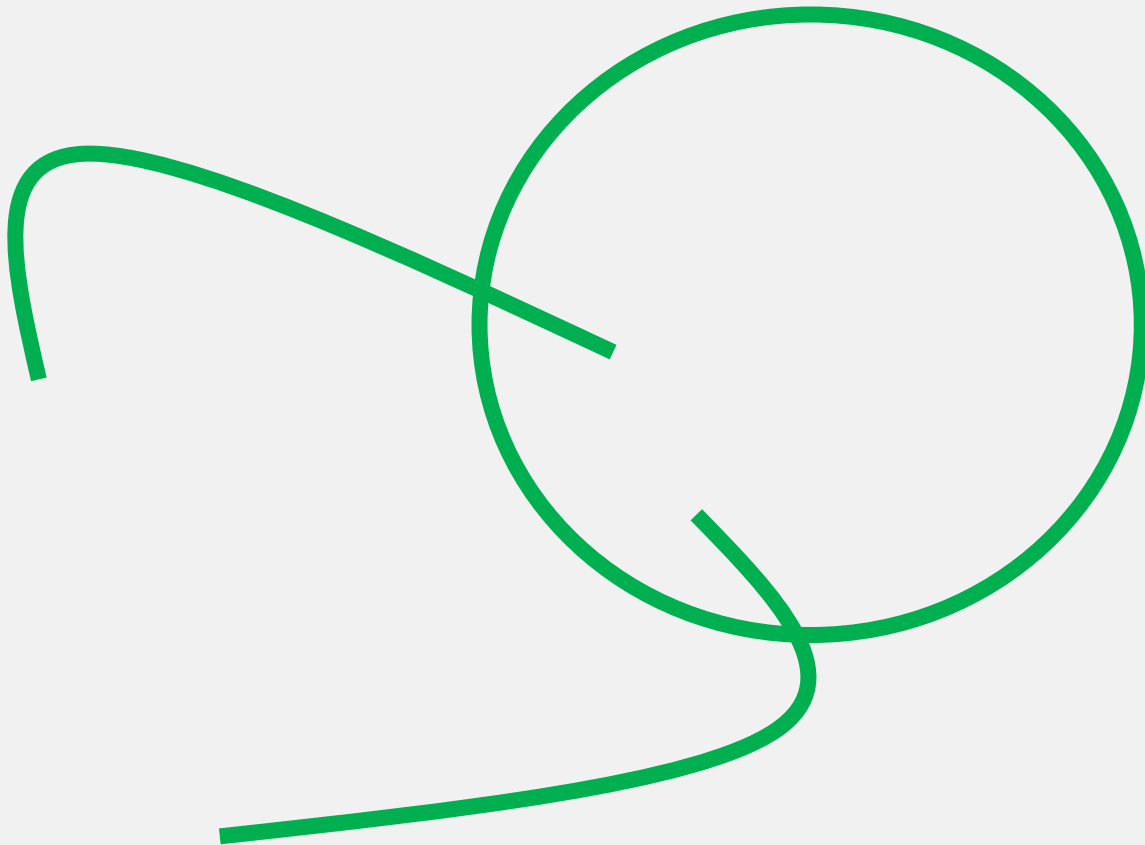
C10K Problem



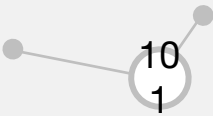
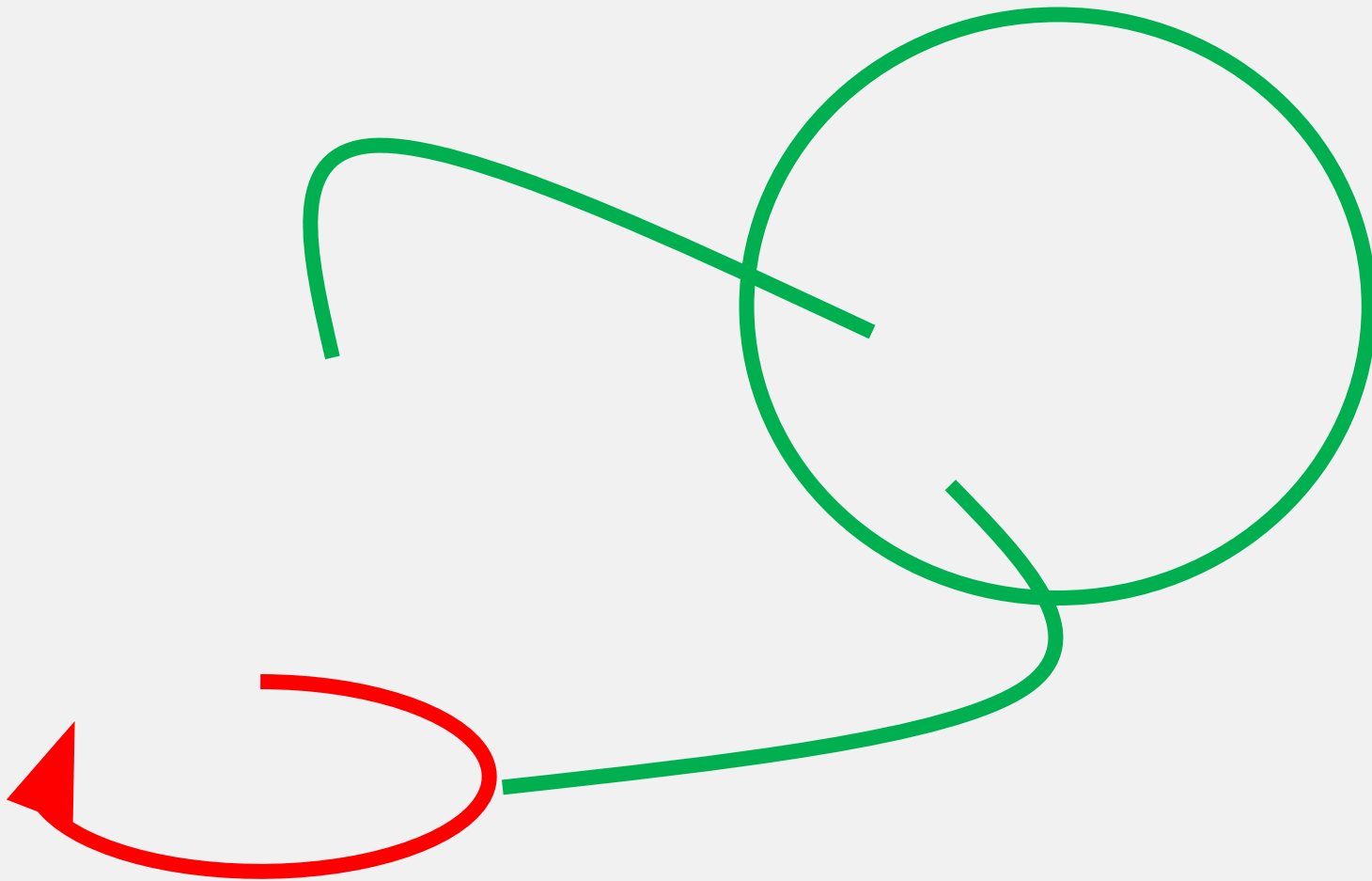
C10K Problem



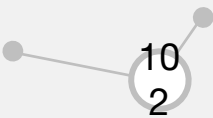
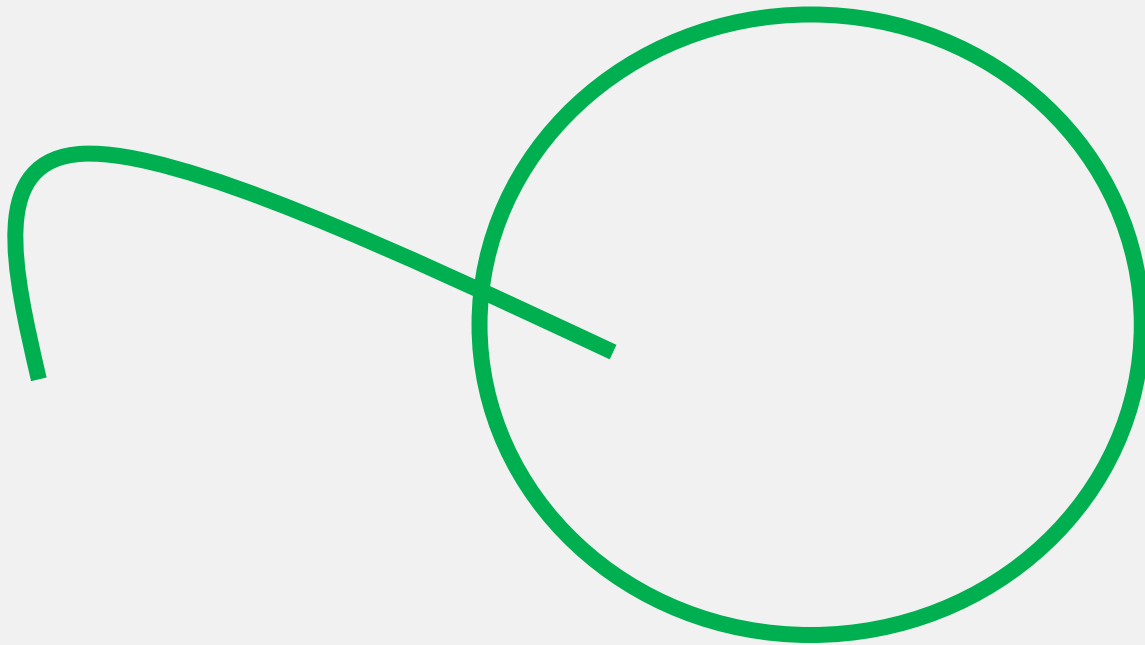
C10K Problem



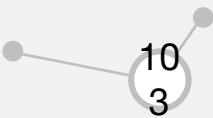
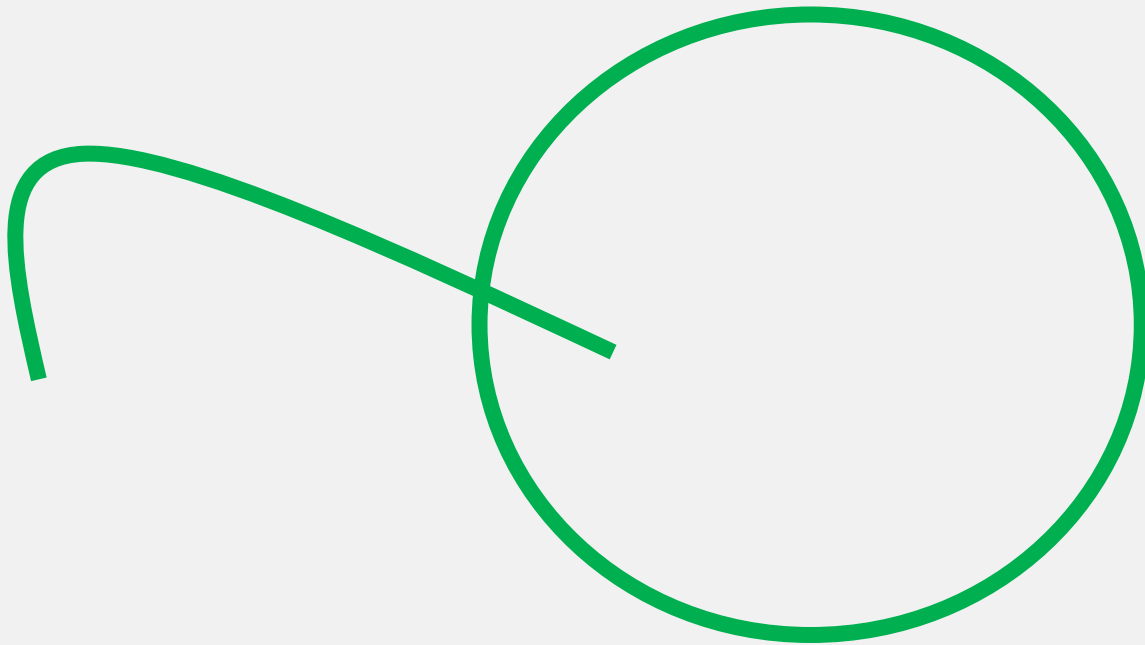
C10K Problem



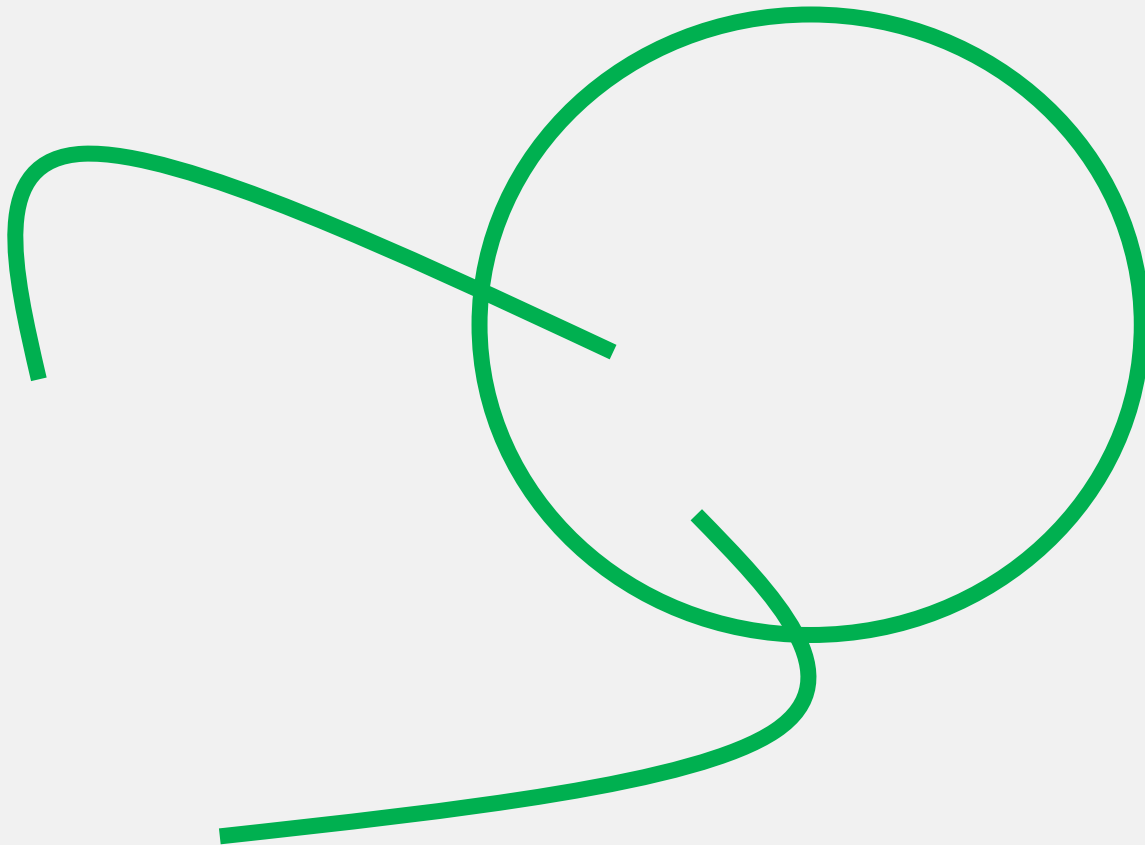
C10K Problem



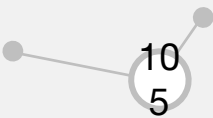
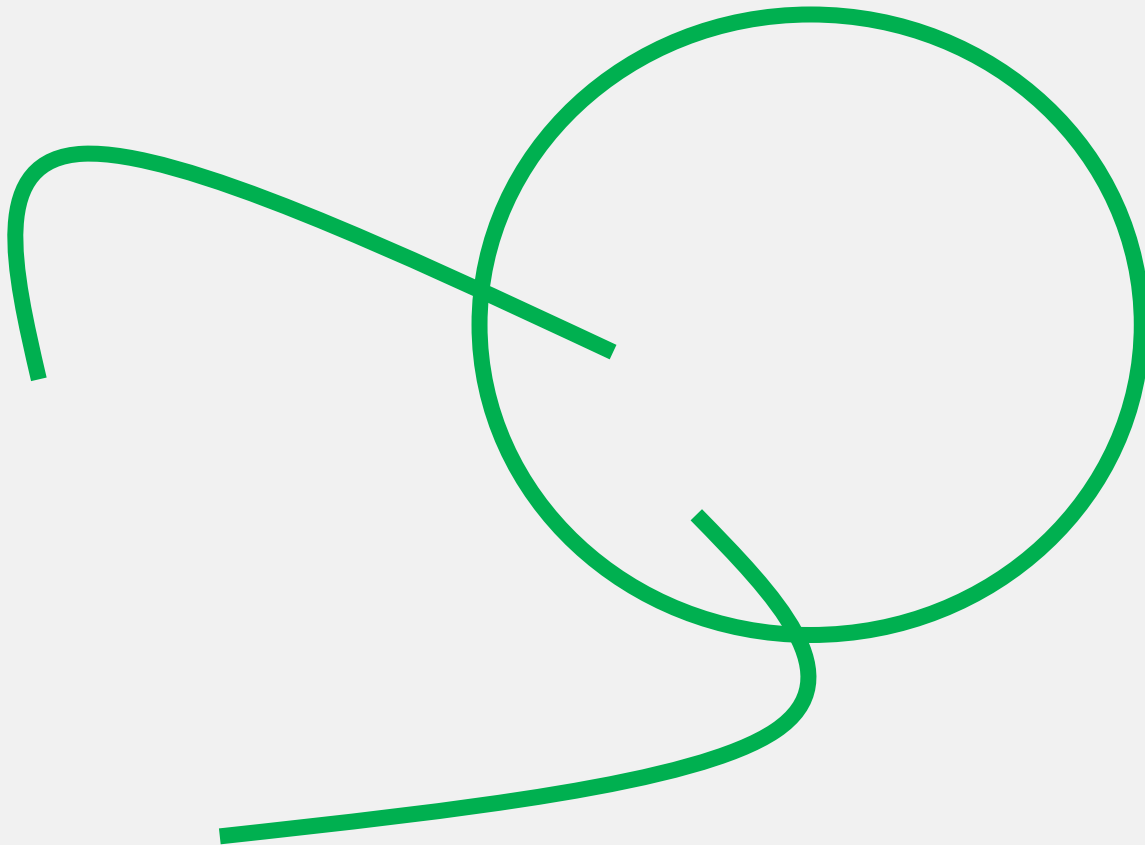
C10K Problem



C10K Problem



C10K Problem





Работа с epoll

```
1.  int EPoll = epoll_create1(0);
```



Работа с epoll

```
1.  int EPoll = epoll_create1(0);  
  
2.  struct epoll_event Event;  
3.  Event.data.fd = MasterSocket;  
4.  Event.events = EPOLLIN | EPOLLET /* edge triggered */;
```



Работа с epoll

```
1.  int EPoll = epoll_create1(0);  
  
2.  struct epoll_event Event;  
3.  Event.data.fd = MasterSocket;  
4.  Event.events = EPOLLIN | EPOLLET; /* edge triggered */  
  
5.  epoll_ctl(EPoll, EPOLL_CTL_ADD, MasterSocket, &Event);
```



Работа с epoll

```
1.  int EPoll = epoll_create1(0);

2.  struct epoll_event Event;
3.  Event.data.fd = MasterSocket;
4.  Event.events = EPOLLIN | EPOLLET; /* edge triggered */

5.  epoll_ctl(EPoll, EPOLL_CTL_ADD, MasterSocket, &Event);

6.  while(true)
7.  {
8.      int N = epoll_wait(EPoll, Events, MAX_EVENTS, -1);
9.      for(unsigned int i = 0; i < N; i++)
10.     {
11.         /* ... */
12.     }
13. }
```



Работа с epoll

```
1.  struct epoll_event * Events;
2.  Events = (struct epoll_event *) calloc(MAX_EVENTS,
      sizeof(struct epoll_event));

3.  /* ... */

4.  while(true)
5.  {
6.      int N = epoll_wait(EPoll, Events, MAX_EVENTS, -1);
7.      for(unsigned int i = 0; i < N; i++)
8.      {
9.          /* ... */
10.     }
11. }
```



Работа с epoll

```
1.  struct epoll_event * Events;
2.  Events = (struct epoll_event *) calloc(MAX_EVENTS,
      sizeof(struct epoll_event));

3.  /* ... */

4.  while(true)
5.  {
6.      int N = epoll_wait(EPoll, Events, MAX_EVENTS, -1);
7.      for(unsigned int i = 0; i < N; i++)
8.      {
9.          if((Events[i].events & EPOLLERR) || (Events[i].events &
      EPOLLHUP))
10.         { /* ... */ }
11.     }
12. }
```



Работа с kqueue

```
1.  int KQueue = kqueue();
```




Работа с kqueue

1. `int KQueue = kqueue();`
2. `struct kevent KEvent;`
3. `bzero(&KEvent, sizeof(KEvent));`
4. `EV_SET(&KEvent, MasterSocket, EVFILT_READ, EV_ADD, 0, 0, 0);`
5. `kevent(KQueue, &KEvent, 1, NULL, 0, NULL);`



Работа с kqueue

```
1.  int KQueue = kqueue();

2.  struct kevent KEvent;
3.  bzero(&KEvent, sizeof(KEvent));
4.  EV_SET(&KEvent, MasterSocket, EVFILT_READ, EV_ADD, 0, 0, 0);
5.  kevent(KQueue, &KEvent, 1, NULL, 0, NULL);

6.  while(true)
7.  {
8.      bzero(&KEvent, sizeof(KEvent));
9.      kevent(KQueue, NULL, 0, &KEvent, 1, NULL);
10.     if(KEvent.filter == EVFILT_READ) { /* ... */ }
11. }
```



Работа с kqueue

```
1.  if(KEvent.ident == MasterSocket)
2.  {
3.      int SlaveSocket = accept(MasterSocket, 0, 0);
4.      bzero(&KEvent, sizeof(KEvent));
5.      EV_SET(&KEvent, SlaveSocket, EVFILT_READ, EV_ADD, 0, 0, 0);
6.      kevent(KQueue, &KEvent, 1, NULL, 0, NULL);
7.  }
8.  else
9.  {
10.     /* ... */
11. }
```

Raw-сокеты



Raw-сокеты



```
int RAWSocket = socket(AF_INET, SOCK_RAW,  
                        IPPROTO_RAW);
```

```
int RAWSocket = socket(AF_INET, SOCK_RAW,  
                        IPPROTO_TCP);
```

```
int tmp = 1;
```

```
setsockopt(sock, 0, IP_HDRINCL, & tmp, sizeof(tmp));
```

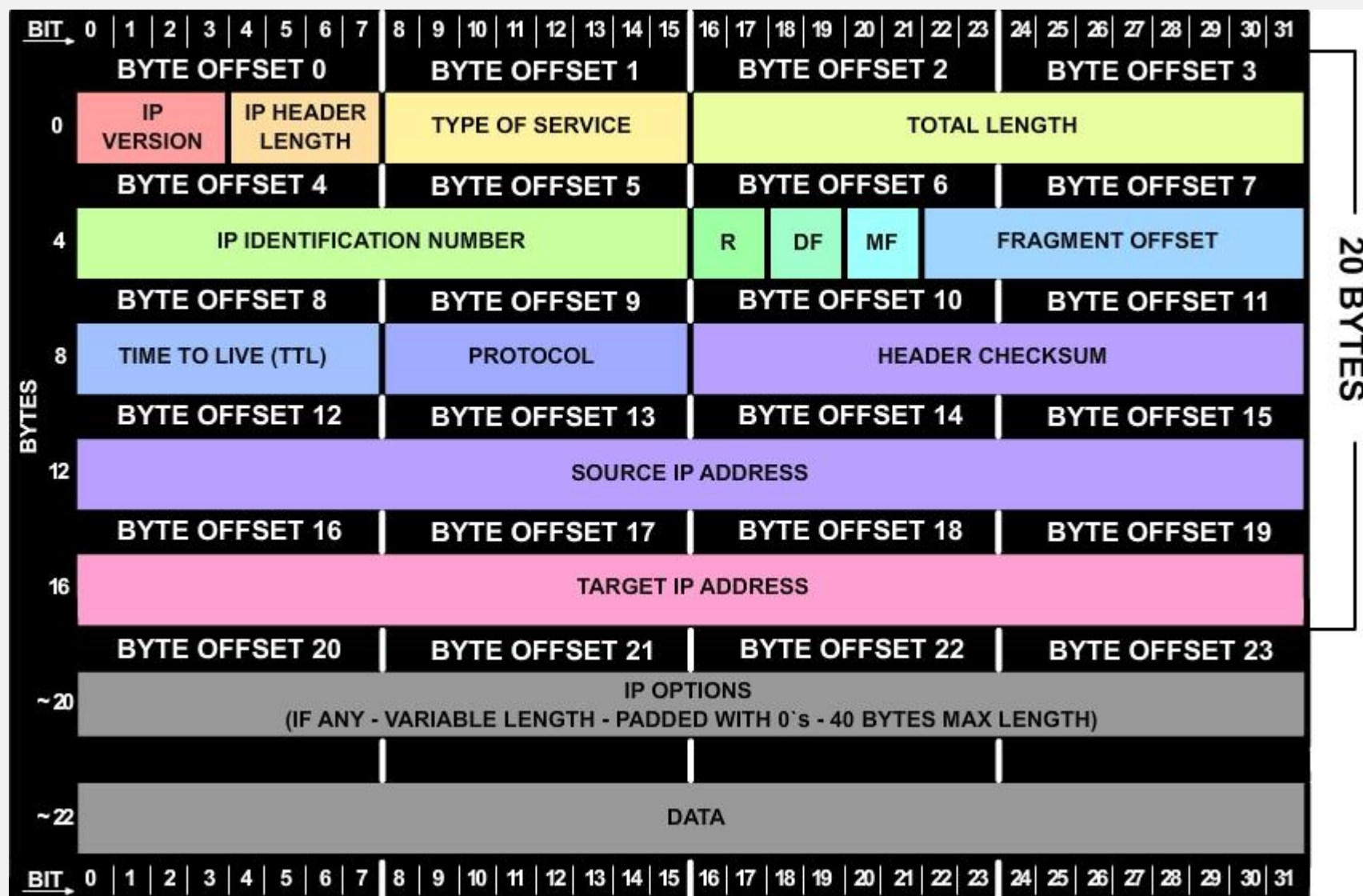
```
int RAWSocket = socket(PF_PACKET, SOCK_RAW,  
                        <protocol>);
```

Raw-сокеты



<http://www.pdbuchan.com/rawsock/rawsock.html>

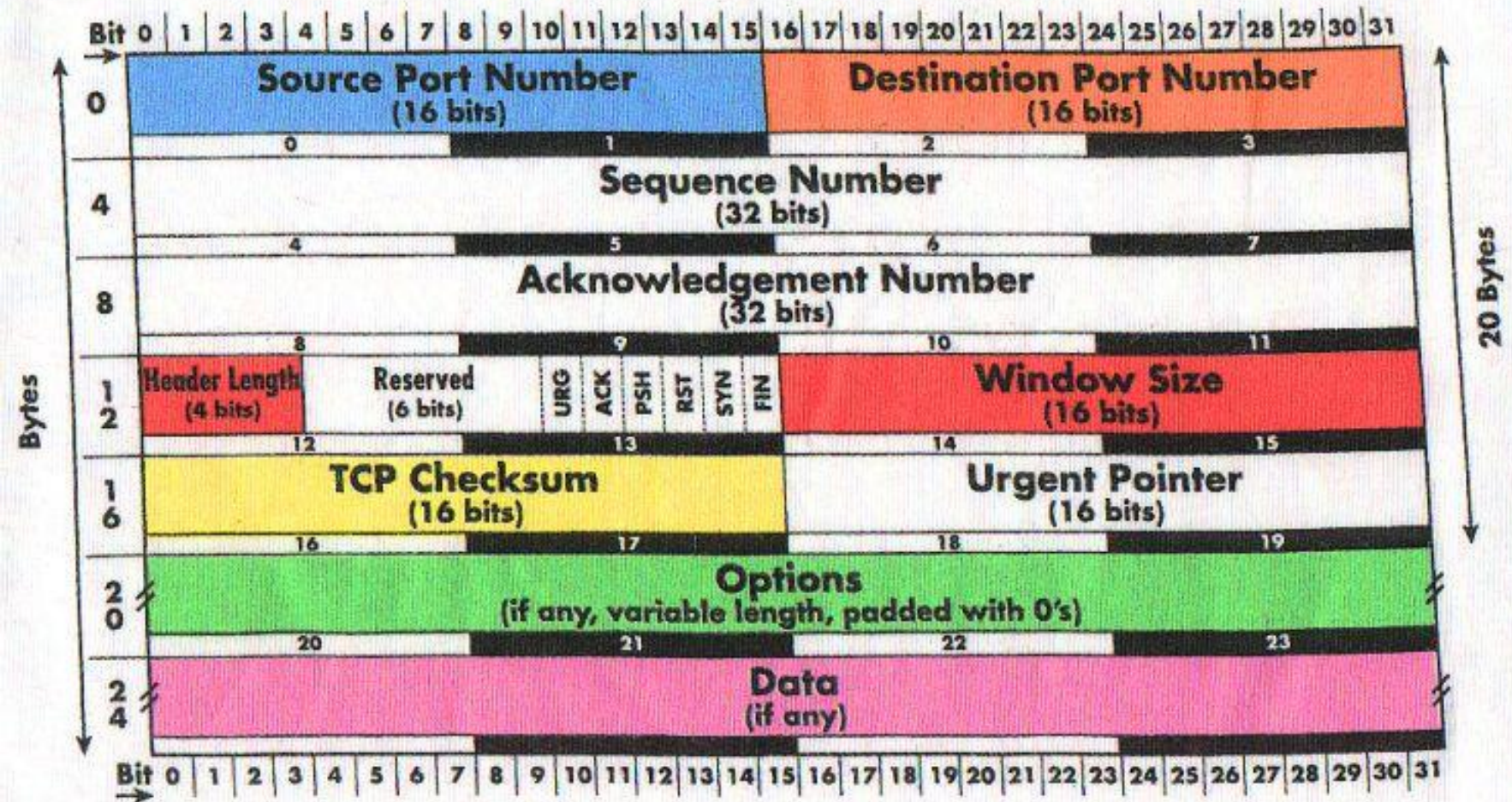
Raw-сокеты





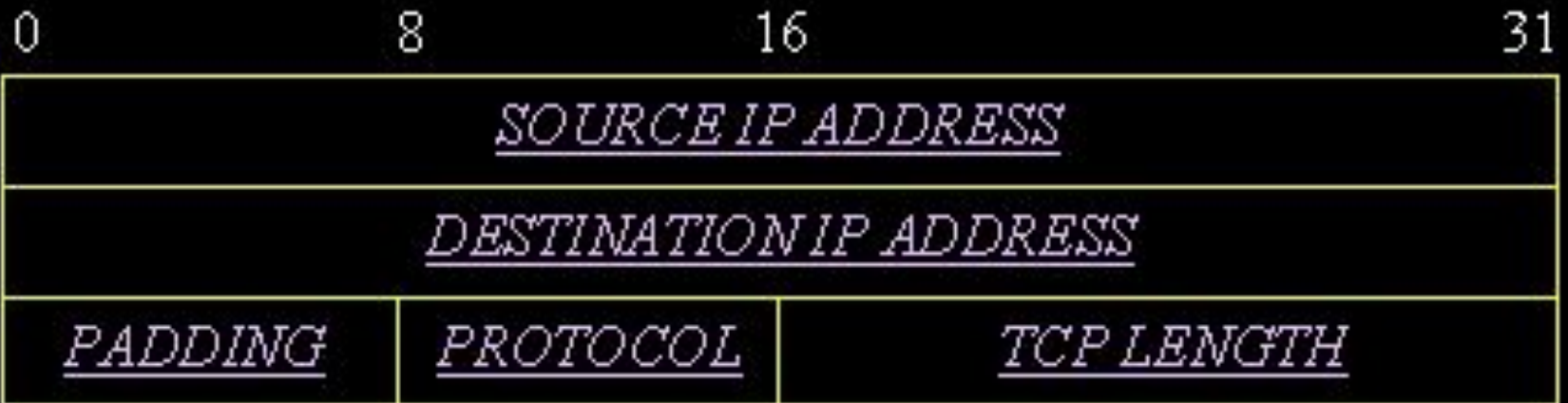
TCP Header

RFC 793 — Transmission Control Protocol





TCP Pseudo-Header:



Домашнее задание №1

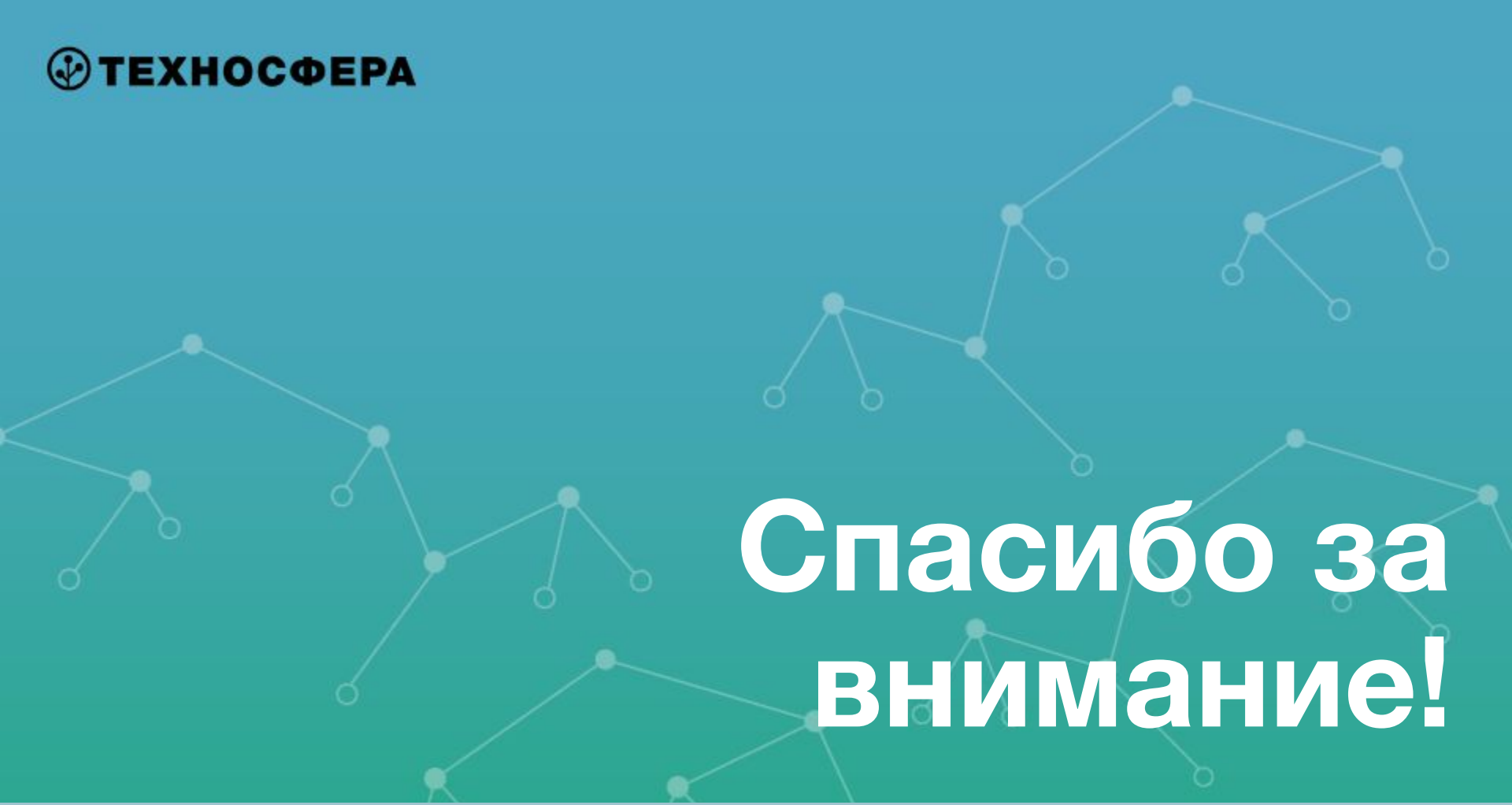


- Создать HTTP-сервер.
- Сборка через make.
- Запуск:

```
./wwwd -d <dir> -h <ip> -p <port>
```
- Реализация HEAD/GET/POST.
- Статусы 200 и 404.
- В каталоге <dir> - html и jpeg файлы.

Срок

сдачи (Коллоквиум)



**Спасибо за
внимание!**

Дмитрий Калугин-Балашов

rvncerr@rvncerr.org