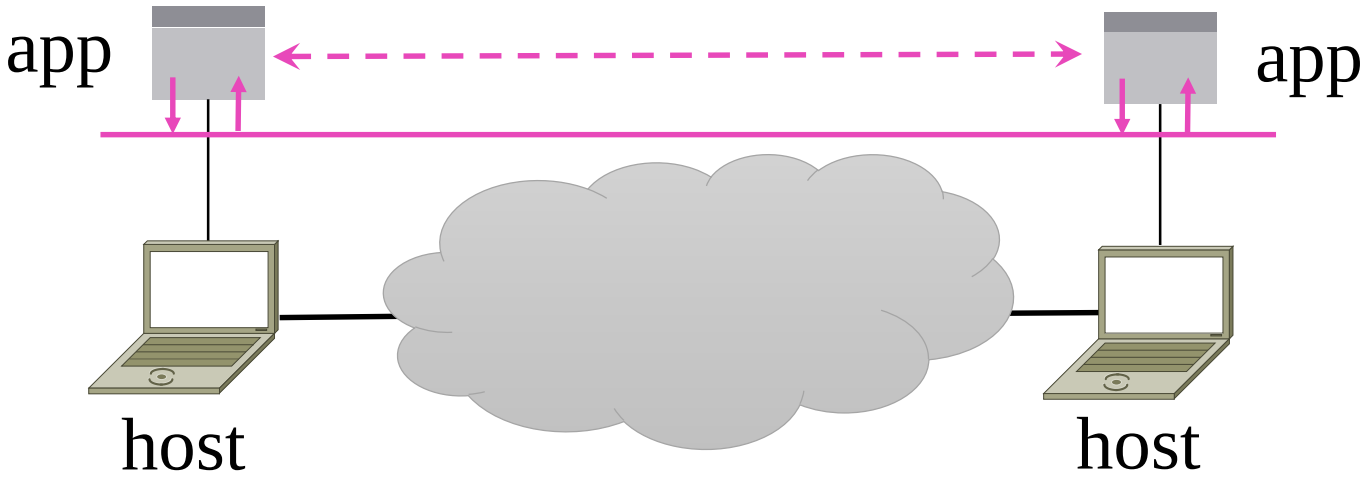

# شبکه های مخابراتی

**سید حمید صفوی**
دانشکده فنی و مهندسی
دانشگاه محقق اردبیلی
نیمسال دوم ۹۸-۹۹

# رابط برنامه کاربردی سوکت
# Socket Application Program Interface (API)

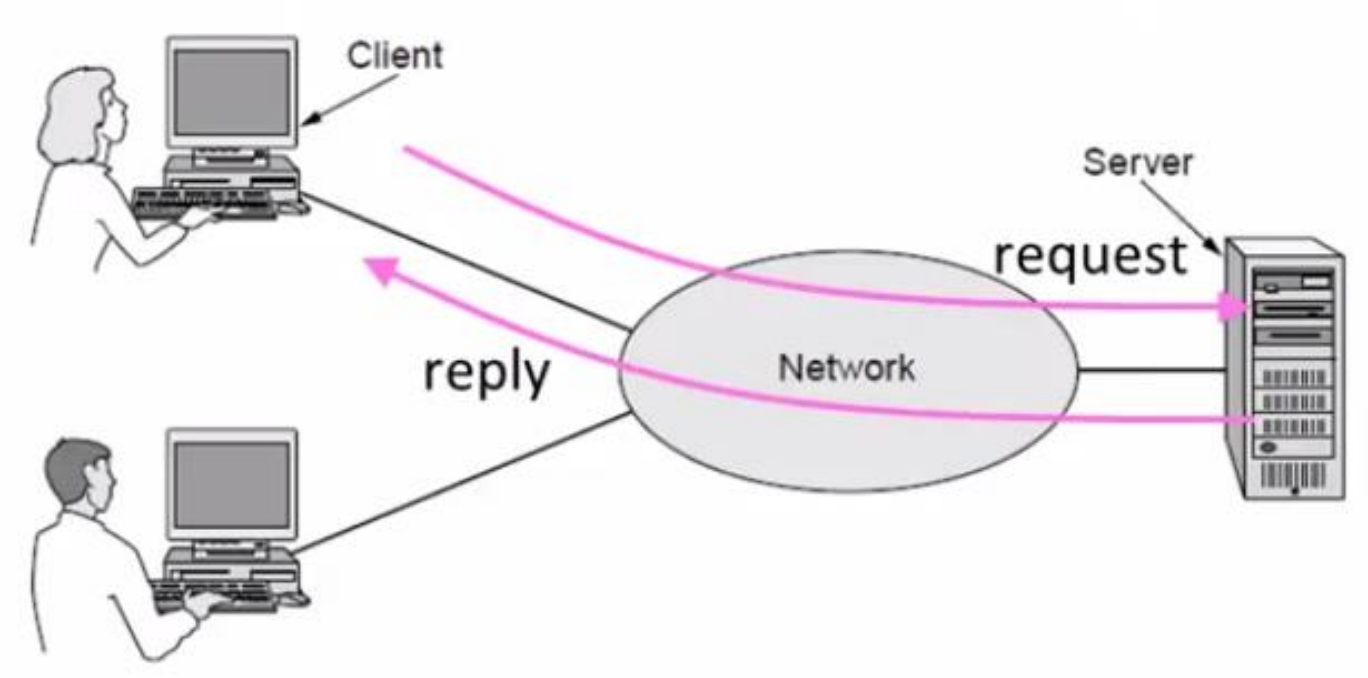
# رابط شبکه–Application

- چگونگی استفاده app ها از شبکه را مشخص می‌کند.
  - برقراری ارتباط بین app ها را از طریق host ها میسر می‌کند؛ جزئیات شبکه را پنهان می‌کند. (برای مثال چه تعداد روتر داخل شبکه است و ...)



# Motivating Application

- یک کلاینت – سرور ساده



# Motivating Application (2)

- یک کلاینت سرور ساده
  - app کلاینت درخواست خود را به app سرور ارسال می‌کند.
  - app سرور پاسخ (طولانی‌تر) را بر می‌گرداند.
- این پایه بسیاری از app هاست.
  - انتقال فایل: نام ارسال می‌شود و فایل دریافت می‌شود.
  - وب‌گردی: لینک URL ارسال می‌شود و صفحه باز می‌شود.
  - Echo: پیام ارسال می‌شود، سپس برگردانده می‌شود. معمولا از این کار برای تست استفاده می‌شود.

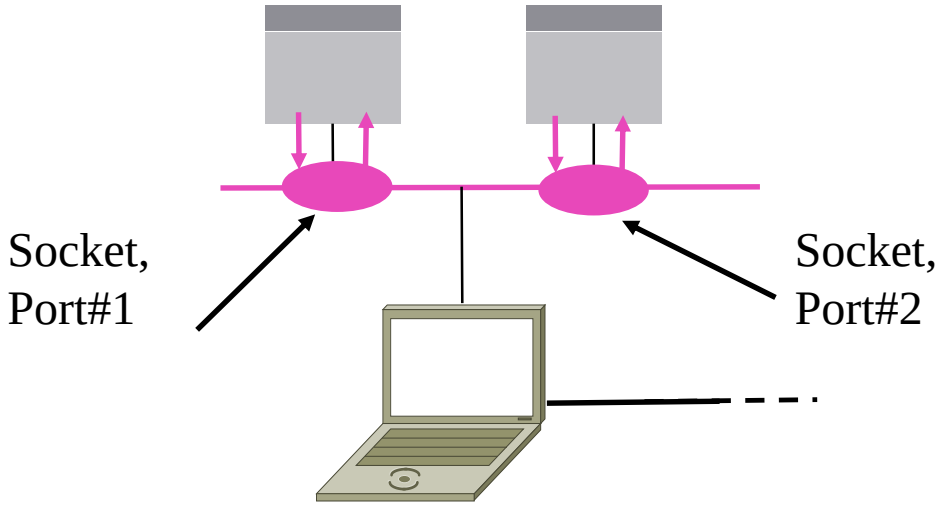
در ادامه می‌بینیم که چطور این app را بنویسیم ...

# Socket API

- یک رابط ساده برای استفاده از شبکه
  - سرویس شبکه API برای نوشتن همه برنامه‌های اینترنت به کار می‌رود.
  - بخشی از همه سیستم‌عامل‌ها و زبان‌های برنامه‌نویسی مشهور؛ برای مثال در ابتدا Unix دانشگاه برکلی در سال ۱۹۸۳
- از دو نوع **سرویس شبکه** پشتیبانی می‌کند:
  - **Streams:** app ها را قادر می‌کند تا رشته‌ای از بایت‌ها را با قابلیت اطمینان به app دیگر ارسال می‌کند.
  - **Datagrams:** پیام‌های جداگانه‌ای را به صورت غیر قابل اطمینان ارسال می‌کند. (فعلا بحث این درس نیست.)

# Socket API (2)

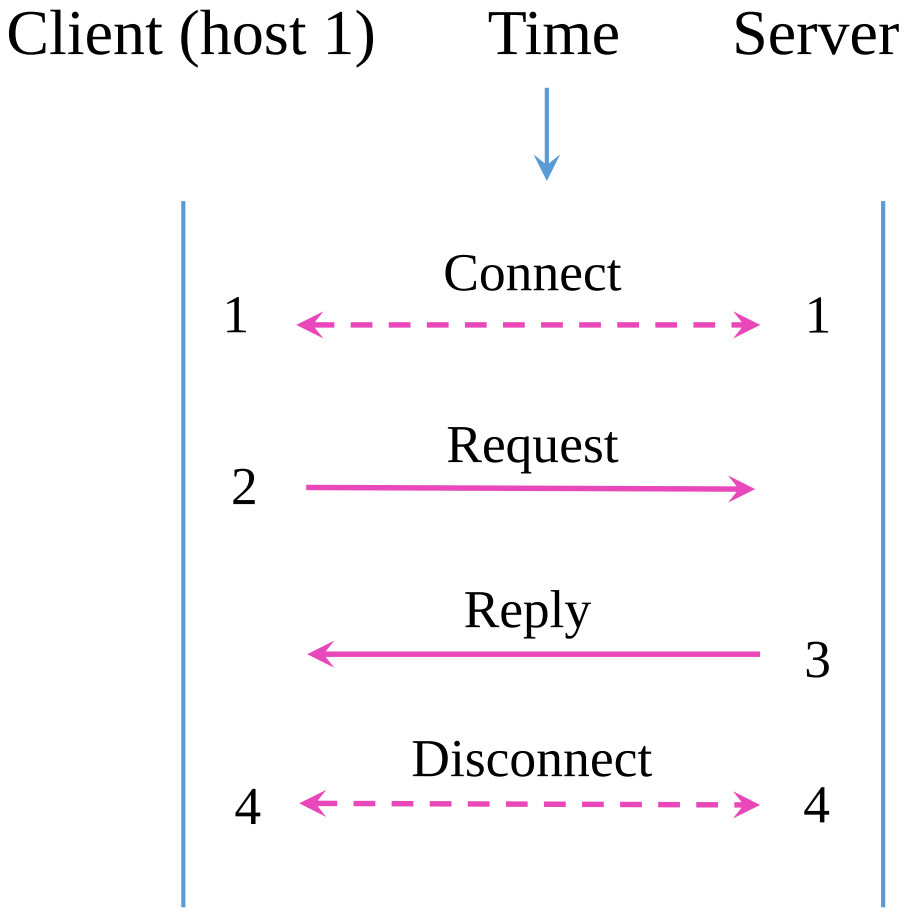
- سوکت‌ها به app ها اجازه برقراری ارتباط با شبکه محلی را از طریق پورت‌های مختلف فراهم می‌کنند.
- شماره پورت‌ها به نوعی فرمی از **آدرس‌دهی** را فراهم می‌کنند.
- همچنین به نوعی **مالتی‌پلکسینگ** را فراهم می‌کند. چند app می‌توانند از یک host استفاده کنند.



# Socket API (3)

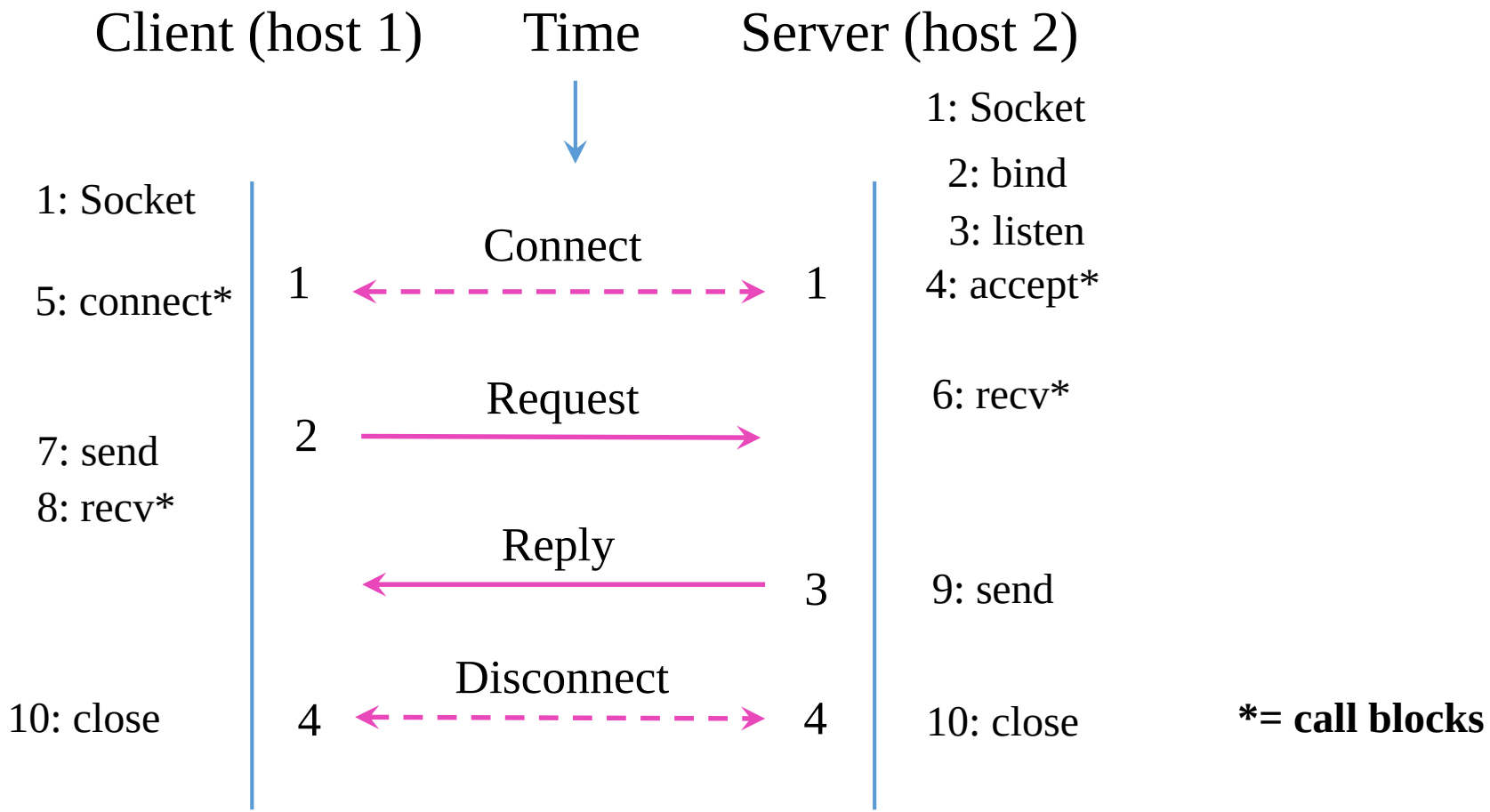
| Primitive | Meaning |
|---|---|
| SOCKET | Create a new communication endpoint |
| BIND | Associate a local address with a socket |
| LISTEN | Announce willingness to accept connections; give queue size |
| ACCEPT | Passively establish an incoming connection |
| CONNECT | Actively attempt to establish a connection |
| SEND | Send some data over the connection |
| RECEIVE | Receive some data from the connection |
| CLOSE | Release the connection |

# Using Sockets




ابتدا اتفاقاتی که به مرور زمان برای درخواست و پاسخ باید رخ دهد را بررسی می‌کنیم.

# Using Sockets (2)



# Client Program (outline)

```
socket()                // make socket
getaddrinfo()           // server and port name
                        // www.example.com:80
connect()               // connect to server [block]
…

send()                  // send request
recv()                  // await reply [block]
…                       // do something with data!
close()                 // done, disconnect
```

# Server Program (outline)

```
socket()        // make socket
getaddrinfo()   // for port on this host
bind()          // associate port with socket
listen()        // prepare to accept connections
accept()        // wait for a connection [block]
…
recv()          // wait for request
…
send()          // send the reply
close()         // eventually disconnect
```

Loop structure

# مثالی از کد سرور و کلاینت

```
/* This page contains a client program that can request a file from the server program
 * on the next page. The server responds by sending the whole file.
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                   /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                       /* block transfer size */

int main(int argc, char *argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];                       /* buffer for incoming file */
  struct hostent *h;                        /* info about server */
  struct sockaddr_in channel;               /* holds IP address */

  if (argc != 3) fatal("Usage: client server-name file-name");
  h = gethostbyname(argv[1]);               /* look up host's IP address */
  if (!h) fatal("gethostbyname failed");

  s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s <0) fatal("socket");
  memset(&channel, 0, sizeof(channel));
  channel.sin_family= AF_INET;
  memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
  channel.sin_port= htons(SERVER_PORT);

  c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
  if (c < 0) fatal("connect failed");

  /* Connection is now established. Send file name including 0 byte at end. */
  write(s, argv[2], strlen(argv[2])+1);

  /* Go get the file and write it to standard output. */
  while (1) {
      bytes = read(s, buf, BUF_SIZE);       /* read from socket */
      if (bytes <= 0) exit(0);              /* check for end of file */
      write(1, buf, bytes);                 /* write to standard output */
  }
}

fatal(char *string)
{
  printf("%s\n", string);
  exit(1);
}
```

```
#include <sys/types.h>                      /* This is the server code */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345                   /* arbitrary, but client & server must agree */
#define BUF_SIZE 4096                       /* block transfer size */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];                       /* buffer for outgoing file */
  struct sockaddr_in channel;               /* holds IP address */

  /* Build address structure to bind to socket. */
  memset(&channel, 0, sizeof(channel));     /* zero channel */
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);

  /* Passive open. Wait for connection. */
  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);   /* create socket */
  if (s < 0) fatal("socket failed");
  setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

  b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
  if (b < 0) fatal("bind failed");

  l = listen(s, QUEUE_SIZE);                /* specify queue size */
  if (l < 0) fatal("listen failed");

  /* Socket is now set up and bound. Wait for connection and process it. */
  while (1) {
      sa = accept(s, 0, 0);                 /* block for connection request */
      if (sa < 0) fatal("accept failed");

      read(sa, buf, BUF_SIZE);              /* read file name from socket */

      /* Get and return the file. */
      fd = open(buf, O_RDONLY);             /* open the file to be sent back */
      if (fd < 0) fatal("open failed");

      while (1) {
          bytes = read(fd, buf, BUF_SIZE); /* read from file */
          if (bytes <= 0) break;            /* check for end of file */
          write(sa, buf, bytes);            /* write bytes to socket */
      }
      close(fd);                            /* close file */
      close(sa);                            /* close connection */
  }
}
```

# Client Program (outline)

کدی که مشاهده کردید، هسته اصلی کدهایی است که برای ارتباط بین هاست و سرور نوشته می‌شوند.

زبان‌های برنامه نویسی استفاده‌شده برای ارتباط بین هاست و سرور:

- ✓ زبان برنامه نویسی C
- ✓ زبان برنامه نویسی جاوا
- ✓ زبان برنامه نویسی PHP
- ✓ زبان برنامه نویسی پایتون
- ✓ ...