U LISBOA | UNIVERSIDADE DE LISBOA

Múltiplos Clientes e Sockets Avançados

Bernardo Ferreira / Alan Oliveira /
António Casimiro

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

Fonte da imagem: http://www.dsn.jhu.edu

Ciências ULisboa Faculdade de Ciências da Universidade de Lisboa

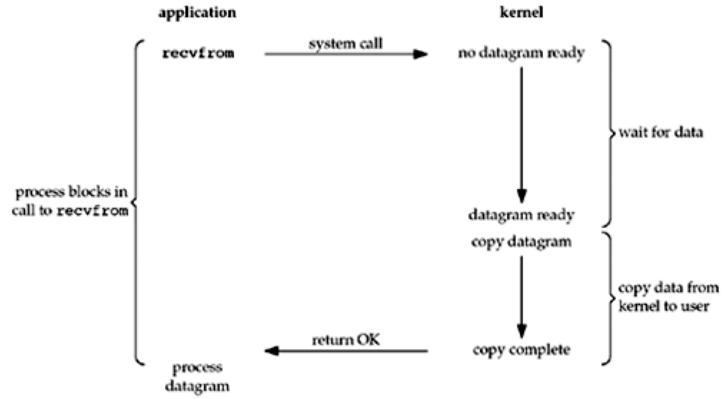
Como lidar com múltiplos clientes?

- **Servidor muito simples**
  - Aceita ligação de vários clientes mas só trata os pedidos de um cliente de cada vez
  - Em geral, inaceitável para servidores reais
- **Servidor concorrente**
  - Trata pedidos de múltiplos clientes em simultâneo

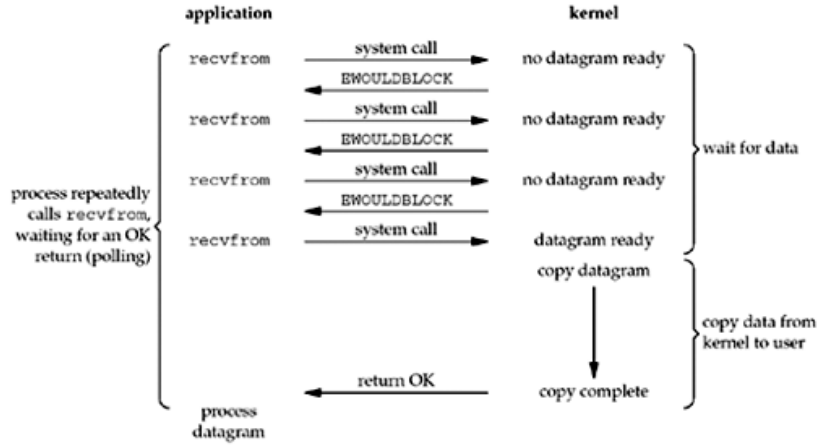
- Opções para concretizar servidores concorrentes:
  - **Bifurcação**
    - » Threads
    - » Fork (criar processos filhos)
  - **Multiplexagem de I/O**
    - » **I/O não bloqueante com select/poll**
    - » I/O não bloqueante com sinais

Modelos de I/O (I)

I/O bloqueante



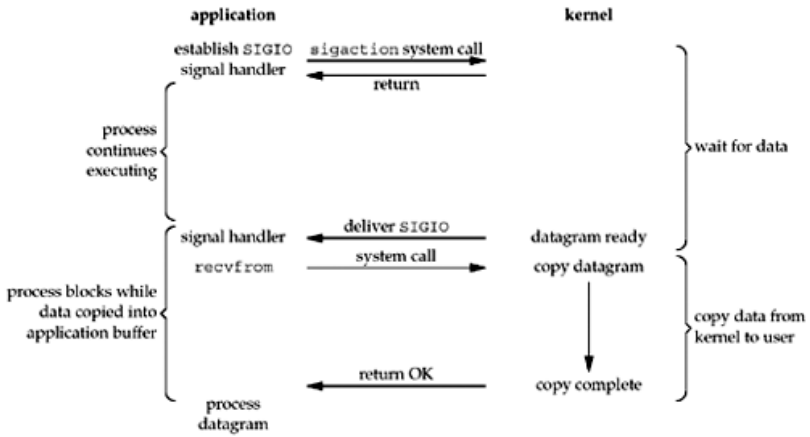
I/O não bloqueante



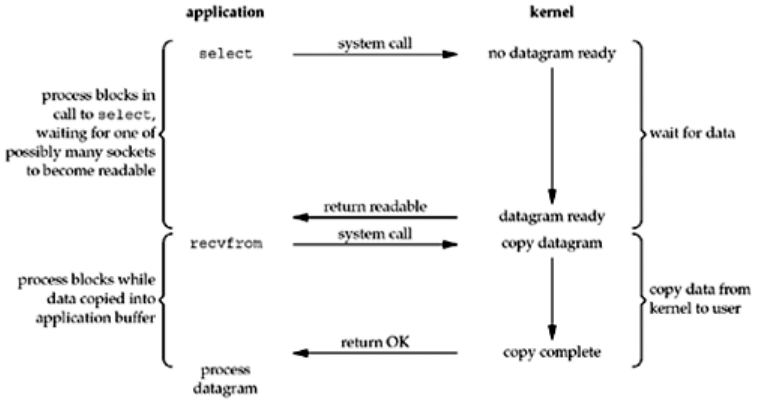
Fonte: [Stevens2004]

Modelos de I/O (II)

I/O não bloqueante com sinais



Multiplexagem de I/O



Fonte: [Stevens2004]

Multiplexagem de I/O com *select()* e *poll()*

- ***select()*** e ***poll()***
  - funções semelhantes que permitem observar o estado de vários sockets em simultâneo

- Recebem:
  - Lista de descritores de **sockets**
  - Conjunto de **eventos** a observar para cada descritor
  - **Timeout**

- Devolvem:
  - **Número** de descritores com eventos
  - Conjunto de **eventos** observados para cada descritor

- Aguardam durante um **timeout** (pode ser 0, positivo, ou infinito)

select() e *poll()*

- **select():** Função mais tradicional (BSD 4.2) que não monitora mais que FD_SETSIZE descritores (tipicamente 1024)

```
#include <sys/select.h>
int select (int nfds, fd_set *read_set, fd_set *write_set, fd_set *except_set,
    struct timeval *timeout);
```

- **poll():** Função mais moderna (POSIX.1-2001), sem limite de nº de descritores, e com interface mais simples

```
#include <poll.h>
int poll (struct pollfd *pfds, nfds_t nfds, int timeout);
```

poll() - Introdução

- **poll()**: Aguarda durante timeout que pelo menos um dos descritores "registados" gere um evento
  - **Evento** = **socket** que tenha **dados** para **leitura** ou **escrita**, que receba mensagens urgentes, ou erros

```
#include <poll.h>
int poll (struct pollfd *pfds, nfds_t nfds, int timeout);

struct pollfd {
   int fd;                  /* file descriptor */
   short events;            /* queried event bit mask */
   short revents;           /* returned event mask */
}
```

- Devolve: número positivo de descritores prontos, 0 se timeout, -1 em caso de erro.

poll() – events e *revents*

- Eventos que se podem indicar em ***events***
  - **POLLIN**: Dados disponíveis para leitura (ou, no caso dum socket servidor: uma nova ligação recebida)
  - **POLLOUT**: Espaço disponível para escrever (write() não vai bloquear, ou, no caso de um connect() assíncrono: ligação estabelecida)
  - **POLLPRI**: Dados urgentes disponíveis para leitura

- Eventos que a função devolve em ***revents***
  - **Todos de *events* mais** os seguintes:
    - » **POLLERR**: Descritor tem erros
    - » **POLLHUP**: Descritor hang up (fechou a ligação)
    - » **POLLNVAL**: Descritor inválido

Usos possíveis da função *poll*

- **Função sleep com alta precisão** (milisegundos em vez de segundos)
  - nfds = 0
  - Timeout positivo
- **Esperar até o primeiro evento**
  - nfds > 0
  - timeout INFTIM ou -1
- **Esperar até o primeiro evento ou timeout**
  - nfds > 0
  - Timeout positivo
- **Verificação instantânea dos descritores** (sem esperar)
  - nfds > 0
  - Timeout = 0

Exemplo

```
struct pollfd my_pfds[2];
my_pfds[0].fd = fileno(stdin);
my_pfds[0].events = POLLIN;
my_pfds[1].fd = sockfd;
my_pfds[1].events = POLLIN;
if (poll(my_pfds, 2, INFTIM) > 0) {
   if( my_pfds[0].revents & POLLIN ) {
     /* data disponível no stdin */
     [...]
  }
  if (my_pfds[1].revents & POLLIN ) {
      /* data disponível na socket sockfd */
      [...]
   }
}
```

FUNÇÕES DE SOCKETS AVANÇADAS

Função *getpeername() / getsockname()*

- Obter informações sobre um *socket* aberto:
  - ***getpeername()*** : devolve o endereço do processo remoto que se encontra ligado ao socket
  - ***getsockname()*** : devolve o endereço local associado a um socket

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

int getpeername(int sockfd, struct sockaddr *peer,
                socklen_t *addrlen);

int getsockname(int sockfd, struct sockaddr *local,
                socklen_t *addrlen);
```

- Devolvem: 0 se tudo corre bem (o endereço é colocado em *peer* ou local, dependendo da função) ou -1 em caso de erro.

Função *gethostbyname()*

- ***gethostbyname()***: dado um nome de uma máquina (e.g., nemo.alunos.di.fc.ul.pt) devolve informação relacionada com o seu endereço

```
#include <netdb.h>

struct hostent *gethostbyname(char *hostname);
/* Para o IPv4 e IPv6 */
struct hostent *gethostbyname2(char *hostname, int family);

struct hostent {
  char    *h_name;        /* official name of host */
  char    **h_aliases;    /* alias list */
  int     h_addrtype;     /* host address type = AF_INET */
  int     h_length;       /* length of address = 4 */
  char    **h_addr_list;  /* NULL terminated list of addr */
}
#define h_addr  h_addr_list[0]  /* first addr on the list */
```

- Devolve: ponteiro não nulo se OK, NULL em caso de erro com respectiva indicação em *h_errno*

EXEMPLO DE USO DA FUNÇÃO *GETHOSTBYNAME()*

```c
/* Print the "hostent" information of the host whose name is
 * specified on the command line. */
#include <stdio.h>
#include <sys/types.h>
#include <netdb.h>/* for struct hostent */
#include <sys/socket.h>     /* for AF_INET */
#include <netinet/in.h>     /* for struct in_addr */
#include <arpa/inet.h>      /* for inet_ntop() */

int main(int argc, char **argv)
{
    struct in_addr *addrptr;
    struct hostent *hostptr;
    int i;
    char s[1024];

    if (argc < 2) {
          printf("Host not given\n"); return -1;
    }

    if ( (hostptr = gethostbyname(argv[1])) == NULL) {
          printf("gethostbyname error for host: %s\n", argv[1]);return -1;
    }
```

```
    printf("official host name: %s\n", hostptr->h_name);

    /* go through the list of aliases */
    for (i=0 ; (hostptr->h_aliases)[i] != NULL ; i++)
          printf(" alias: %s\n", (hostptr->h_aliases)[i]);

    printf("addr type = %d, addr length = %d\n",
          hostptr->h_addrtype, hostptr->h_length);
    if(hostptr->h_addrtype == AF_INET) {
          /* Print first address in dotted-decimal notation. */
          addrptr=(struct in_addr*)hostptr->h_addr;
          printf("First Internet address: %s\n",
                  inet_ntop(AF_INET,addrptr,s,1024));

          /* Print other address in dotted-decimal notation. */
          for (i=1 ; hostptr->h_addr_list[i] != NULL ; i++) {
                  addrptr = (struct in_addr*)hostptr->h_addr_list[i];
                  printf("Internet address: %s\n",
                          inet_ntop(AF_INET,addrptr,s,1024));
          }
    } else {
          printf("unknown address type\n"); return -1;
    }
}
```

```
struct hostent {
  char    *h_name;
  char    **h_aliases;
  int     h_addrtype;
  int     h_length;
  char    **h_addr_list;
}
```

```
fvramos@fvramos-virtualbox:~/Desktop$ gcc gethostbyname_exemplo.c -o gethostbyname_exemplo
fvramos@fvramos-virtualbox:~/Desktop$ ./gethostbyname_exemplo
Host not given
fvramos@fvramos-virtualbox:~/Desktop$ ./gethostbyname_exemplo nemo.alunos.di.fc.ul.pt
official host name: nemo.alunos.di.fc.ul.pt
addr type = 2, addr length = 4
First Internet address: 10.101.85.18
fvramos@fvramos-virtualbox:~/Desktop$ ./gethostbyname_exemplo www.cnn.com
official host name: www.cnn.com
addr type = 2, addr length = 4
First Internet address: 157.166.226.26
Internet address: 157.166.255.18
Internet address: 157.166.255.19
Internet address: 157.166.226.25
```

Função *gethostbyaddr()*

- ***gethostbyaddr()*** : dado um endereço devolve informação relacionada com a máquina

```
#include <netdb.h>

struct hostent *gethostbyaddr(const char *addr,
                              size_t len, int family);
```

- Devolve: ponteiro não nulo se OK, NULL em caso de erro com respectiva indicação em *h_errno*.

Função *getservbyname()*

- ***getservbyname()*** : obtém informação sobre um serviço (guardada em /etc/services)

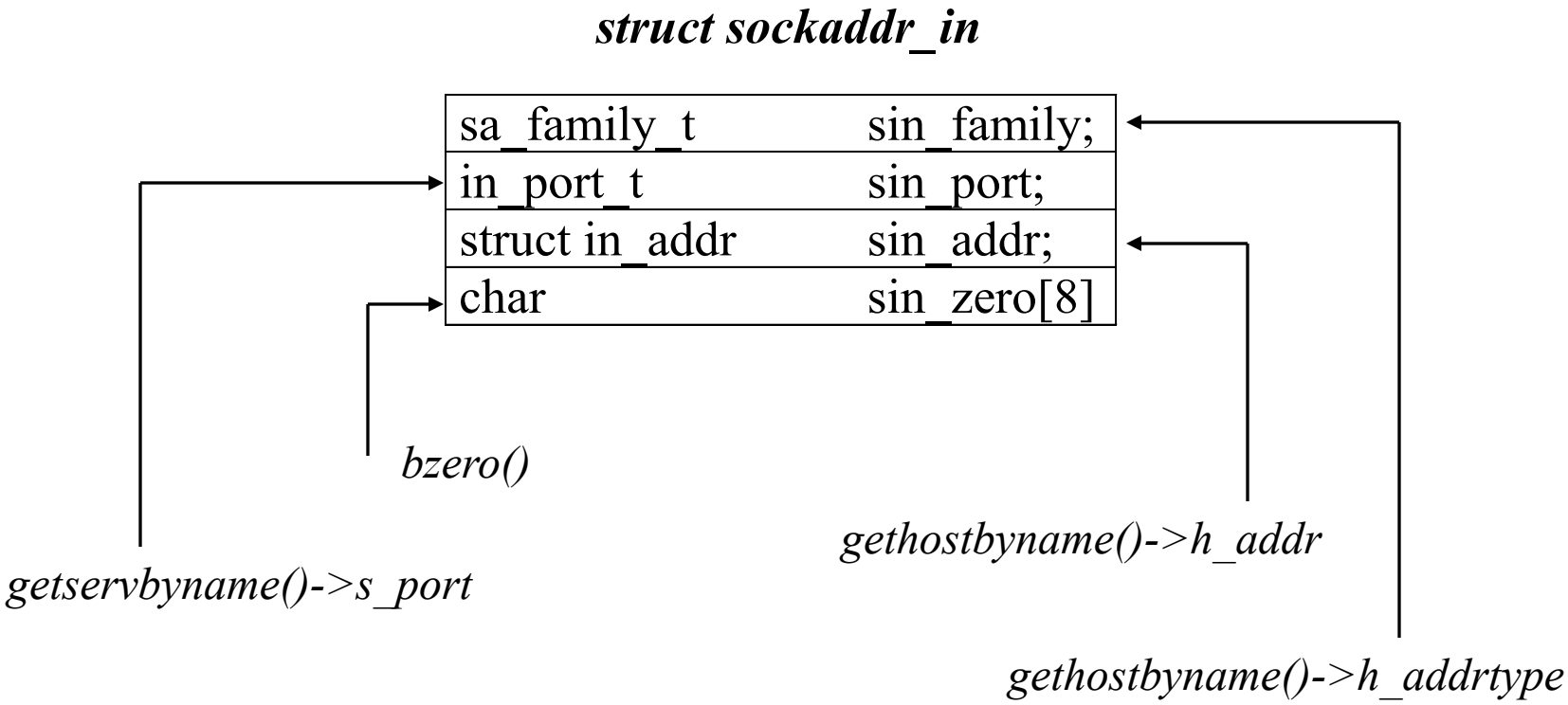
```
#include <netdb.h>

struct servent *getservbyname(const char *sname, const char *protname);
struct servent *getservbyport(int port, const char *proto);

struct servent {
  char *s_name;          /* nome oficial do serviço */
  char **s_aliases;      /* lista de outros nomes */
  int  s_port;           /* n.º do porto (já em formato da rede) */
  char *s_proto;         /* protocolo a usar */
```

exemplo

sptr= getservbyname("ftp", "tcp");

- Devolve: ponteiro não nulo se OK, NULL em caso de erro com respectiva indicação em *h_errno*.

Preencher o endereço de um servidor sabendo o seu nome e o nome do serviço



OPÇÕES DE SOCKETS AVANÇADAS

Obter e alterar *socket options*

- `getsockopt()/setsockopt()`
  - Obter ou alterar parâmetro de configuração de sockets

- `fcntl()`
  - Obter ou alterar parâmetro de configuração de um ficheiro

- `ioctl()`
  - Alterar características de um descritor aberto

Funções `getsockopt()/setsockopt()`

- ***getsockopt()*** : obter o valor de um parâmetro de configuração das sockets
- ***setsockopt()*** : alterar um parâmetro de configuração das sockets

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

int getsockopt(int sockfd, int level, int optname,
  void *optval, socklen_t *optlen);
int setsockopt(int sockfd, int level, int optname,
  void *optval, socklen_t optlen);
```

`level` : camada que vai interpretar a opção – TCP, IP , socket, etc.

`optname` : nome da opção que se quer obter ou alterar

`optval` : valor da opção

`optlen` : tamanho do valor da opção

Algumas opções

- Socket (genéricas)
  - SO_BROADCAST: permite o envio de pacotes em broadcast (SOCK_DGRAM)
  - SO_LINGER: determina o que fazer aos pacotes que existam nos *buffers* do sistema quando se executa o close() de uma socket. Por omissão, o close() retorna imediatamente mas o sistema tenta enviar os dados que existam pendentes (SOCK_STREAM)
  - SO_RCVBUF e SO_SNDBUF: tamanho dos *buffers* de envio e recepção das sockets - utilizados para melhorar o desempenho
  - SO_REUSEADDR: indica ao sistema que o porto de uma socket pode ser reutilizado.
    - » Útil em inúmeras situações (ver [Stevens2004], secção 7.5).
- IPv4
  - IP_TOS, IP_TTL, IP_HDRINCL, etc.
- TCP
  - TCP_NODELAY : desactiva o *algoritmo Nagle*. O objectivo do algoritmo é reduzir o número de pacotes pequenos transmitidos pela rede. Se houver dados em trânsito e o cliente tiver um pacote pequeno para enviar, este tem de esperar pelo ACK de todos os dados. Estes pacotes pequenos vão assim sendo guardados num *buffer* até perfazerem o tamanho de um pacote "normal". Esta opção permite assim alterar este comportamento, fazendo com que os pacotes sejam imediatamente enviados. (SOCK_STREAM)

Exemplo

```
#include<sys/types.h>
#include<sys/socket.h>     /* for SOL_SOCKET and SO_xx values */

main()
{
   int  sockfd, sim;

   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("can't create socket"); exit(1);
   }
   /* Permite reutilização do socket */
   sim = 1;
   if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (int *)&sim,
                  sizeof(sim)) < 0 ) {
        perror("SO_REUSEADDR setsockopt error");
   }
   /* Cria server socket */
   bind(sockfd, ...);
}
```

Função `fcntl()`

- Obter ou alterar um parâmetro de configuração de um descritor (e.g., um socket)

```
#include <fcntl.h>
int fcntl(int fd, int cmd, ... /* int arg */);
```

`fd`: descritor de um ficheiro (ou descritor de um socket)

`cmd`: comando a ser executado

`F_GETOWN e F_SETOWN`: especificar quem recebe os SIGIO/SIGURG

`F_GETFL e F_SETFL`: modificar algumas flags associadas ao descritor

Flags: O_NONBLOCK, O_ASYNC

`arg`: valor da opção

- Devolve: depende de cmd, -1 em caso de erro.

Função `ioctl()`

- Alterar as características de um descritor aberto

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

int ioctl(int fd, int request, ... /* char *arg */);
```

`fd`: descritor de um ficheiro (ou socket)

`request`: nome de opção que se quer obter ou alterar

- operação sobre sockets (definir/obter dono do socket)
- operação sobre ficheiros (I/O não bloqueante, *signal-driven* I/O)
- operação sobre interfaces (descobrir endereços das interfaces, descobrir se a interface suporta broadcast e multicast, etc.)
- Etc.

`arg`: argumento da operação

Exercício

① Analise a aplicação cliente-servidor `exemplo_poll.tar.gz` disponível no site da disciplina. O que faz a aplicação?

② Conecte um segundo cliente e verifique que o servidor atende pedidos de **ambos os clientes**.

③ **Opcional**: Implemente um mecanismo simples de **tolerância a faltas**: se o servidor for abaixo o cliente deve esperar 3 segundos e voltar a tentar, repetidamente, até o servidor ficar de novo disponível.

Referências

- [Stevens2004]
  - W. R. Stevens, B. Fenner, A.M. Rudoff, *Unix Network Programming, The Sockets Networking API,* Volume 1, 3rd Edition, Addison Wesley, 2004