

Sockets UDP



gdb threads tree char *ptr: **Programação Avançada** for #include sockets
i++ mutex ponteiro ciclogcc (c) Patricio Domingues
IPL++ linked list doxygen lock/unlock
#define malloc

Autor: Patricio Domingues

(c) Patricio Domingues, Vitor Carreira

UDP – Recordando...

- Não orientado à ligação
- Não garante a entrega dos *datagrams*
- Não garante a sequenciação na entrega dos *datagrams*
- Utilizado em
 - DNS
 - NFS
 - NTP
 - SNMP
 - ...

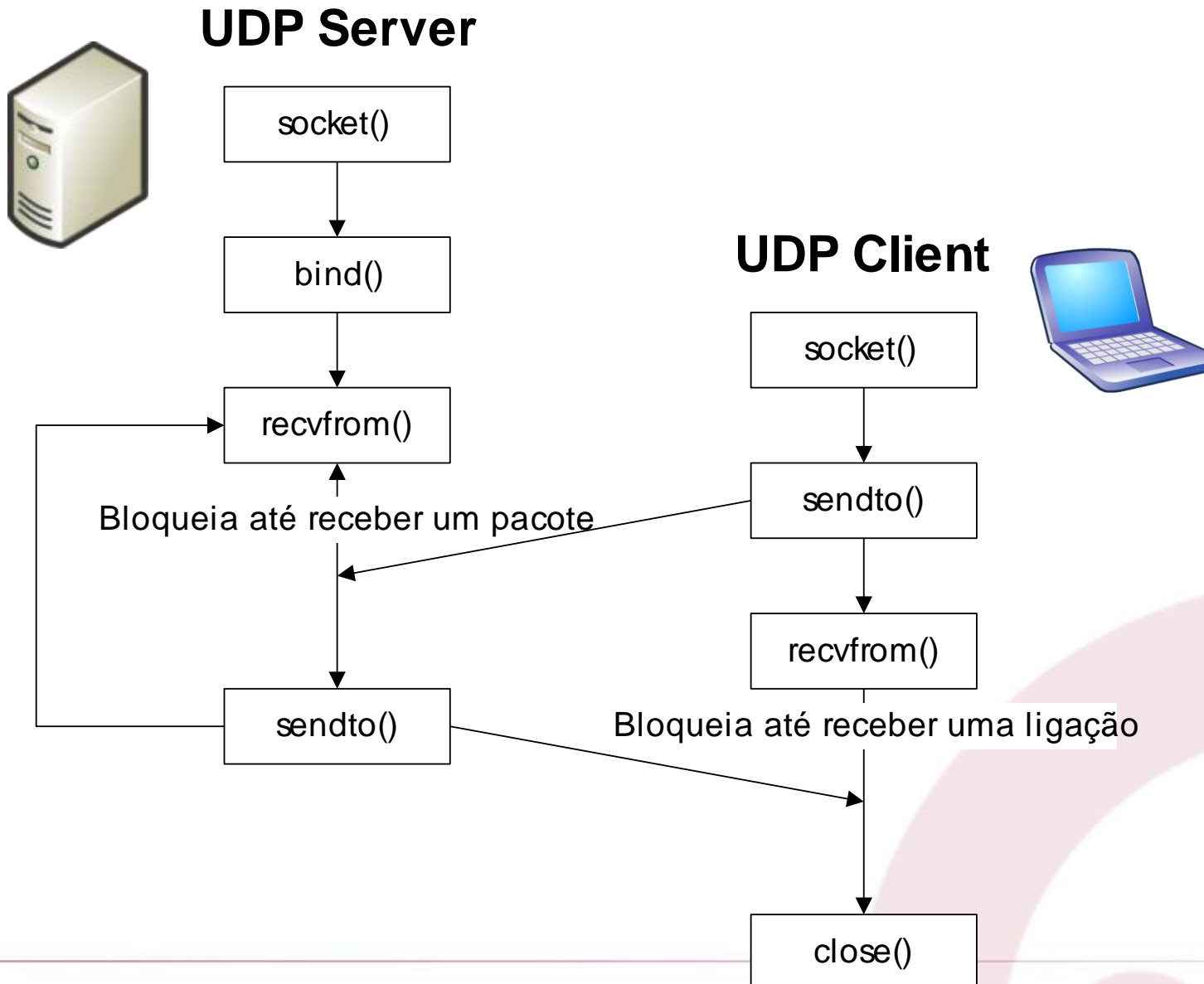




IPL

escola superior de tecnologia e gestão
instituto politécnico de leiria

Funções API Socket: UDP



- Criação sockets UDP (cliente e servidor)
 - Função: `socket`
- Registo socket UDP (servidor e opcionalmente pelo cliente)
 - Função: `bind`
- Envio e recepção de dados (cliente e servidor)
 - Funções: `sendto` e `recvfrom`
- Situações de erro e soluções
 - *Timeout*
 - Utilização de sinais
- Modo ligado
 - Função: `connect`
 - socket passa apenas a poder enviar e receber de uma mesma entidade remota
 - O envio e receção de dados passa a utilizar as funções: `send` e `recv`

■ Cliente

```
int sockfd;  
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1){  
    ERROR(-1, "socket failed");  
}
```

■ Servidor

```
int sockfd;  
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1){  
    ERROR(-1, "socket failed");  
}
```

- A criação do socket é idêntica para ambos os casos – cliente e servidor

Registo do socket UDP (*bind*)

- Apenas é necessário para o servidor

```
#define PORTO 8986
```

```
struct sockaddr_in serv_addr;
```

```
memset(&serv_addr, 0, sizeof(struct sockaddr_in));
```

```
serv_addr.sin_family = AF_INET;
```

```
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
serv_addr.sin_port = htons(PORTO);
```

```
if (bind(sockfd, (struct sockaddr*)&serv_addr,  
                                sizeof(serv_addr))==-1){
```

```
    ERROR(-1, "Bind failed");
```

```
}
```

Envio de dados: sendto (1)

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

- Escreve no socket *s*, *len* bytes do conteúdo da zona de memória apontada por *msg*
 - *to* - Endereço do destinatário (IP + porto)
 - *tolen* - Tamanho do endereço
- Retorno:
 - OK: número de bytes enviados;
 - Erro: *-1*

- É possível enviar uma mensagem de 0 bytes
 - Mensagem de “*heartbeat*”

```
sendto(sockfd, NULL, 0, 0,  
      (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

- Erros mais comuns (variável `errno` `#include <errno.h>`)
 - **EBADF**, **ENOTSOCK** – O descritor de socket é inválido
 - **EFAULT** – Endereço do buffer inválido
 - **ENOBUFS** – Os *buffers* do sistema estão cheios
 - **EMSGSIZE** – Mensagem demasiadamente grande

- Tamanho de uma mensagem UDP
 - Limite teórico: 65535 bytes (65503)
 - Tamanho máximo para um *datagram* IP (incluindo o cabeçalho IP = 24 bytes + cabeçalho UDP = 8 bytes)
 - Sobreposição dos 16 bits do campo *total length* do cabeçalho IP sobre os 16 bits do campo *length* do cabeçalho UDP
 - Rede: 1468 bytes (1500 – 32)
 - Ethernet MTU (Maximum Transmission Unit) = 1500 bytes
 - Superior ao tamanho da rede ocorre fragmentação o que aumenta significativamente a probabilidade da perda do *datagram*
 - Útil 576 bytes (*Minimum Reassembly Buffer Size*)
 - Limite mínimo que qualquer implementação deve suportar

Envio de dados: sendto (4)

- O valor devolvido pelo `sendto()` indica o número de bytes aceites pelo sistema operativo para envio através de um *datagram* UDP
 - Não indica o número de bytes que chegaram ao destino
- Não existe código de erro que indique que o destinatário não recebeu os dados enviados
 - O UDP não é confiável!
 - Exceção
 - socket UDP “ligado” (**connect**) quando não existe serviço remoto (ver mais adiante)



Recepção de dados: recvfrom (1)

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

- Lê do socket *s*, *len* bytes para a zona de memória apontada por *buf*
 - *from* é preenchido com o endereço origem da mensagem (IP + porto)
 - *fromlen* contém o tamanho do endereço
- Retorno:
 - OK: número de bytes recebidos;
 - Erro: **-1**

Receção de dados: recvfrom (2)

■ Algumas notas

- A função `recvfrom` fica bloqueada até receber uma mensagem
 - Exceto se o socket foi configurado para não bloqueante ou se for indicada a flag `MSG_DONTWAIT` na chamada do `recvfrom`
- Caso se pretenda ignorar o endereço origem da mensagem basta utilizar o valor `NULL` nos parâmetros

■ Dados recebidos

- Se o buffer não tiver espaço suficiente para o *datagram* (o tamanho do buffer é especificado pelo parâmetro *Len*)...
 - Os dados que não cabem são perdidos...
 - Caso a flag `MSG_TRUNC` esteja ativa, a função `recvfrom` devolve o tamanho real do datagrama e não o número de bytes escritos no buffer de receção
 - Contudo, apenas são escritos *Len* bytes no buffer, perdendo-se na mesma os dados em excesso
- É possível receber 0 bytes de dados (mensagem de “heartbeat”)



- Erros mais comuns
 - **EBADF**, **ENOTSOCK** – O descritor de socket é inválido
 - **ECONREFUSED** – O endereço destino não está disponível
 - **EFAULT** – Endereço do buffer é inválido
 - **EINTR** – A chamada foi interrompida por um sinal
 - Há que tratar o erro
 - Se (`errno == EINTR`) repetir `recvfrom`;
 - Ou, `sigaction`: `sa.sa_flags = SA_RESTART` (não é suportado em todos os sistemas)

- Código típico de um cliente UDP
 - Cria socket UDP: `socket()`
 - Cria e preenche objeto “sockaddr” com endereço do servidor
 - Endereço IP
 - Porto do servidor
 - Envia pedido ao servidor através do `sendto()`
 - Possivelmente chama `recvfrom()` (no caso de esperar uma resposta).

- Código típico servidor UDP
 - Cria socket UDP: `socket`
 - Regista o socket no sistema local: `bind`
 - **Chama `recvfrom()` à espera de pedidos (chamada bloqueante)**
 - Recebe pedido anotando o endereço do cliente
 - Envia resposta ao pedido através de `sendto()`, recorrendo ao endereço recebido em `recvfrom()`
- Nota:
 - Normalmente, os servidores UDP são iterativos dado que o processamento é do tipo pedido / resposta



Verificar a resposta do servidor

(excerto de código do cliente)

```
if (recvfrom(sockfd, msg, MSG_SIZE, 0,  
            (struct sockaddr*)NULL, NULL) == -1){  
    ERROR("recvfrom failed", -1);  
}
```

- Como limitar a resposta de forma a que tenha apenas como origem o servidor?
 - Resposta: uso do "connect () "

- Exemplo UDP IPv4
 - Ficheiros
 - cliente_udp.c & servidor_udp.c
- Exemplo UDP IPv6
 - Ficheiros
 - cliente_udp_ipv6.c & servidor_udp_ipv6.c

Situações de erro (1)

- 1) Perda de datagramas – o *datagrama* do cliente alcança o servidor mas a resposta do servidor perde-se
 - O cliente irá ficar bloqueado “*ad eternum*” à espera da resposta do servidor (que se perdeu)
- Como prevenir essa situação ?
 - Especificar um tempo máximo de espera no cliente – *timeout*
 - **Opção A** -- Pode implementar-se um mecanismo de *timeout* com o sinal SIGALRM (função `alarm()`)
 - `alarm(10);`
 - » Configura o sistema por forma que o processo recebe 10 segundos volvidos o sinal SIGALRM
 - » A chamada bloqueante `recvfrom` irá ser interrompida devolvendo o erro `EINTR`
 - » O uso de sinais têm a desvantagem da perda de contexto: a execução do processo passa para a rotina de tratamento de sinal

Opção B >>

Situações de erro (2)

- (cont.) Especificar um tempo máximo de espera no cliente – timeout

Exemplo >>

- **Opção B** – configuração dos sockets via `setsockopt`

- Existe ainda `getsockopt` para a leitura das opções de um socket
- Protótipos

```
int setsockopt(int s, int level, int optname, const void *optval, int optlen);
```

```
int getsockopt(int s, int level, int optname, void *optval, int *optlen);
```

SO_SNDTIMEO is an option to set a timeout value for output operations. It accepts a struct timeval parameter with the number of seconds and microseconds used to limit waits for output operations to complete. If a send operation has blocked for this much time, it returns with a partial count or with the error EWOULDBLOCK if no data were sent.

SO_RCVTIMEO is an option to set a timeout value for input operations. It accepts a struct timeval parameter with the number of seconds and microseconds used to limit waits for input operations to complete. If a receive operation has been blocked for this much time without receiving additional data, it returns with a short count or with the error EWOULDBLOCK if no data were received.

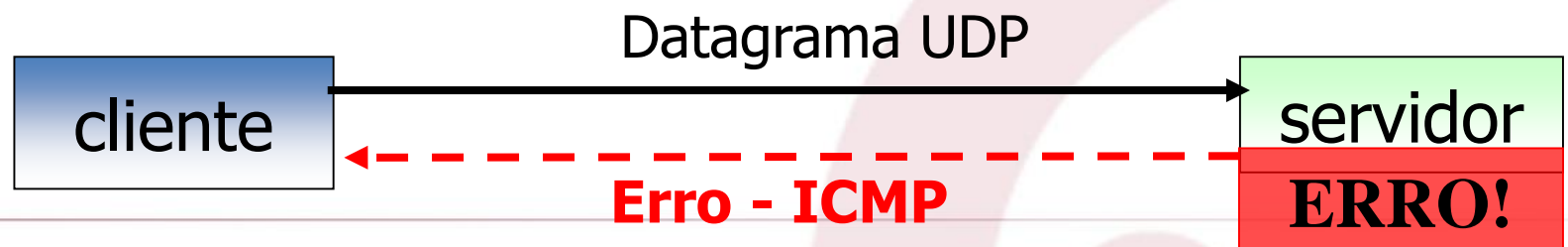
Configura timeout de 2 segundos

```
struct timeval tv;  
tv.tv_sec = 2; tv.tv_usec = 0;  
  
/* configura timeout para RECV */  
setsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, &tv, (socklen_t) sizeof(tv));  
  
/* configura timeout para SEND */  
setsockopt(fd, SOL_SOCKET, SO_SNDTIMEO, &tv, (socklen_t) sizeof(tv));  
  
/* Leitura do valor de timeout especificado para RECV*/  
socklen_t n;  
getsockopt(fd, SOL_SOCKET, SO_RCVTIMEO, &tv, &n);  
printf("sec=%ld, usec=%ld\n", tv.tv_sec, tv.tv_usec);
```

- Consultar: man 7 socket

Situações de erro (4)

- 2) O processo servidor não está a correr
 - Cliente fica bloqueado no `recvfrom`, o tempo correspondente ao *timeout* (solução para a situação 1)
 - Contudo, neste caso, o próprio sistema operativo (não confundir com o processo) deteta o erro e gera uma mensagem de erro assíncrona (~4ms depois)
 - Pacote ICMP (Internet Control Message) indicando que o porto não está disponível
 - O pacote ICMP só é entregue se o socket UDP estiver “ligado”
 - Apenas a chamada `recvfrom` receberá o erro



- Para utilizar as sockets UDP em modo ligado utiliza-se a função `connect`
 - Difere do TCP dado que não é estabelecida nenhuma ligação
 - não existe o 3WS!
 - O kernel apenas guarda o endereço (IP + porto) do destinatário
 - Passa-se a utilizar o `recv()` e `send()` em vez de `recvfrom()` e `sendto()`
- Para que serve?
 - Para permitir que o *socket* seja notificado de erros assíncronos
 - Exemplo
 - situação de erro 2 (ICMP))

- Só para relembrar

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

- Lê do socket `s` `len` bytes para a zona de memória apontada por `buf`

- Retorno:

- OK: número de bytes recebidos;
 - Erro: `-1`

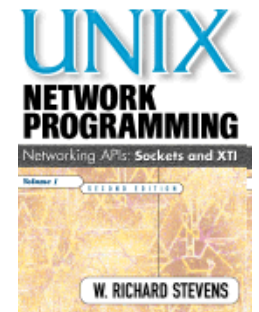
- `ssize_t send(int s, const void* msg, size_t len, int flags);`

- Escreve `len` bytes do conteúdo da zona de memória apontada por `msg` para o socket `s`

- Retorno:

- OK: número de bytes enviados;
 - Erro: `-1`

- “UNIX Network Programming”, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998, ISBN 0-13-490012-X.



(<http://www.kohala.com/start/unpv12e.html>)

- Portugal a Programar – Sockets de berkeley
http://wiki.portugal-a-programar.org/dev_geral:c:sockets_de_berkeley



- “Basic Socket Interface Extensions for IPv6”, RFC 3493, February 2003 (<https://www.ietf.org/rfc/rfc3493.txt>)

- man 7 udp
- man 7 socket