Sockets
Parte II: tópicos avançados

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa

Múltiplos clientes

Servidor muito simples:

- Aceita ligação dum cliente, comunica com este cliente, termina ligação, e só depois aceita o próximo cliente
- Em geral, inaceitavel para servidores reais

Servidor concorrente:

- Aceita ligações de múltiplos clientes em simultâneo

Opções para concretizar servidores concorrentes:

- Bifurcação
  - Threads
  - Fork (criar processos filhos)
- Multiplexação de I/O
  - **I/O não bloqueante com select/poll**
  - I/O não bloqueante com sinais

Multiplexação de I/O com *select()/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)

Multiplexação de I/O com *select()/poll()*

- *select()*

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

Função mais tradicional (BSD 4.2), não aceita descritores maiores que FD_SETSIZE (típicamente 1024)

- *poll()*

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

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

Função *poll()*

- ***poll()*** : Aguarda durante *timeout* que pelo menos um dos descritores dos conjuntos tenha um evento (dados para leitura, não bloqueie na escrita, 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.

Função *poll() - cont.*

- Eventos que se pode indicar em `events`:
  - POLLIN: Dados disponíveis para leitura (ou, no caso duma server socket: 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` e mais os seguintes…
  - POLLERR: Descritor tem errors
  - POLLHUP: Descritor *hang up* (fechou ligação)
  - POLLNVAL: Descritor inválido

Função *poll() - cont.*

- Alguns 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: *poll()*

- Utilização do ***poll()*** para monitorizar `stdin` e socket em parallelo

```
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 */
     [...]
  }
}
```

Configuração adicional de sockets

- `getsockopt()/setsockopt()`

  Funções para obter valor de/para alterar um parâmetro de configuração de sockets

- `fcntl()`

  Funções para obter ou alterar um parâmetro de configuração de um ficheiro

- `ioctl()`

  Funções para alterar as 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 ou socket

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

`optval` : valor da opção

`optlen` : tamanho do valor da opção

Algumas Opções de *getsockopt() / setsockopt()*

- IP_ADD_MEMBERSHIP/IP_DROP_MEMBERSHIP (grupos de multicast)
- TCP_NODELAY : para reduzir o número de pacotes pequenos transmitidos pela rede, algumas das realizações do TCP/IP apenas permitem o envio de um pacote pequeno sem estar a ser ACK. Se se tentar enviar outros pacotes pequenos antes do ACK chegar, estes serão guardados num *buffer*. Esta opção permite alterar este comportamento, fazendo com que os pacotes sejam imediatamente enviados (SOCK_STREAM)
- 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 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 das sockets
- SO_REUSEADDR : indica ao sistema que o porto de uma socket pode ser reutilizado por outros processos (SOCK_STREAM e SOCK_DGRAM)

Exemplo *getsockopt() / setsockopt()*

```
#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, (char *)&sim,
                  sizeof(sim)) < 0 ) {
        perror("SO_REUSEADDR setsockopt error");
   }
   /* Cria server socket */
   bind(sockfd, ...);
}
```

Função *fcntl()*

- ***fcntl()*** : obter ou alterar um parâmetro de configuração de um ficheiro

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

`fd` : descritor de um ficheiro (ou descritor de uma 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
`arg` : valor da opção

- Devolve: depende de cmd, -1 em caso de erro.
- Útil para fazer I/O assíncrono (flag FASYNC)

Função *ioctl()*

- ***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 descritor de uma socket)
`request` : nome de opção que se quer obter ou alterar
- operação sobre ficheiros
- operação sobre sockets
- operação relacionadas com o encaminhamento (routing)
- operação relacionadas com a interface
- operações à cache do ARP
- sistema de streams

`arg` : argumento da operação

Verificar errors

- (Quase) todas as funções da API do SO podem falhar
  - É importante verificar todos os erros possíveis
  - Típicamente, devolvem -1 (int) para indicar um erro (ou ponteiro NULL em alguns casos)

- Em caso de erro:
  - `errno`: inteiro global (por thread) com código de error
  - `perror()`: função para imprimir descrição textual do erro
  - Sem erro, o valor de `errno` não é alterado

- `errno.h` define constantes para os códigos de erro

- Erros "especiais" não indicam erro, programa pode (e deve) repetir a função:
  - EINTR (se usar sinais)
  - EWOULDBLOCK / EAGAIN (operação I/O não bloqueante)

Verificar erros: exemplo `socket()`

- `EACCES`
  Permission to create a socket of the specified type and/or protocol is denied.
- `EAFNOSUPPORT`
  The implementation does not support the specified address family.
- `EINVAL`
  Unknown protocol, or protocol family not available.
- `EMFILE`
  Process file table overflow.
- `ENFILE`
  The system limit on the total number of open files has been reached.
- `ENOBUFS` or `ENOMEM`
  Insufficient memory is available. The socket cannot be created until sufficient resources are freed.
- `EPROTONOSUPPORT`
  The protocol type or the specified protocol is not supported within this domain.

Verificar erros: exemplo

```
int readfully(int sock, char *buf, int len) {
    int bufsize = len;
    while(len>0) {
        int res = read(sock, buf, len);
        if(res<0) {
            if(errno==EINTR) continue;
            perror("read failed: ");
            return res;
        }
        buf += res;
        len -= res;
    }
    return bufsize;
}
```