

Ficha 8 – Multiplexagem Síncrona de Entrada/Saída

Tópicos abordados:

- *Multiplexagem Síncrona E/S*
- Função `select`
- Exercícios

Duração prevista: 1 aula

©2019: {patricio.domingues,, carlos.grilo, carlos.machado, rui.ferreira, gabriel.silva, luis.correia}@ipleiria.pt

1. *Multiplexagem Síncrona de Entrada/Saída*

Alguns casos de usos requerem a possibilidade de uma dada aplicação ser capaz de processar simultaneamente eventos vindos de fontes distintas. Por exemplo, uma aplicação servidora de *chat* assente numa arquitetura cliente/servidor, deve processar os eventos (mensagens) de todos os clientes, incluindo as situações em que vários clientes enviam simultaneamente conteúdo para o servidor. Considerando que num serviço de *chat*, cada cliente está ligado ao servidor através de um `socket` TCP, não é viável colocar a aplicação servidora a chamar a função `recv` para um determinado `socket`, pois por omissão a chamada `recv` é bloqueante. O uso de *threads* é uma possibilidade, mas requer sincronização na receção e transmissão de dados e poderá não ser escalável (e.g., serviço de chat com milhares de utilizadores). O que é necessário é um mecanismo que possibilite a monitorização simultânea de vários canais, permitindo *multiplexagem* de Entradas/Saídas. Nesta ficha de trabalho é estuda a multiplexagem síncrona de entrada/saída indicadas por descritores de ficheiros/sockets

1.1 Função `select`

```
/* According to POSIX.1-2001, POSIX.1-2008 */
#include <sys/select.h>
/* According to earlier standards */
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
```

```
int select(int nfd, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

A função `select` permite efetuar monitorização simultânea de vários descritores de ficheiros/sockets. O caso de uso é simples: o programador configura os descritores de ficheiros/sockets que pretende monitorizar e chama a função. Esta bloqueia até que ocorra um evento – e.g., novo pedido de ligação num `socket` TCP ou a chegada de um datagrama num `socket` UDP – retornando a indicação do(s) evento(s) que ocorreu/am. Para evitar espera sem término, a função `select` disponibiliza ainda um temporizador que possibilita limitar a espera no tempo.

1.2 Funcionamento geral

```
int select(int nfd, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

Os parâmetros da função `select` são os seguintes:

- **int nfd**: corresponde ao valor do maior descritor acrescido em uma unidade. Cabe ao programador determinar o valor deste parâmetro com base nos valores numéricos dos descritores que pretende monitorizar
- **fd_set *readfds**: endereço de variável do tipo `fd_set`. Deve conter a indicação de quais os descritores que se pretendem monitorizar para **eventos de leitura**. Um evento de leitura pode indicar que existe um novo pedido de ligação (`socket` TCP), dados para ler (descritor de ficheiro, sockets TCP ou UDP). Caso não se pretenda monitorizar eventos de escrita, deve ser passado o valor `NULL`.
- **fd_set *writefds**: similar ao parâmetro `readfds`, mas para eventos de escritas. Um evento de escrita é, por exemplo, a indicação que já é possível escrever num `socket`, ou que a parte de escrita do `socket` foi fechado (`socket semi-aberto`). Caso não se pretenda monitorizar eventos de escrita, deve ser passado o valor `NULL`. A manipulação do tipo de dado `fd_set` é feito através das macros `FD_CLR`, `FD_ISSET`, `FD_SET` e `FD_ZERO`, como é explicado mais adiante.
- **fd_set *exceptfds**: similar aos parâmetros `readfds` e `writefds`, mas para assinalar eventos ditos de exceções. Um exemplo de evento classificado como exceção é a

existência de dados “fora-de-banda” num socket. Caso não se pretenda monitorizar nenhum evento de exceção, deve ser passado o valor NULL.

- `struct timeval *timeout`: parâmetro que permite especificar o tempo máximo de espera da função `select` sem que ocorra um evento nos descritores monitorizados. Caso se pretenda uma chamada bloqueante, sem limite de tempo de espera, deve ser passado o valor NULL. Caso se pretenda uma chamada com efeitos imediatos (não bloqueante), deve ser passado o valor de temporização a zero, colocando a zero os dois campos da `struct timeval`.

Valores de retorno

A função `select` devolve:

- 1: se ocorreu um erro do sistema, atribuindo à variável `errno` um código apropriado ao erro
- 0: caso não exista nenhum evento nos descritores monitorizados. Tal ocorre quando o temporizador expira, sem que tenha ocorrida atividade nos descritores monitorizados.
- > 0: número de eventos detetados pela função.

1.3 Manipulação do tipo de dado `fd_set`

O tipo de dado `fd_set` permite i) especificar quais os descritores que se pretendem monitorizar e ii) determinar em que descritores ocorreram eventos. A situação i) corresponde ao valor que é passado pelo programador à função `select`, ao passo que a situação ii) representa a forma com a função `select` indica ao código chamante quais os descritores em que ocorreram eventos.

A manipulação de um `fd_set` faz-se através das macros `FD_ZERO`, `FD_SET`, `FD_CLR` e `FD_ISSET`, descritas nas secções que se seguem:

```
void FD_ZERO(fd_set *set);
void FD_SET(int fd, fd_set *set);
void FD_CLR(int fd, fd_set *set);
int  FD_ISSET(int fd, fd_set *set);
```

1.3.1 `FD_ZERO`

```
void FD_ZERO(fd_set *set);
```

Zera o conjunto `set`, isto é, inicializa-o.

1.3.2 `FD_SET`

```
void FD_SET(int fd, fd_set *set);
```

Ativa a monitorização para o descritor `fd` no conjunto `set`.

1.3.3 FD_CLR

```
void FD_CLR(int fd, fd_set *set);
```

Inibe/remove a monitorização para o descritor fd no conjunto set.

1.3.4 FD_ISSET

```
int FD_ISSET(int fd, fd_set *set);
```

Testa se o descritor fd está ativo no conjunto set.

1.3.5 Exemplo de uso

O exemplo da Listagem 1 ilustra i) a configuração da chamada ao `select` e a consequente ii) análise de resultados da monitorização. Em concreto, são continuamente monitorizados três sockets: `sock_UDP`, `sock_TCP1` e `sock_TCP2`, sendo empregue um temporizador de 30 segundos.

```
int sock_UDP = socket(...);
int sock_TCP1 = socket(...);
int sock_TCP2 = socket(...);
(...)
int max = MAX(sock_UDP, sock_TCP1, sock_TCP2);
fd_set readset;
struct timeval timeout;
while(1){
    FD_ZERO(&readset);
    // ativa monitorização para os descritores sock_UDP, sock_TCP1 e sock_TCP2.
    FD_SET(sock_UDP, &readset);
    FD_SET(sock_TCP1, &readset);
    FD_SET(sock_TCP2, &readset);
    timeout.tv_sec = 30;
    timeout.tv_usec = 0;
    int ret_select = select(max+1, &readset, NULL, NULL, &timeout);
    if(ret_select == -1 ){
        ERROR(1, "Cannot select");
    }
    if(ret_select == 0 ){
        printf("Timeout has expired\n");
        continue;
    }
    if( FD_ISSET(sock_UDP, &readset) ){
        printf("Event in sock_UDP\n");
        continue;
    }
    if( FD_ISSET(sock_TCP1, &readset) ){
        printf("Event in sock_TCP1\n");
        continue;
    }
    if( FD_ISSET(sock_TCP2, &readset) ){
        printf("Event in sock_TCP2\n");
        continue;
    }
}
} // while (1);
```

Listagem 1: Exemplo de uso da função select

2. Exercícios

Na resolução de cada exercício deverá utilizar o *template* de exercícios cliente/servidor, que inclui uma *makefile* bem como todas os ficheiros para as aplicações cliente e servidor, bem como as dependências necessárias à compilação. As funções comuns às aplicações cliente/servidor deve ser colocadas no ficheiro `common.c` e os respetivos protótipos no ficheiro `common.h`.

2.1. Para a aula

1. Recorrendo à linguagem C e à função `select`, elabore a aplicação servidora `par_impar_serv`. A aplicação deve registar o porto indicado com o argumento `--port/-p <porto>` em TCP e em UDP. Sempre que receber uma string com um número no socket TCP, deve devolver uma *string* representando um número **par** aleatório compreendido entre 0 e o número que recebeu. Similarmente, sempre que receber uma *string* com um número no socket UDP, deve uma *string* representando um número **impar** aleatório compreendido entre 1 e o número que recebeu. Para testar a aplicação, sugere-se o uso do utilitário `nc (netcat)`.

2.2. Extra-aula

2. Recorrendo à função `select`, implemente o serviço *broadchatTCP*. Como o nome sugere, o serviço *broadchatTCP* permite que um utilizador envie uma mensagem (string) para todos os utilizadores ligados ao serviço. Para o efeito deve implementar:
 - `broadchatTCP_serv`: aplicação servidora. Recebe através da opção `--port/-p <porto>` o porto de escuta. Sempre que um cliente solicita uma ligação, deve aceitar a ligação, ficando à espera de uma mensagem desse cliente. Sempre que tal ocorre, a mensagem do cliente deve ser enviada para todos os outros clientes ligadas ao serviço *broadchatTCP*, exceto para o cliente emissor que deve receber a indicação “mensagem a ser enviada para X utilizadores”, em que X indica o número de utilizadores para os quais foi enviada a mensagem-
 - `broadchatTCP_clnt`: aplicação cliente do serviço *broadchatTCP*. Deve i) enviar para o servidor do *broadchatTCP* as mensagens que são inseridas pelo utilizador através da entrada padrão; ii) mostrar ao utilizador todas as mensagens

que são enviadas pelos outros utilizadores; iii) terminar quando deteta que o utilizador escreve a palavra **END**. A aplicação cliente recebe o porto (--porto/-p <porto>) e o endereço IP da aplicação servidora (--ip/-i <endereçoIP>) através dos parâmetros da linha de comando.

3. Bibliografia

- [1] UNIX Network Programming, Volume 1, 3rd edition: Networking APIs: Sockets and XTI, Prentice Hall, 2003, 978-0131411555. *Section 6 / . I/O Multiplexing: The select and poll functions*
- [2] Transparências das aulas teórico-práticas de Programação Avançada
- [3] Páginas do manual (man pages – man <nome da função>)
- [4] “Beej's Guide to Network Programming - Using Internet Sockets”, Brian “Beej Jorgensen” Hall, 2016 (<http://beej.us/guide/bgnet/>)