

Ficha 4 – Comunicação tipo *stream* através de *pipe***I**

Considere os dois programas apresentados na Figura 1. Um destes programas (**fifo_receiver**) tem a função de imprimir no ecrã todas as mensagens que recebe através de um *pipe* com nome (também designado por *fifo*). O outro programa (**fifo_sender**) tem a função de fazer a leitura das mensagens de texto introduzidas pelo utilizador e enviá-las para o primeiro, através do *pipe*.


A função `int mkfifo(const char *pathname, mode_t mode)` faz a criação do *pipe* com nome (para mais detalhes, man 2 mkfifo).

<pre>// fifo_sender-v1.c int main() { int fd, n, sz; char buffer[MAXSZ]; fd = open_fifo(FIFO_NAME, O_WRONLY); if(fd == -1) { perror("open_fifo"); exit(1); } while(1) { fgets(buffer, MAXSZ, stdin); sz = strlen(buffer); n = write(fd, buffer, sz); if(n!=sz) perror("write"); } }</pre>	<pre>// fifo_receiver-v1.c int main() { int fd, n; char buffer[MAXSZ]; fd = open_fifo(FIFO_NAME, O_RDONLY); if(fd == -1) { perror("open_fifo"); exit(1); } while(1) { n = read(fd, buffer, MAXSZ-1); if(n > 0) { buffer[n] = 0; printf("Msg (%d bytes): %s\n", n, buffer); sleep(5); } else sleep(1); } }</pre>
<pre>int open_fifo(char *name, int flags) { struct stat st; int fd; /* Create fifo, if no file with that name exists or */ if existing file is not a fifo (man 7 inode) if(stat(name, &st) == -1 S_ISFIFO(st.st_mode) == 0) { unlink(name); //delete any existing file with that name if(mkfifo(FIFO_NAME, 0600) == -1) return -1; } printf("Opening pipe and waiting for connection on remote endpoint\n"); fd = open(FIFO_NAME, flags); return fd; }</pre>	

Figura 1 – Código fonte dos programas **fifo_sender** (à esquerda) e **fifo_receiver** (à direita).

1. Analise os programas contidos em `fifo_sender-v1.c` e `fifo_receiver-v1.c` e gere os respetivos executáveis.
2. Abra duas janelas de terminal e execute cada programa num terminal diferente. Teste os programas através da introdução de algumas linhas de texto.
3. Termine a execução do programa **fifo_receiver** e introduza uma nova mensagem no programa **fifo_sender**. Por que motivo o programa termina? Dica: SIGPIPE.
4. Inicie novamente os programas e introduza três mensagens em rápida sucessão (em menos de 5 segundos). Verifique que as duas últimas mensagens serão interpretadas pelo recetor como uma única mensagem.

Esse comportamento deve-se ao facto da 3ª mensagem chegar ao recetor antes deste ter lido sequer a 2ª, pois encontra-se bloqueado na pausa de 5 segundos. Quando o recetor finalmente faz a segunda leitura, irá receber toda a informação em espera, isto é, a 2ª e 3ª mensagens:

Tempo	fifo_sender	fifo_receiver
	Tempo de escrita da 1ª mensagem Envio da 1ª mensagem (write) Tempo de escrita da 2ª mensagem Envio da 2ª mensagem (write) Tempo de escrita da 3ª mensagem Envio da 3ª mensagem (write)	1º read (aguarda chegada de dados) recebe 1ª mensagem Espera de 5 segundos (sleep(5)) 2º read : receção de 2ª e 3ª mensagens

*Figura 2 – Exemplo de relação temporal entre a execução dos programas **fifo_sender** e **fifo_receiver**.*

Neste exemplo, a espera de 5 segundos foi introduzida artificialmente. No entanto, na prática, o tempo de processamento dos dados e taxas de transmissão mais altas poderão provocar o fenómeno aqui ilustrado. Ou seja, neste tipo de comunicações devemos ter em conta que o fluxo de dados do lado de recetor não conta com qualquer tipo de delimitação correspondente à sequência de escritas feitas do lado do emissor. Os dados são recebidos como um fluxo constante de *bytes* pelo que, cada vez que se faz um `read`, essa função obtém todos os *bytes* recebidos até essa altura até ao máximo indicado no 3º parâmetro da função (tamanho do *buffer*¹).

Se o tamanho do *buffer* não for suficiente para ler todos os dados entretanto recebidos, o programa poderá também correr o risco de não conseguir ler a mensagem toda de uma vez. Esse fenómeno pode ser facilmente verificado através da alteração do tamanho do *buffer* do recetor, tal como descrito no exercício seguinte.

5. Altere o tamanho do *buffer* usado no programa **fifo_receiver** para 16 *bytes*. Isto significa que as mensagens só poderão ter 14 caracteres úteis (sendo os restantes 2 *bytes* reservados para o `\n` e o `\0`). Gere o novo executável e repita o procedimento do ponto anterior, usando mensagens com pelo menos 8 caracteres (e.g., “mensagem”).

¹ A buffer is a region of memory used to temporarily store data while it is being moved from one place to another.

6. De seguida, pretende-se alterar o programa **fifo_receiver** de modo a que este fique imune aos fenómenos ilustrados nos dois pontos anterior e, dessa forma, faça sempre a impressão de uma mensagem completa de cada vez. Para tal, iremos considerar três abordagens:
- Leitura *byte a byte* com a função `read` até que seja encontrado o carácter `\n`. Esta abordagem é relativamente simples, mas é a menos eficiente do ponto de vista de uso do processador, pois, em sistemas tipo UNIX, cada chamada à função `read` implica um acesso ao núcleo do sistema operativo. Em sistemas tipo UNIX, esta abordagem deverá ser evitada caso se prevejam taxas de transmissão elevadas. Por exemplo:

```
char *pos = buffer;
while(1) {
    n = read(fd, pos, 1);
    if(n > 0) {
        if(*pos == '\n') {
            pos[1] = 0; //terminate string
            printf("Msg (%ld bytes): %s\n", strlen(buffer), buffer);
            pos = buffer;
            sleep(5);
        }
        else
            ++pos;
    }
    else
        sleep(1);
}
```

*Figura 3 – Possível alteração do ciclo de leitura do programa **fifo_receiver** para garantia de leitura linha a linha, assumindo que as mensagens têm tamanho inferior a MAXSZ-2 bytes.*

- Análise *byte a byte* do *buffer* de leitura. Nesta abordagem, o código fica ligeiramente mais complexo, mas é possível garantir uma boa eficiência. Por exemplo:

```
char buffer2[MAXSZ];
char *pos = buffer2;

while(1) {
    n = read(fd, buffer, MAXSZ-1);
    if(n > 0) {
        for(int i = 0; i < n; ++i) {
            *pos = buffer[i];
            if(*pos == '\n')
            {
                pos[1] = 0; //terminate string
                printf("Msg (%ld bytes): %s\n", strlen(buffer2), buffer2);
                pos = buffer2;
                sleep(5);
            }
            else
                ++pos;
        }
    }
    else
        sleep(1);
}
```

*Figura 4 – Possível alteração do ciclo de leitura do programa **fifo_receiver** para garantia de leitura linha a linha, assumindo que as mensagens têm tamanho inferior a MAXSZ-2 bytes.*

- Uso da função `fgets` da biblioteca C standard. Para tal ser possível, o *pipe* com nome deverá ser aberto com a função `fopen`. Esta abordagem permite manter o código simples e garantir uma boa eficiência.

6.1) Altere o programa **fifo_receiver** de modo a que a leitura dos dados seja feita linha a linha com a função `fgets`.

II

1. Analise o código do ficheiro `fifo-sender-v2.c`.

```
// fifo-sender-v2.c
int main() {
    int fd, n;
    int16_t sz; // 16 bit integer (short int)
    char buffer[MAXSZ];

    fd = open_fifo(FIFO_NAME, O_WRONLY);
    if( fd == -1 ) {
        perror("open_fifo");
        exit(1);
    }

    while(1) {
        fgets(buffer, MAXSZ, stdin);
        sz = (short int) strlen(buffer);
        --sz;
        buffer[sz] = 0; //remove '\n'

        n = write(fd, &sz, sizeof(sz));
        if(n != sizeof(sz))
            perror("write");
#ifdef USE_SLOW_WRITE
        n = slow_write(fd, buffer, sz);
#else
        n = write(fd, buffer, sz);
#endif
        if(n != sz)
            perror("write");
    }
}

int slow_write(int fd, void *buf, int len) {
    int sent = 0;
    if(len <= 0)
        return len;
    while(sent != len && write(fd, buf + sent, 1) == 1)
        ++sent;
    return sent;
}
```

Figura 5 – Envio de mensagens com cabeçalho de 2 bytes indicando o comprimento do texto.

Do ponto de vista do utilizador, este programa tem um comportamento idêntico ao do programa analisado anteriormente (`fifo-sender-v1.c`). No entanto, o protocolo de envio das mensagens para o recetor é diferente. Neste caso, cada mensagem é precedida por um valor de 2 bytes indicando o comprimento da mesma, e a mensagem não é terminada com `\n`. Os 2 bytes consistem num valor inteiro, do tipo **short int**. Este pequeno cabeçalho

facilita a implementação da receção da mensagem, pois, desta forma, já se sabe à partida qual o número de caracteres que é necessário ler

- 1.1) Implemente uma nova versão do programa **fifo-receiver** que permita a correta receção das mensagens e a sua apresentação mensagem a mensagem. O programa deverá fazer uma primeira leitura (função `read`) de 2 bytes, seguido de uma segunda leitura (também com a função `read`) indicando o número exato de caracteres da mensagem a ser lida.

2. Adicione a seguinte diretiva no topo do ficheiro `fifo-sender-v2.c`:

```
#define USE_SLOW_WRITE
```

Esta diretiva fará com que o programa use a função `slow_write` para o envio da mensagem. A função `slow_write` simula um mecanismo de comunicação em que os caracteres não são enviados como um único bloco de dados (tal como será visto em futuras aulas).

- 2.1) Teste novamente a comunicação entre os dois programas. Se implementou o programa **fifo-receiver** exatamente como descrito acima, a comunicação deverá falhar.
- 2.2) Altere o programa **fifo-receiver** de forma a garantir a correta leitura da totalidade dos caracteres de cada mensagem. De forma análoga ao que foi visto para a versão anterior (`fifo-receiver-v1.c`), essa leitura pode ser feita *byte a byte*, com *buffer* auxiliar ou usando uma função adequada da biblioteca C standard (neste caso, a função `fread`).