Comunicação

O Modelo de Fluxo de Dados (Streams)

Os programas escritos em C utilizam o conceito de fluxo de dados (em, inglês, eles são chamados de **streams**) para comunicarem-se com dispositivos do computador ou com outros programas. Este modelo é suficientemente poderoso para descrever com simplicidade a maioria dos dispositivos de envio ou recebimento de dados. Entre eles: teclado, terminal (ou prompt do DOS), arquivos, conexões de rede, comunicação entre programas, compressão de dados, entre outros.

Modelo

O modelo de fluxo de dados tem como objetivo oferecer uma única forma de realizar envio e recebimento de dados em um programa escrito em C. Com este modelo, o programa poderá ser compilado em qualquer arquitetura de computadores, e todos os dispositivos aparentarão operar da mesma maneira.

Um fluxo de dados é a passagem de dados entre o programa e outra entidade (que pode ser um dispositivo do computador ou um outro programa). Uma das entidades, chamada de **produtor**, será responsável por oferecer os dados, operação que denominamos **escrever no fluxo**. A outra entidade, denominada **consumidor**, retira estes dados, ou seja, é a operação de **ler do fluxo**. A Figura 1 ilustra o envio de dados entre um produtor e um consumidor.

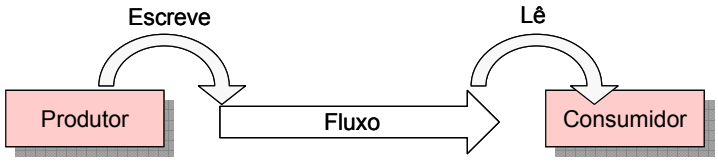

Figura 1 – Modelo de operação do Fluxo de Dados

O fluxo funciona como uma fila, pois os dados inseridos pelo produtor são retirados na mesma ordem pelo consumidor. Como as duas entidades não necessariamente operam no mesmo ritmo, o fluxo armazena temporariamente os dados que foram escritos pelo produtor e ainda não foram lidos pelo consumidor.

Um exemplo simples é o tratamento das teclas digitadas pelo usuário. Existe um fluxo entre o teclado e o programa. Toda vez que uma tecla é pressionada, o programa que controla o teclado coloca (escreve) um caractere no fluxo. Em seguida, o programa consumidor retira um ou mais caracteres acumulados do fluxo e os interpreta de acordo com sua lógica de funcionamento.

Leitura

O **fluxo somente de leitura** é o modelo mais simples. Os bytes são obtidos de uma outra entidade e podem ser lidos um a um, em blocos, ou em segmentos que representam números, palavras ou linhas de texto. Exemplos são arquivos, teclado e conexão de rede.

Nos **fluxos de acesso seqüencial**, a operação de leitura consome os bytes do fluxo, ou seja, não é possível voltar para trás com a intenção de lê-los novamente. Se seu programa necessitar esses dados mais tarde, ele terá de armazená-los na memória.

Os arquivos são um exemplo de um **fluxo de comprimento limitado**. A leitura começa no primeiro byte. Ao alcançar o final do arquivo, as operações de leitura retornarão sempre o

valor EOF (*end of file*), indicando que o final do fluxo foi atingido. A sigla EOF aplica-se a qualquer fluxo, mesmo os que não sejam idealizados como arquivos. O teclado é um **fluxo de comprimento ilimitado**, pois não existe limite para a quantidade de caracteres que o usuário pode digitar.

Como a taxa com a qual o teclado disponibiliza novos bytes é extremamente lenta (quando comparada com a velocidade de processamento do computador), ele opera como um **fluxo bloqueante**. Quando se tenta ler novos bytes de um fluxo bloqueante, e não há novos dados disponíveis, o programa para sua execução e aguarda até a chegada de novos dados. A leitura de arquivos é um **fluxo não bloqueante**, pois, supondo que o final do arquivo não tenha sido alcançado, novos bytes podem ser lidos imediatamente.

Em modelos mais elaborados de fluxo, os bytes lidos não são consumidos e existe a possibilidade de retroceder ou avançar, ou até de saltar para uma posição arbitrária dentro do fluxo. Eles são chamados de **fluxos de acesso aleatório**, pois permitem realizar a leitura a partir de qualquer posição. Existem poucos dispositivos que operam com esta lógica, o principal exemplo sendo os arquivos. Neste tipo de fluxo, existe um contador que indica qual será a posição da próxima leitura. Uma operação de leitura avança (ou retrocede) este contador de acordo com o número de bytes lidos. É claro que não é possível retroceder além do início do fluxo. Essa tentativa fará com que a operação de leitura retorne BOF (*begining of file*).

Escrita

O **fluxo somente de escrita** é um receptor, no qual bytes podem ser escritos um a um, em blocos ou em segmentos representando números ou palavras. Os fluxos de escrita possuem propriedades semelhantes aos de leitura.

O fluxo de escrita corresponde a algum dispositivo consumidor de bytes. Exemplos são arquivos (que estão sendo escritos), um terminal (ou promt do DOS), uma impressora ou uma conexão de rede. Note que a conexão de rede é um exemplo especial, pois ela oferece dois fluxos, um para leitura (receber dados de um outro computador) e um de escrita (enviar dados).

Os **fluxos de escrita seqüencial** absorvem os bytes escritos, não sendo mais possível retroceder para sobreescrevê-los.

Em princípio, os programas podem escrever uma quantidade infinita de dados em um fluxo de escrita seqüencial (por enquanto, vamos ignorar o fato que os arquivos são limitados pelo tamanho da mídia de armazenamento). O tamanho do fluxo cresce à medida que novos bytes são escritos. O programa deve escrever o valor EOF (end of file) para indicar o término da operação de escrita.

Os **fluxos de escrita não bloqueantes** garantem que o programa pode escrever imediatamente os dados. No entanto, no **fluxo bloqueante**, o programa precisa esperar até que o receptor esteja desocupado. Durante a espera, o programa permanece inoperante. A escrita em um arquivo pode ser considerada não bloqueante. A transmissão de dados pela rede é bloqueante, pois o programa precisa esperar até que o outro computador esteja disponível para receber antes de enviar novos dados.

Os modelos mais elaborados de fluxo de escrita oferecem a possibilidade de retroceder, avançar e saltar para uma posição arbitrária dentro do fluxo. Eles são chamados de **fluxos de escrita aleatória**. Ao retroceder no fluxo, a escrita substituirá os bytes já existentes. Não é permitido retroceder aquém do início do fluxo, tal como não é possível avançar além do total de bytes escritos até o momento. Neste tipo de fluxo, existe um contador que indica a

posição da próxima escrita. Uma operação avança (ou retrocede) este contador de acordo com o número de bytes escritos.

Leitura e escrita

Alguns poucos dispositivos oferecem uma combinação das características pertinentes tanto ao fluxo de leitura quanto ao fluxo de escrita. Este é o modelo mais poderoso: o **fluxo de leitura e escrita**.

Ele permite que o programa leia um determinado trecho do fluxo, realize algum processamento sobre os dados, e o reescreva os resultados sobre o fluxo.

Este fluxo apresenta necessariamente acesso aleatório pois, caso contrário, não seria possível retroceder para ler um valor previamente escrito no fluxo. O único exemplo deste modelo são os arquivos.

Note que uma conexão de rede não é um fluxo de leitura e escrita, mas sim dois fluxos totalmente independentes, um de leitura e outro de escrita. Observe também que, por exemplo, não é possível ler novamente um dado que foi enviado para outro computador.

Pecularidades de Fluxos

Em princípio, todos os bytes de um fluxo são tratados de forma semelhante. No entanto, alguns compiladores optaram por distinguir entre **fluxos binários** (imagens, dados comprimidos) de **fluxos de texto**.

Mas por que existe esta distinção? A motivação dos fluxos de texto é contornar automaticamente as diferenças na forma como computadores de arquiteturas diferentes representam caracteres e delimitam linhas e arquivos.

O exemplo mais comum é a marcação de novas linhas. Arquiteturas baseadas no sistema operacional DOS/Windows utilizam dois bytes para marcar o fim de uma linha. Já os sistemas Unix/Linux usam apenas um.

Os compiladores combinam os comandos `printf` e `scanf` com filtros para codificarem operações de leitura e escrita em fluxos, de forma a aparentar o mesmo funcionamento em todos os sistemas.

Abrir e Fechar Fluxos de Dados

Antes de realizar leituras ou escritas, é necessário **abrir o fluxo**. Esta operação tem vários propósitos:

- Associa (ou conecta) o fluxo com o dispositivo desejado.
- Define as operações permitidas (leitura, escrita ou ambas).
- Especifica o tipo do fluxo (binário ou texto).
- Escolhe entre acesso seqüencial ou aleatório (somente se disponível).

Após terminar as operações de leitura ou de escrita, o programa é obrigado a **fechar o fluxo**. Isto desassocia (ou desconecta) o fluxo do dispositivo e permite que outros programas utilizem este dispositivo.

Leitura e Escrita de Arquivos

Tipo de Dados

Em C, um fluxo de dados associado com arquivos é declarado como uma variável de tipo `FILE *`. Note que o asterisco faz parte do nome do tipo. É possível operar simultaneamente sobre vários arquivos, desde que cada um deles esteja associado com sua própria variável de tipo `FILE *`. Veremos mais detalhes sobre esse tipo de dados mais à frente.

A variável contém uma referência para o fluxo de dados. Essa variável será indicada como argumento em cada um dos comandos de leitura e de escrita. Desta forma, o programa sabe em qual fluxo se realizará a operação.

Abrir e Fechar Arquivo

Antes de realizar leitura ou escrita em um arquivo, é necessário abri-lo usando a função `fopen`. Os comandos estão no quadro a seguir:

```
FILE *arquivo;
arquivo = fopen("nome do arquivo", "modo");
...
fclose(arquivo);
```

A função `fopen` associa a variável `arquivo` com o fluxo que representa o conteúdo do arquivo desejado. Se `fopen` falhar, então o valor da variável `arquivo` será NULL.

Observação: Chamamos esta variável de `arquivo`, mas o nome poderia ser qualquer outro identificador válido da linguagem C.

O comando `fopen` possui dois argumentos: *nome do arquivo* e *modo*.

O *nome do arquivo* é um texto (ou variável tipo `char *`) que contém o nome do arquivo com o qual o programa deseja operar. Este nome pode incluir também o nome do diretório e da unidade de disco, bem como referências relativas ao diretório atual. O arquivo não precisa necessariamente existir previamente.

O *modo* é um texto (ou variável tipo `char *`) que informa quais operações serão realizadas sobre o arquivo. Deve-se escolher exatamente uma das opções abaixo:

- "r" – (*read*) Abre o arquivo para leitura somente. Se o arquivo não existir ou não puder ser encontrado, então `fopen` falha. A leitura iniciará no primeiro byte do arquivo. O fluxo será somente para leitura e de acesso sequencial.
- "w" – (*write*) Abre o arquivo para escrita somente. Cria o arquivo, se necessário. Se o arquivo já existir, ***seu conteúdo anterior será apagado***. O fluxo será somente para escrita e de acesso sequencial.
- "a" – (*append*) Abre o arquivo somente para escrita. Cria o arquivo se necessário. Se o arquivo já existir, *mantém o conteúdo anterior*. As operações de escrita serão realizadas sempre no final do arquivo, acrescentando novos dados. O fluxo será somente para escrita e de acesso sequencial.
- "r+" – Abre o arquivo para leitura e escrita. Se o arquivo não existir ou não puder ser encontrado, então `fopen` falha. O fluxo será de leitura e escrita, acesso aleatório.

- "w+" – Abre o arquivo para leitura e escrita. Cria o arquivo se necessário. Se o arquivo já existir, ***seu conteúdo anterior será apagado***. O fluxo será de leitura e escrita, acesso aleatório.
- "a+" – Abre o arquivo para leitura e escrita. Se o arquivo já existir, *mantém o conteúdo anterior*. As operações de escrita serão realizadas sempre no final do arquivo. O fluxo será de leitura com acesso aleatório, mas a escrita é sempre sequencial e no fim do arquivo.

Além destas seis opções, é possível adicionar-se duas letras, para escolher se o arquivo será de texto ou binário:

- "t" – (texto) O conteúdo do arquivo será interpretado como texto. Isto significa que operações de leitura e escrita sofrerão modificações de compatibilidade no conteúdo para ajustar alguns caracteres, dependendendo do sistema operacional.
- "b" – (binário) O conteúdo do arquivo será lido ou escrito sem modificações.

Após realizar todas as operações de leitura e/ou escrita, é necessário fechar o arquivo usando a função `fclose`. O único argumento de `fclose` é a variável com a referência do fluxo que desejamos fechar.

Ler do arquivo

A leitura de dados de um arquivo é semelhante à leitura de dados do teclado:

```
FILE *arquivo;

...
fscanf(arquivo, "formato", &variavel);
...
```

Ao invés da função `scanf`, utiliza-se uma variante especial, chamada `fscanf` (de *file scanf*). Os argumentos são:

- *Arquivo*: A referência para o fluxo que será lido.
- *Formato*: Um texto com indicadoes %d, %f, %c, %s, etc, para informar o que desejamos ler.
- *Variável*: Uma ou mais variáveis (separadas por vírgula), que armazenarão os valores lidos. Todas as variáveis (exceto as de tipo `char[]`), devem ser precedidas por &.

Para ler somente um caractere, podemos utilizar a função `fgetc`:

```
char c;
...
c = fgetc(arquivo);
...
```

A função `fgetc` retorna o próximo caractere que está no fluxo e avança a leitura em uma posição. Seu único argumento é a referência para o fluxo que está sendo lido.

Para ler uma linha inteira, existe a função `fgets`:

```
char linha[102];
...
fgets(linha, 100, arquivo);
...
```

A função `fgets` tem três argumentos. O primeiro é a variável (tipo vetor de *char*) que armazenará os caracteres que formam a linha a ser lida. No exemplo acima, chamamos esta variável de `linha`. O segundo argumento é o tamanho máximo da linha. Isto é necessário para evitar que `fgets` leia mais caracteres do que realmente cabem na variável `linha`. Neste caso, a linha será lida de forma incompleta. Note que o tamanho do vetor precisa ser duas unidades maior que o máximo de caracteres que podem ser lidos: um caractere para armazenar o '\n' que sinaliza o final da linha, outro para o '\0' que indica o final do vetor de caracteres. O último argumento é a referência para o fluxo correspondente ao arquivo.

Obeservações:

- Se o fluxo for de acesso sequencial, todos os caracteres lidos com `fscanf`, `fgetc` ou `fgets` serão "consumidos", ou seja, não será mais possível voltar para lê-los novamente.
- No caso de fluxos de acesso aleatório, existe um contador que indica a posição onde se inciará a próxima leitura. Nesse caso, `fscanf`, `fgetc` ou `fgets` aumentam o valor deste contador conforme o número de caracteres lidos. Para realizar novamente a leitura de um valor, é necessário retroceder este contador, operação que será estudada mais adiante.

Verificar fim do arquivo

Quando não conhecemos de antemão o tamanho de um arquivo, precisamos verificar a cada leitura se não ultrapassamos o final do arquivo.

A função `feof` retorna verdadeiro se a última operação (seja ela `fscanf`, `fgetc` ou `fgets`) falhou por não haver mais dados para serem lidos, pois foi atingido o final do arquivo. A função `feof` é utilizada tipicamente para controlar um **`while`** que realiza a leitura do arquivo:

```
...
while (!feof(arquivo)) {
      ...
      Operação de leitura
      ...
}
...
```

Outra forma de verificar o final do arquivo é comparar se o número de variáveis cujos valores foram atribuídos pelo `fscanf` é igual ao número de variáveis que esperávamos ler.

A função `fscanf` retorna quantas variáveis foram lidas. Este resultado pode ser utilizado para decidir se interrompemos ou não a leitura:

```
int variaveis_lidas;
int a, b, c;

while(fscanf(arquivo, “%d %d %d”, &a, &b, &c)!=0) {

      ...
}
```

Escrever no arquivo

A escrita também é semelhante à operação para imprimir dados na tela:

```
FILE *arquivo;

...
fprintf(arquivo, “fomato”, variavel);
...
```

Ao invés da função `printf`, utiliza-se uma variante especial, chamada `fprint` (de *file printf*). Os argumentos são:

- *Arquivo*: A referência para o fluxo no qual se vai escrever.
- *Formato*: Um texto com indicadores %d, %f, %c, %s, etc, para informar o tipo de dados que desejamos escrever.
- *Variável*: Uma ou mais expressões, variáveis ou constantes (separados por vírgula), cujos valores serão formatados de acordo com os respectivos indicadores.

Para escrever somente um caractere, podemos utilizar a função `fputc`:

```
char c;
...
fputc(c, arquivo);
...
```

A função `fputc` escreve o caractere armazenado em `c` no fluxo referenciado por `arquivo`.

Para escrever uma linha inteira, existe a função `fputs`:

```
char texto[] = “Conteudo do texto”;
...
fputs(texto, arquivo);
fputs(“Conteudo de outro texto”, arquivo);
...
```

A função `fputs` tem dois argumentos. O primeiro é o texto que deve ser escrito e o segundo é a referência para o fluxo associado com o arquivo.

Normalmente, o conteúdo escrito no fluxo apresenta um tempo de latência até ser definitivamente escrito no arquivo. Por exemplo, o computador pode aguardar várias chamadas para a função `fprintf` até realmente escrever os dados no arquivo. Desta forma, ele economiza operações de acesso ao disco, que são mais lentas.

Para forçar a escrita de todos os dados pendentes, utiliza-se a função `fflush`:

```
...
fprintf(arquivo, “fomato”, variavel);
...
fflush(arquivo);
...
```

Obeservações:

- Se o fluxo for de escrita sequencial, todos os caracteres escritos com `fprintf`, `fputc` ou `fputs` não poderão ser sobreescritos.
- No caso de fluxos de acesso aleatório, existe um contador que indica a posição onde se inciará a próxima escrita. As funções `fprintf`, `fputc` e `fputs` aumentam o valor deste contador conforme o número de caracteres escritos. Para sobrescrever valores já existentes, é necessário retroceder este contador, operação que será estudada mais adiante.
- Se o arquivo for aberto com o modo “a” ou “a+”, toda a escrita será realizada no final do arquivo, independente do valor do contador de posição de escrita.
- Quando estiver testando seu programa, lembre-se que os dados demoram para serem realmente escritos no arquivo. Para garantir que não existe nenhum dado pendente, chame a função `fflush`. Por outro lado, um número excessivo de chamadas de `fflush` causará uma queda de desempenho do seu programa.

Avançar e retroceder a posição de leitura ou escrita

Nos arquivos abertos com acesso sequencial, existe um contador indicando a posição da próxima leitura ou da próxima escrita. Note que o contador é compartilhado pelas duas operações.

Uma vez iniciada a leitura de um arquivo, com a função `rewind` pode-se sempre retroceder para a posição inicial:

```
FILE *arquivo;
arquivo = fopen(“nome do arquivo”, “modo”);
...
rewind(arquivo);
...
fclose(arquivo);
```

A função `rewind` é útil em situações quando escrevemos resultados obtidos pelo processamento em um arquivo (ao invés de um vetor na memória) e mais tarde desejamos lê-los novamente. Uma alternativa menos eficiente seria fechar o arquivo (com `fclose`) e abri-lo novamente (com `fopen`).

A qualquer momento, podemos consultar a função `ftell` para saber a posição atual da leitura ou da escrita:

```
long int posicao;
...
posicao = ftell(arquivo);
...
```

O valor retornado pela função `ftell` é sempre do tipo *`long int`*. A variável `arquivo` é a referência para o fluxo associado com o arquivo.

Os arquivos de acesso aleatório podem mudar a posição de leitura e escrita com a função `fseek`:

```
...
fseek(arquivo, deslocamento, referencia);
...
```

Os três argumentos de `fseek` são:

- *Arquivo*: referência para o fluxo associado com o arquivo.
- *Deslocamento*: quantos caracteres desejamos avançar (se positivo) ou retroceder (se negativo) em relação à posição referencial.
- *Posição referêncial*: de onde o deslocamento deve ser realizado.
  Deve ser uma das três opções:
  - *SEEK_CUR*: O deslocamento é em relação à posição de leitura e escrita atual. Portanto, `fseek(arquivo, 10, SEEK_CUR)`, pula dez caracteres sem ler os mesmos. Já `fseek(arquivo, -10, SEEK_CUR)`, retrocede dez caracteres.
  - *SEEK_END*: O deslocamento é em relação ao final do arquivo. Neste caso fazem sentido apenas deslocamentos negativos, uma vez que não é possível avançar além do fim do arquivo.
  - *SEEK_SET*: O deslocamento é em relação ao início do arquivo. Neste caso fazem sentido apenas deslocamentos positivos, uma vez que não é possível retroceder aquém do início do arquivo.

A função `fseek` raramente permite todas as combinações de argumentos. Os deslocamentos são exatos em arquivos binários. Em arquivos tipo texto, traduções realizada pelos comandos `fscanf` e `fprintf` podem introduzir imprecisão.

Por exemplo, em arquivos texto (que foram abertos com `fopen` e modo "t"), somente é possível fazer as seguintes operações com segurança: retroceder para o início e avançar até o final.

Fluxos padrão de entrada e saída

No início da execução do programa, existem três fluxos de dados pré-estabelecidos: `stdin`, `stdout` e `stderr`. Eles se comportam de forma bem semelhantes a arquivos, estudados na seção anterior, e aceitam as mesmas funções de leitura e escrita.

As funções `printf`, `putchar` e `puts` escrevem automaticamente em `stdout`. As funções `scanf`, `getchar` e `gets` lêem sempre de `stdin`. Até este momento, utilizamos estes fluxos sem conhecimento dos mesmos.

- `stdin`: Este fluxo está associado com a entrada padrão do programa. Normalmente isto corresponde ao teclado.
- `stdout`: Está associada com a saída padrão do programa. Normalmente, isto é um terminal ou o prompt do DOS.
- `stderr`: Também está associado com a saída padrão, mas este fluxo deve receber somente mensagens de erro.

Concluímos, portanto, que a chamada

```
printf("texto")
```

é, na verdade, equivalente à chamada

```
fprintf(stdout, "texto")
```

Da mesma forma,

```
scanf("indicadores", &variavel, ...)
```

é, na verdade, equivalente à

```
fscanf(stdin, "indicadores", &variavel, ...)
```