Arquivos

Instituto de Informática
Universidade Federal de Goiás
Introdução à Programação

12 de janeiro de 2024

Fluxo de Dados - Stream

Fluxo de Dados

- Os programas em C utilizam o conceito de fluxo de dados (streams) para comunicarem-se com dispositivos do computador, com arquivos em disco, ou com outros programas.
- Os fluxos permitem uma padronização das operações de entrada e saída:
  - Um conjunto fixo de funções existem para ler e escrever nos fluxos.
  - Há um mecanismo para associar fluxos a um dispositivo ou programa.
  - Esse modelo de comunicação permite ao programador enviar e ler dados para/de diversos dispositivos usando as mesmas funções de E/S.

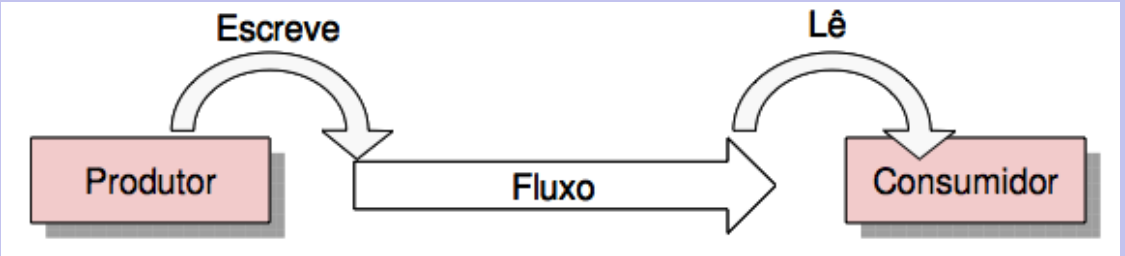
O Modelo de Fluxo

Modelo de um Stream

- O fluxo corresponde a uma abstração, implementada por uma área de memória e um conjunto de funções, que dão a idéia de existir um fluxo de dados entre um **produtor** e um **consumidor**.
- O fluxo funciona como um *buffer*, uma área de dados na memória principal utilizadas para guardar dados durante a comunicação entre o produtor e consumidor.
- Funciona também como uma fila: o produtor insere dados em uma extremidade do buffer e o consumidor obtém os dados na outra extremidade.

O Modelo de Fluxo

Ilustração



Produtor e Consumidor

- Produtor: programa ou dispositivo de E/S
- Consumidor: programa ou dispositivo de E/S

Tipos de Fluxo

Quanto ao Tipo de Operação

- Fluxo de leitura apenas.
- Fluxo de escrita apenas.
- Leitura e escrita.

Quanto ao Tipo de Acesso

- Sequencial
- Aleatório

Quanto ao Tipo de Dado

- Texto
- Binário

Denominados Arquivos

- Em C um fluxo é criado usando um ponteiro para o tipo FILE (definido em stdio.h)
- Ex. FILE * arq;
- FILE é definida como uma struct cujos campos contêm várias informações sobre o fluxo:
  - Um ponteiro para o buffer que armazena os dados do fluxo.
  - Tipo de operação permitida no fluxo.
  - Códigos de erro.
  - Ponteiros para funções.
  - Um *indicador da posição atual* no buffer.
  - Etc.

Operação de Abertura

fopen()

- Deve ser a primeira operação a ser executada sobre o fluxo e serve para:
  - Tenta alocar na memória uma struct FILE para armazenar dados sobre o fluxo.
  - Alocar o buffer que armazenará dados do fluxo e armazenar seu endereço na struct FILE correspondente ao fluxo
  - Associar um dispositivo de E/S ao fluxo
  - Definir as operações de E/S que são permitidas no fluxo.
  - Definir o tipo de acesso no fluxo
  - Definir o tipo de dados do fluxo (texto ou binário);

Operação de Abertura

Formato Geral

FILE *f = fopen (nome-**do**-arquivo, modo);

- nome-do-arquivo é uma constante string ou uma variável char* que armazena o caminho no disco onde se encontra o arquivo.
- modo é uma constante string ou uma variável char* que descreve o tipo de operação possível no arquivo, o tipo de acesso e o tipo de dados

Operação de Abertura

Modos de abertura - Acesso sequencial

- *r* - abre o arquivo para leitura sequencial apenas. A função fopen() retorna NULL se o arquivo não existir.
- *w* - abre o arquivo para escrita sequencial apenas. Se o arquivo não existir ele é criado. Se existir seu conteúdo original é apagado.
- *a* - abre o arquivo somente para escrita. Se o arquivo não existir, ele é criado. Se existir, não é apagado. Permite escrever apenas ao final do arquivo.

Operação de Abertura

Modos de abertura - Acesso aleatório

- *r+* - abre o arquivo para leitura e escrita. Se arquivo não existir, retorna NULL. O fluxo é de leitura e escrita e acesso aleatório.
- *w+* - abre o arquivo para leitura e escrita. Se o arquivo não existe, ele é criado. Se existe, seu conteúdo será apagado. O acesso é aleatório.
- *a+* - abre o arquivo para leitura e escrita. Se o arquivo existir, mantém o conteúdo existente. A leitura pode ser feita de modo aleatório, mas a escrita só pode ser feita ao final do arquivo.

Operação de Abertura

Modos de abertura - Tipo de dados

- Acrescentando-se um *b* na string de modo indica-se que o fluxo é binário.
- Acrescentando-se um *t* na string de modo indica-se que o fluxo é um arquivo de texto. Na ausência de *b* e *t* no modo, o arquivo é aberto como arquivo tipo texto.

- Arquivos binários armazenam os dados em formato binário. São utilizados para armazenar em disco os dados armazenados em formato binário na memória do computador: int, floats, structs, etc.
- Arquivos texto armazenam sequências de caracteres.

Operações de Leitura de Arquivo tipo Texto

Função fgetc()

- Cabeçalho: **int** fgetc(File* pFile)
- retorna o próximo caractere de um fluxo tipo texto e avança o ponteiro do arquivo para o próximo caractere.
- Embora retorne valores da tabela ASCII. O tipo retornado é um **int**, porque retorna EOF (-1) se o fim do arquivo for encontrado (não há um caractere para representar fim de arquivo). Por isso, a função não pode ser do tipo **char**, pois qualquer um dos 256 valores da tabela ASCII ou -1 podem ser retornados.
- O arquivo deve ter sido aberto em modo texto e com alguma das opções que permite leitura

Exemplo

Leitura de Arquivo e sua Apresentação na Tela

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 int main ()
4 {
5   FILE * pFile;
6   int c;
7   int n = 0;
8         //Abrir um arquivo tipo texto apenas para leitura sequencial
9   pFile=fopen ("Arquivo.txt","rt");
10        //Testar se o arquivo existe
11  if (pFile==NULL){
12    printf("Arquivo nao foi encontrado - Terminando o programa.\n");
13    exit(1);
14  }
15        //Leitura caractere por caractere
16  c = fgetc (pFile);
17  while (c!=EOF) {
18    putchar (c);
19    c=fgetc(pFile);
20  }
21  // Fechando o arquivo.
22  fclose (pFile);
23  return 0;
24 }
```

Operações de Leitura de Arquivo tipo Texto

Entrada formatada com fscanf()

- Cabeçalho:
  int fscanf (FILE *arq, **const char** *format, ...);
- Tem a mesma funcionalidade da função scanf (), porém a entrada é feita a partir do arquivo indicado por *arq.
- Retorna o número de itens lido se conseguir ler algum dado. Se não conseguir ler nenhum dado por causa de um erro na entrada ou por ter encontrado o fim do arquivo, retorna EOF.
- O arquivo deve ter sido aberto em modo texto e com alguma das opções que permite leitura.

Operações de Escrita de Arquivo tipo Texto

Função fputc()

- Cabeçalho: **int** fputc (**int** c, FILE * arq);
- Imprime o caractere passado pelo parâmetro c no arquivo indicado por arq.
- Retorna o mesmo caractere recebido no parâmetro c, se conseguir realizar a escrita.
- Retorna EOF caso ocorra um erro durante a escrita.
- O arquivo deve ter sido aberto em modo texto e com alguma das opções que permita a escrita.

Exemplo

Cópia de Texto

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main ()
4 {
5   FILE * pFile1, *pFile2;
6   int c;
7   int n = 0;
8   pFile1=fopen ("Arquivo.txt","rt");
9   if (pFile1==NULL){
10    printf("Arquivo nao foi encontrado - Terminando o programa.\n");
11    exit(1);
12  }
13  pFile2=fopen("Copia.txt", "wt");
14  c = fgetc (pFile1);
15  while (c!=EOF) {
16    fputc (c, pFile2);
17    c=fgetc(pFile1);
18  }
19  fclose (pFile1);
20  fclose(pFile2);
21  return 0;
22 }
```

Operações de Escrita de Arquivo tipo Texto

Saída formatada com `fprintf()`

- Cabeçalho:
  int fprintf (FILE *arq, **const char** *formato, ...);
- Tem a mesma funcionalidade da função `printf ()`, porém a saída é feita a partir do arquivo indicado por *arq.
- Retorna o número de caracteres escritos, caso consiga realizar a escrita com sucesso. Retorna um número negativo em caso contrário.
- O arquivo deve ter sido aberto em modo texto e com alguma das opções que permite escrita.

Fluxos Especiais do tipo Texto

Fluxos Padrões

- Como dito anteriormente, os fluxos permitem padronizar a entrada e a saída de modo que fiquem independentes do dispositivo usado. Assim, operações semelhantes são feitas tanto para arquivos em disco como para o teclado ou terminal, por exemplo.
- Para todos os programas em C, mesmo os que não abram arquivos explicitamente, são criados alguns fluxos automaticamente. Quatro desses fluxos são os seguintes: *stdin, stdout, stderr, stdprn*.

Fluxos Especiais do Tipo Texto

Fluxos Padrões - Detalhamento

- stdin - fluxo de entrada padrão - geralmente associado ao teclado, é utilizado pelas funções de entrada que não usam explicitamente arquivos: `scanf()`, `getchar()`.
- stdout - fluxo de saída padrão - geralmente associado ao terminal. Utilizado por funções que não usam explicitamente arquivos: `printf()`, `putchar()`.
- stderr - fluxo de mensagens de erro - pode ser utilizado para enviar mensagens de erro para um fluxo diferente do stdout.
- stdprn - fluxo destinado a impressão ao qual se conecta um impressora.

Uso dos Fluxos Padrões

Entrada, Saída, Erro e Impressão

- Os quatro fluxos são automaticamente abertos e automaticamente fechados. São do tipo texto.
- Equivalência:

`printf("Um dois");`	`fprintf(sdtdout, "Um dois");`
`scanf("%d", &x);`	`fscanf(sdtdin, "%d", &x );`

- Impressão de mensagem de erro em *stderr*:
  - `fprintf(stderr, "Nao ha memoria suficiente");`
  - `perror("Nao ha memoria suficiente");` (definida na stdio.h).
- Saída na impressora padrão: `fprintf(stdprn, "Um dois");`

Arquivos binários

Fluxos de bytes

- Em um arquivo binário, os bytes são transmitidos, em uma operação de escrita, como estão na memória RAM. Assim, se um valor do tipo int for gravado em um arquivo binário, seus bytes são copiados como estão armazenados e não são traduzidos em caracteres como o ocorre quando se imprime no formato "%d"da função `fprintf()`;
- De forma análoga, durante uma operação de leitura de um arquivo binário os bytes armazenados no arquivo são copiados como estão no arquivo para a memória RAM do computador.
- Dado que é feita uma transmissão de bytes quando o arquivo é binário, pode-se transferir grandes blocos de bytes de uma vez entre a memória e o dispositivo de E/S.

Operação de Escrita em Arquivos Binários

Função fwrite()

- Cabeçalho: `size_t fwrite(const void * ptr, size_t tam, size_t cont, FILE * arq);`
- Transfere tam bytes a partir do endereço ptr da RAM $cont$ vezes para o arquivo cujo fluxo está indicado por arq.
- O indicador de posição do arquivo é deslocado para frente $cont * tam$ bytes.
- A função retorna o número de blocos de tamanho tam que foram gravados. Se o valor de retorno for diferente do parâmetro $cont$ é porque houve um erro de escrita e a operação falhou em transferir a quantidade de blocos especificada em $cont$.

Operação de Leitura em Arquivos Binários

Função fread()

- Cabeçalho: size_t fread(**const void** * ptr, size_t tam, size_t cont, FILE * arq);
- Transfere $cont$ blocos de tam bytes do arquivo arq e os armazena a partir do endereço ptr da RAM .
- O indicador de posição do arquivo é deslocado para frente $cont * tam$ bytes.
- A função retorna o número de blocos de tamanho tam que foram lidos. Se o valor de retorno for diferente do parâmetro $cont$ é porque o arquivo terminou, ou houve um erro de leitura e a operação falhou em transferir a quantidade de blocos especificada em $cont$.

Verificando o Fim em Arquivos Binários

Função feof()

- Arquivos binários não podem usar EOF para checar fim de arquivo, pois os bytes podem representar qualquer coisa na memória. A escrita de uma variável do tipo char contendo o valor -1 (EOF) seria interpretada como fim de arquivo.
- Na linguagem C há uma função para detectar fim de arquivo. É a função feof().
- Cabeçalho: `int feof ( FILE * arq );`
- A função retorna um valor diferente de zero se o fim do arquivo foi atingido e zero, em caso contrário.
- Apesar de ser necessária para detecção de fim em arquivos binários, feof() pode ser utilizada também com arquivos texto.

Acesso Aleatório a Arquivos

Mudança do Indicador de Posição do Arquivo

- O *indicador de posição* de arquivo é um valor que até agora foi utilizado indiretamente: (a) após a abertura do arquivo ele indica o início do arquivo (b) Após cada operação de leitura/escrita ele avança de acordo com o número de bytes lidos/escritos.
- É possível mudar diretamente o indicador de posição de arquivo através da função `fseek()`
- Após a mudança ,é possível ler ou gravar em uma outra posição do arquivo, fazendo com que o processamento da leitura/escrita deixe de ser sequencial. Denominamos esse tipo de acesso a qualquer posição de *acesso aleatório*.
- A escrita/leitura nesse caso só é possível se o arquivo foi aberto no modo acesso aleatório.

Acesso Aleatório a Arquivos

A função fseek()

- Cabeçalho:
  int fseek (FILE * arq, **long int** desloc, **int** origem);
- O argumento *desloc* corresponde a número de bytes que o indicador de posição deve ser deslocado em relação a *origem*.
- *origem* pode ser uma das seguintes constantes definidas em stdio.h:

SEEK_SET	Início do arquivo
SEEK_CUR	Posição atual do indicador de posição
SEEK_END	Final do arquivo

- A função fseek() retorna zero se executar corretamente e um outro valor em caso contrário.

Outras Funções para Arquivos

A função ftell()

- Cabeçalho: **long int** ftell (FILE * arq);
- Retorna o valor do indicador de posição atual do arquivo cujo ponteiro é indicado por arq.
- Para arquivos binários o **valor do indicador corresponde ao número de bytes entre o início do arquivo e a posição atual do indicador**.
- Para arquivos texto, o valor pode não ter significado, mas ainda pode ser usado para voltar à posição atual, por guardar o valor atual em uma variável **long int** e usar futuramente essa variável como parâmetro para fseek().
- Em caso de falha, a função retorna -1.

Outras Funções para Arquivos

A função ferror()

- Cabeçalho: **int** ferror (FILE * arq);
- Verifica se um erro foi indicado na struct FILE para o fluxo arq.
- Um valor diferente de zero retornado indica que houve algum erro com a última operação executada sobre o arquivo.
- O valor de indicação de erro na struct FILE é limpado sempre que uma das operações: rewind() e freopen() ocorre sobre o fluxo

Outras Funções para Arquivos

A função fflush()

- Cabeçalho: **int** fflush (FILE * stream);
- Se fluxo foi aberto para escrita ou se foi aberto para leitura/escrita e a última operação foi de escrita, a função descarrega o buffer do fluxo no dispositivo de saída a ele associado.
- Se arq for NULL, todos os buffers de todos fluxos na situação anterior são descarregados.
- A função não fecha os fluxos.
- Retorna zero se obteve sucesso e EOF em caso contrário.

Outras Funções para Arquivos

A função rewind()

- Cabeçalho: **void** rewind (FILE * arq);
- Faz com que o indicador de posição do arquivo aponte para o início do fluxo arq.

Outras Funções para Arquivos

A função freopen()

- Cabeçalho: FILE * freopen (**const char** * nomearq, **const char** * modo, FILE *arq);
- Tenta fechar o fluxo atual indicado por arq. Reutiliza o fluxo, abrindo-o com associação ao arquivo especificado em $nomearq$ e no modo especificado na string $modo$.
- Essa função é especificamente útil para redirecionar a saída padrão dentro do programa para um outro dispositivo ou arquivo em disco.
- Se executar com sucesso, a função retorna o ponteiro para arq, caso contrário, retorna NULL.

Fechamento de Fluxos

A função fclose()

- O fechamento de um fluxo consiste em:
  - Realizar descarga do conteúdo do buffer no dispositivo de saída.
  - Fazer a desvinculação do fluxo com o dispositivo através de chamadas ao sistema operacional
  - Liberar a struct FILE alocada para representar o fluxo.
- Cabeçalho da função: `int fclose ( FILE * stream );`
- Retorno:
  - Zero, se o fluxo foi fechado com sucesso.
  - EOF em caso de falha