Acelerando a Entrada e Saída de Dados com as Funções *getchar_unlocked* e *putchar_unlocked*

Rafael G. Trindade[1], Iago da Cunha Corrêa[1], João V. F. Lima[2], Rafael Boufleuer[2]

[1]Núcleo de Ciência da Computação – Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

[2]Departamento de Linguagens e Sistemas de Computação – Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

{rtrindade,icorrea,jvlima,rboufleuer}@inf.ufsm.br

Abstract. *This article presents function tests in order to optimize the performance of reading and writing data and make them viable alternatives in a controlled environment for other reading and writing functions. The developed functions uses the character capture function "getchar_unlocked" and the character impression function "putchar_unlocked", and in the article are presented the performance tests of the developed functions and the comparison with the obtained results with natives functions of the C and C++ languages.*

Resumo. *Este artigo apresenta testes de funções buscando otimizar a execução da leitura e escrita de dados e torná-las alternativas viáveis dentro de um ambiente controlado para demais funções de entrada e saída. As funções desenvolvidas utilizam a função de captura de caracteres "getchar_unlocked" e a função de impressão de caracteres "putchar_unlocked", e ao decorrer do artigo serão apresentados testes de desempenho das funções desenvolvidas e a comparação com os resultados obtidos com funções nativas das linguagens C e C++.*

1. Introdução

Em computação, é comum a busca por melhores resultados para algoritmos em questão de tempo e quantidade de processamento. Ambientes no qual a busca pelo speedup (proporção de ganho tempo de execução) ideal é constante, buscam fazer proveito de toda a sua capacidade de processamento em paralelo utilizando-se de bons programas e bibliotecas. Um pouco mais próximo da vida acadêmica, encontram-se campeonatos de programação - como a Maratona de Programação da SBC - onde competidores (estudantes) buscam estratégias rápidas, inteligentes e eficientes para resolução dos problemas propostos pela organização da competição. Neste cenário, é perceptível que dois dos 'gargalos' de qualquer algoritmo são a entrada e saída de dados.

Também pode-se perceber que sistemas criados para lidar com big data, como por exemplo, SADs (Sistemas de Apoio à Decisão) e, ou, SIGs (Sistemas de Informações Gerenciais), são na maioria das vezes, implementados nas linguagens de C e C++, ambas por serem menos custosas em tempo de processamento e mais ágeis, isso se comparadas com linguagens mais completas como Java, C# e Python. Embora sejam eficientes, é dificil evitar que o processamento de dados seja ineficiente, logo, é perceptível que as funções desenvolvidas podem ajudar a evitar o gargalo provocado pela quantia de dados

[Beuren 2001], além de permitir que o speedup resultante da implementação das funções apresente melhores resultados. Através das funções que utilizam *putchar_unlocked* e *getchar_unlocked*, procura-se melhorar a eficiência em que as linguagens C e C++ lêem e escrevem dados pois, agilizando tais processos, o gargalo provocado pelo processamento dos dados poderá ser reduzido, resultando em sistemas mais fluídos e suaves.

Geralmente as funções nativas de leitura e escrita de dados da maioria das linguagens são construídas de forma que possam ser utilizadas genericamente (para diversos tipos de dados), além de toda sua arquitetura conter testes e tratamentos para possíveis erros relacionados ao processamento de IO (input/output). A linguagem de programação C, muito utilizada por estar teoricamente em uma zona intermediária entre linguagens de baixo e alto nível [Schildt 1996], possui diversas funções para leitura e escrita de dados, e entre elas estão as funções *getchar_unlocked* e *putchar_unlocked*, as quais tem seu uso abordado nesse artigo.

Os objetivos específicos desse artigo podem ser encontrados na seção 2, que apresenta como essas funções serão utilizadas para a diminuição do gargalo de tempo com IO. A seção 3 contém as formas como as funções foram implementadas para manipular a leitura e a escrita de diferentes tipos de dados, assim como trechos de códigos. As médias de tempo resultantes de suas execuções e gráficos contendo comparações com tempos de outras funções das linguagens C e C++ podem ser conferidas na seção 4. Ao final, nas seções 5 e 6, interpretam-se os valores obtidos e disserta-se sobre os mesmos.

2. Metodologia

Antes de aplicar qualquer solução, precisa-se entender o que são e quais as vantagens e desvantagens das funções *getchar_unlocked* e *putchar_unlocked*. As subseções a seguir apresentam as descrições de ambas as funções bem como formas de implementá-las para leitura e escrita de diferentes tipos primitivos de dados.

2.1. *putchar_unlocked* e *getchar_unlocked*

As funções *getchar_unlocked* e *putchar_unlocked* são versões modificadas das funções *getchar* e *putchar*, presentes na biblioteca de entrada e saída padrão da linguagem C (*stdio*). A função *getchar_unlocked* lê um caractere do fluxo de entrada padrão da linguagem (*stdin*), assim como a função *putchar_unlocked* escreve um caractere no fluxo de saída padrão da linguagem (*stdout*). São equivalentes ao uso de *getc_unlocked*(*stdin*) e *putc_unlocked*(*stdin*), respectivamente. Ambas não são thread-safe (seguras para uso em threads), logo os usos das funções aqui descritas são aconselhados quando o conjunto de dados a ser escrito ou lido é grande e a velocidade de execução é importante, mas a segurança para uso de threads não.

A função *getchar_unlocked* retorna o próximo caractere disponível no *stdin*. Se o *stdin* está no final do arquivo, o indicador de fim de arquivo (EOF) é retornado, assim como se algum erro de leitura ocorrer, o indicador de erro do *stdin* é ativado e a função retorna um EOF. A função *putchar_unlocked* escreve o caractere recebido como argumento no *stdout*, e se a escrita falhar por alguma razão, ela retorna EOF. [IBM 2015]

2.2. Implementação para entrada de dados

Apesar das funções somente tratarem caracteres, é possível utilizá-las para a implementação de funções que não necessariamente sejam voltadas para leitura e im-

pressão do conteúdo de variáveis somente com o tipo de dado char (caractere). É possível implementar a leitura e escrita de diversos outros tipos de dados, como por exemplo *int* (inteiro), *unsigned int* (inteiro positivo), *float* (números reais), *double* (números reais com precisão dupla) e *string* (cadeias de caracteres).

A Figura 1 apresenta a forma como a leitura de números inteiros (variáveis do tipo *int*) pode ser executada retirando-se caracteres vindos do *stdin* através do uso da função *getchar_unlocked*. Apesar de aparentar complexidade, a lógica por trás é simples, assim como sua velocidade em relação às funções concorrentes é superior visto que limita-se ao uso de operações aritméticas muito simples do ponto de vista do processador (soma e subtração) e operações binárias (comparações, shifts e negação *bitwise* - *bit* a *bit*). Para processar a leitura, a função inicialmente ignora todos os caracteres diferentes dos referentes à números (linhas 11 a 14). Se encontrar um hífen ('-'), a função seta uma *flag* indicadora que negativará o resultado ao final da leitura (linha 12). Uma vez alcançado um caractere referente à um número, a função multiplica por 10 o valor contido na variável de destino usando *shifts* e então soma o valor numérico referente ao lido (linha 16). Ao encontrar um caractere não numérico, o laço de leitura se encerra. Se a *flag* que indica que o resultado precisa ser negativo está ativa, então a função realiza um complemento de dois no valor, invertendo logicamente cada *bit* do valor e então somando com 1 (linha 17).

```
1  /*
2  * Escreve um numero inteiro no stdout
3  * int n     -> Valor a ser impresso
4  * return    -> Sem retorno.
5  */
6  void escreve(int n){
7      long int i=1, q;
8      while (i<n) i = (i<<3) + (i<<1);
9      i /= 10;
10     while (i>0){
11         q = n/i;
12         putchar_unlocked(q + '0');
13         n -= q*i;
14         i /= 10;
15     }
16 }
```

Figura 1. Trecho de Código da leitura de números inteiros com a função *getchar_unlocked*

Para a implementação com números de ponto flutuante, basta adicionar um teste para verificar se um dos caracteres lidos foi um ponto ('.') e então basta acumular os números lidos a seguir em uma variável diferente da de destino, logo após dividí-los por 10^n, onde n é a quantia de casas decimais após a virgula, para posteriormente somar à variável de destino.

2.3. Implementação para saída de dados

Apesar de análogo às implementações para entrada, usar a função *putchar_unlocked* de uma forma que gere uma saída mais específica tem seus problemas de complexidade, apesar de que, quando aplicada a uma cadeia de caracteres, uma lógica simples pode ser aplicada utilizando-se de apenas um laço de repetição para emitir sequencialmente os caracteres da cadeia para o *stdout*. Entretanto, para valores numéricos, a função necessita de diversas operações aritméticas computacionalmente custosas como divisões e multiplicações. A Figura 2 apresenta o código implementado para escrita de números inteiros utilizando a função *putchar_unlocked*.

```
/*
* Le numeros inteiros.
* int *n    -> Ponteiro a armazenar o valor lido.
* return    -> True (1) ou EOF se fim do arquivo.
*/
int le(int *n) {
    *n = 0;
    char ch = getchar_unlocked(), sign = 0;
    if (ch==EOF)
        return EOF;
    while (ch < '0' || ch > '9') {
        if (ch == '-') sign = 1;
        ch = getchar_unlocked();
    }
    while (ch >= '0' && ch <= '9')
        *n = (*n<<3) + (*n<<1) + ch - '0', ch = getchar_unlocked();
    if (sign) *n = ~(*n) + 1;
    return 1;
}
```

Figura 2. Trecho de Código responsável pela escrita de números inteiros com a função *putchar_unlocked*

Multiplica-se uma variável i inicializada como 1 (um) por 10 (dez) sucessivamente até que o valor supere o do valor n a ser escrito. Após superar, divide-se por 10 para que fique com a casa decimal mais alta no mesmo nível do valor a ser escrito. Então, repetidamente - enquanto i for maior que 0 (zero) -, pega-se o piso do quociente q da divisão de n por i e imprime-se na tela, diminui-se a casa decimal mais alta de n através de uma subtração e a seguir divide-se i por 10 para que ele represente a casa decimal inferior.

3. Resultados

Realizaram-se testes de leitura e escrita com as funções que utilizavam *getchar_unlocked*, comparando os resultados obtidos com as funções *cin* e *cout* (nativas do C++) e *scanf* e *printf* (nativas da linguagem C). Foram feitos testes para 1, 10, 100, 1000, 10 mil, 100 mil, 1 milhão e 10 milhões de leituras e escritas da macro *INT_MAX*, da biblioteca "*limits.h*". A discrepância dos valores dos testes ajudou na conclusão do trabalho pois, era o objetivo do mesmo abranger uma grande gama de resultados. Para a leitura dos números com precisão dupla, utilizou-se a mesma quantidade de leituras para um valor fixo em ponto flutuante com precisão dupla, e, para a escrita de cadeia de caracteres, foi utilizada uma cadeia fixa com 50 caracteres. As Figuras 3 e 4 apresenta os gráficos que mostram a relação entre tempo e quantidade de leituras para inteiros e números com precisão dupla. As Figuras 5 e 5 apresenta os gráficos que mostram a relação entre tempo e quantidade de escritas para inteiros e cadeias de caracteres. As funções implementadas utilizando as funções *getchar_unlocked* e *putchar_unlocked* tem seus tempos representados nos gráficos por linhas de cor azul, enquanto que as funções nativas da linguagem C (*scanf* e *printf*) são representadas pela cor vermelha e as de C++ (*cin* e *cout*) pela cor amarela.

4. Discussão

Nota-se, através dos resultados obtidos, que as funções que utilizam *getchar_unlocked* e *putchar_unlocked* não são aconselháveis para todos os casos. Embora ambas abstenham-se da característica *thread-safe*, a eficiência delas na leitura e na escrita de dados difere.

A velocidade superior da função *getchar_unlocked* em comparação à outros métodos de leitura deve-se ao fato de que todos os gastos de tempo com exclusão mútua são evitados, pois a função não verifica *locks*, e quando tem-se uma única linha de

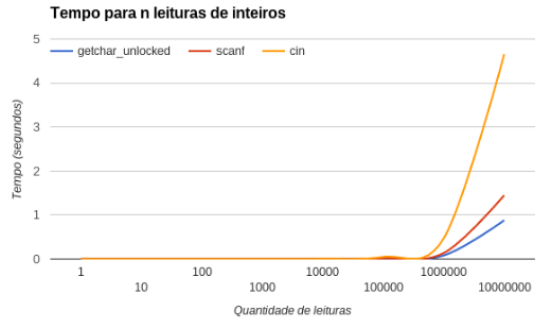

Figura 3. Gráfico dos Tempos de execução pela quantidade de leituras de números inteiros

Figura 4. Gráfico dos Tempos de execução pela quantidade de leituras de números com precisão dupla

execução isso não faz diferença. Entretanto, é possível usá-la com segurança se o programador implementar a *thread* que a invoca com travamento do *stdin* usando *flockfile* (ou *ftrylockfile*) e *funlockfile*.

Enquanto a *getchar_unlocked* se demonstra mais eficiente que a *scanf* e da *cin* para a leitura de dados, com e sem otimização, a função *putchar_unlocked* não apresenta o mesmo ganho de tempo, mesmo não verificando *locks*. Para a escrita de inteiros, a função composta de *putchar_unlocked* necessita de um uso abusivo de inúmeras divisões e multiplicações, o que de fato atrasa a execução da escrita. Já para uma determinada cadeia de caracteres, a função *putchar_unlocked* sofre atrasos devido ao excesso de *overheads*. Um *overhead* ocorre quando há inúmeras chamadas de funções e de sistema por trás de uma função aparentemente simples, saturando o conjunto de instruções de baixo nível a ser executado e resultando em um gargalo na saída de dados. Funções mais complexas como *printf* e *cout* emitem o conteúdo a ser escrito para o *stdout* fazendo-se uso de *buffers* de caracteres, podendo emitir toda uma cadeia à impressão sem ter de usar um laço de repetição para isso, fazendo apenas uma chamada de sistema e diminuindo o *overhead*.

5. Conclusão

Logo, pode-se concluir com o presente estudo que se tratando de uma quantidade massiva de dados, o programador deve utilizar a forma mais viável para lidar com tais dados, porém, a segurança dos dados também deve ser levado em consideração.

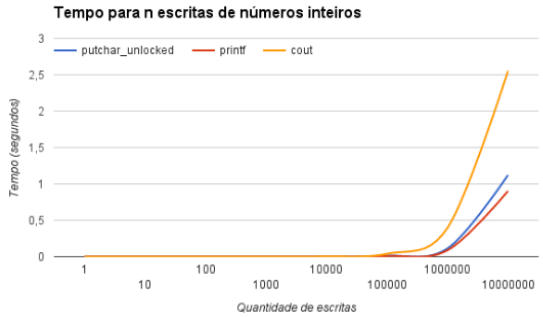

Figura 5. Gráfico dos Tempos de execução pela quantidade de escritas de números inteiros

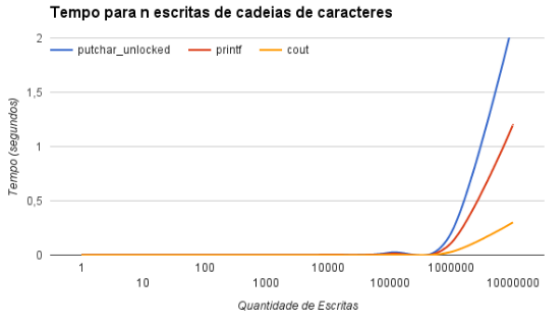

Figura 6. Gráfico dos Tempos de execução pela quantidade de escritas de cadeia de caracteres

Para ambientes controlados, onde o programador sabe quais tipos de dado o programa receberá e tem a certeza que esses dados sempre chegarão corretos (passando por alguma verificação antes ou não), as funções de entrada de dados propostas neste trabalho podem se tornar de grande valia, se implementadas corretamente. Um possível ambiente de utilização são campeonatos de programação, onde entradas e saídas geralmente não devem possuir qualquer tipo de erro, acelerando o tempo final de execução sem tornar o algoritmo menos eficiente.

Como já explanado no presente artigo, vivemos em uma época em que a informação e o conhecimento são de valores expressivos, logo, devem ser tratados com a meticulosidade necessária para que a informação continue contribuindo com o avanço tecnológico da humanidade.

Referências

Beuren, I. M. (2001). Sistema de informações executivas: Suas características e reflexões sobre sua aplicação no processo de gestão. Disponível em: http://www.scielo.br/pdf/rcf/v12n26/v12n26a01.

IBM (2015). Stdio with explicit client locking. Disponível em: http://www-01.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.bpxbd00/getclock.htm.

Schildt, H. (1996). *C - Completo e Total*. Makron Books, 3rd edition.