5º Exercício Prático
Desenvolvido no Laboratório

Objetivo

Utilizar a ferramenta Valgrind de levantamento de perfis de execução de programas, levantando *hotspots* para otimizar, e modificar o programa tetando melhorar o seu desempenho.

Materiais

1. Compilador GCC
2. Ferramentas de análise de desempenho Callgrind do Valgrind
3. Arquivo Acessos.cpp
   - **ATENÇÃO**, use o Acessos.cpp desta aula pois ele é ligeiramente diferente do Acessos.cpp usado na aula passada.
4. Arquivo com textos: biblia-em-txt.txt.

Desenvolvimento

O experimento de hoje, como no da aula passada, usa um programa que faz a contagem de ocorrências de uma letra em um arquivo de texto. No desenvolvimento deste experimento serão analisadas as funções que mais precisam de tempo para executar e como elas podem ser substituídas para melhorar o desempenho.
Para identificar os pontos onde a otimização pode gerar melhores resultados (*hotspots*), será usada a ferramenta Callgrind do Valgrind.

Executando o programa

1. Compile o programa Acessos.cpp
   - gcc -g Acessos.cpp -o Acessos -masm=intel
   - O “-masm=intel” define que o dialeto – mnemônicos e sintaxe – fazendo que o compilador aceite as definições encontradas nos manuais da Intel.
2. Execute o programa.
   - ./Acessos
3. Observe a saída do programa Acessos e responda:

   Quantas vezes a letra ‘a’ aparece no texto?

   Ocorrências da letra ‘a’: _____

   Qual foi o tempo observado na execução?

   Tempo 1: _____ s

Primeira otimização

Na aula passada foi usado o Valgrind para identificar a função que tinha maior impacto na execução do programa. Esta função é a *fgetc*! A função *fgetc* precisa realizar várias verificações relativas ao arquivo que está sendo lido e isso ocorre a cada caractere do arquivo de entrada, o que demanda tempo. Para reduzir esse tempo, pode-se ler mais dados do arquivo de uma só vez, uma possibilidade é ler uma linha de cada vez. Para isso usaremos a função *fgets*.

1. Abra o arquivo Acessos.cpp em um editor de texto
2. Altere a função contaLetra() com o código abaixo

```
int contaLetra(FILE *pArq, char c) {
  int cont = 0;
  char linha[MAX_CARACTERES_POR_LINHA];
  while (fgets(linha, MAX_CARACTERES_POR_LINHA, pArq)) {
    for (int i = 0; i < strlen(linha); i++) {
      if (linha[i] == c) {
        cont++;
      }
    }
  }
  return cont;
}
```

   Quantas vezes a letra ‘a’ aparece no texto?

   Ocorrências da letra ‘a’: _____

   Qual foi o tempo observado na execução?

   Tempo 2: _____ s

3. Execute o programa usando o Valgrind
   - valgrind --tool=callgrind ./Acessos
4. Após a execução, para visualizar o perfil, use comando:
   - callgrind_annotate --auto=yes --show-percs=yes callgrind.out.<id>

   Observando a saída do callgrid_annotate, responda qual é a função das bibliotecas padrões da linguagem C que agora tem o maior impacto no tempo de execução do programa?

   Nome da função: _____

Segunda otimização

A função *strlen* é chamada em cada interação e a cada chamada ele busca na cadeia de caracteres o caractere nulo para identificar o fim da cadeia e calcular o seu tamanho, ou seja, em uma linha de *n* caracteres, esta função é chama *n* vezes, buscando o nulo a cada vez nos *n* caracteres da linha, mesmo que o valor retornado para a linha seja sempre o mesmo. Para reduzir o tempo observado no programa anterior, pode-se chamar a função *strlen* uma única vez logo após a leitura da linha,

atribuir o tamanho a uma variável e usá-la na condição do *for*.

1. Altere o programa da primeira otimização chamando o *strlen* uma única vez para cada linha. Quantas vezes a letra ‘a’ aparece no texto?

   Ocorrências da letra ‘a’: _____

   Qual foi o tempo observado na execução?

   Tempo 3: _____ s

   Observando a saída do callgrid_annotate, responda qual é a função das bibliotecas padrões da linguagem C que agora tem o maior impacto no tempo de execução do programa?

   Nome da função: _____

Terceira otimização

Vamos tentar agora fazer uma única leitura do arquivo, colocar todos os dados na memória e então fazer a contagem da letra.

Em um arquivo de texto sem formatação, as letras estão armazenadas uma após a outra, sendo que cada final de linha é marcado com um caractere ‘\n’ que é seguido pela primeira letra da linha seguinte.

1. Altere a função contaLetra() do programa anterior com o código abaixo

```
int contaLetra(FILE *pArq, char c) {
  int cont = 0;
  int tam = fread(tudo, MAX_CARACTERES_NA_BIBLIA, 1, pArq);
  for (int i = 0; i < MAX_CARACTERES_NA_BIBLIA; i++) {
    if (tudo[i] == c) {
      cont++;
    }
  }
  return cont;
}
```

   Quantas vezes a letra ‘a’ aparece no texto?

   Ocorrências da letra ‘a’: _____

   Qual foi o tempo observado na execução?

   Tempo 4: _____ s

Última otimização

Muitas vezes é possível utilizar linguagem de montagem para otimizar uma seção de um programa. Altere a busca na função contaLetra para que esta busca seja feita em linguagem de montagem.

1. Altere a função contaLetra() do programa anterior com o código abaixo

```
int contaLetra(FILE *pArq, char c) {
  int cont = 0;
  int tam = fread(tudo, MAX_CARACTERES_NA_BIBLIA, 1, pArq);
  asm("   mov   rdi, %[tudo]    \n"
      "   mov   rcx, %[MAX_CARACTERES_NA_BIBLIA]\n"
      "   mov   al, %[c]        \n"
      "   xor   rdx, rdx        \n"
      "   xor   rbx, rbx        \n"
      "   add   rcx, rdi        \n" // calcula endereço final
      "repete:                  \n"
      "   cmp   [rdi], al       \n"
      "   sete  dl              \n"
      "   add   rdi, 1          \n"
      "   add   rbx, rdx        \n"
      "   cmp   rdi, rcx        \n"
      "   jnz   repete          \n"
      "   mov   %[cont], ebx    \n"
      : [cont] "=r"(cont) // resultado em cont
      : [tudo] "r"(tudo), // vetor com texto
        [c] "r"(c),       // caractere a procurar
        [MAX_CARACTERES_NA_BIBLIA] "i"(MAX_CARACTERES_NA_BIBLIA)
      : "rax", "rbx", "rcx", "rdx", "rdi");
  return cont;
}
```

Quantas vezes a letra ‘a’ aparece no texto?

Ocorrências da letra ‘a’: _____

Qual foi o tempo observado na execução?

Tempo 5: _____ s

Observando a função contaLetra acima, procure duas instruções de linguagem de montagem que você conhece e explique o que elas fazem no contexto que você já aprendeu em linguagem de montagem.

Avaliando os resultados

Envie a avaliação dos resultados como descrito no arquivo “Avaliacao Dos Resultados.pdf”.

Conclusão

Uma ferramenta de levantamento de perfil pode ajudar na escolha das funções onde focar esforços para otimizar um programa. Reduzir o número de vezes que funções são chamadas pode oferecer vantagens no tempo de execução. A linguagem de montagem pode ser importante quando bem aplicada.