Instituto de Matemática e Estatística da USP

MAC0216 - Técnicas de Programação I - 2s2022

EP1
Entrega até 8:00 de 19/9/2022
(INDIVIDUAL)

Prof. Daniel Macêdo Batista

1 Problema

Números primos são muito úteis em diversos campos da computação, como por exemplo na criptografia. De forma bem resumida, alguns sistemas criptográficos são difíceis de serem quebrados porque para isso seria necessário que, dado um número não primo muito grande $a = b \times c$, com b e c primos, b e c fossem descobertos por uma pessoa que tivesse apenas conhecimento de a.

Este EP vai explorar o uso de diversas operações da linguagem Assembly (transferência de dados, operações aritméticas, operações lógicas, saltos e *syscalls*) para a descoberta de alguns números primos. Adicionalmente, um programa em C também terá que ser escrito para ser usado numa análise comparativa de desempenho.

2 Requisitos

2.1 Códigos em C e em Assembly

Dois programas deverão ser escritos, um em C e um em Assembly. Os seguintes requisitos deverão ser cumpridos:

- Os programas terão dois modos de execução que serão informados na entrada padrão juntamente com um número inteiro maior que 1 e menor ou igual a 999999936. No modo de execução 0, o programa deverá imprimir na saída padrão o próximo primo imediatamente superior ao valor inteiro passado na entrada padrão. No modo de execução 1, o programa deverá verificar se o valor inteiro passado na entrada padrão é múltiplo de dois números primos ou não. Caso seja, esses dois números primos deverão ser impressos na saída padrão. Caso não seja, nenhum valor deverá ser impresso. Seguem alguns exemplos de execução do programa:

```
lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel/ep1-Daniel$ ./epC
```

```
0 610666660 <--- Isso foi digitado
610666687   <--- Essa foi a saída do programa

lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel/ep1-Daniel$ ./epS
0 10000000
10000019

lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel/ep1-Daniel$ ./epC
1 908118221 <--- Isso foi digitado
30133 30137 <--- Essa foi a saída do programa

lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel/ep1-Daniel$ ./epS
1 6
2 3

lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel/ep1-Daniel$ ./epC
1 908118222

lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel/ep1-Daniel$ ./epS
1 2
```

- Os programas não podem usar nenhuma tabela pré-calculada de números primos. Todos os cálculos devem ser feitos em tempo de execução
- Os programas devem ser equivalentes. Por exemplo, se você implementar o programa em C com um laço que termina quando o divisor chega na metade do valor, o programa em Assembly precisa de um laço equivalente. Isso é importante para que a comparação de desempenho seja justa (Recomenda-se escrever o programa em C primeiro e o programa em Assembly depois)
- O programa em Assembly só pode usar instruções e construções da arquitetura x86_64 vistas em sala de aula. Os programas serão montados e ligados com os seguintes comandos na correção:

```
nasm -f elf64 fonte.s
ld -s fonte.o
```

- O programa em Assembly precisa ter comentários que expliquem o que está sendo feito. Programas em Assembly sem comentários não serão corrigidos (não reproduza no comentário, de forma literal, o que está sendo feito em uma dada linha. Por exemplo, dizer que um comando está movendo o número 4 para o registrador rdx não ajuda. Explique a intenção daquele mov. Recomenda-se que os comentários sejam feitos para cada bloco do programa e não linha a linha)
- Em ambos os programas, mas principalmente no caso do programa em Assembly, use nomes para rótulos e variáveis que auxiliem no entendimento do programa. Nomes como `salto1` ou `Var1` não ajudam no entendimento. Se a compreensão do programa ficar muito difícil por conta dos nomes usados, serão descontados pontos

- O programa em C não pode usar alocação dinâmica e nenhum tipo de vetor de qualquer dimensão (com exceção das strings onde serão lidas as informações de entrada e onde serão formatadas as saídas)

- O programa em C só poderá usar comandos presentes nos cabeçalhos `stdio.h` e `stdlib.h`. Ele será compilado com o seguinte comando na correção:

  ```
  gcc -Wall fonte.c
  ```

- *Warnings* ou erros que apareçam na compilação, montagem e ligação dos programas levarão a descontos na nota

Para facilitar os testes dos programas confira alguns números primos conhecidos acessando `http://compoasso.free.fr/primelistweb/page/prime/liste_online_en.php`.

Recomenda-se fortemente o uso do depurador `gdb` durante a implementação do programa em Assembly. Ainda sobre o programa em Assembly, vale lembrar que números lidos da entrada padrão e escritos na saída padrão na verdade são strings de caracteres e cada dígito é representando por um código ASCII correspondente. Por exemplo, o valor 149, que é um número inteiro armazenado na memória do computador como 10010101, quando digitado na entrada padrão é uma string de três caracteres que corresponde a 001100010011010000111001. Ou seja, 8 bits para cada caractere. Para saber os códigos ASCII de cada dígito numérico, consulte a manpage do ascii rodando no terminal o comando `man ascii`.

2.2 Relatório

O desempenho, em termos de tamanho e de tempo de execução, dos programas em Assembly e em C deverá ser avaliado no modo de execução 0 com as seguintes entradas:

```
0 10000000
0 110111120
0 210222220
0 310333358
0 410444432
0 510555560
0 610666660
0 710777961
0 810888972
0 999999936
```

Para cada valor desses, você deverá rodar o programa em C dez vezes e o programa em Assembly dez vezes. Ou seja, no total, o programa em C será executado cem vezes e o programa em Assembly será executado cem vezes. O tempo de execução em segundos deverá ser medido para cada entrada e a média das dez execuções para cada entrada deverá ser colocada em uma tabela no seu relatório conforme apresentado na Tabela 1.

Para descobrir o tempo de execução recomenda-se usar o utilitário `/usr/bin/time` da seguinte forma ao invocar o programa para uma dada entrada (por exemplo, para o primeiro valor da tabela):

```
lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel/ep1-Daniel$ echo 0 10000000 | /usr/bin/time -f "%e" ./epS
10000019
0.29 <--- Esse é o tempo em segundos retornado pelo /usr/bin/time
```

Tabela 1: Tempo de execução em segundos (média de dez execuções para cada entrada)

Entrada	**Média (C)**	**Média (Assembly)**
10000000	Média 1 do programa C	Média 1 do programa ASM
110111120	Média 2 do programa C	Média 2 do programa ASM
210222220	Média 3 do programa C	Média 3 do programa ASM
310333358	Média 4 do programa C	Média 4 do programa ASM
410444432	Média 5 do programa C	Média 5 do programa ASM
510555560	Média 6 do programa C	Média 6 do programa ASM
610666660	Média 7 do programa C	Média 7 do programa ASM
710777961	Média 8 do programa C	Média 8 do programa ASM
810888972	Média 9 do programa C	Média 9 do programa ASM
999999936	Média 10 do programa C	Média 10 do programa ASM

Tabela 2: Tamanhos dos executáveis em bytes

C	**Assembly**
Tamanho do executável C	Tamanho do executável ASM

Os tamanhos em bytes dos executáveis devem ser colocados em uma tabela no seu relatório conforme apresentado na Tabela 2.

Para descobrir o tamanho em bytes recomenda-se usar o comando `ls -l`.

Além das tabelas, o seu relatório precisa ter um cabeçalho com seu nome e NUSP e uma conclusão explicando se os resultados obtidos em termos de tempo de execução e de tamanho estão dentro do esperado ou não. As conclusões precisam ser justificadas. Também informe a configuração do computador (processador, memória e sistema operacional) onde todas as execuções foram realizadas. Todas as execuções devem ser feitas no mesmo computador para que a comparação tenha validade.

2.3 LEIAME

Junto com os programas e com o relatório você terá que entregar também um arquivo LEIAME em texto puro (Ele pode ser nomeado apenas como LEIAME ou pode ser LEIAME.txt). Arquivos LEIAME (ou README) são extremamente importantes para que outras pessoas saibam quem fez o programa, qual o objetivo do programa, como compilar, como executar e quais as dependências, caso haja alguma. Portanto, crie o seu arquivo com no mínimo estas seis seções dentro dele:

```
AUTOR:
<Seu nome, NUSP e endereço de e-mail>

DESCRIÇÃO:
<Explique o que o programa faz (Não diga apenas que ele é o EP1 da
disciplina tal, explique o que de fato o programa faz)>

COMO COMPILAR:
<Informe o passo a passo de como compilar o programa. De preferência
```

```
com todas as sequências de linha de comando>

COMO EXECUTAR:
<Informe como rodar o programa, como passar informações de entrada
para ele e como interpretar a saída>

TESTES:
<Dê exemplos de alguns pares de entradas e saídas que sejam
suficientes para que alguém confirme que o programa está funcionando
corretamente>

DEPENDÊNCIAS:
<Informe o que é necessário para compilar e rodar o programa.
Informações como: versão do montador, versão do compilador, versão do
linker, informações do SO onde você executou e sabe que ele funciona
são importantes de serem colocadas aqui>
```

Um bom arquivo de LEIAME faz parte de mais uma importante Técnica de Programação que deve ser exercitada sempre: **Documentação**. Os **Testes**, por si só, também representam uma outra importante Técnica de Programação que será mais exercitada adiante no curso.

3 Entrega

Você deverá entregar um arquivo tarball comprimido (.tar.gz) contendo os seguintes itens:

- 1 único arquivo .c com o código fonte do programa em C;
- 1 único arquivo .s com o código fonte do programa em Assembly;
- 1 arquivo LEIAME (pode ser LEIAME.txt) em texto puro;
- 1 arquivo .pdf com o relatório.

Todos esses arquivos devem estar presentes. Caso algum arquivo esteja faltando, o EP não será corrigido e a nota dele será zero.

O desempacotamento do tarball deve produzir um diretório contendo os itens. O nome do diretório deve ser ep1-seu_nome. Por exemplo: ep1-joao_dos_santos.

A entrega do tarball deve ser feita no e-Disciplinas.

Se você nunca criou um tarball, segue abaixo um passo a passo de como você pode fazer isso supondo que todos os seus arquivos estejam no diretório `~/usp/mac0216/ep1-Daniel`:

```
# Rodando um ls para confirmar que realmente está tudo no lugar certo
lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel$ ls
LEIAME        ep1.c         ep1.s         relatorio.pdf

# Descendo um nível para que o tarball seja criado com o diretório
lidenbrook@asteroid:~/usp/mac0216/ep1-Daniel$ cd ..
```

```
# Gerando o arquivo
lidenbrook@asteroid:~/usp/mac0216$ tar zcvf ep1-Daniel.tar.gz ep1-Daniel
a ep1-Daniel
a ep1-Daniel/ep1.s
a ep1-Daniel/LEIAME
a ep1-Daniel/ep1.c
a ep1-Daniel/relatorio.pdf

# Confirmando que ele foi criado sem problemas
lidenbrook@asteroid:~/usp/mac0216$ tar ztf ep1-Daniel.tar.gz
ep1-Daniel/
ep1-Daniel/ep1.s
ep1-Daniel/LEIAME
ep1-Daniel/ep1.c
ep1-Daniel/relatorio.pdf

# Confirmando mais ainda (descompactando no /tmp e indo lá ver)
lidenbrook@asteroid:~/usp/mac0216$ tar zxvf ep1-Daniel.tar.gz -C /tmp/
x ep1-Daniel/
x ep1-Daniel/ep1.s
x ep1-Daniel/LEIAME
x ep1-Daniel/ep1.c
x ep1-Daniel/relatorio.pdf
lidenbrook@asteroid:~/usp/mac0216$ cd /tmp/ep1-Daniel/
lidenbrook@asteroid:/tmp/ep1-Daniel$ ls
LEIAME        ep1.c         ep1.s         relatorio.pdf
```

O EP deve ser feito individualmente.

Obs.1: Serão descontados 2,0 pontos de EPs com tarballs que não estejam nomeados como solicitado ou que não criem o diretório com o nome correto após serem descompactados. Confirme que o seu tarball está correto, descompactando ele no shell (não confie em interfaces gráficas na hora de verificar o seu tarball pois alguns gerenciadores de arquivos criam o diretório automaticamente mesmo quando esse diretório não existe).

Obs.2: A depender da qualidade do conteúdo que for entregue, o EP pode ser considerado como não entregue, implicando em MF=0,0. Isso acontecerá também se for enviado um tarball corrompido, códigos fonte vazios ou códigos fonte que resolvam um problema completamente diferente do que foi solicitado.

Obs.3: O prazo de entrega expira às 8:00:00 do dia 19/9/2022.

4 Avaliação

80% da nota será dada pela implementação, 10% pelo LEIAME e 10% pelo relatório. Os critérios detalhados da correção serão disponibilizados apenas quando as notas forem liberadas.