MAC0216 - Técnicas de Programação I
EP1
Data de Entrega: 13/10/2020

Prof. Alfredo Goldman

1 Problema

Neste EP você deverá implementar diferentes versões de um programa chamado `sieve`, que determina se um dado número é **primo** ou não. Qualquer algoritmo correto será aceito [3]. Esse programa será parte implementado na linguagem C, e parte implementado em linguagem de máquina x86 de 32-bits.

2 Requisitos

2.1 Parte 1 – Escrevendo em C – `sieve.c`

Esse primeiro programa `sieve.c` deverá ser totalmente escrito em C, e chamar **obrigatoriamente** uma função com a seguinte assinatura:

```
int is_prime(int n)
```

onde n é o inteiro positivo a ser testado. A função devolverá:

- 1, se n for primo.
- 0 caso contrário.

Escreva também um `Makefile` simples que compile esse arquivo, gerando um binário chamado `sieve`.

2.2 Parte 2 – Misturando com Assembler – `sieve_asm.c`

Nessa versão, a função *is_prime* deverá ser reescrita em **linguagem de montagem x86 de 32-bits**, porém o resto do programa deverá ser mantido em C. Sendo assim, crie uma cópia de `sieve.c` com o nome `sieve_asm.c`.

Agora, você deve criar um novo arquivo chamado `is_prime.s` e implementar a versão em *assembler* de `is_prime`, e chamar essa função diretamente de `sieve_asm.c`. Para isso, remova o corpo de `is_prime` do arquivo `sieve_asm.c`, mas mantenha a declaração dela, pois isso dirá ao compilador que essa função estará disponível em outra Unidade de Compilação.

Para chamar corretamente a função a partir do arquivo em C, a função `is_prime` precisa ser codificada de maneira a respeitar a convenção de chamada usada pelo compilador, que no nosso caso é a `CDECL` [2] [1].

Atualize também o `Makefile` para essa parte. Você deverá compilar o arquivo `sieve_asm.c` com um compilador C, e o arquivo `is_prime.s` com um assembler, como o GNU Assembler

se preferir a sintaxe da GNU, ou o Netwide Assembler se preferir a sintaxe da Intel. **Não use um compilador C++**, pois você terá problemas. Ligue todos os `.o` com o próprio GCC, pois isso evitará problemas. Ao terminar essa parte, o seu `Makefile` também deve gerar o binário `sieve_asm`.

2.3 Parte 3 – Sem Bibliotecas Padrão – `sieve_nostdlib.c`

Na Parte 2, nosso simples programa contém várias dependências, você pode checar isso com:

```
$ ldd sieve_asm
```

temos a `linux-gate.so`, a `libc`, o `ld-linux.so`, e talvez outras. Todos os programas em C contêm essas dependências por causa de funções como `printf` e a rotina `_start` necessária para chamar a função `main`. O que você deve fazer agora é eliminar essas dependências, codificando a sua própria versão de `print` e `_start`, além de qualquer outra função da libc que você tenha usado.

Para isso, crie uma cópia do arquivo `sieve_asm.c` da parte anterior, renomeando-a para `sieve_nostdlib.c`. Para eliminar todas as dependências você deve escrever:

- Uma função em assembler que contenha a seguinte assinatura:

  ```
  void print_asm(int length, char *string)
  ```

  onde *length* é o comprimento de *string*. Codifique-a no arquivo **print.s**. Para de fato imprimir a string, faça uma chamada à rotina `write` através do vetor de interrupções de sistema `0x80`. Note que essa função **não** precisa suportar uma string de formatos como o `printf`. Você pode achar os materiais [6] e [5] interessantes para essa tarefa.

- A rotina `_start`, que deverá ser implementada em `_start.s`. Como não estamos usando `malloc` ou nada complicado para gerenciar memória, ela deve apenas preparar os argumentos `argc`, `argv`, definir corretamente o alinhamento da pilha, chamar a `main`, e por fim fazer uma requisição de sistema para encerrar o programa com o código de retorno da `main` chamando a rotina `exit`, também pelo vetor de interrupções `0x80`. Para isso, veja [4].

Por fim, atualize o seu `Makefile` para compilar, montar e ligar tudo. Para evitar que o `GCC` tente ligar com as bibliotecas padrões, use a flag `-nostdlib` para compilar o arquivo `sieve_nostdlib.c`, gerando o binário `sieve_nostdlib`. Se tudo der certo, a saída do `ldd sieve_nostdlib` será "statically linked".

3 Linguagem

Os arquivos .c devem ser escritos em C e os arquivos .s devem ser escritos em Assembly x86 de 32-bits. Certifique-se de que eles funcionam no GNU/Linux pois eles serão compilados e avaliados apenas neste sistema operacional

4 Entrada/Saída

Os três executáveis `sieve`, `sieve_asm` e `sieve_nostdlib` gerados pelo `Makefile` serão chamados da seguinte forma:

```
$ ./{nome_do_binário} <NUMBER>
```

onde `<NUMBER>` é um número inteiro positivo. Como saída, seu binário deverá imprimir 1 se `<NUMBER>` for primo, e 0 caso contrário. **Não se esqueça da quebra de linha na string**.

5 Entrega

- Você deverá entregar um arquivo `.tar.gz` contendo os seguintes arquivos:

  1. `sieve.c`
  2. `Makefile`
  3. `sieve_asm.c`
  4. `is_prime.s`
  5. `sieve_nostdlib.c`
  6. `print.s`
  7. `_start.s`
  8. `README.md`

  e outros arquivos que julgar necessário.

- O nome do pacote deve ser `ep1-membros-da-equipe.tar.gz`. Ex.: `ep1-joao-maria.tar.gz`.
- O arquivo `README.md` deve conter o nome e número USP de todos os integrantes do grupo.
- O EP pode ser feito individualmente ou em **grupos de até 3 pessoas**.
- **O prazo de entrega expira às 23:59:00 do dia 13/10/2020.**

6 Avaliação

90% da nota será dada pela implementação e 10% pela documentação. Os critérios detalhados da correção serão disponibilizados apenas quando as notas forem liberadas.

Para garantir a nota de documentação, não esqueça de **comentar seu código** e preencher o `README.md`.

Referências

[1] Alan Batson Adam Ferrari. The 32 bit x86 c calling convention, September 2020. URL: `https://aaronbloomfield.github.io/pdr/book/x86-32bit-ccc-chapter.pdf`.

[2] Wikibooks Contribuitors. X86 calling conventions, September 2020. URL: `https://en.wikibooks.org/wiki/X86_Disassembly/Calling_Conventions#CDECL`.

[3] Wikipedia Contribuitors. Primality test algorithms, September 2020. URL: `https://en.wikipedia.org/wiki/Primality_test`.

[4] Patrick Horgan. Linux x86 program startup, September 2020. URL: `http://dbp-consulting.com/tutorials/debugging/linuxProgramStartup.html`.

[5] Roger Jegerlehner. Intel code table, September 2020. URL: `https://montcs.bloomu.edu/Information/LowLevel/Assembly/IntelCodeTable.pdf`.

[6] Robert Montante. "hello, world" in x86 assembly language, September 2020. URL: `https://montcs.bloomu.edu/Information/LowLevel/Assembly/hello-asm.html#a_Linux-compatible_version`.