Atividade de Laboratório 1

Objetivos

O objetivo desta atividade é a familiarização com as ferramentas e o ambiente de trabalho da disciplina.

Instalação dos Requisitos

Nesta atividade de laboratório você aprenderá ou relembrará como compilar um código em C e como extrair os resultados intermediários do processo de compilação. Para isso, vamos usar ferramentas de compilação, montagem e ligação que são Software Livres e foram desenvolvidos pelo projeto GNU. Além disso, esperamos que essas ferramentas sejam utilizadas em um ambiente Bash/UNIX-Like (Freebsd, GNU/Linux, Linux for Windows).

Tanto as ferramentas de compilação da GNU quanto os ambientes de desenvolvimento Bash/UNIX-like são referência em desenvolvimento tanto na academia quanto na indústria. Antes de começarmos a utilizá-las, entretanto, precisamos garantir que você tenha acesso a elas em seu computador. Para isso, sugerimos 3 formas: usar uma distribuição GNU/Linux como Ubuntu; usar WSL2 (Linux for Windows); ou, usar máquinas no IC por meio de acesso remoto (SSH).

1. Usando uma distribuição GNU/Linux

   Você pode instalar na sua máquina uma distribuição GNU/Linux (64 bits) como a Ubuntu. Ela pode ser instalada diretamente na sua máquina ou instalada em uma máquina virtual como o Oracle VirtualBox.

   Uma vez dentro da sua distribuição GNU/Linux. Use o gerenciador de pacotes da sua distribuição para instalar o pacote **build-essential**. Caso esteja usando Ubuntu ou algum linux parecido com Debian digite os seguintes comandos no terminal:

   ```
   sudo apt update
   sudo apt install build-essential
   ```

   O sistema deve pedir sua senha (ou a senha do superusuário da máquina). Uma vez que o processo for concluído, você terá o ambiente configurado para a atividade de laboratório.

2. Usando WSL2

   A Microsoft, a partir do Windows 10, permite que o usuário instale um ambiente GNU/Linux no Windows. Para isso, você pode seguir o seguinte tutorial: https://sempreupdate.com.br/como-instalar-o-windows-subsystem-for-linux-2-no-windows-10/

   Depois de instalado, vá em iniciar e busque por Ubuntu. Um terminal GNU/Linux será aberto. Nele, digite:

   ```
   sudo apt update
   sudo apt install build-essential
   ```

   O sistema deve pedir sua senha (ou a senha do superusuário da máquina). Uma vez que o processo for concluído, você terá o ambiente configurado para a atividade de laboratório.

3. Usando máquinas do IC por acesso remoto (ssh)

Você pode usar as máquinas do laboratório do IC remotamente. Para isso, você precisa de um cliente SSH. O Windows possui diversos clientes de ssh (o git pode ser usado para acessar servidor por ssh: https://git-scm.com/download/win)

O servidor de ssh do IC tem o seguinte endereço: ssh.students.ic.unicamp.br
O usuário(seu ra) e senha são os mesmos que você usa para logar nas máquinas Linux no IC-3.

Em um terminal você pode acessar usando o seguinte comando:

```
ssh seuusuario@ssh.students.ic.unicamp.br
```

As máquinas do IC já possuem as ferramentas de compilação da GNU instaladas.

Uma vez que você configurou seu ambiente de desenvolvimento, realize as atividades descritas a seguir.

Descrição

Este documento contém informações sobre como funciona o processo de compilação de *software* em mais detalhes. Ele também o auxiliará na tarefa de organizar e automatizar um processo de compilação que envolve várias etapas. Neste processo, você irá perceber que o programa fica gradativamente mais próximo da linguagem de máquina até o momento em que um arquivo com as instruções que o processador irá executar é criado, o executável.

Como estudo de caso, crie um arquivo com um programa em código C de nome **`ola.c`**. O conteúdo do arquivo deve ser:

```
#include <stdio.h>
int main ()
{
  printf("Ola!\n");
  return 0;
```

```
}
```

Para compilar o programa C apresentado, a abordagem mais simples é utilizar, em um terminal do linux, a seguinte linha de comando para invocar o programa **gcc**:

```
gcc ola.c -o ola.x
```

O **gcc** irá compilar o programa fonte **ola.c** produzindo o programa executável **ola.x**. O programa executável é uma representação do programa pronta para ser executada pelo sistema. Ele possui trechos de código em linguagem de máquina (prontos para serem executados pelo processador), informações sobre o ponto de entrada do programa (a primeira função que deve ser chamada ao iniciar-se a execução), as constantes do programa e outras informações. Para executar este programa basta digitar o caminho completo do executável (pode ser ./ se estive no mesmo diretório) e teclar ENTER. Por exemplo:

```
usuario@maquina$ ./ola.x
Ola!
usuario@maquina$
```

Os passos acima sugerem que o programa **gcc** converteu o programa em linguagem de alto nível (linguagem C) diretamente para um programa em linguagem de máquina. Entretanto, o **gcc** é na verdade um programa que articula a chamada de diversos outros programas para realizar a compilação do código fonte. Nesta disciplina, você irá se familiarizar com a construção de *softwares* em nível mais baixo do que o da linguagem C. Para isso, é útil entender quais as etapas realizadas pelo **gcc** até chegar na linguagem de máquina, aquela que realmente é entendida pelo processador no momento de executar seu programa.

O *toolchain* GNU

Toolchain é o conjunto de ferramentas do compilador para produzir *software* para um sistema (no nosso caso, um *desktop* compatível com a arquitetura Intel x86). Todo compilador começa por traduzir cada arquivo de linguagem C para código de linguagem de montagem (arquivos com a extensão .s no Linux ou .asm na maioria dos compiladores Windows). Geralmente o compilador produz um arquivo em linguagem de montagem para cada arquivo fonte em linguagem de alto nível. Em seguida, uma ferramenta chamada "montador" (ou *assembler*, em inglês) lerá os arquivos em linguagem de montagem e produzirá um código-objeto (extensão .o no Linux ou .obj na

maioria dos compiladores Windows), que possuem código em linguagem de máquina. Existe um arquivo-objeto para cada arquivo em linguagem de montagem. Note que um *software* complexo pode conter diversos arquivos de código fonte, o que irá levar a vários arquivos-objeto durante o processo de compilação. Apesar de possuir código em linguagem de máquina, os arquivos-objeto não são executáveis, pois o código binário ainda está separado em diversos arquivos-objeto e precisa ser "ligado" em um único arquivo, que contenha todo o código. Dessa forma a etapa final consiste em "ligar" todos os arquivos-objeto em um único arquivo final, o executável. Essa etapa é realizada pelo ligador (*linker*, em inglês).

O ligador lê diversos arquivos-objeto como entrada, os liga entre si, e também liga código de bibliotecas que você usa. Um exemplo clássico de biblioteca é a biblioteca padrão C (no Linux, é a glibc ou *GNU C Library*). Ela contém, por exemplo, a implementação da função `printf()`, usada por nosso programa. Sem o trabalho do ligador, nosso programa não conseguiria chamar a função `printf`. Após conectar todos os módulos e bibliotecas e fundi-los em um mesmo arquivo, o resultado é o executável final, o programa que pode ser executado pelo usuário. A Figura 1 ilustra o processo de compilação de um *software* com dois arquivos fonte: `fonte1.c` e `fonte2.c`.

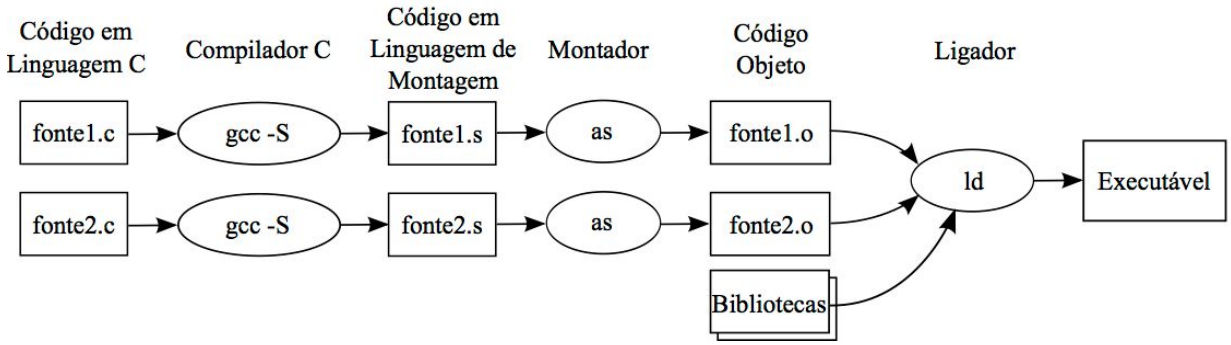

Figura 1: O processo de compilação de um programa utilizando as ferramentas da GNU.

Exercício

Você deverá percorrer o processo de compilação executando cada programa individualmente (compilador C, montador e ligador) até chegar no executável final do programa. O código fonte que você irá utilizar contém dois arquivos C, chamados de `arquivo1.c` e `arquivo2.c` com o seguinte conteúdo:

`arquivo1.c`

```
#include <stdio.h>

void funcao();

int main()
{
    printf("Ola!\n");
    funcao();
    printf("Adeus!\n");
    return 0;
}
```

`arquivo2.c`

```
#include <stdio.h>

void funcao()
{
    printf("Estou no arquivo 2!\n");
}
```

Você deverá fazer o processo uma vez manualmente, depois automatizar o processo com um *script* makefile. Para isso, você deve criar uma regra para cada arquivo intermediário, até chegar no arquivo final.

Tarefa 1

Para compilar um código-fonte C e produzir o código em linguagem de montagem .s utilize o comando:

```
gcc -S arquivo.c -o arquivo.s
```

Você pode verificar o conteúdo do arquivo `arquivo.s` abrindo este arquivo em seu editor de texto favorito. Ele é um arquivo texto e contém o mesmo programa que você escreveu em C, porém transcrito para linguagem de montagem para o processador de arquitetura Intel x86. Note que a linguagem de montagem faz

referência a instruções (add, mov, etc.) e outros elementos específicos de cada tipo de processador e, consequentemente, é dependente da interface do mesmo. Para converter o código em linguagem de montagem para linguagem de máquina você pode usar o montador (*assembler*) da seguinte forma:

```
as arquivo.s -o arquivo.o
```

Você não pode abrir o arquivo produzido (arquivo.o) em seu editor de texto, pois é um arquivo binário. Para analisar esse arquivo, você precisa de programas especiais chamados "desmontadores", que interpretam o conteúdo do arquivo e convertem sua representação para texto. Você pode utilizar a ferramenta `objdump` para desmontar o arquivo binário em linguagem de máquina. Execute o comando

```
objdump -D arquivo.o
```

e compare o resultado com o arquivo em linguagem de montagem produzido pelo compilador (arquivo.s).

Uma vez que você gerou todos os arquivos objeto, você precisa ligá-los entre si e com as bibliotecas desejadas para produzir o executável final utilizando o ligador. O seguinte comando mostra como ligar os arquivos `arquivo1.o` e `arquivo2.o` com a biblioteca padrão C e outros arquivos com código de inicialização e finalização do programa (Antes de executar esta linha em seu computador, veja as observações abaixo).

```
ld -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib64/crt1.o
/usr/lib64/crti.o -L/usr/lib64 arquivo1.o arquivo2.o [...] -lc
/usr/lib64/crtn.o -o saida.x
```

OBS 1: apesar de ser mostrado em três linhas, a linha de comando acima não pode ter quebra de linhas, ou seja, é uma única linha de comando. `ld` é o nome do ligador (linker) e o restante são parâmetros para o mesmo, incluindo os arquivos `arquivo1.o` e `arquivo2.o`.

OBS 2: os arquivos extras ligados aos arquivos de seu programa (`/lib64/ld-linux-x86-64.so.2`, `/usr/lib64/crti.o`, etc) contêm código para inicialização e finalização do programa e código para permitir que seu programa use a biblioteca padrão C. Estes arquivos podem estar localizados em pastas diferentes em outras distribuições Linux. Para identificar onde estes arquivos estão localizados, você pode compilar um programa qualquer usando o `gcc` com a opção -v, que mostrará

como os programas (compilador, montador e ligador) do processo de compilação são executados (incluindo seus parâmetros).

NOTA: Quando for submeter seu Makefile, você deve usar os caminhos descritos no exemplo acima, já que seu Makefile será testado em um sistema com estes caminhos.

OBS 3: o parâmetro -L não adiciona módulos ao seu programa, mas apenas especifica uma pasta onde encontrar bibliotecas (esse caminho pode variar entre distribuições Linux). O parâmetro -lc, por sua vez, informa que a biblioteca C é necessária. Os outros arquivos-objeto .o (ponto ó) mencionados acima são pequenos módulos auxiliares necessários para todo programa em linguagem C. O trecho `[ . . . ]` não pode ser digitado na linha de comando. Ele apenas informa que você pode colocar quantos arquivos-objeto .o quiser. Os arquivos-objeto fornecidos como parâmetro são dispostos no executável final na mesma ordem dos parâmetros. Por isso o arquivo `crtn.o`, que contém código de finalização do suporte em tempo de execução C (*C RunTime - eNd*), aparece por último, e o `crt1.o`, que contém código de inicialização do *C runtime* aparece primeiro.

Note que o `gcc` é um programa que invoca outros programas por você e já provê os parâmetros corretos para o ligador (ld). Nesse caso, como estamos construindo o programa sem o auxílio do gcc, nós precisamos passar todos os módulos adicionais que devem ser ligados ao seu executável para que ele funcione. Na dúvida, você pode adicionar a *flag* -v na linha de comando do gcc para visualizar todos os comandos executados pelo gcc durante a compilação de uma aplicação em seu sistema. Na disciplina, quando estiver criando programas diretamente em linguagem de montagem (arquivos .s), você pode usar apenas as ferramentas as e ld, sem a necessidade do gcc. O arquivo `saida.x` será o programa executável e pode ser executado com o comando:

```
./saida.x
```

GNU makefile

O processo de desenvolvimento de *software* envolve diversas iterações de correções de *bugs* e recompilações. Entretanto, muitos destes projetos possuem uma quantidade grande de arquivos de programa e a compilação de todos os arquivos é um processo lento. Os arquivos .o precisam ser ligados novamente para formar o novo binário, no entanto, apenas os arquivos modificados precisam ser recompilados. Dessa forma é importante ter um mecanismo automático para recompilar apenas os arquivos

necessários. Para isso, existe uma modalidade de *script* específica para automatizar a compilação de *softwares*. O GNU makefile é um exemplo largamente utilizado no mundo GNU/Linux.

Para fazer o seu próprio *script* que irá orientar o GNU make a construir o seu programa, você deve criar um arquivo texto chamado Makefile, que deve estar na mesma pasta dos códigos-fonte, contendo regras para a criação de cada arquivo. Por exemplo, você pode criar regras para especificar como o arquivo .s (em linguagem de montagem) é criado (utilizando o compilador gcc), especificar como os arquivos-objeto .o (códigos-objeto) são criados (utilizando o montador) e assim em diante. Exemplo de criação de regras:

```
ola.s: ola.c
     gcc -S ola.c -o ola.s
ola.o: ola.s
     as -o ola.o ola.s
```

Neste exemplo existem duas regras: ola.o e ola.s. A regra ola.o deve corresponder ao arquivo que é produzido com essa regra. Os arquivos necessários para produzir o arquivo ola.o devem aparecer em uma lista (separada por espaços) após o caractere ":" (no nosso caso, ola.s é necessário para criar ola.o). Em seguida, você deve, na linha seguinte, usar uma tabulação (apertar a tecla tab) e digitar o comando que será executado no *shell* para produzir esse arquivo. No nosso exemplo, chamamos o compilador gcc para traduzir um arquivo em linguagem C para linguagem de montagem, e em outra regra, chamamos o montador GNU as para transformar um arquivo em linguagem de montagem .s em um arquivo-objeto .o. Note que você pode especificar como arquivo de entrada de uma regra o nome de outra regra, e esta outra regra será chamada antes para produzir o arquivo de entrada necessário. ATENÇÃO: O *script* não funcionará se não houver uma tabulação (tab) antes dos comandos "`gcc -S ...`" e "`as -o ...`"! Não use espaços! Além disso, note que alguns editores de texto incluem espaços em vez do caracteres de tabulação quando a tecla tab é pressionada. Para evitar este tipo de problema, é recomendado o uso do Emacs ou VI para a criação do Makefile.

Você pode criar várias regras em um mesmo arquivo Makefile. Para executar o *script*, na linha de comando, digite make nome-da-regra. Por exemplo:

```
make nome-da-regra
```

O programa make irá executar os comandos associados à regra nome-da-regra, descrita no Makefile. Note que o programa make sempre lê o arquivo de nome Makefile na pasta em que você está e o usa como *script*. Se você não utilizar esse nome de arquivo (Makefile com "M" maiúsculo), o *script* irá falhar. Se você invocar o comando make sem parâmetros ele executará a primeira regra do arquivo Makefile.

Tarefa 2

Agora você criará o arquivo Makefile para gerar um programa a partir do seu código fonte. O programa será construído a partir de 3 arquivos fonte com código escrito na linguagem C. São eles:

- main.c
- math.c
- log.c

A primeira regra do arquivo deve se chamar prog.x e o arquivo produzido por esta regra deve ser o executável "prog.x". Seu Makefile deve ter regras individuais para construir o programa final e cada um dos arquivos intermediários necessários para a compilação, ou seja, os arquivos em linguagem de montagem (main.s, math.s e log.s) e os arquivos-objeto (main.o, math.o e log.o), como exercitado na tarefa 1. Além disso, seu Makefile deve ter uma regra chamada clean, que remove os arquivos em linguagem de montagem, os arquivos-objeto e o arquivo executável. Você pode usar o comando

```
rm -f main.s math.s log.s main.o math.o log.o prog.x
```

para removê-los.

Para testar seu Makefile, execute as seguintes linhas de comando no terminal do linux e verifique se a ferramenta make produziu APENAS os arquivos necessários pela regra especificada.

```
make main.s
make clean
make math.s
make clean
make log.s
make clean
make main.o
make clean
make math.o
```

```
make clean
make log.o
make clean
make prog.x
```

Submeta o Makefile gerado utilizando a atividade no Moddle.

Referência para Makefile:

- Guia em português: http://pt.wikibooks.org/wiki/Programar_em_C/Makefiles
- Manual original (em inglês): http://www.gnu.org/software/make/manual/make.html#Simple-Makefile

Entrega e avaliação

O arquivo Makefile deve ser submetido no sistema Moddle para avaliação.