INSTRUÇÕES PARA INSTALAÇÃO E UTILIZAÇÃO DAS FERRAMENTAS NECESSÁRIAS PARA A CRIAÇÃO E EXECUÇÃO DE PROGRAMAS UTILIZANDO LINGUAGEM DE MONTAGEM.

Gabriel Rodrigues dos Santos
Universidade Estadual de Santa Cruz - UESC
Rodovia Jorge Amado, Km 16 – CEP 45662-900 – Ilhéus, BA
gabrielrsantoss@icloud.com

Resumo. *Este documento apresenta instruções detalhadas para instalação e utilização de ferramentas necessárias para criação e execução de programas utilizando linguagem de montagem. Foi desenvolvido a critério de avaliação para disciplina Software Básico da Universidade Estadual de Santa Cruz, com a orientação do professor Leard de Oliveira Fernandes.*
Palavras-chave: *Software básico, Linguagem de montagem, Programação*

1. INTRODUÇÃO

A arquitetura dos computadores compreende somente a linguagem de máquina, que são instruções baseadas em 0s e 1s (*bits*), porém, os humanos não possuem tais habilidades para entender essas instruções de forma rápida e eficiente, sendo necessário um método capaz de facilitar a comunicação entre homens e máquinas.

Surgiu assim a linguagem de montagem, que é uma notação legível por humanos para o código de máquina, possibilitando a sua programação por símbolos chamados mnemónicos, o que tornou possível a utilização do comando mov (mover) no lugar de cadeias de bits ilegíveis.

Software Básico é uma disciplina que possui a finalidade de descrever essa camada de funcionamento de um sistema computacional, que se situa entre o hardware e o sistema operacional. Sendo necessário algumas ferramentas de desenvolvimento que facilitam essa abordagem como: gcc, gdb, nasm, objdump, kgdb, editores de texto, bless editor *e o docker*. Que serão abordados adiante.

2. METODOLOGIA

Todos procedimentos utilizados na construção desse relatório foram feitos no sistema operacional Linux Ubuntu e no Windows 10 utilizando o *docker*.
Necessário um computador com acesso à internet para a instalação das ferramentas e, ao menos, o mínimo grau de instrução de arquiteturas de computadores, linguagens de programação C e entendimento de comandos básicos do gerenciador de pacote do *Linux*.
Para o Windows 10, é necessário um processador com suporte a virtualização.

3. INSTALAÇÃO E UTILIZAÇÃO DAS FERRAMENTAS

Abaixo está listado as ferramentas necessárias para o desenvolvimento em linguagem de montagem, com instruções de instalação e a utilização de comando básicos de cada uma.

As ferramentas serão instaladas no terminal do ubuntu com o gerenciador de pacotes apt. Exemplos serão listados para facilitar a compreensão. O caractere '*$*' indica linha de comando do terminal, entre '/*' e '*/' os comentários de bloco e '//' comentários de linha.

Ao final, durante a abordagem do *docker*, será demonstrado sua instalação no Windows 10 e também a instalação das ferramentas listadas abaixo no mesmo.

3.1 Editor de texto

Antes de prosseguir com as instalações de utilizações das ferramentas, é necessário definir um editor de texto para utilização. O ubuntu conta com alguns disponíveis:

Gedit. Similar ao bloco de notas do windows. Basta criar seu código e salvar respeitando a extensão da linguagem utilizada.

Nano. Utilizado por terminal.

```
$ nano 'nome_do_arquivo'.'extensão'
/* Acima está um exemplo de utilização do nano para criar ou editar um documento existente. */
```

Existem vários outros editores disponíveis, como vim ou vi, pluma, sublime, bluefish, geany, entre outros.

3.2 GNU compiler collection (GCC)

É um conjunto de compiladores de linguagens de programação escrito por Richard Stallman em 1987. Originalmente suportava apenas a linguagem de programação C, mas com o tempo ganhou suporte às linguagens C++, Fortran, Ada, Java e Objetive-C. Neste tutorial utilizaremos apenas com C.

Instalação. Existem duas formas de realizar a instalação do compilador GCC no *ubuntu*. Abaixo segue os dois exemplos utilizando o gerenciador de pacotes descrito anteriormente.

```
$ sudo apt-get install gcc //ou
$ sudo apt-get install build-essential

/* Necessário utilizar 'sudo' para executar o comando com privilégios de root, caso contrário o sistema não irá permitir a instalação. */
```

Utilização. Depois do compilador instalado, poderá criar linhas de código através de um editor. Abaixo segue exemplo de um código criado através do nano para imprimir uma mensagem na tela.

```
//Criar um arquivo c.
$ nano prog.c

//Código em C
#include <stdio.h>

int main(){
    printf("Hello World\n");
    return 0;
}
```

Após ter seu programa salvo, necessário gerar o executável e a execução.

//Compilar
$ gcc prog.c -o prog

/ prog.c é o programa que contém seu código, -o primeiro_programa é a saída do compilador (arquivo executável que conterá o programa). */*

//Executar
$ ./prog

/ './' indica que o programa está no diretório atual. */*

O GCC executa 4 passos para obter o executável de um programa. É possível executar cada um desses passos separadamente. Vejamos como pode ser feito para o arquivo prog.c.

Pré-processamento – processa cabeçalhos e macros. Em C ocorre da seguinte maneira.

- Substitui tri grafos por equivalentes.
- Concatena ficheiros de código-fonte.
- Substitui comentários por espaços em branco.
- Reage a linhas iniciadas com cardinal (#), efetuando substituições de macros, inclusão de ficheiros, inclusão condicional e outras operações.

//Comando
$ gcc -E prog.c -o prog.i

/ Basta abrir o arquivo gerado com editor de texto para verificar o que foi gerado. */*

Compilação – compila e converte o arquivo prog.i para a linguagem assembly (montagem).

//Comando
$ gcc -S prog.i -o prog.s

Assemble – cria o arquivo objeto (código de máquina)

//Comando
$ gcc -c prog.s -o prog.o

Link-edição – cria o executável adicionando as bibliotecas

//Comando
$ gcc prog.o -o prog

$ gcc -v prog.c -o prog

/ Comando acima permiti verificar quais programas externos o gcc utiliza para executar essas tarefas. Caso acrescente '-save-temps' os arquivos temporários também poderão ser verificados. */*

Mais informações podem ser obtidas através dos comandos.

$ man gcc //Documentação
$ gcc --help //Utilização

3.3 GNU Debugger

Um *debugger* é um programa que supervisiona a execução da aplicação para que seja verificado como ela está funcionando. Assim, possibilita a execução da aplicação linha por linha, bem como descobrir o valor das variáveis em cada instante da execução. Existem também algumas funcionalidades específicas que ajudam a ir direto para uma linha determinada ou ainda alterar o valor de uma variável forçosamente e observar os resultados.

Apesar do compilador ser informativo a respeito de erros de sintaxe em códigos, ele

não é capaz de identificar loops infinitos, ou modificações de variáveis incorretas. Assim, o *debugger* se torna útil, evitando a reanalise do código de forma cansativa e demorada.

Para que o gdb possa controlar a execução do programa, é necessário a inserção de informações especiais (tabela de símbolos) ao traduzir o código fonte para executável. No gcc utilizasse o flag de compilação '-g', solicitando a inclusão dessas informações.

Instalação.

$ sudo apt-get install gdb

Utilização.

//Comando
$ gcc prog.c -o prog -g

//Inicializando gdb
$ gcc prog

/ O gdb informaŕa uma série de mensagens, e se ao compilar não for informado a inclusão dos símbolos com '-g', nessa etapa será notificado a mensagem 'no debugging symbols found'. */*

Alguns comandos básicos e úteis para se usar no gdb.

//Comandos básico
run: indica ao dgb para que a execução seja iniciada.
kill: força o término da execução do programa.
quit: sai do gdb.

//Execução pausadas
break <lugar>: inseri um 'breakpoint' (ponto de parada) no programa. Pode ser usado para parar em uma função ou linha específica.
step <n>: executa a linha atual e passa para a próxima. Caso encontre uma função, a executa completamente linha por linha, n indica quantas linhas devem ser executadas (o padrão é 1).
next <n>: igual ao step, difere nas chamadas de função, no qual são tratadas como se fossem um comando só.

//Impressões
display <expressão>: imprimi o valor atual da variável passada como argumento, se for função, o resultado será impresso. Caso não haja argumentos, todos os resultados impressos com display serão somados.
undisplay <n>: remove a entrada 'n' da lista de exibição.
print <expressão>: imprimi o valor de uma variável ou expressão sem adicioná-los à lista de exibição.

Obs.: Deve-se tomar cuidado com variáveis que não foram inicializadas, pois terão valores esquisitos devido ao "lixo" na memória.

//Trabalhando com breakpoints
continue: executa o programa até algum breakpoint.
break <lugar> if <condicao>: breakpoint condicionais.
watch <expressão>: insere a expressão na "lista de vigília". Se o valor mudar, a execução é interrompida.
Info breakpoints/watchpoints: exibi os breakpoints inseridos até o momento.
Delete <n>: remove o breakpoint/ watchpoint de índice "n" da lista.

Mais informações sobre o gdb podem ser encontrados através dos comandos:

$ man gdb //Documentação
$ gdb -h //Utilização

3.4 Kdbg

Caso seja necessária uma melhor visualização das operações feitas com debugger gdb, o kdbg oferece uma interface para que essas operações sejam mais visíveis.

Instalação.

$ sudo apt-get install kdbg

Utilização. Basta executar o programa clicando no atalho de acesso ou digitando "kdbg" no terminal linux.

3.5 Nasm

Nasm é um *software Assembly* livre, para arquitetura x86. É utilizado para criar um executável a partir de um código-fonte de linguagem de montagem (.asm).

Instalação.

$ sudo apt-get install nasm

Utilização. Após ter criado o código assembly (prog.asm) em um editor de texto, basta executar os comandos.

//Compilar - cria o arquivo prog.o
$ nasm -f elf prog.asm

//Criando o executável
$ ld prog.o -o prog

//Executando
$ ./prog

//Mais informações
$ man nasm //Documentação
$ nasm -h //Utilização

3.6 Objdump

É utilizado para investigar e exibir informações sobre os dumps (programa de computador) do sistema, desmontando objetos compartilhados e bibliotecas, porém geram os arquivos em linguagem de montagem.

Instalação. Os sistemas operacionais linux já possuem.

Utilização. Tomando como exemplo o arquivo executável "prog" criado anteriormente.

//Comando principal
$ objdump -d prog

//Para salvar a saída em um arquivo
$ objdump -d prog > "nome_arquivo"

//Mais informações
$ man objdump //Documentação
$ objdump -H //Utilização

3.7 Bless editor (Editor hexadecimal)

Por conta de a linguagem de montagem manipular endereços em hexadecimal, há necessidade de um método prático para obter e calcular posições de memória, por conta disso a ferramenta "bless editor" facilita o desenvolvimento em *assembly*.

Instalação.

$ sudo apt-get install bless

Utilização. Basta clicar no atalho de acesso ou digitar "bless" no terminal linux.

3.8 Docker

Projeto de código aberto que automatiza a implantação de aplicativos dentro de recipientes de software, em poucas palavras, é um virtualizado de processos. Uma das vantagens é a possibilidade de iniciar sua aplicação em qualquer máquina que sempre irá rodar da forma esperada.

Instalação Linux.

$ sudo apt-get installl docker.io

Utilização Linux. Para baixar uma imagem, é necessário que ela esteja no *docker hun* ou em algum outro *registry*. É possível montar um ambiente completo baixando e linkando as imagens umas às outras.

//Baixar a imagem
$ docker pull "nome_da_imagem"

//Listar todas a imagens baixadas
$ docker images

//Iniciar um container da imagem escolhida
$ docker run "nome_da_imagem"

/ Existem alguns parâmetros do docker run: "-v" para mapear uma pasta da máquina para dentro do container, "-p" porta externa no qual deseja que a máquina faça requisições. */*

//Listar os containers em execução
$ docker ps

//Executar comando dentro do container
$ docker exec "container_id" "comando"

//Parar a execução de todos containers
$ docker stop $(docker ps -aq)

//Excluir todos containers criados
$ docker rm $(docker ps -aq)

//Mais informações
$ man docker //Documentação
$ docker -h //Utilização

Instalação Windows 10. Baixe o arquivo *Docker Toolbox* através do link: https://store.docker.com/editions/community/docker-ce-desktop-windows?tab=description

As etapas de instalação são simples, basta executar o arquivo após o *download* e seguir os passos listados.

1. A caixa de texto inicia (*Help Docker improve Toolbox*) é opcional, clique *next.*
2. Seleciona qual pasta deseja instalar o programa, clique *next.*
3. Escolha o tipo de instalação, caso deseje acrescentar outros programas opcionais, basta seleciona-los ou desmarca-los e clicar em *next.*
4. As opções seguintes dispõem uma série de ferramentas extras que auxiliam durante a utilização do *docker*, podem ser desmarcadas dependendo da finalidade de uso do programa, clique *next.*
5. Basta clicar em *install* e aguardar a finalização.

Utilização no Windows 10. Após a instalação ser concluída, será demonstrado como utiliza-lo e instalar as ferramentas descritas

anteriormente de maneira adequada. Não existe nenhuma dificuldade em sua utilização no Windows 10, já que toda manipulação é realizada por uma interface amigável.

Após a execução, necessário possuir uma conta no *Docker Hub* (repositório de imagens do *docker*), *link* para cadastro: https://hub.docker.com.

1. Realize o *login* com a conta *docker hub*.

Todos os passos seguintes são igualmente utilizados para se obter quaisquer ferramentas abordadas anteriormente. Existem ferramentas que ainda não existe uma imagem no *docker hub*, como é o caso do "bless editor". Porém é possível criar um repositório no *docker* com a imagem do ubuntu e realizar a instalação das ferramentas descritas anteriormente.

2. Buque no campo de pesquisa a ferramenta desejada para que o *docker* procure uma imagem no repositório.
3. As imagens oficiais das desenvolvedoras estarão identificadas.
4. Clique em *"creat"* para acrescenta-la aos seus *containers*, depois basta executar e usa-la.

4. CONCLUSÕES

Com base nas informações descritas nesse documento, pode-se perceber a importância que algumas ferramentas trazem ao desenvolvedor, não somente a nível de linguagem de montagem, mas para qualquer tipo de aplicação, minimizando o tempo de produção, identificando e evitando problemas inesperados.

Embora atualmente existem linguagens de alto nível que facilitam e aumentam a produtividade do *software*, possibilitando que poucos desenvolvedores tenham contato com a linguagem *assembly*, ainda há grande importância na camada de baixo nível para identificação, correção e melhorias na eficiência dos algoritmos, o que acaba se tornando um diferencial para esses profissionais. Conhecer algumas dessas ferramentas podem trazer benefícios.

[1] Usando Ferramentas de Dump do Sistema [Internet]. IBM. [2017]. Disponível em: https://www.ibm.com/support/knowledgecenter/pt-br/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/problem_determination/linux_system_dumps.html

[2] Compilar um programa de linguagem de montagem com o Nasm. CCM. [2017]. Disponível em: http://br.ccm.net/faq/8005-compilar-um-programa-de-linguagem-de-montagem-com-o-nasm

[3] Pré-processador. Wikipédia. [2017]. Disponível em: https://pt.wikipedia.org/wiki/Pré-processador

[4] Fábio M. Oliveira. Usando o compilador GCC. Viva o Linux. [2007]. Disponível em: https://www.vivaolinux.com.br/dica/Usando-o-compilador-gcc

[5] João L. M. Pinto. Como usar o GCC?. Ubuntu Fórum. [2006]. Disponível em: http://ubuntuforum-br.org/index.php?topic=9337.0

[6] Compilando programas em C com o GCC. USP. [2017]. Disponível em: http://fig.if.usp.br/~esdobay/c/gcc.html

[7] GNU Compiler Collection. Wikipédia. [2016]. Disponível em: https://pt.wikipedia.org/wiki/GNU_Compiler_Collection

[8] GDB. Uni Rio Tec. [2016]. Disponível em: http://www.uniriotec.br/~morganna/guia/gdb.html

[9] Ricardo T. Tutorial inicial para o GDB. Unicamp. [2017]. Disponível em: http://www.lrc.ic.unicamp.br/~luciano/courses/mc202-2s2009/tutorial_gdb.txt

[10] Mnemónica. Wikipédia. [2017]. Disponível em: https://pt.wikipedia.org/wiki/Mnemónica

[11] Assembly. Wikipédia. [2017]. Disponível em: https://pt.wikipedia.org/wiki/Assembly

[12] Software básico. Wikipédia. [2017]. Disponível em: https://pt.wikipedia.org/wiki/Software_básico