Sistemas Embarcados

Linux

Introdução ao desenvolvimento com Linux

- Conteúdo:
  - Metodologia/Filosofia de desenvolvimento com o Linux/UNIX;
  - Licenças de software livre (GPL e LGPL) ;
  - Comandos básicos do Linux;
  - Organização da estrutura de diretórios;
  - Obtendo informações sobre o sistema;
  - Instalação de programas;
  - Ferramentas de desenvolvimento em linguagem C

O que é Linux?

- Variante *free-software* do Unix;
- Criado por Linus Torvalds em 1990;
- Disponibilizado em forma de distribuições:
  - Debian;
  - Fedora;
  - Slackware;
  - Knoppix;
- A diferença entre as distribuições está no conjunto de utilitários oferecidos.

O que é realmente o Linux?

- O Linux é apenas um bloco de um sistema operacional completo e livre;
- Linux = kernel do sistema operacional GNU;
- Os mais ortodoxos chamam os sistemas operacionais de GNU/Linux: sistema operacional livre com kernel Linux;

Como o kernel Linux opera?

- O kernel toma controle do computador.
- Gerencia o processador: sistema multi-usuário, multi-processo, multi-processado (*scheduler*).
- Gerencia a memória com precisão: sana as demandas de memória, gerencia o espaço de troca (*swap*).

Como o kernel Linux opera?

- Gerencia dispositivos:
  - Suporta milhares de dispositivos: *drivers*;
  - Para não carregar demais o kernel, carrega *drivers* sobre demanda: os módulos.

- Gerencia sistemas de arquivos: reconhece vários tipos;

- Segurança: sistema de permissões para um ambiente multi-usuário.

Como o kernel Linux opera?

- Em síntese:
  - O kernel inicia quando se liga o computador;
  - Gerencia programas;
  - Permite comunicação simples e eficiente com o hardware.

Distribuições Linux

- Sistema operacional possui/pode ter:
  - Kernel
  - Interfaces gráficas com o usuário;
  - Utilitários administrativos;
  - Aplicações;
  - Ferramentas de programação.

Filosofia de desenvolvimento com Linux

- Simplicidade:
  - Ferramentas pequenas, simples e fáceis de entender;
  - KISS: Keep It Small and Simple;
  - Programas complexos são mais suscetíveis a bugs;

Filosofia de desenvolvimento com Linux

- Foco:
  - O programa executa a tarefa com perfeição;
  - Manter um programa responsável por muitas tarefas é difícil;
  - Modularização e composição são freqüentes.

Filosofia de desenvolvimento com Linux

- Reuso:
  - Não reinvente a roda!
  - Procure disponibilizar suas aplicações na forma de bibliotecas.

Filosofia de desenvolvimento com Linux

- Filtros:
  - Muitas das aplicações do Linux são filtros;
  - Isso permite a combinação de programas simples em um sistema complexo.

Filosofia de desenvolvimento com Linux

- Formato de arquivo aberto:
  - Uma das razões de sucesso (e ódio também);
  - Arquivos de configuração e dados são codificados em ASCII;
  - Maior liberdade aos usuários.

Filosofia de desenvolvimento com Linux

- Flexibilidade:
  - Não se pode prever o que um usuário deseja com um programa;
  - Procure ser flexível:
    - evite limites nos tamanhos dos campos;
    - tenha em mente um ambiente de rede;
  - Documente bem o seu programa.

Alguns conceitos

- Usuários: o Linux, por ser um sistema multi-usuário, apresenta um conjunto característico de usuários.

- root: o usuário raiz, super-usuário, administrador, ...
  - Logando como o usuário root você tem controle total sobre o sistema: ligar/desligar, adicionar/remover usuários, alterar senhas, instalar/remover aplicações, ...

- Outros usuários: definidos sobre demanda
  - usuários comuns;
  - daemons;
  - serviços.

Alguns conceitos

- Cada usuário possui um conjunto de permissões e privilégios.
- Isso faz com que um sistema de grande porte seja mais seguro intrinsicamente: nativamente o sistema disponibiliza um conjunto de privilégios.
- Uma conta para cada usuário;
- Não é elegante operar como root ou deixar usuário sem senha.

Alguns conceitos

- Para usar mídias removíveis, o Linux requer a **montagem** dos dispositivos:
  - Montar = mapear o conteúdo da mídia para uma pasta do sistema de arquivos;
  - Depois de montado, as operações sobre o dispositivo são semelhantes às operações sobre uma pasta do sistema;
  - Deve-se respeitar o esquema de permissões.

Iniciando o uso do Linux

- Interfaces gráficas maduras: KDE, GNOME, Wmaker.
- Aplicações em linha de comando:
  - Os comandos são os mesmos, independente do UNIX;
  - São disponibilizadas poderosas ferramentas de filtragem de dados;

Iniciando o uso do Linux

- Alguns motivos para aprender a usar o shell (linha de comando):
  - Shell inteligente: histórico e auto-completar;
  - Sistema eficiente de automação de tarefas;

Iniciando o uso do Linux

- Brinque com o Linux online:

  https://www.tutorialspoint.com/unix_terminal_online.php

Comandos simples

- Para desligar um computador com Linux:
  - Logar como root;
  - Digitar:

```
# shutdown -h now
# poweroff
```

- Para reiniciar:
  - Logar como root;
  - digitar:

```
# shutdown -r now
# reboot
```

Comandos simples

- Para sair:

```
$ exit
```

- Comandos simples:

```
$ date
   qui,  9 de mar de 2017 10:21:22

$ cal
      março 2017
do se te qu qu se sá
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Comandos simples

```
$ who

$ finger

$ cat

$ su

$ alias
```

Obtendo informações

- O Linux, por ser um sistema operacional livre e dedicado a ambientes de rede, possui como vasta fonte de informações a própria Internet.

- Em caso de dúvidas, pergunte ao oráculo: http://www.google.com

- Ajuda on-line: `$ man`

Manipulando arquivos

- `ls`
- `file`
- `chmod`
    - `$ chmod permissoes alvo`

Manipulando arquivos

O sistema de permissões Linux

- As permissões são associadas a arquivos e diretórios;
- Destinam-se à manter a privacidade dos usuários e integridade do sistema;
- A cada arquivo, está associada uma string de nove bits que define as permissões;
- Além disso, cada arquivo possui um proprietário e um grupo proprietário.

Manipulando arquivos

O sistema de permissões Linux

- String de permissão:
  - **rwxrwxrwx**

- Os três primeiros bits definem as permissões do proprietário;

- Os bits do meio, do grupo proprietário;

- Os últimos três bits, de todos os outros usuários.

Manipulando arquivos

O sistema de permissões Linux

- Cada bit de uma trinca significa:
  - **`r` : leitura;**
  - **`w` : escrita;**
  - **`x` : execução.**

- O comando `chmod` permite a mudança de permissões.

- `chmod` é geralmente usado com um primeiro argumento numérico que especifica as permissões.

Manipulando arquivos

O sistema de permissões Linux

- Por exemplo:
  - `$chmod 744 curriculum_vitae.rtf`

- Cada número representa um conjunto de permissões;

- Como cada conjunto é formado por três bits, as permissões são codificadas binariamente com números de 0 a 7.

- **r** tem peso 4, **w** tem peso 2 e **x** tem peso 1.

Manipulando arquivos

O sistema de permissões Linux

- Portanto, 7 4 4:
  - 7 : 4 + 2 + 1 => proprietário possui poder de leitura, escrita e execução;
  - 4 : 4 => grupo proprietário possui poder de leitura;
  - 4 : 4 => outros usuários possuem poder de leitura
- O proprietário (e o root) podem mudar as permissões de um arquivo.

Manipulando arquivos

- cp

  ```
  $ cp -atributos origem destino
  Atributos comuns:
     -r (recursivo)
     -f (forçar)
     -a (manter permissões)
  ```

- mv

  ```
  $ mv origem destino
  Muito usado para renomear arquivos
  ```

- rm

  ```
  $ rm -atributos arquivo
  Atributos comuns:
     -r (recursivo)
     -f (forçar
  ```

Manipulando arquivos

- chown

  ```
  $ chown -atributos novo_proprietario alvo
  ```

  Atributo comum: -R (recursivo)

- chgrp

  ```
  $ chgrp -atributos novo_grupo alvo
  ```

  Atributo comum: -R (recursivo)

Recursos do shell

- Redirecionando a saída de comandos:
  - `$ ls -la > out.txt`
  - `$ cat out.txt`

- Concatenando a saída de comandos:
  - `$ date >> out.txt`
  - `$ cat out.txt`

- Tunelando comandos:
  - `$ cat out.txt | grep Ago`

Recursos do shell

- Direcionando a entrada de comandos:

```
$ cat < out.txt
$ cat out.txt
```

- Comandos sequenciais:

```
$ clear; ls
```

Recursos do shell

- Completamento de comandos:
  - inicie o nome do comando e pressione tab
- Histórico de comandos:
  - pressione teclas cima ou baixo;
  - pressione ctrl+r e inicie a digitação;
  - pressione ctrl+s e inicie a digitação.

Recursos do shell

- Metacaracteres:

*	qualquer string
?	qualquer caracter isolado
[...]	qualquer caracter isolado dentro dos colchetes
[a-z]*	qualquer caracter alfabético minúsculo
*	o caractere '*'

Usando os recursos do shell

- Caracteres especiais:

  ; separador seqüencial

  & comando em segundo plano

  # comentário

Atividade do sistema

- Comandos de manipulação de processos:

ps	Lista dos processos atuais dos usuários
jobs	Lista as tarefas disparadas por um usuário
top	Apresenta a lista de processos do sistema dinamicamente
kill	Mata um processo

Atividade do sistema

- Comandos de manipulação de processos:

`nice` Modifica a prioridade de escalonamento

`fg` Trás um processo de background ou parado para foreground

Comandos de amostras de arquivos

- Às vezes, não precisamos verificar todo o conteúdo de um arquivo:

```
$ head -atributos arquivo
   Atributo comum: -f (continuamente)

$ tail -atributos arquivo
   Atributo comum: -f (continuamente)
```

Estrutura de diretórios do Linux

- Nos sistemas de arquivos do Unix podemos encontrar:
  - Processos;
  - Dispositivos de hardware;
  - Canais de comunicação entre processos;
  - Segmentos de memória compartilhada.

Estrutura de diretórios do Linux

- O que compõe um sistema de arquivos:
  - Espaço de nomes: forma de nomear e organizar as coisas hierarquicamente;
  - API: conjunto de chamada de sistema para navegar e manipular nós: interface padrão;
  - Modelo de segurança: esquema de proteção e compartilhamento de recursos;
  - Implementação: código que associa o modelo lógico ao circuito eletrônico.

Estrutura de diretórios do Linux

- Os nomes de caminho e algumas regras:
  - Diretório raiz: /
  - Caminhos absolutos:
    - `/home/user/UnB/lista.txt`
  - Caminhos relativos:
    - `UnB/lista.txt`
  - A hierarquia pode ser profunda:
    - cada nível pode conter 255 caracteres;
    - o caminho completo é limitado em 1023 caracteres.

Estrutura de diretórios do Linux

Hierarquia visitada

- Existe um padrão para a estrutura de diretórios no Linux.

- Esse padrão – FHS -, permite uma compatibilidade entre aplicações de diferentes distribuições.

Estrutura de diretórios do Linux

Hierarquia visitada

- /
  - Conteúdo destinado a boot, recuperação e reparo do sistema;
  - Deve conter os seguintes diretórios:
    - `bin, boot, dev, etc, lib, media, mnt, opt, sbin, srv, tmp, usr, var`
    - `home, root`

Estrutura de diretórios do Linux

Hierarquia visitada

- `/bin`
  - Binários de comandos essenciais ao usuário;
  - Não devem ocorrer subdiretórios em `/bin`.
- `/boot`
  - Contém ferramentas necessárias ao processo de inicialização.

Estrutura de diretórios do Linux

Hierarquia visitada

- `/dev`
  - Contém arquivos de dispositivos;
  - Programadores procuram usar uma mesma interface para resolver diversos problemas;
  - O acesso aos dispositivos de hardware é feito por funções semelhantes às funções usadas para a manipulação de arquivos

Estrutura de diretórios do Linux

Hierarquia visitada

- `/etc`
  - Contém configuração específica do computador;
  - Arquivos de configuração são arquivos locais usados para controlar a operação de um programa.
  - Não devem ser encontrados binários em `/etc`.

Estrutura de diretórios do Linux

Hierarquia visitada

- `/home`
  - Diretório home dos usuários;
- `/lib`
  - Contém bibliotecas compartilhadas essenciais e módulos do kernel;
- `/media`
  - Ponto de montagem para mídias removíveis: floppy, cdrom, cdrecorder, usbdisks.

Estrutura de diretórios do Linux

Hierarquia visitada

- `/mnt`
  - Ponto de montagem para sistemas de arquivos temporariamente montados;
- `/opt`
  - Contém pacotes adicionais de software;
- `/root`
  - Diretório home do root, super-usuário.

Estrutura de diretórios do Linux

Hierarquia visitada

- `/sbin`
  - Programas usados apenas por administradores do sistema;
  - Binários essenciais para a inicialização, recuperação e reparo do sistema.
- `/srv`
  - Dados de serviços providos pelo sistema.

Estrutura de diretórios do Linux

Hierarquia visitada

- `/tmp`
  - Espaço do sistema disponibilizado a programas e usuários que requeiram arquivos temporários.

- `/usr`
  - Informações estáticas a respeito do computador;
  - Armazena informações e programas.

Estrutura de diretórios do Linux

Hierarquia visitada

- `/var`
  - Contém arquivos de dados variáveis;
  - Inclui diretórios de spool e arquivos, dados administrativos e de log, arquivos transientes e temporários;
  - Um bom administrador deve acompanhar a execução do sistema por meio dos arquivos de log.

Estrutura de diretórios do Linux

Hierarquia visitada

- `/proc`
  - Informações legíveis sobre o sistema;
  - Informações legíveis sobre cada processo executado

`/proc/cpuinfo`	Informações sobre o processador
`/proc/devices`	Lista de dispositivos
`/proc/pci`	Lista de dispositivos PCI
`/proc/version`	Versões do kernel
`/proc/meminfo`	Uso de memória no sistema

Estrutura de diretórios do Linux

Hierarquia visitada

- `/proc`

  `/proc/filesystems` Sistemas de arquivos conhecidos pelo kernel;

  `/proc/ide` Interface ide;

  `/proc/mounts` Sumário de sistemas de arquivo montados

  `/proc/uptime`

Arquivos de Log

- Documentação de cada passo dado pelo do sistema;
- Importantíssimo para a manipulação de um sistema operacional complexo;
- Alguns arquivos interessantes:
  - boot: informações de serviços durante a inicialização do sistema;
  - messages: mensagens gerais relacionadas à operação do sistema;
  - auth: informações relacionadas à segurança, como logins.

Arquivos de Dispositivos

- O Linux interage com o hardware por meio de device drivers, que fazem parte do kernel;
- O device driver oferece uma interface padrão, extremamente parecida com a interface para manipulação de arquivos;
- Expandindo esse conceito, existem device drivers que comunicam-se com o kernel, proporcionando comportamentos especializados.

Arquivos de Dispositivos

- Tipos de dispositivos:
  - dispositivos de caracter: trata fluxo serial de bytes;
  - dispositivos de bloco: trata fluxo de dados em blocos;

- O diretório `/dev` possui as entradas para os dispositivos de caracter e bloco.

- As entradas seguem um certa padronização de nomes e números: major e minor numbers.

Arquivos de Dispositivos

- Dispositivos interessantes:

`/dev/null`	Descarta qualquer dado copiado para ele
`/dev/zero`	Fluxo infinito de zeros
`/dev/full`	Simula arquivo cheio
`/dev/random`	Gerador de números aleatórios
`/dev/urandom`	Gerador de números aleatórios

Mais comandos

du: estima o tamanho dos arquivos

```
$ du -atributos
```

Atributos comuns: -h (humano)

df: apresenta a ocupação do sistema de arquivos

```
$ df -atributos
```

Atributos comuns: -h (humano)

touch: cria um arquivo vazio

```
$ touch arquivo
```

Mais comandos

mkdir: cria diretório

```
$ mkdir -atributos alvo
   Atributos comuns: -p (cria os pais)
```

rmdir: remove diretório vazio

```
$ rmdir -atributos alvo
   Atributos comuns: -p (pais)
```

ln: cria links (atalhos)

```
$ ln -s destinoDoLink nomeDoLink
```

Mais comandos

tar: encapsulador; unido a compressor de dados (TI)

```
$ tar -czvf arquivo.tar.gz origem
$ tar -xzvf arquivo.tar.gz
$ tar -cjvf arquivo.tar.bz2 origem
$ tar -xjvf arquivo.tar.bz2
```

grep: filtro

```
$ grep padrao
```

sort: processamento de ordenação

```
$ sort arquivo
```

Mais comandos

- O `grep` é um filtro poderoso: entendê-lo com profundidade pode poupar bastante trabalho;

- Busca por padrão simples:
  `$ grep padrao arquivo`

- Expressão regular ponto (`.`)
  `$ grep "padrao." arquivo`
  O ponto tem a mesma função do ? no shell

Mais comandos

- Expressão regular `[]`

  `$ grep “padrao[2-5].” arquivo`

  Os valores entre `[]` limitam o espaço de filtragem

- Expressão regular `[^]`

  `$ grep “padrao[^2-5].” arquivo`

  Os valores entre `[]` limitam o espaço de filtragem: lógica inversa

- Expressão regular *

  `$ grep “padrao[2]*.” arquivo`

  A expressão regular precedente pode estar repetida zero ou mais vezes

Mais comandos

- Expressão regular ^

  `$ grep "^padrao" arquivo`

  O caracter ^ representa o início de linha

- Expressão regular $

  `$ grep "padrao$" arquivo`

  O caracter $ representa o fim da linha

Mais comandos

- Argumentos interessantes para `grep`:

  `n` Mostra o número da linha da ocorrência

  `ls` Mostra os arquivos em que foram encontradas ocorrências sem advertência

  `r` Recursivo

Mais comandos

- `sed`: editor de fluxo, muito usado na substituição de padrões e extração de linhas de arquivos;

- Argumentos comuns:

  `s` Substituição

  `g` Substituição global

  `n` Substituição de `n` ocorrências

  `d` Deleta

Mais comandos

sed: editor de fluxo

- Alguns exemplos:

```
$ sed -e 's/Amaral/Julia/' nomes.txt
```

Substitui Julia por Amaral na primeira ocorrência

```
$ sed -e 's/Amaral/Julia/g' nomes.txt
```

Substitui Julia por Amaral em todas as ocorrências

```
$ cat nomes.txt |sed -e
   's/Amaral/Julia/;/Wilson/d'
```

Muda Amaral por Julia e deleta Wilson

Mais comandos

sed: editor de fluxo

- Alguns exemplos:

```
$ sed '1,/Wilson/d' nomes.txt
```

Deleta todas as linhas, a partir da primeira, até Wilson

```
$ sed -e '1,3s/a/%/g' nomes.txt
```

Substitui, da primeira à terceira linha, o caracter `a` pelo caracter `%`

Tarefas úteis com o Linux

Tarefas agendadas

- Automação de tarefas repetitivas
- A `crontab` é usada para programar o `cron`, o serviço resposável pelo disparo de tarefas
- Cada entrada possui o seguinte formato:

`[minutos] [horas] [dias do mês] [mês] [dias da semana] [usuário] [comando]`

Tarefas úteis com o Linux

Tarefas agendadas

- O preenchimento de cada campo é feito da seguinte maneira:

  - Minutos: informe números de 0 a 59;
  - Horas: informe números de 0 a 23;
  - Dias do mês: informe números de 0 a 31;
  - Mês: informe números de 1 a 12;
  - Dias da semana: informe números de 0 a 7;
  - Usuário: é o usuário que vai executar o comando (não é necessário especificá-lo se o arquivo do próprio usuário for usado);
  - Comando: a tarefa que deve ser executada.

Tarefas úteis com o Linux

Tarefas agendadas

- Usar '*' em qualquer campo implica execução constante;
- Usar '–' (exemplo, 0-3) implica execução entre os extremos especificados;
- Usar ',' (exemplo, 3,7) implica execução entre dos extremos especificado;
- Um exemplo:
  - `$ crontab -e`
  - `$ * * * * * /bin/echo "Olha a hora" >> /home/linux/out; date >> /home/linux/out`

Instalando programas

apt-get

- Distros mais tradicionais disponibilizam pacotes pré-compilados e sistemas de gerenciamento de pacotes;
- O Debian usa o formato *.deb para disponibilizar pacotes pré-compilados e o sistema apt para gerenciamento dos mesmo;
- O Fedora usa o formato rpm e o sistema yum.
- Os sistemas de gerenciamento de pacotes permitem a seleção e instalação automática dos mesmos.

Instalando programas

apt-get

- Se o pacote estiver no computador, o sistema irá buscá-lo e instalá-lo;

- Se o pacote não estiver no computador, o sistema procurará na Internet em mirrors as versões mais atuais do pacote e, encontrando-os, irá iniciar o processo de download e instalação.

Instalando programas

apt-get

- Usando o apt-get:

```
$ apt-get update
```

Atualiza base de dados de pacotes

```
$ apt-cache search pacote
```

Busca pacote na base de dados

Instalando programas

apt-get

- Usando o apt-get:

```
$ apt-get install pacote
```

Baixa o pacote e o instala

```
$ apt-get remove pacote
```

Desinstala o pacote

```
$ apt-get upgrade
```

Atualiza o sistema, baixando e instalando novas versões

Instalando programas
*.deb

- Existe a opção de baixar os pacotes *.deb e instalá-los manualmente:

  `$ dpkg -i pacote.deb`

  Instala o pacote

  `$ dpkg -r pacote`

  Desinstala o pacote: note que não há extensão nesse comando.

  `$ dpkg -l [pacote]`

  Lista todos os pacotes instalados ou verifica se `pacote` está instalado

Ferramentas de desenvolvimento

gcc

- GCC: Gnu C Compiler;
- O `gcc` foi um dos primeiros passos na criação de um sistema operacional livre;
- É por meio dele que os códigos fonte são criados em binários executáveis.

Ferramentas de desenvolvimento

gcc

- Um pequeno programa:

```
// Arquivo main.c
#include <stdio.h>
#include “reciprocal.hpp”
int main (int argc, char **argv)
{
    int i;
    i = atoi (argv[1]);
    printf (“The reciprocal of %d is %g\n”, i, reciprocal (i));
    return 0;
}
```

Ferramentas de desenvolvimento

gcc

- Um pequeno programa:

```
// Arquivo reciprocal.cpp
#include <cassert>
#include “reciprocal.hpp”
double reciprocal (int i)
{
    // I should be nonzero.
    assert (i != 0);
    return 1.0/i;
}
```

Ferramentas de desenvolvimento

gcc

- Um pequeno programa:

```
// Arquivo reciprocal.hpp
#ifdef __cplusplus
extern "C" {
#endif
extern double reciprocal (int i);
#ifdef __cplusplus
}
#endif
```

Ferramentas de desenvolvimento

gcc

- Esse é um programa que irá calcular o inverso de um inteiro;

- Ele é misto: possui códigos em C e C++.

Ferramentas de desenvolvimento

gcc

- Para criar a aplicação, é necessário compilar os códigos-fonte.
  - main.c:
    - `$ gcc -c main.c`
  - reciprocal.cpp
    - `$ g++ -c reciprocal.cpp`
- A opção `-c` solicita ao compilador a compilação dos fontes.

Ferramentas de desenvolvimento

gcc

- Se quiser ativar otimização:
  - main.c:
    - `$ gcc -O4 -c main.c`
  - reciprocal.cpp
    - `$ g++ -O4 -c reciprocal.cpp`
- A opção `-O4` solicita ao compilador a otimização nível 4 do código objeto.

Ferramentas de desenvolvimento

gcc

- Uma vez compilados os objetos, devemos linká-los. Como o nosso projeto é misto C/C++, deveremos usar o g++.

  - `$ g++ -o reciprocal main.o reciprocal.o`

- A opção `-o` solicita o *link* dos objetos `main.o` e `reciprocal.o` no binário `reciprocal`.

Ferramentas de desenvolvimento

gcc

- Pronto! Agora é só rodar.

```
$ ./reciprocal 7
```

- Note o ./ antes do nome da aplicação.

- Essa aplicação não usou nenhuma biblioteca diferente da biblioteca padrão C.

Ferramentas de desenvolvimento

gcc

- Para linkar com bibliotecas, usamos a opção `-l`.
- Por exemplo:

```
$ g++ -o reciprocal main.o reciprocal.o
-lm
```

- Esse comando linkará reciprocal à biblioteca matemática.

Ferramentas de desenvolvimento

gcc

- O compilador não se preocupa na resolução de símbolos: ele se preocupa com questões sintáticas e de tradução.
- É o linker que resolverá os símbolos.
- Portanto, no Linux, os passos de compilação e de linkagem são quase estanques.

Ferramentas de desenvolvimento

gcc

- Quando o projeto cresce muito, use:
  - Uma IDE (*integrated development environment*), como o Kdeveloper.
  - O GNU Make. Geralmente as IDEs irão apenas simplificar a aplicação do GNU Make.

Ferramentas de desenvolvimento

gcc

- Para automatizar a compilação e a linkagem de grandes projetos, usamos o `make`.
- O `make` interpretará um arquivo texto chamado `Makefile`;
- Nesse arquivo encontraremos rótulos com algumas instruções para o `make`;

Ferramentas de desenvolvimento

gcc

- Makefile

```
reciprocal: main.o reciprocal.o
   g++ $(CFLAGS) -o reciprocal main.o reciprocal.o
main.o: main.c reciprocal.hpp
   gcc $(CFLAGS) -c main.c
reciprocal.o: reciprocal.cpp reciprocal.hpp
   g++ $(CFLAGS) -c reciprocal.cpp
clean:
   rm -f *.o reciprocal
```

Ferramentas de desenvolvimento

gcc

- Estrutura do Makefile:
  - temos os alvos na esquerda;
  - seguidos dos alvos, após os :, temos a lista de dependências;
  - abaixo do alvo, temos a regra: note o TAB na linha da regra;

- Usando o Makefile:

```
$ make
```

Ferramentas de desenvolvimento

gcc

- O make é esperto o suficiente para identificar quais foram os arquivos modificados;
- Isso permite com que não se perca tempo recompilando arquivos sem modificações;
- Note que, com apenas um comando, compilamos vários arquivos e os linkamos .

Ferramentas de desenvolvimento

gcc

- Variáveis:
  - Note o `$(CFLAGS)`, uma variável que pode ser usada para customizar a compilação.
  - Um exemplo:

```
$ make CFLAGS=-O4
```

Ferramentas de desenvolvimento

gcc+gdb

- Além de um compilador, o que seria de um programador sem depuradores?
- GDB: GNU Debugger
- É um depurador completo, contudo pagamos um preço por sua versatilidade: ele não é nada intuitivo;
- Contudo, é muito melhor investir um tempo aprendendo como usá-lo do que usar os velhos traces usando printf.

Ferramentas de desenvolvimento

gcc+gdb

- Para tanto, usaremos novamente a variável CFLAGS:

```
$ make CFLAGS=-g
```

  `-g`: compila com informações de debug

- Uma vez compilado o program, iniciamos o uso do `gdb`.

```
$ gdb reciprocal
(gdb)
```

Ferramentas de desenvolvimento

gcc+gdb

- Agora, disparamos o programa :

```
(gdb) run
```

- Analisando

```
(gdb) where
(gdb) up 2
(gdb) print argv[1]
```

Ferramentas de desenvolvimento

gcc+gdb

- Inserindo breakpoint:

```
(gdb) break main
```

- Agora

```
(gdb) run 7
(gdb) next
```

- Mais um passo

```
(gdb) step
```

- Passo a passo (executa uma linha)

Ferramentas de desenvolvimento

gcc+gdb

- Outros comandos úteis:

```
(gdb) print variavel
(gdb) display variavel
(gdb) watch variavel
(gdb) set variable variavel=10
(gdb) break arquivo.x:num_linha
(gdb) backtrace
(gdb) file arquivo.x
(gdb) help comando
```