Sistemas Operacionais
Aula Prática 1 - Introdução ao Linux

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PERNAMBUCO

Objetivo

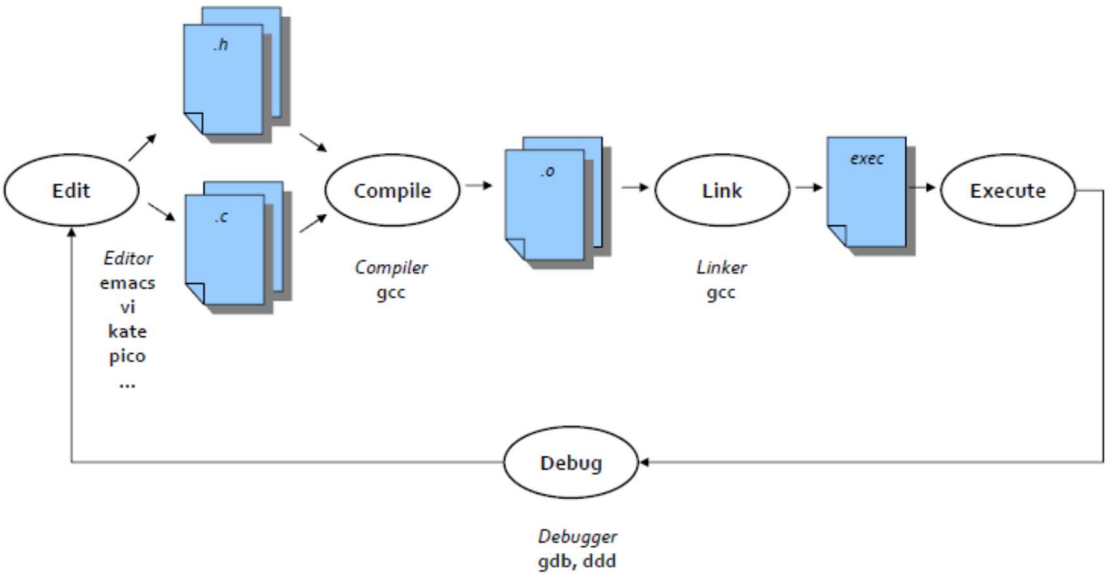
- Introdução ao desenvolvimento de aplicações em linguagem C no ambiente Linux/POSIX
- Familiarização com as ferramentas necessárias: *gcc*, *gdb*, *make*.

Material de Apoio

- Para aula 1 baixe o material disponível em http://siep.ifpe.edu.br/anderson/arquivos/so_aula1_pratica.zip
- É indispensável o entendimento do SO Linux. Caso precise de ajuda acesse o site http://siep.ifpe.edu.br/anderson/arquivos/linux.zip e busque informações de utilização.

Introdução

Observe o ciclo de desenvolvimento de programas em C:



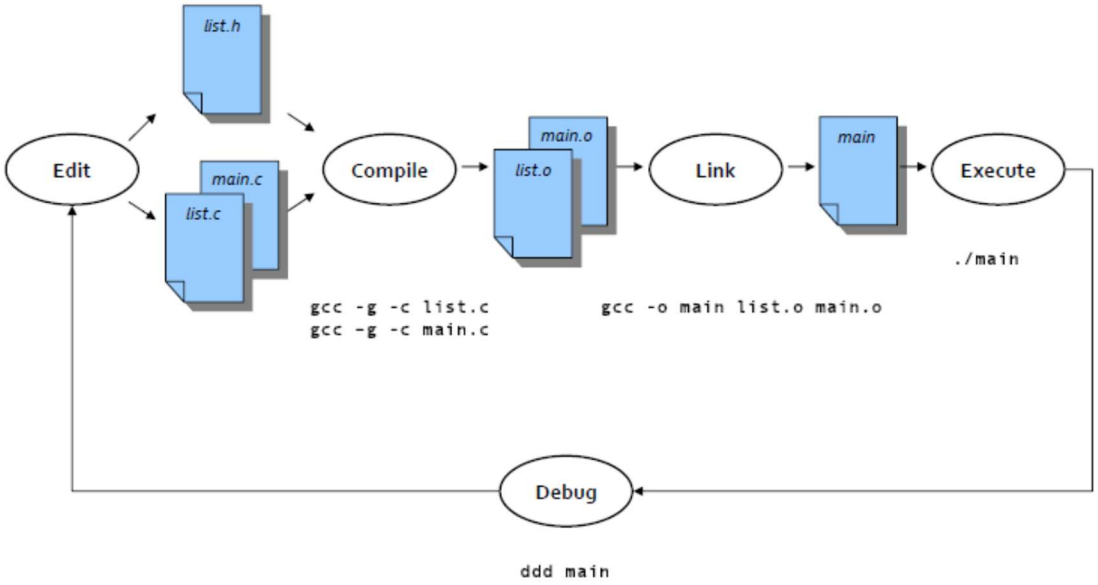
Analise o que o esquema acima representa.

Baixe e descompacte o arquivo baixado no ambiente Linux. O comando a seguir é uma sugestão, se você quiser opte pelo seu conhecimento.

```
unzip so_aula1_pratica.zip
```

1. Geração do executável

a) Visualize e compreenda o conteúdo dos arquivos.



b) Compile os arquivos e faça a sua ligação de modo a produzir o executável denominado *main*.

```
gcc -g -c list.c main.c
gcc -o main list.o main.o
```

c) Execute a aplicação *main* e compreenda o resultado.

d) Discuta os resultados com seu grupo e em sala de aula.

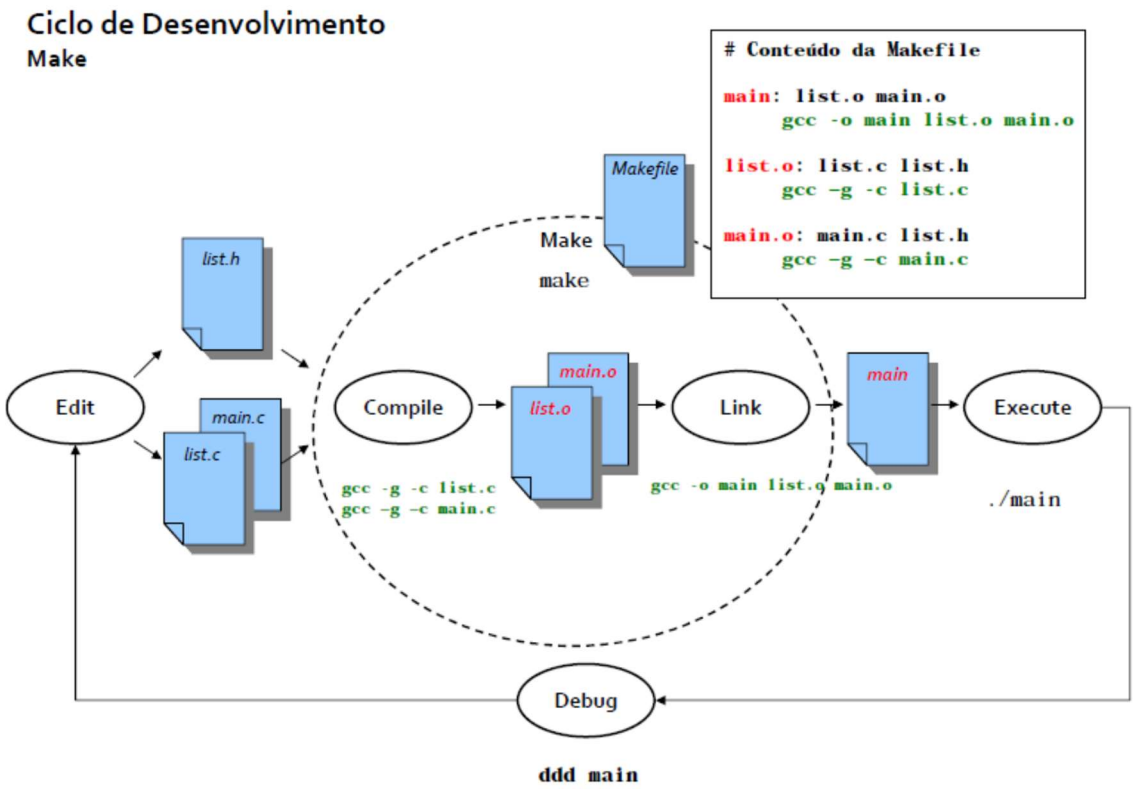
2. Utilização do debugger (opcional)

a) Execute a aplicação no debugger de alguma IDE. No caso do exemplo utilizamos o Eclipse IDE com CDT. Porque tem esta saída? Explique.

b) Coloque um *breakpoint* na primeira instrução `lst_insert` e execute a aplicação pressionando o botão *run*. O que observa?

c) Execute a aplicação utilizando os botões *step into* e *step over*. Qual é a diferença?

3. Utilização da ferramenta *make*



a) Leia o arquivo *Makefile*, pelo comando `cat Makefile` execute *make*. O que aconteceu? O que significa as opções –o, –g e –c na linha de comando do gcc?

b) Apague o arquivo *list.o*. Re-execute *make*. Interprete o resultado.

c) Simule uma alteração do arquivo *main.c* com o comando abaixo e re-execute *make*. O que aconteceu com o resultado?

```
touch main.c
```

Exercícios

1. Implemente a função `lst_insert` do exemplo anterior que adiciona um valor no final da lista. **Sugestões**:

- Utilize a *Makefile* anterior para gerar o executável sempre que alterar o código fonte.
- Teste se os valores são devidamente adicionados à lista utilizando algum *debugger*: coloque um *breakpoint* numa invocação à função `lst_insert` e observe o conteúdo atual das variáveis.
- Se a aplicação terminar com uma violação de memória o *debugger* permite identificar a instrução responsável. Aprenda como.

2. Implemente a função `lst_print` do arquivo *list.c* que imprime o conteúdo de uma lista (inteiros separados por vírgulas). Utilizando um debugger, verifique que a lista impressa pela função `lst_print` corresponde ao conteúdo efetivo da lista em memória.

3. Compreenda melhor o funcionamento do *make.*

a) Simule a alteração do ficheiro *list.h* e execute `make`. Porque razão todos os arquivos foram gerados?

b) Simule a alteração do arquivo *list.o*. O que acontece quando faz `make list.o`? E se agora fizer `make`?

c) Retire a dependência do arquivo *list.h* da regra *list.o* da *Makefile*. Repita procedimento da alínea a). Explique a diferença no resultado?

d) Adicione a regra seguinte no fim do arquivo. O que descreve esta regra? Identifique: o alvo, as dependências e o comando.

```
clean:
	rm -f *.o main
```

e) Execute `make clean`. O que aconteceu? Porque razão o comando é executado sempre que esta regra é invocada explicitamente?

4. Implemente as função `lst_destroy` e `lst_remove`. A função `lst_destroy` liberta toda a memória alocada pela lista. A função `lst_remove` recebe uma lista e um valor e remove o primeiro item com aquele valor.

Exercício Desafio 1

Calculadora RPN (Reverse Polish Notation)

Criar uma calculadora capaz de fazer operações de soma, subtracção, multiplicação e divisão, recebendo como entrada expressões na notação RPN. A notação RPN permite definir expressões aritméticas evitando o uso de parêntesis para definir as prioridades na avaliação das operações. As operações/expressões ilícitas - como, por exemplo, a falta de operandos e/ou operadores; divisão por zero - devem ser identificadas com mensagens .Utilize uma pilha como estrutura de dados auxiliar (*sugestão*: estender a biblioteca lista com operações `push` e `pop`).

Exemplos:

```
/* 8 - 4 (notação tradicional) */
> 8 4 -
Resultado: 4

/* 3 + (20 + 5) (notação tradicional) */
```

```
> 3 20 5 + +
Resultado: 28

/* (10 - 1) - (6 - 3) (notação tradicional) */
> 10 1 - 6 3 - -
Resultado: 6

/* 10 - 1 - 6 - 3 (notação tradicional) */
> 10 1 - 6 - 3 -
Resultado: 0

/* 10 : 0 (notação tradicional) */
> 10 0 :
Resultado: DIVISAO POR ZERO

/* Expressõos ilícitas) */
> 10 1
Resultado: FALTAM OPERADORES

> 10 1 + 1 5
Resultado: FALTAM OPERADORES

> + 1 5
Resultado: OPERAÇÂO NÃO VÁLIDA

> 1 5 + A
Resultado: OPERAÇÂO NÃO VÁLIDA
```