MC-202 — Aula de Laboratório
Compilação e Depuração

Yulle Glebbyo[1]

Universidade Estadual de Campinas

1º semestre/2018

[1] Esta versão possui algumas modificações realizadas pelos monitores das turmas ABCDE do semestre corrente.

Na Aula Anterior

Software de Máquina Virtual para estabelecer um ambiente de trabalho

- VirtualBox

Nosso ambiente de desenvolvimento:

- Debian 9.1 - Stretch;
- Emulador de Console: Konsole;
- Editor de Texto Moderno: Atom;
- Navegador: Mozilla Firefox;
- Ferramentas de compilação e *debugging*.

Na Aula Anterior (Cont.)

Comando	Uso	Descrição
`pwd`	`pwd`	print working directory: imprime o diretório atual
`ls`	`ls <opcional_dir>`	list contents: lista o conteúdo do diretório atual
`cd`	`cd <diretorio>`	change directory: muda para diretório passado como parâmetro;
`mkdir`	`mkdir <novo_dir>`	make directory: cria um novo diretório;
`mv`	`mv <arq> <dir>`	move: move arquivo para diretório destino;
`cp`	`cp <arquivo> <copia>`	copy: cria uma cópia de um arquivo;
`rm`	`rm <arquivo>`	remove: remove permanentemente um arquivo
`rm -rf`	`rm -rf <diretorio>`	recursive, force: remove recursivamente um diretório e todos seus conteúdos
`diff`	`diff <arq1> <arq2>`	difference: imprime na tela a diferença entre arq1 e arq2

Compilação

É a tradução do código em linguagem humana para linguagem de máquina, para que possa ser executado pelo computador

Utilizamos o programa `gcc` para realizar esse processo com o padrão `C-ANSI`

Para compilar um código usamos o comando:

- `$ gcc arquivo.c -ansi -Wall -pedantic-errors -g -o arquivo.x`

Se precisamos compilar múltiplos arquivos, usamos:

- `$ gcc arquivo01.c -ansi -Wall -pedantic-errors -g -c arquivo01.o`
- `$ gcc arquivo02.c -ansi -Wall -pedantic-errors -g -c arquivo02.o`
- `$ gcc arquivo01.o arquivo02.o -o arquivo.x`

make

`make` é um programa que auxilia e automatiza o processo de compilação

Para compilar projetos com múltiplos arquivos, podemos criar um script `Makefile` para ser executado pelo `make`

```
1  CC=gcc
2  CFLAGS=-ansi -Wall -pedantic-errors -g -lm
3
4  all: arquivo.x
5
6  arquivo.x: arquivo01.o arquivo02.o
7   $(CC) arquivo01.o arquivo02.o -o arquivo.x
8
9  arquivo01.o: arquivo01.c arquivo01.h
10  $(CC) arquivo01.c -c $(CFLAGS)
11
12 arquivo02.o: arquivo02.c arquivo02.h
13  $(CC) arquivo02.c -c $(CFLAGS)
14
15 clean:
16   rm -f *.o *.x
```

make (Cont.)

É possível compilar individualmente cada um dos alvos:

- `$ make arquivo01.o`
  - gera o código de máquina arquivo01.o a partir dos arquivos `.c` e `.h`

Ou compilar todos os arquivos e gerar o executável:

- `$ make`
  - executa o alvo `all`, gerando todos os `.o` e ligando-os para gerar o `.x`

Por fim, é possível remover todos os arquivos de compilação gerados até então:

- `$ make clean`
  - executa o alvo `clean`, que remove todos os arquivos de extensão `.o` e `.x`

Depuração

É o processo de encontrar, corrigir ou mitigar os efeitos de características indesejadas no código

Utilizamos o programa `gdb` para realizar este processo com o padrão `C-ANSI`

- `gdb`: GNU Debugger
- Ajuda a encontrar erros semânticos e comportamentos indesejados no programa
- Alternativa a utilizar vários `printf` para verificar o fluxo e valores de variáveis do programa
- Para ser habilitado, é necessário que o código a ser depurado seja compilado com a flag `-g`

gdb

Funciona como um terminal de linha de comandos

- autocomplete usando a tecla `tab`
- histórico de comandos com ↑ e ↓
- use `help` ou `help <comando>` para ajuda

Carregue programas a partir do terminal:

- utilize o comando `$ gdb arquivo.x`

Ou de dentro do `gdb`:

- inicialize o `gdb` com `$ gdb` seguido pelo comando `(gdb) file arquivo.x`

gdb (Cont.)

Breakpoints

- cria pontos de parada em localizações específicas dos arquivos de código
- quando o gdb encontra o um breakpoint, ele interrompe a execução do programa para que ele possa ser analisado

Watchpoints

- um ponto de observação em alguma variável do programa
- sempre que a variável é atualizada, a execução é interrompida e seus valores anteriores são mostrados

Querying

- quando a execução esta interrompida, podemos executar ações sob o código no escopo atual de execução
- é possível imprimir conteúdo de variáveis, e até mesmo forçar a invocação de funções do seu código

Alguns Comandos do gdb

`(gdb) run`

- executa o programa carregado anteriormente

`(gdb) break <arquivo.c>:<linha>`

- cria um breakpoint na linha `<linha>` do arquivo `<arquivo.c>`

`(gdb) break <nome_funcao>`

- cria um breakpoint na declaração da função `<nome_funcao>`

`(gdb) watch <variavel>`

- cria um watchpoint para a variável `<variavel>`

Alguns Comandos do gdb (Cont.)

`(gdb) continue`

- resume a execução do programa após uma interrupção

`(gdb) step`

- executa a próxima linha do programa após uma interrupção

`(gdb) next`

- executa o próximo bloco de comandos após uma interrupção

`(gdb) print <variavel>`

- imprime o conteúdo da variável `<variável>`

`(gdb) call <expressao>`

- força a execução de `<expressao>` a partir dos símbolos definidos no escopo atual

Verificação de Memória

Utilizamos o programa `valgrind` para:

- Verificar se toda a memória da Heap está sendo adequadamente desalocada, ou seja, se para cada chamada à função `malloc` há um `free` correspondente;
- Identificar acessos indevidos à memória alocada na Heap.

Supondo que a chamada do seu programa seja:
`./arquivo.x arg1`

O diagnóstico é feito com o comando:
`valgrind --leak-check=yes ./arquivo.x arg1`

A opção `--leak-check=yes` serve para listar separadamente todos os problemas de memória não liberada.

Yulle Glebbyo

glebbyo@ic.unicamp.br

IC/UNICAMP