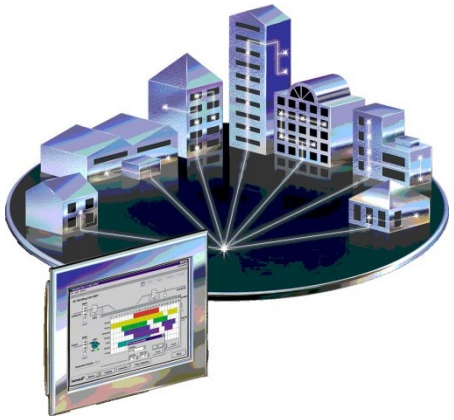
Revisão da Linguagem C



Programação Paralela Avançada - PPA

Mestrado em Computação Aplicação – MCA
Programa de Pós-Graduação em Computação Aplicada – PPGCA
Centro de Ciências Tecnólogicas - CCT
Universidade do Estado de Santa Catarina – UDESC

Profs Maurício A. Pillon e Guilherme P. Koslovski
Linha de Sistemas Computacionais
Grupo de Pesquisa de Redes de Computadores e Sistemas Distribuídos
Laboratório de Pesquisa LabP2D

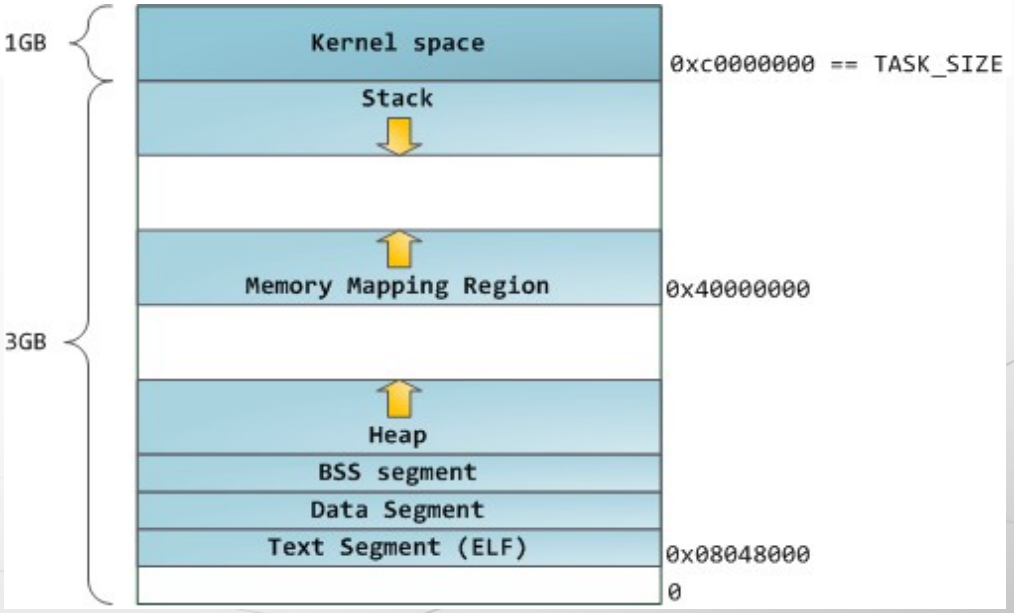
Agenda

- Anatomia de um programa na memória.
- Bem-vindo a linguagem de programação C.
- Ponteiros em C.
- Estudo de caso: Matrizes.
- Exercícios de revisão.

Anatomia de um programa na memória

Processos têm um espaço de endereçamento virtual próprio (blocos de 4GB)

- ***Stack***: armazena variáveis locais e parâmetros das funções;
- ***Memory mapping region***: região com o mapeamento direto do conteúdo de arquivos; e mapeamento anônimo de memória (“grandes” malloc());
- ***Heap***: alocação de memória em tempo de execução; em C, a interface de alocação é malloc();
- ***BSS***: armazena o conteúdo de variáveis globais não inicializadas no código fonte;
- ***Data***: mesmo que BSS, porém sobre variáveis inicializadas no código fonte;
- ***Text***: armazena o código fonte (read-only) e textos do programa.



Layout de um segmento padrão de um processo no Linux.

Bem-vindo a Linguagem C

- Parâmetros do Shell (main);
- Procedimento em C;
- Função em C;
- *Cast* em C;

Compilando/Executando:

- Opções de compilação

```
~ $ gcc -Wall -g -O3 10_revisaoC.c -o rev_10.exe
~ $
~ $
~ $ ./rev_10.exe 20 3
```

- Aconselha-se um ***man gcc***

Resultado:

```
(main) retorno função = 60
(procedimento) var_local = 60
```

```c
#include <stdio.h>
#include <stdlib.h>
// Formato de uma função em C;
// <tipo de retorno> <nome_da_função> (<parâmetro1,parâmetro2>)
int funcao (int par1, float par2) {
  int var_local; // válida somente dentro da função;
  var_local = par1 * (int) par2; // cast de float p/ inteiro
  return var_local; // retorno de um inteiro;
}
// Formato de um procedimento em C;
// void <nome_da_função> (<parâmetro1,parâmetro2>)
void procedimento (int par1, float par2) {
  int var_local; // válida somente dentro da função;
  var_local = par1 * (int) par2;
  printf ("(procedimento) var_local = %d\n",var_local);
}

// Função principal
// argc = número de elementos passados como parâmetro;
// argv = um vetor com o conteúdo dos parâmetros;
int main(int argc, char **argv) {

  if (argc != 3) {
    printf("ERRO: Número de parâmetros deve ser 3.\n");
    return 1; // Indicativo de erro.
  }

  printf ("(main) retorno função = %d\n",funcao (atoi(argv[1]),atof(argv[2])));
  procedimento (atoi(argv[1]),atof(argv[2]));

  return 0; // Retorno com sucesso.
}
```

Linguagem C: Condicionais.

- Tipo de condicionais:
  - **switch** (<expressão>) {
    case <valor>:
    <bloco1>
    case <valor>:
    <bloco2>
    default:
    <bloco3>
    }
  - **if** (<condição>) {
    <bloco1>
    } **else if** (<condição>) {
    <bloco2>
    } **else** {
    <bloco3>
    }
  - Operador ternário:
    <expressão> ? <SIM> : <NÃO>;

```c
#define N 10

int main(int argc, char **argv) {
  int var_i=0;
  int var_j=10;
  char operador = '/';

  operador = argv[1][0];
  switch (operador) {
    case '+':
      printf("soma\n");
      break;
    case '-':
      printf("substracao\n");
      break;
    default:
      printf ("Erro: operador não reconhecido!\n");
    }

  var_i = atoi (argv[2]);
  if (var_i > N) {
    var_i += 10;
  } else if (var_i > var_j) {
    var_j *= var_i;
  } else {
    var_i -= var_j;
  }
  var_i >= var_j ? var_i++ : var_i--;
  printf("var_i/var_j = %d/%d\n", var_i,var_j);
  return 0;
}
```

Qual é a função do break?

O que acontece se:
(i) operador = '-' e
(ii) removermos o break?

Linguagem C: Loops.

- Declaração de uma constante
- Tipo de *Loop* em C:

  - **for** (i = <inicio>; i < <fim>; i++) {
    <código>
    }

  - **while** (<condição>) {
    <código>
    }

  - **do** {
    <código>
    } **while** (<condição>);

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main(int argc, char **argv) {
  int wi=15;
  int di=0;

  for (int fi = 5; fi <= N; fi++){
     printf ("(for) %d\n",fi);
  }

  while ( N <= wi) {
    printf ("(while) %d\n",wi);
    wi--;
  }

  do {
    printf ("(do/while) %d\n",di);
    di+=2;
  } while(di <= N);
  return 0;
}
```

```
(for) 5
(for) 6
(for) 7
(for) 8
(for) 9
(for) 10
(while) 15
(while) 14
(while) 13
(while) 12
(while) 11
(while) 10
(do/while) 0
(do/while) 2
(do/while) 4
(do/while) 6
(do/while) 8
(do/while) 10
```

Ferramenta de análise dinâmica

- Valgrind (http://valgrind.org/)

```
mpillon@MintPositivo ~ $ gcc -Wall -g 10_revisaoC.c -o rev10.exe
mpillon@MintPositivo ~ $ valgrind ./rev10.exe 10 10.9
==23020== Memcheck, a memory error detector
==23020== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et
==23020== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyrigh
==23020== Command: ./rev10.exe 10 10.9
==23020==
(main) retorno função = 100
(procedimento) var_local = 100
==23020==
==23020== HEAP SUMMARY:
==23020==     in use at exit: 0 bytes in 0 blocks
==23020==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==23020==
==23020== All heap blocks were freed -- no leaks are possible
==23020==
==23020== For counts of detected and suppressed errors, rerun with: -v
==23020== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mpillon@MintPositivo ~ $
```

```c
#include <stdio.h>
#include <stdlib.h>
// Formato de uma função em C;
// <tipo de retorno> <nome_da_função> (<parâmetro1,parâmetro2>)
int funcao (int par1, float par2) {
  int var_local; // válida somente dentro da função;
  var_local = par1 * (int) par2; // cast de float p/ inteiro
  return var_local; // retorno de um inteiro;
}
// Formato de um procedimento em C;
// void <nome_da_função> (<parâmetro1,parâmetro2>)
void procedimento (int par1, float par2) {
  int var_local; // válida somente dentro da função;
  var_local = par1 * (int) par2;
  printf ("(procedimento) var_local = %d\n",var_local);
}

// Função principal
// argc = número de elementos passados como parâmetro;
// argv = um vetor com o conteúdo dos parâmetros;
int main(int argc, char **argv) {

  if (argc != 3) {
    printf("ERRO: Número de parâmetros deve ser 3.\n");
    return 1; // Indicativo de erro.
  }

  printf ("(main) retorno função = %d\n",funcao (atoi(argv[1]),atof(argv[2])));
  procedimento (atoi(argv[1]),atof(argv[2]));

  return 0; // Retorno com sucesso.
}
```

Ferramenta de análise dinâmica

- Onde está o erro neste programa?
- Por que o GCC não indica o erro?

Ops! Não tem erro!
Desculpe, pessoal!

13_revisaoC.c

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main(int argc, char **argv) {
  int *vet_i;
  vet_i = malloc(sizeof(int)*10);
  for (int fi = 0; fi <= N; fi++){
     vet_i[fi] = fi * N;
     printf ("(for) vet_i[%d]%d\n",fi,vet_i[fi]);
  }
  return 0;
}
```

```
mpillon@MintPositivo ~ $
mpillon@MintPositivo ~ $ gcc -Wall -g 13_revisaoC.c -o rev13.exe
mpillon@MintPositivo ~ $ ./rev13.exe
(for) vet_i[0]0
(for) vet_i[1]10
(for) vet_i[2]20
(for) vet_i[3]30
(for) vet_i[4]40
(for) vet_i[5]50
(for) vet_i[6]60
(for) vet_i[7]70
(for) vet_i[8]80
(for) vet_i[9]90
(for) vet_i[10]100
mpillon@MintPositivo ~ $
```

Ferramenta de análise dinâmica

- Qual é o tamanho do vetor?
- Quantas posições são afetadas no loop?

13_revisaoC.c

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main(int argc, char **argv) {
  int *vet_i;
  vet_i = malloc(sizeof(int)*10);
  for (int fi = 0; fi <= N; fi++){
     vet_i[fi] = fi * N;
     printf ("(for) vet_i[%d]%d\n",fi
  }
  return 0;
}
```

```
mpillon@MintPositivo ~ $ valgrind ./rev13.exe
==24348== Memcheck, a memory error detector
==24348== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==24348== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==24348== Command: ./rev13.exe
==24348==
(for) vet_i[0]0
(for) vet_i[1]10
(for) vet_i[2]20
(for) vet_i[3]30
(for) vet_i[4]40
(for) vet_i[5]50
(for) vet_i[6]60
(for) vet_i[7]70
(for) vet_i[8]80
(for) vet_i[9]90
==24348== Invalid write of size 4
==24348==    at 0x4005AD: main (13_revisaoC.c:9)
==24348==  Address 0x5203068 is 0 bytes after a block of size 40 alloc'd
==24348==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==24348==    by 0x40057E: main (13_revisaoC.c:7)
==24348==
==24348== Invalid read of size 4
==24348==    at 0x4005C3: main (13_revisaoC.c:10)
==24348==  Address 0x5203068 is 0 bytes after a block of size 40 alloc'd
==24348==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==24348==    by 0x40057E: main (13_revisaoC.c:7)
==24348==
(for) vet_i[10]100
==24348==
==24348== HEAP SUMMARY:
==24348==     in use at exit: 40 bytes in 1 blocks
==24348==   total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
==24348==
==24348== LEAK SUMMARY:
==24348==    definitely lost: 40 bytes in 1 blocks
==24348==    indirectly lost: 0 bytes in 0 blocks
==24348==      possibly lost: 0 bytes in 0 blocks
==24348==    still reachable: 0 bytes in 0 blocks
==24348==         suppressed: 0 bytes in 0 blocks
==24348== Rerun with --leak-check=full to see details of leaked memory
==24348==
==24348== For counts of detected and suppressed errors, rerun with: -v
==24348== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
mpillon@MintPositivo ~ $
```

Vamos perguntar para o Valgrind?

GNU Project Debugger (GDB)

- Valgrind não permite que o programa seja interrompido!
  - o programa não é analisado passo-a-passo.

Então, o que devo usar para depurar o meu código?

- GNU Project Debugger – GDB (https://www.gnu.org/software/gdb/)

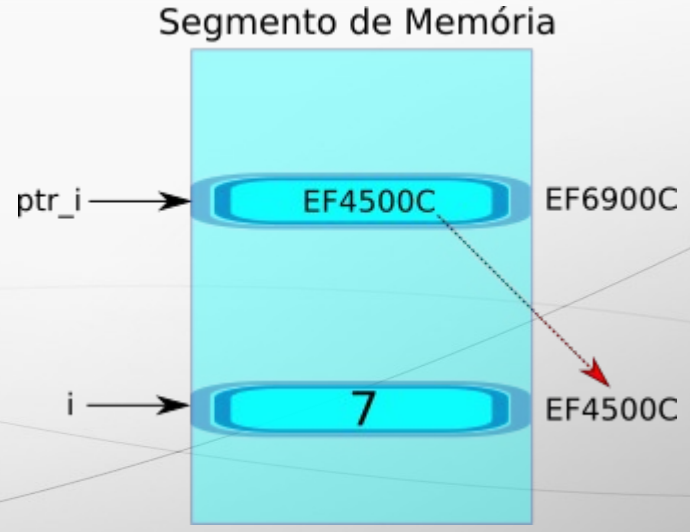
- Tutorial do GDB
  http://www.lrc.ic.unicamp.br/~luciano/courses/mc202-2s2009/tutorial_gdb.txt

Ponteiros em C

- Ponteiros são variáveis cujos valores são endereços de memória;
- A referência a um valor, por meio de um ponteiro, é feita de forma indireta;
- O **operador de endereço (&)** é unário e retorna o endereço da variável que o procede.
- O **operador de indireção (*)** é unário e retorna o valor do conteúdo apontado pelo ponteiro.

```
1 int main () {
2       int i = 7;
3       int *ptr_i = NULL;
4
5       ptr_i = &i;
6       i++;
7 }
```

- *Qual é o valor de ptr_i na linha 7?*



Ponteiros em C

Operações sobre ponteiros:

- Deslocamento de ponteiro;

- O que será impresso na tela?

```
5   int main(int argc, char **argv) {
6     int   valor[3] = {10,20,30};
7     int   *ptr_valor = NULL;
8
9     ptr_valor = &valor[0];
10    if (*(++ptr_valor) == valor[1])
11            printf ("OK (1)\n");
12    ptr_valor = &valor[0];
13      printf ("%d\n", *ptr_valor);
14    if (*(ptr_valor++) == valor[1])
15            printf ("OK (2)\n");
16      return 0;
17  }
```

Passagem de parâmetros

- ***Passagem por valor:*** os valores de *int_p1* e *float_p2* na linha 9 são 5 e 5.1, respectivamente.

```
1 int pass_valor (int par1, float par2) {
2       par1 = 10;
3       par2 = 10.10;
4}

5 (…)
6 int_p1 = 5;
7 float_p2 = 5.1;
8 pass_valor (int_p1, float_p2);
9 (…)
```

Passagem de parâmetros

- ***Passagem por referência:*** os valores de *int_p1* e *float_p2* na linha 9 são 10 e 10.1, respectivamente.

```
1 int pass_valor (int *par1, float *par2) {
2       *par1 = 10;
3       *par2 = 10.10;
4}

5 (…)
6 int_p1 = 5;
7 float_p2 = 5.1;
8 pass_valor (&int_p1, &float_p2);
9 (…)
```

- Neste caso, observa-se que as variáveis ***par1*** e ***par2***, com tipos, ***int*** e ***float***, passaram a ser referenciadas por ponteiros no interior da função. Dessa forma, o conteúdo atribuído dentro da função altera diretamente o endereço de memória alocada no ***main***.

Passagem de parâmetros

- Se a passagem de parâmetro de uma variáveis ***int*** exige a utilização de um ponteiro, como farei a passagem por referência de um ponteiro?

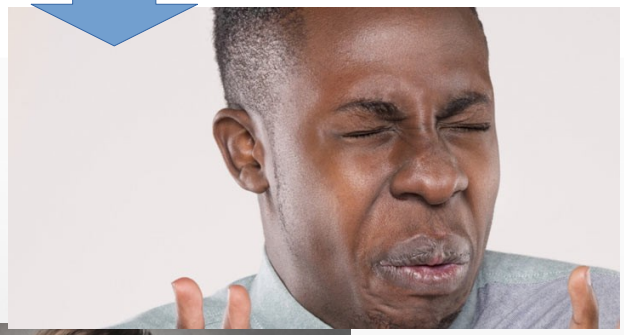
- ***Passagem por referência de um ponteiro:*** os valores de **int_p1* e **float_p2* na linha 9 são 10 e 10.1, respectivamente.

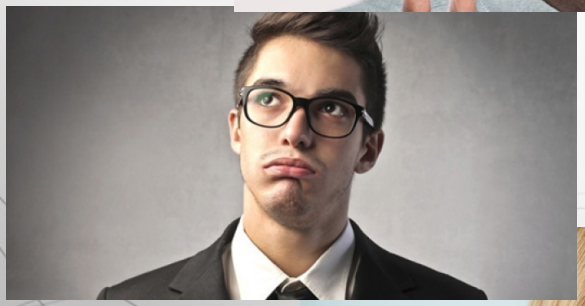
```
1 int pass_valor (int **par1, float **par2) {
2       *(*par1) = 10;
3       *(*par2) = 10.10;
4}

5 (…)
6 *int_p1 = 5;
7 *float_p2 = 5.1;
8 pass_valor (&int_p1, &float_p2);
9 (…)
```

Exercícios de revisão:

Vocês





Programação Paralela

Paraíso ...