Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

TP2
Ferramentas de Debug e Serialização

Vinicius Cogo

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

Fonte da imagem: http://www.dsn.jhu.edu

GDB

- GNU project debugger (http://www.gnu.org/s/gdb/)
  - Permite ver o que se passa dentro do programa enquanto ele está a ser executado (ou quando *crashou*!).

- Quatro tarefas principais
  - Executar o programa
  - Fazê-lo parar em determinado pontos ou quando determinada condição ocorre
  - Examinar o que ocorreu quando o programa parar
  - Corrigir bugs

GDB: exemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
        char *s1 = "Hello World";
        char *s2;

        int len = strlen(s1);
        s2 = (char *) malloc(len);
        strcpy(s2,s1);
        printf("%s\n",s2);
        return 0;
}
```

```
$ gcc -g –o teste teste.c
 ./teste
Hello World
```

Ciências ULisboa

GDB: exemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
        char *s1 = "Hello World";
        char *s2;

        int len = strlen(s1);
        s2 = (char *) malloc(len);
        strcpy(s2,s1);
        printf("%s\n",s2);
        return 0;
}
```

```
$ gcc -g –o teste teste.c
 ./teste
Hello World
```

Parece estar tudo bem!

Ciências ULisboa

```
$ gdb teste
GNU gdb (Ubuntu/Linaro 7.2-1ubuntu11) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/fvramos/workspace/teste/src/teste...done.
(gdb) list
1               #include <stdio.h>
2               #include <stdlib.h>
3               #include <string.h>
4
5               int main() {
6                               char *s1 = "Hello World";
7                               char *s2;
8
9                               int len = strlen(s1);
10                              s2 = malloc(len);
(gdb)
11                              strcpy(s2,s1);
12                              printf("%s\n",s2);
13                              return 0;
14              }
(gdb)
```

```
(gdb) break 11
Breakpoint 1 at 0x8048485: file teste.c, line 11.
(gdb) run
Starting program: /home/fvramos/workspace/teste/src/teste

Breakpoint 1, main () at teste.c:11
11                      strcpy(s2,s1);
(gdb) print s2
$1 = 0x804b008 ""
(gdb) print s1
$2 = 0x8048570 "Hello World"
(gdb) print *s2
$3 = 0 '\000'
(gdb) print *s1
$4 = 72 'H'
(gdb) print *(s1+3)
$5 = 108 'l'
(gdb) print s1[3]
$6 = 108 'l'
```

```
(gdb) break 13
Breakpoint 2 at 0x400645: file teste.c, line 13.
(gdb) continue
Hello World

Breakpoint 2, main () at teste.c:13
13                  return 0;
(gdb) print s2
$1 = 0x602010 "Hello World"
(gdb) next
14          }
(gdb) quit
```

GDB: Sumário de Comandos

- **gdb** *program* – executar o programa *program* com o gdb
- **help –** obter informação de ajuda sobre o gdb
- **quit** - terminar a sessão
- **run** - executar o programa desde o início
- **run** *arg1 arg2 argx* **–** executar o programa desde o inicio e com argumentos
- **list** – listar código fonte
- **list** *func / line* – listar o código da função *func* / à volta da linha *line*
- **break** *fun* / n – definir breakpoint no ínicio da função *fun* / linha *n*
- **info** *break* – saber que breakpoints existem
- **delete** *n* - remover o breakpoint número *n*

GDB: Sumário de Comandos

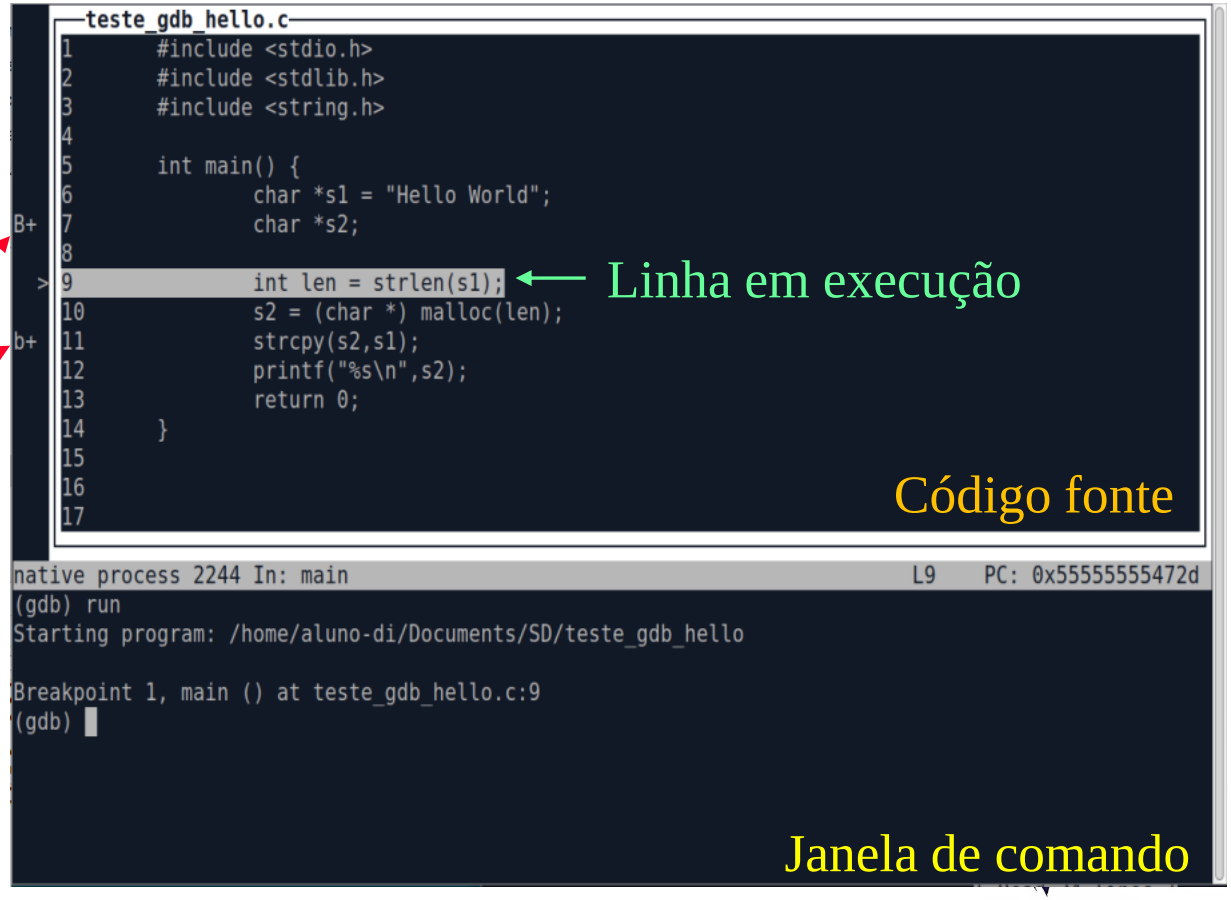
- **next** - executar a próxima linha
- **next** *n* – executar as próximas *n* linhas
- **step** - executar a próxima linha (mas entra dentro de funções)
- **continue** - continuar a execução até ao próximo breakpont (ou até ao final)
- **print** *var* - calcular e mostrar o valor duma variável *var*
- **print** **pvar* – mostrar o conteúdo da zona de memória apontada por p*var*
- **display** *expr* - mostrar o valor da expressão sempre que parar
- **set var** *nome* = *expr* - modificar o valor duma variável
- **where** – mostrar em que função se está a executar e conteúdo da stack
- **up / down** – subir / descer na stack

  - Nota Final: podem abreviar instruções usando apenas a sua primeira letra
    - » Ex: usar apenas **r** em vez de **run**

GDB: Sumário de Comandos

- **layout next / prev** – escolhe o layout da Text User Interface (TUI). Diferentes informações podem ser apresentadas simultaneamente na TUI conforme o layout escolhido (5 opções de layout):
  1. source
  2. assembly
  3. source and assembly
  4. source and registers
  5. assembly and registers

Exemplo



GDB: Exercício

1. Obtenha as notas da prova e trabalho do *aluno_2* através do GDB (sem alterar o código fonte);
2. Verifique a execução do código com o GDB. Observe o que ocorre com o *aluno_1* e efetue as alterações necessárias para que a sua classificação final seja a correta (“10”) e a sua situação seja justa (“Aprovado”), considerando as notas que recebeu (trabalho = 10.5; prova = 9.5).

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>


typedef struct {
    char nome[20];
    float trabalho;
    float prova;
    float classificacao_final;
    char situacao[10];
} registo;

void regista_aluno_1(registo *aluno){

    char nome[20];

    // Regista o nome do aluno
    printf("Digite o nome do aluno 1.\n");
    scanf("%s",nome);
    strcpy(aluno->nome,nome);

    // Regista as notas do aluno
    aluno->trabalho = 10.5;
    aluno->prova = 9.5;
}


void regista_aluno_2(registo *aluno){

    char nome[20];

    // Regista o nome do aluno
    printf("Digite o nome do aluno 2.\n");
    scanf("%s",nome);
    strcpy(aluno->nome,nome);

    // Regista as notas do aluno
    srand(2); // Fixa a semente. Neste caso, poderia ser omitido.
    aluno->trabalho = 0.5 * (rand() % 41);
    aluno->prova = 0.5 * (rand() % 41);
}
```

```c
void classifica_aluno(registo *aluno){

    // Calcula a classificacao final do aluno
    int prova = aluno->prova;
    int trabalho = aluno->trabalho;
    aluno->classificacao_final = (prova + trabalho)/2;

    // Define a situacao do aluno
    if (aluno->classificacao_final <10)
        strcpy(aluno->situacao,"Reprovado");
    else
        strcpy(aluno->situacao,"Aprovado");


}

int main(){

    registo *aluno_1;
    aluno_1 = (registo *)  malloc(sizeof(registo));

    registo *aluno_2;
    aluno_2 = (registo *) malloc(sizeof(registo));

    regista_aluno_1(aluno_1);
    regista_aluno_2(aluno_2);

    classifica_aluno(aluno_1);
    classifica_aluno(aluno_2);


    printf("A classificacao do aluno %s e: %f.\n", aluno_1->nome, aluno_1->classificacao_final);
    printf("A situacao do aluno %s e: %s.\n\n", aluno_1->nome, aluno_1->situacao);

    printf("A classificacao do aluno %s e: %f.\n", aluno_2->nome, aluno_2->classificacao_final);
    printf("A situacao do aluno %s e: %s.\n\n", aluno_2->nome, aluno_2->situacao);

    free(aluno_1);
    free(aluno_2);

    return 0;

}
```

Valgrind

- Valgrind (http://valgrind.org/)
  - Auxilia o trabalho de depuração de programas.

- Diversas funcionalidades que permitem inspecionar os programas em grande detalhe
  - Deteta erros decorrentes do uso incorreto da memória dinâmica
  - Deteta erros na criação e gestão de threads
  - Etc., etc.

Valgrind: retomando o exemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
        char *s1 = "Hello World";
        char *s2;

        int len = strlen(s1);
        s2 = malloc(len);
        strcpy(s2,s1);
        printf("%s\n",s2);
        return 0;
}
```

```
$ gcc -g –o teste teste.c
$ ./teste
Hello World
```

Parece estar tudo bem!

Ciências ULisboa

```
$ valgrind --leak-check=yes ./teste
==2144== Memcheck, a memory error detector
==2144== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2144== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2144== Command: ./teste
==2144==
==2144== Invalid write of size 1
==2144==    at 0x40270C3: strcpy (mc_replace_strmem.c:311)
==2144==    by 0x8048498: main (teste.c:11)
==2144==  Address 0x419e033 is 0 bytes after a block of size 11 alloc'd
==2144==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2144==    by 0x8048480: main (teste.c:10)
==2144==
==2144== Invalid read of size 1
==2144==    at 0x4027040: __GI_strlen (mc_replace_strmem.c:284)
==2144==    by 0x4098C94: puts (ioputs.c:37)
==2144==    by 0x80484A4: main (teste.c:12)
==2144==  Address 0x419e033 is 0 bytes after a block of size 11 alloc'd
==2144==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2144==    by 0x8048480: main (teste.c:10)
```

```
==2144==
Hello World
==2144==
==2144== HEAP SUMMARY:
==2144==     in use at exit: 11 bytes in 1 blocks
==2144==   total heap usage: 1 allocs, 0 frees, 11 bytes allocated
==2144==
==2144== 11 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2144==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2144==    by 0x8048480: main (teste.c:10)
==2144==
==2144== LEAK SUMMARY:
==2144==    definitely lost: 11 bytes in 1 blocks
==2144==    indirectly lost: 0 bytes in 0 blocks
==2144==      possibly lost: 0 bytes in 0 blocks
==2144==    still reachable: 0 bytes in 0 blocks
==2144==         suppressed: 0 bytes in 0 blocks
==2144==
==2144== For counts of detected and suppressed errors, rerun with: -v
==2144== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 11 from 6)
```

```
==2144==
Hello World
==2144==
==2144== HEAP SUMMARY:
==2144==     in use at exit: 11 bytes in 1 blocks
==2144==   total heap usage: 1 allocs, 0 frees, 11 bytes allocated
==2144==
==2144== 11 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2144==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2144==    by 0x8048480: main (teste.c:10)
==2144==
==2144== LEAK SUMMARY:
==2144==    definitely lost: 11 bytes in 1 blocks
==2144==    indirectly lost: 0 bytes in 0 blocks
==2144==      possibly lost: 0 bytes in 0 blocks
==2144==    still reachable: 0 bytes in 0 blocks
==2144==         suppressed: 0 bytes in 0 blocks
==2144==
==2144== For counts of detected and suppressed errors, rerun with: -v
==2144== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 11 from 6)
```

Nem tudo está bem!

Valgrind: resolvendo o problema

==2144== Invalid write of size 1
==2144== at 0x40270C3: strcpy (mc_replace_strmem.c:311)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
        char *s1 = "Hello World";
        char *s2;

        int len = strlen(s1);

        s2 = malloc(len);

        strcpy(s2,s1);
        printf("%s\n",s2);
        return 0;
}
```

Ciências ULisboa

Valgrind: resolvendo o problema

==2144== Invalid write of size 1
==2144== at 0x40270C3: strcpy (mc_replace_strmem.c:311)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
        char *s1 = "Hello World";
        char *s2;

        int len = strlen(s1);

        s2 = malloc(len+1);

        strcpy(s2,s1);
        printf("%s\n",s2);
        return 0;
}
```

Ciências ULisboa

```
$ vim teste.c
$ gcc –o teste teste.c
$ valgrind --leak-check=yes ./teste
==2168== Memcheck, a memory error detector
==2168== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2168== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2168== Command: ./teste
==2168==
Hello World
==2168==
==2168== HEAP SUMMARY:
==2168==     in use at exit: 12 bytes in 1 blocks
==2168==   total heap usage: 1 allocs, 0 frees, 12 bytes allocated
==2168==
==2168== 12 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2168==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2168==    by 0x8048483: main (in /home/fvramos/workspace/teste/src/a.out)
==2168==
==2168== LEAK SUMMARY:
==2168==    definitely lost: 12 bytes in 1 blocks
==2168==    indirectly lost: 0 bytes in 0 blocks
==2168==      possibly lost: 0 bytes in 0 blocks
==2168==    still reachable: 0 bytes in 0 blocks
==2168==         suppressed: 0 bytes in 0 blocks
==2168==
==2168== For counts of detected and suppressed errors, rerun with: -v
==2168== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 6)
```

```
$ vim teste.c
$ gcc –o teste teste.c
$ valgrind --leak-check=yes ./teste
==2168== Memcheck, a memory error detector
==2168== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2168== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2168== Command: ./teste
==2168==
Hello World
==2168==
==2168== HEAP SUMMARY:
==2168==     in use at exit: 12 bytes in 1 blocks
==2168==   total heap usage: 1 allocs, 0 frees, 12 bytes allocated
==2168==
==2168== 12 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2168==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2168==    by 0x8048483: main (in /home/fvramos/workspace/teste/src/a.out)
==2168==
==2168== LEAK SUMMARY:
==2168==    definitely lost: 12 bytes in 1 blocks
==2168==    indirectly lost: 0 bytes in 0 blocks
==2168==      possibly lost: 0 bytes in 0 blocks
==2168==    still reachable: 0 bytes in 0 blocks
==2168==         suppressed: 0 bytes in 0 blocks
==2168==
==2168== For counts of detected and suppressed errors, rerun with: -v
==2168== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 11 from 6)
```

Melhor, mas ainda não resolvido!

Valgrind: resolvendo o problema

```
==2168== 12 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2168==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2168==    by 0x8048483: main (in /home/fvramos/workspace/teste/src/teste)
==2168==
==2168== LEAK SUMMARY:
==2168==    definitely lost: 12 bytes in 1 blocks
```

```
#include <stdlib.h>
#include <string.h>

int main() {
        char *s1 = "Hello World";
        char *s2;

        int len = strlen(s1);
        s2 = malloc(len+1);
        strcpy(s2,s1);
        printf("%s\n",s2);


        return 0;
}
```

Ciências ULisboa

Valgrind: resolvendo o problema

```
==2168== 12 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2168==    at 0x4026864: malloc (vg_replace_malloc.c:236)
==2168==    by 0x8048483: main (in /home/fvramos/workspace/teste/src/teste)
==2168==
==2168== LEAK SUMMARY:
==2168==    definitely lost: 12 bytes in 1 blocks
```

```
#include <stdlib.h>
#include <string.h>

int main() {
        char *s1 = "Hello World";
        char *s2;

        int len = strlen(s1);
        s2 = malloc(len+1);
        strcpy(s2,s1);
        printf("%s\n",s2);
        free(s2);

        return 0;
}
```

Ciências ULisboa

```
$ vim teste.c
$ gcc teste.c
$ valgrind --leak-check=yes ./teste
==2178== Memcheck, a memory error detector
==2178== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2178== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2178== Command: ./teste
==2178==
Hello World
==2178==
==2178== HEAP SUMMARY:
==2178==     in use at exit: 0 bytes in 0 blocks
==2178==   total heap usage: 1 allocs, 1 frees, 12 bytes allocated
==2178==
==2178== All heap blocks were freed -- no leaks are possible
==2178==
==2178== For counts of detected and suppressed errors, rerun with: -v
==2178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 11 from 6)
```

```
$ vim teste.c
$ gcc teste.c
$ valgrind --leak-check=yes ./teste
==2178== Memcheck, a memory error detector
==2178== Copyright (C) 2002-2010, and GNU GPL'd, by Julian Seward et al.
==2178== Using Valgrind-3.6.1 and LibVEX; rerun with -h for copyright info
==2178== Command: ./teste
==2178==
Hello World
==2178==
==2178== HEAP SUMMARY:
==2178==     in use at exit: 0 bytes in 0 blocks
==2178==   total heap usage: 1 allocs, 1 frees, 12 bytes allocated
==2178==
==2178== All heap blocks were freed -- no leaks are possible
==2178==
==2178== For counts of detected and suppressed errors, rerun with: -v
==2178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 11 from 6)
```

À partida, tudo resolvido!

Exercício

- Analisar o programa

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void test1() {
        const int NUM_HEIGHTS = 3;
        int *heights = malloc( NUM_HEIGHTS );
        for (int i=0; i < NUM_HEIGHTS; i++) {
                heights[i] = i * i;
                printf("%d: %d\n", i, heights[i]);
        }
}

void test2() {
        const int NUM_WEIGHTS = 5;
        long long *weights = malloc( NUM_WEIGHTS * sizeof(weights));
        for (int i=0; i < NUM_WEIGHTS; i++) {
                weights[i] = 100 + i;
                printf("%d: %lld\n", i, weights[i]);
        }
        free (weights);
        weights[0] = 0;
}

char *get_string(){
        char message[100] = "Hello world!";
        char *ret = message;
        return ret;
}

void test3(){
        printf( "String: %s\n", get_string() );
}

int main(int agra, char* args[]) {
        test1();
        test2();
        test3();

        return 0;
}
```

```
aluno-di@linux-di-fcul-1718:~/Documents/SD$ ./bad_memory
0: 0
1: 1
2: 4
0: 100
1: 101
2: 102
3: 103
4: 104
String: Hello world!
aluno-di@linux-di-fcul-1718:~/Documents/SD$
```

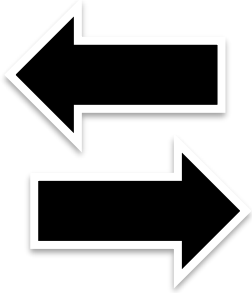
Output

Será que não existem mesmo problemas?

Valgrind

Marshalling e *Unmarshalling (Serialização)*

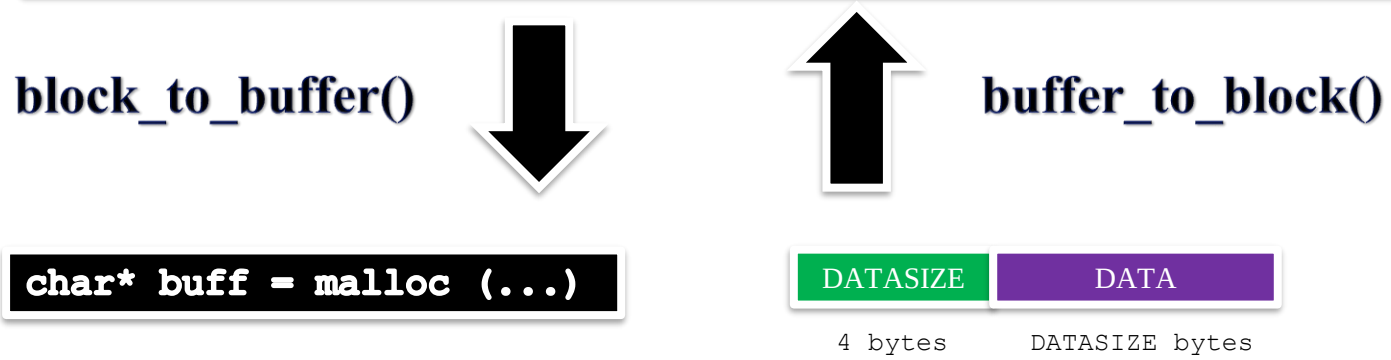
- Processo pelo qual se transforma uma estrutura de dados para o seu formato binário (byte[])
  - Muito usado para comunicação entre máquinas de um Sistemas Distribuído
  - Também útil para efeitos de persistência
    - » gravar cópia em disco



Marshalling e *Unmarshalling (Serialização)*

Ex: block_t

```
struct block_t {
        int datasize; /* Tamanho do bloco de dados */
        void *data;   /* Conteúdo arbitrário */
};
```

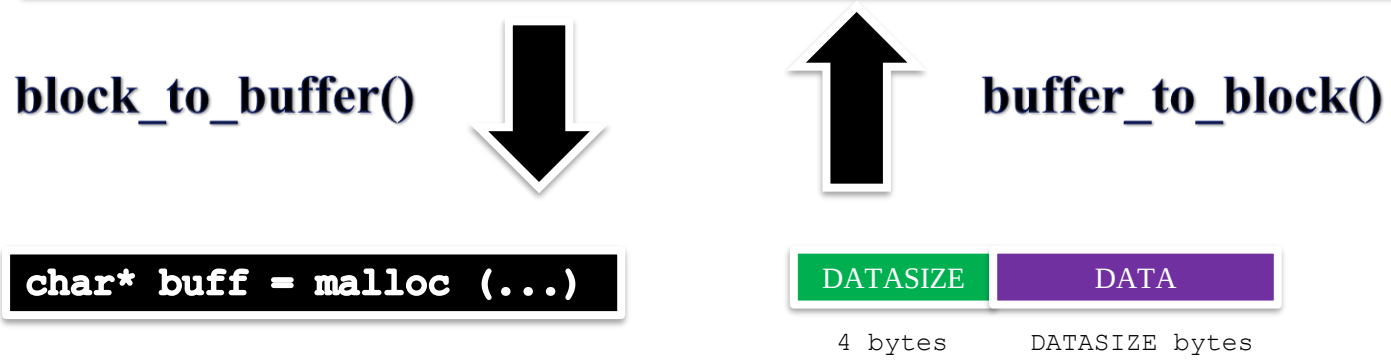


- Formato mais comum:
  - primeiro escreve-se 1 inteiro (4 bytes) com o tamanho da estrutura
  - Depois então escreve-se o conteúdo da estrutura

Marshalling e *Unmarshalling (Serialização)*

Ex: block_t

```
struct block_t {
        int datasize; /* Tamanho do bloco de dados */
        void *data;   /* Conteúdo arbitrário */
};
```

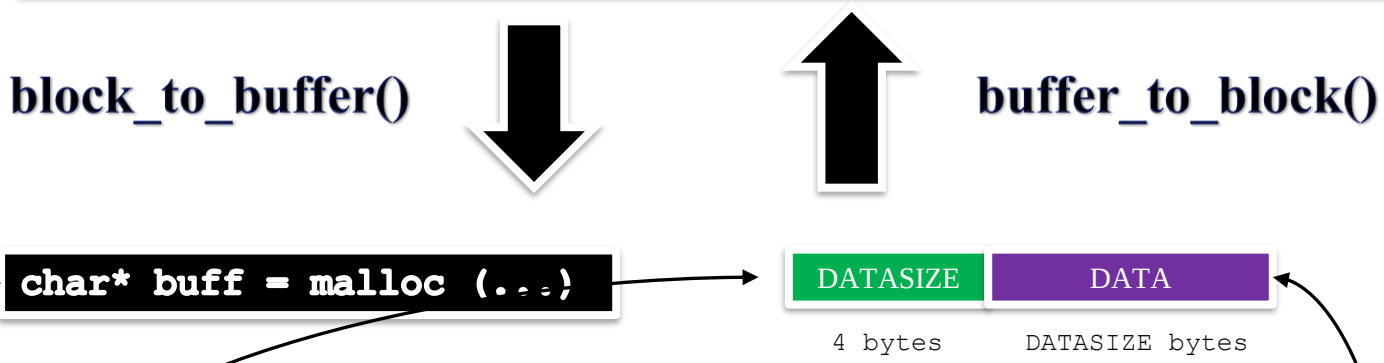


- Como copiar o conteúdo de uma variável para o array de bytes?
  - void *memcpy(void *dest, const void * src, size_t n)
    - » void* dest – o endereço de memória para onde queremos copiar
    - » void* src – o endereço de memória da variável que queremos copiar
    - » size_t n – o tamanho da variável (em bytes)

Marshalling e *Unmarshalling (Serialização)*

Ex: block_t

```
struct block_t {
        int datasize; /* Tamanho do bloco de dados */
        void *data;   /* Conteúdo arbitrário */
};
```



- Exemplo – queremos serializar a variável string

```
char* string = "ola!";
int size = strlen(string)+1;
char *buff = malloc(sizeof(int)+size);
memcpy(buff, &size, sizeof(int));
memcpy(buff+sizeof(int), &string, size);
```

Ciências ULisboa

Referências

- [Kernighan1988]
  - Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, 2nd Edition, Prentice Hall, 1988