

Sistemas Hardware-Software

Aula 11 – Programação em nível de máquina (I)

2019 – Engenharia

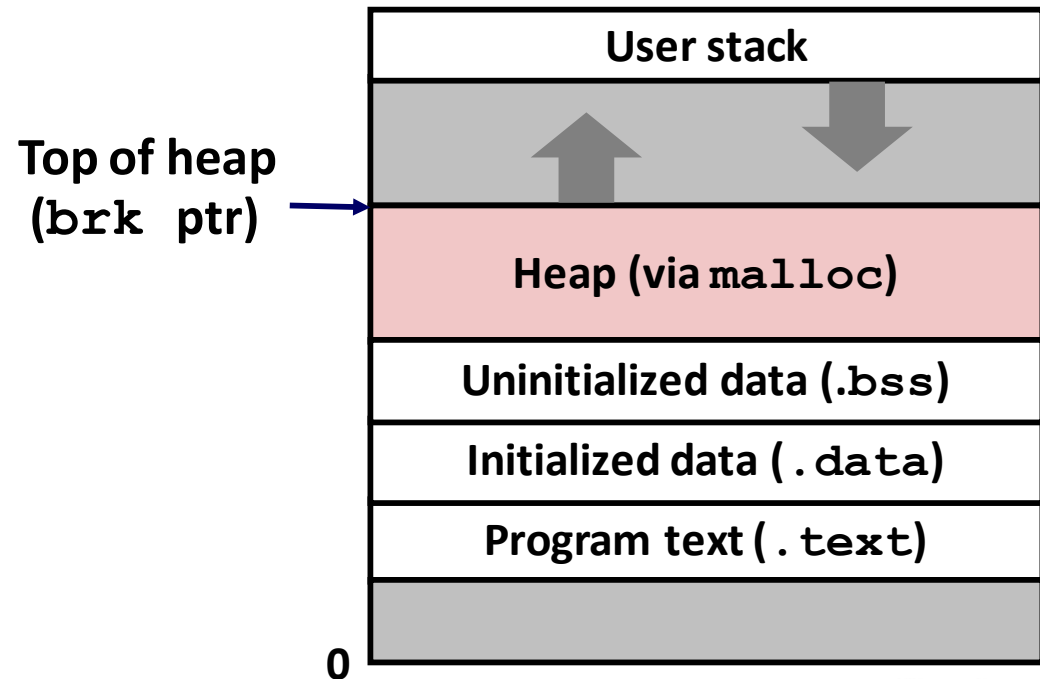
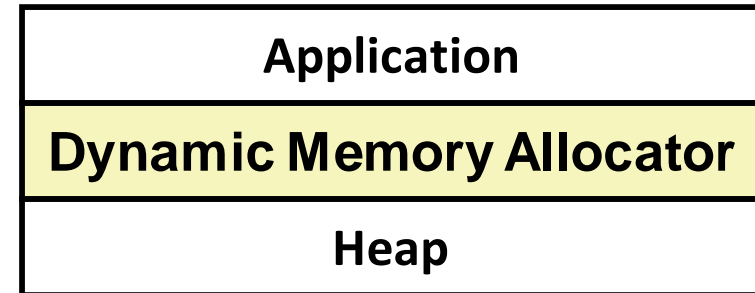
Maciel Calebe, Igor Montagner [<igorsm1@insper.edu.br>](mailto:igorsm1@insper.edu.br), Fábio Ayres

Alocação estática

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAXW 512
5  #define MAXH 512
6
7  int main(int argc, char *argv[]) {
8      int mat[MAXH][MAXW];
9
10     /*
11        trabalhar com arquivo PGM
12     */
13
14     return 0;
15 }
```

Alocação dinâmica de memória

- Programas usam **alocadores de memória dinâmica** para criar e gerenciar novos espaços de memória virtual
 - C: malloc, free
 - C++: new, delete
- A área do espaço de memória virtual gerenciada por estes alocadores é chamada de **heap**



Alocação dinâmica de memória

- Alocadores organizam o heap como uma coleção de blocos de memória que estão **alocados** ou **disponíveis**
- Tipos de alocadores
 - Explícitos: usuário é responsável por **alocar** e **deallocar** (ou liberar) a memória.
Exemplo: malloc, new
 - Implícitos: usuário não precisa se preocupar com a liberação da memória.
Exemplo: **garbage collector** em Java

malloc

```
#include <stdlib.h>  
void *malloc(size_t size)
```

Se bem sucedido: retorna ponteiro para bloco de memória com pelo menos **size** bytes reservados, e com alinhamento de 8 bytes em x86, ou 16 bytes em x86-64. Se **size** for zero, retorna **NULL**.

Se falhou: retorna **NULL** e preenche **errno**

Alocação estática

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MAXW 512
5  #define MAXH 512
6
7  int main(int argc, char *argv[]) {
8      int mat[MAXH][MAXW];
9
10     /*
11        trabalhar com arquivo PGM
12     */
13
14     return 0;
15 }
```

free

```
#include <stdlib.h>  
void free(void *p)
```

Devolve o bloco apontado por **p** para o *pool* de memória disponível

Alocação dinâmica

- Vantagens
 - Controle feito em tempo de execução
 - Economia de memória
 - Expandir / diminuir / liberar conforme necessário
- Desvantagens
 - Riscos da gerência
 - Liberar espaços não mais necessários
 - Não acessar espaços já liberados
 - Acessar apenas a quantidade requisitada
 - Etc.

Outras funções

calloc: Versão de malloc que inicializa bloco alocado com zeros.

realloc: “Re-aloca” um bloco – muda o tamanho do bloco garantindo a integridade dos dados. Note que o bloco realocado pode mudar de lugar na memória!

sbrk: usado internamente pelos alocadores para aumentar ou diminuir o heap

Exemplo

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

void foo(int n) {
    int i, *p;

    /* Allocate a block of n ints */
    p = (int *) malloc(n * sizeof(int));
    if (p == NULL) {
        perror("malloc");
        exit(0);
    }

    /* Initialize allocated block */
    for (i = 0; i < n; i++) {
        p[i] = i;
    }

    /* Return allocated block to the heap */
    free(p);
}
```

Atividade

Exercícios 1, 2 e 3 da atividade no papel

20 minutos

Bugs de memória comuns

- De-referenciando ponteiros ruins
- Ler memória não-inicializada
- Sobrescrever memória
- Referenciar variáveis inexistentes
- De-alocar bloco de memória múltiplas vezes
- Referenciar memória desalocada
- Não de-alocar memória

Dereferenciando ponteiros ruins

O bug clássico do `scanf`!

```
int val;  
  
...  
  
scanf("%d", val);
```

Ler memória não-inicializada

Bug clássico: assumir que dados no heap são pré-inicializados com zero

```
/* return y = Ax */
int *matvec(int **A, int *x) {
    int *y = malloc(N*sizeof(int));
    int i, j;

    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            y[i] += A[i][j]*x[j];
    return y;
}
```

Sobrescrever memória

Bug insidioso! Eis um exemplo de alocação de tamanho errado:

```
int **p;  
  
p = malloc(N*sizeof(int));  
  
for (i=0; i<N; i++) {  
    p[i] = malloc(M*sizeof(int));  
}
```

Sobrescrever memória

Bug clássico: off-by-one!

```
int **p;  
  
p = malloc(N*sizeof(int *));  
  
for (i=0; i<=N; i++) {  
    p[i] = malloc(M*sizeof(int));  
}
```


Sobrescrever memória

Bug clássico sinistro: não verificar tamanho de string! A base dos ataques clássicos de buffer overflow!

```
char s[8];  
int i;  
  
gets(s);  /* reads "123456789" from stdin */
```

char * fgets (char * str, int num, FILE * stream);

Sobrescrever memória

Erro no entendimento de aritmética de ponteiros

```
int *search(int *p, int val) {  
  
    while (*p && *p != val)  
        p += sizeof(int);  
  
    return p;  
}
```

Referenciar variáveis inexistentes

Variáveis locais desaparecem após o retorno da função

```
int *foo () {  
    int val;  
  
    return &val;  
}
```

Desalocar múltiplas vezes

```
x = malloc(N*sizeof(int));  
    <manipulate x>  
free(x);  
  
y = malloc(M*sizeof(int));  
    <manipulate y>  
free(x);
```

Referenciar blocos desalocados

```
x = malloc(N*sizeof(int));  
  <manipulate x>  
free(x);  
  ...  
y = malloc(M*sizeof(int));  
for (i=0; i<M; i++)  
    y[i] = x[i]++;
```

Memory leaks

```
foo() {  
    int *x = malloc(N*sizeof(int));  
    ...  
    return;  
}
```

C++ tem uma boa solução para esse problema: smart pointers!

Memory leaks

Bug: desalocar apenas parte da estrutura

```
struct list {
    int val;
    struct list *next;
};

foo() {
    struct list *head = malloc(sizeof(struct list));
    head->val = 0;
    head->next = NULL;
    <create and manipulate the rest of the list>
    ...
    free(head) ;
    return;
}
```

Atividade

Exercícios 4, 5 e 6 da atividade no papel

Se acabar antes pode ir embora, mas tem que validar as respostas.

Insper

www.insper.edu.br