Alocação Dinâmica com Reaproveitamento de Memória

Fábio Naconeczny da Silva
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
fabio.silva1@ufpr.br

***Resumo*—Este trabalho tem como objetivo o entendimento da alocação dinâmica em baixo nível através da manipulação do endereço de memória responsável pela limitação da seção *Heap*, o brk.**

***Index Terms*—alocação, dinâmica, memória, *Heap*, *first-fit*, brk.**

I. Introdução

A seção *Heap* é uma parte da memória dedicada à alocação dinâmica, isto é, a memória usada no programa é alocada em tempo de execução, podendo ser livremente alterada de acordo com a necessidade. Outra característica importante da *Heap* é que ela cresce em sentido oposto da seção Stack, sendo limitada pela mesma. O brk é o endereço de memória que indica o limite da memória alocada para cada processo e é através dele que a manipulação da memória é feita.

II. Trabalho

No trabalho proposto para a disciplina de Software Básico desse semestre, o objetivo era implementar uma biblioteca de gerenciamento de memória dinâmica em *Assembly* x86-64. A biblioteca devia conter quatro funções:

- *Setup Brk*: Obtém o endereço original de brk
- *Dismiss Brk*: Reseta o brk para seu endereço original
- *Memory Alloc*: Executa brk para abrir um bloco de *bytes* livres
- *Memory Free*: Desaloca um bloco de *bytes*

III. Memory Alloc

Para a alocação dinâmica de memória foi adotada a seguinte estratégia: Além da memória requisitada ser alocada, ela também contaria com um bloco de registro acoplado que nos traria duas informações adicionais bastante relevantes: se essa memória alocada está sendo utilizada ou não e o tamanho desse espaço reservado. Essas informações podem facilitar e otimizar o método de alocação.

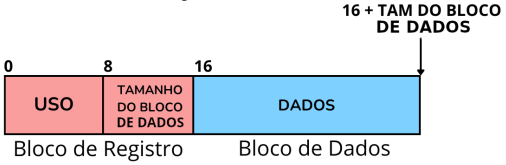

Figura 1: Bloco de Memória

Como o bloco de registro conta com 16 *bytes*, sempre que uma alocação for requisitada, a alocação real dentro da seção *Heap* será da memória requisitado por parâmetro da função *Memory Alloc* + 16 *bytes*.

Isso também nos traz outra informação importante, a menor memória requisitada para alocação, 1 byte, contará com 17 *bytes* alocados dentro da *Heap*.

IV. First - Fit

O método de alocação escolhido para a implementação desse trabalho foi o *First-Fit* que consiste em encaixar o bloco de memória no primeiro espaço desalocado que possa satisfazer os critérios de tamanho, evitando o desperdício de espaço.

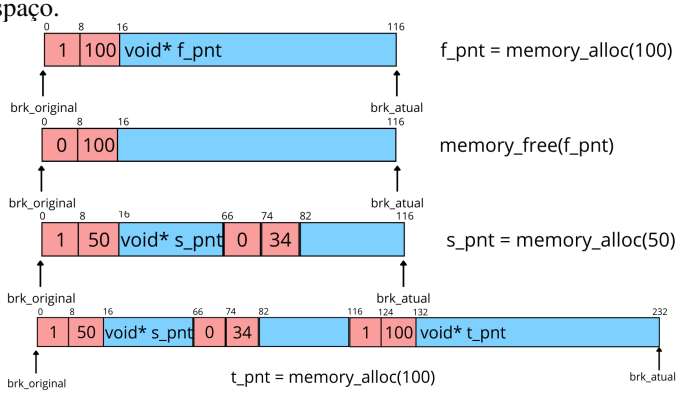

Figura 2: Método *First-Fit*

Esse diagrama representa a execução de uma parte do teste disponibilizado no moodle. A técnica de *First-Fit* ocorre após a desalocação de f_pnt, pois o algoritmo encaixa o bloco de memória de s_pnt no primeiro bloco desalocado que seja maior do que ele, no caso, o antigo f_pnt. Após a alocação de s_pnt, o algoritmo trunca o espaço restante e o transforma em um novo bloco de memória disponível para alocação se seu tamanho total for maior ou igual a 17 (16 para o bloco de registro e 1 byte mínimo para alocação).

V. Implementação

A. Setup Brk

Para a primeira função da biblioteca em assembly, a qual desejamos saber o endereço inicial de brk, 0 é colocado no registrador rdi e realiza-se a chamada do sys_brk colocando 12 no rax. O endereço de brk é retornado em rax e colocado

nas variáveis globais $brk\_original$ e $brk\_current$, afinal o brk atual é igual ao original neste momento.

B. Dismiss Brk

Nesta simples função, ocorre o *reset* do brk, então o valor de $brk\_original$ é passado para $brk\_current$.

C. Memory Alloc

A estratégia para a implementação da função Memory Alloc se baseia no que foi detalhado neste relatório até agora.

A função começa comparando o endereço de $brk\_current$ com o endereço de $brk\_original$. Se esses endereços são iguais, significa que é a primeira alocação, e a função segue para a *label* $\_alloc$ para alocar o bloco de registro.

Se não for a primeira alocação, a função entra em um laço denominado $\_search\_block$, que itera sobre os blocos de memória existentes para encontrar um bloco livre que possa atender à solicitação de alocação. O loop compara o tamanho do bloco atual com o tamanho necessário e verifica se o bloco está em uso. Se um bloco livre adequado é encontrado, ele é marcado como em uso e seu tamanho é ajustado. Em seguida, se o espaço restante tiver mais de 16 *bytes* livres, ele é quebrado para criar um novo bloco de memória, caso contrário, retorna-se o bloco por inteiro.

Na *label* $\_alloc$, a idéia básica é primeiramente criar um novo bloco de registro somando 16 *bytes* no endereço de $brk\_current$ se a requisição da alocação não cumprir os requisitos do *First-Fit* (caso a requisição seja maior que o tamanho de qualquer outro bloco livre disponível), então realiza-se a primeira chamada do sys_brk. Neste bloco de registro, os oito primeiros *bytes* indicam a *flag* de uso e a marca com 1 (uso) e os últimos oito *bytes* indicam o tamanho do bloco de dados. Logo após, a memória requisitada por parâmetro é acrescida no $brk\_current$, aumentando o limite da *Heap* por meio de uma segunda chamada de sys_brk. Este novo bloco de dados criado ao final da Heap, foi acoplado ao bloco de registro, criando um novo bloco de memória.

A estratégia de fazer duas chamadas de sys_brk, uma para o bloco de registro e outra para o bloco de dados é válida, pois ao final da primeira chamada, o endereço do último *byte* do bloco de registro é salvo em um registrador, então a partir dele podemos tanto manipular o bloco de registro, quanto retornar o começo do bloco de dados mais eficientemente, sem precisar de mais aritmética de ponteiros desnecessária.

D. Memory Free

Por fim, a função *Memory Free*, que serve para desalocar memória previamente alocada, funciona de forma simples: a *flag* do bloco de registro responsável por nos informar se o bloco está em uso ou não, recebe o valor 0, indicando que os *bytes* do bloco de dados estão ociosos, podendo serem substituídos por outros, economizando memória.

Entretanto, devemos tomar cautela, não podemos desalocar um bloco que não esteja na seção *Heap*, então tomamos dois limitantes, um inferior e outro superior, para garantir que o bloco desalocado esteja entre o $brk\_original$ e o $brk\_current$.

VI. Conclusão

Em conclusão, este trabalho teve como objetivo a implementação de uma biblioteca de gerenciamento de memória dinâmica em Assembly x86-64, com foco na manipulação do endereço de memória responsável pela limitação da seção Heap, o brk. Foram desenvolvidas quatro funções principais: Setup Brk, que obtém o endereço original do brk; Dismiss Brk, que reseta o brk para seu endereço original; Memory Alloc, responsável pela alocação dinâmica de memória; e Memory Free, destinada à desalocação de memória previamente alocada.

A estratégia adotada para a alocação dinâmica envolveu a criação de blocos de registro acoplados à memória alocada, fornecendo informações adicionais sobre o uso e o tamanho do espaço reservado. A implementação seguiu a estratégia de First-Fit para alocar blocos de memória, visando evitar desperdício de espaço.

A função *Memory Alloc* realiza duas chamadas de sys_brk, uma para o bloco de registro e outra para o bloco de dados, otimizando a manipulação dos blocos. Por sua vez, a função *Memory Free* desaloca blocos de memória, marcando a *flag* de uso como 0 e garantindo que a desalocação ocorra apenas dentro dos limites da seção Heap.

Em resumo, a implementação demonstra uma abordagem eficiente para a gestão dinâmica de memória em baixo nível, proporcionando alocação e desalocação de maneira controlada e otimizada. O uso da estratégia *First-Fit* e a integração de blocos de registro contribuem para uma alocação de memória mais eficaz e um melhor aproveitamento dos recursos disponíveis.