Implementação de um sistema de gerenciamento de alocação dinâmica de memória(HEAP)

Luis Felipe Risch

Departamento de Informática – Universidade Federal do Paraná (UFPR)

lfr20@inf.ufpr.br (GRR20203940)

1. Introdução

O presente trabalho consiste em uma implementação de um sistema gerenciador de alocação dinâmica de memória em assembly AMD64.

A abordagem adotada neste projeto destaca-se pela inclusão de um registro de informações gerenciais diretamente nos blocos de memórias alocados. Esse registro, composto por duas *quadwords*(totalizando 16 bytes), permite a administração dos blocos de memória. A primeira *quadword* sinaliza se o bloco de memória está em uso ou livre, utilizando 0 para livre e 1 para em uso. A segunda *quadword* indica o tamanho do bloco, representado pela quantidade de bytes de dados.

A estratégia adotada para localizar blocos de memória livres é conhecida como "first-fit". Essa abordagem envolve a exploração sequencial de todos os blocos na lista, independentemente de estarem ocupados ou não, com o objetivo de identificar o primeiro bloco livre que possua a capacidade necessária em bytes de dados. Se não for possível encontrar um bloco livre que atenda às exigências, a solução será criar um novo bloco de memória.

É relevante destacar que, caso o bloco livre identificado seja maior que o tamanho requisitado e contenha bytes excedentes suficientes para acomodar um novo registro, com pelo menos 1 byte no bloco de dados, essa alocação adicional será realizada.

2. Implementação

Esta seção tem o objetivo explicar de forma abstraída a implementação da API, que consiste de 4 funções principais:

1. void setup_brk();
2. void dismiss_brk();
3. void* memory_alloc(unsigned long int bytes);
4. int memory_free(void *pointer).

2.1. Inicializador do alocador

A função encarregada de inicializar o gerenciador de alocação dinâmica de memória é a setup_brk(). Este método invoca o serviço do sistema operacional, conhecido como *brk*, para pegar o endereço atual do início da *heap*. Os resultados desta chamada são armazenados em duas variáveis globais. A primeira dessas variáveis mantém o endereço retornado durante toda a execução do programa. Isso é essencial porque, ao finalizar o programa, a função dismiss_brk utilizará este endereço para restaurar a configuração original da *heap*.

Enquanto isso, a segunda variável tem a função de rastrear continuamente o endereço atual da *heap*. Durante as operações de alocação de memória, essa variável é atualizada com o estado atual da heap, sempre apontado para o endereço final;

2.2. Finalizador do alocador

A função encarregada de finalizar o gerenciador de alocação dinâmica de memória é a dismiss_brk. Como já foi dito, este método restaura a configuração original da heap utilizando o endereço armazenado na função setup_brk.

2.3. Alocação de memória

A função encarregada de fazer a alocação de memória é a memory_alloc. Este método possui a seguinte lógica:

1. Percorre-se todos os blocos na heap, independentemente de estarem livres ou ocupados, em busca do primeiro bloco livre que atenda à quantidade de bytes desejada. Vale observar que a transição de um bloco para o próximo é efetuada de maneira direta, adicionando (16 bytes + x) ao endereço inicial do bloco atual, onde x representa a quantidade de bytes de dados no bloco atual. Esse processo continua até que o endereço corrente alcance o topo da heap ou até que um bloco livre, capaz de atender às necessidades especificadas, seja identificado;
    a. Caso um bloco seja encontrado e a diferença entre a quantidade de bytes de dados deste bloco e a quantidade de bytes requisitada seja maior ou igual a 17, cria-se um novo bloco a partir da seção de dados do bloco identificado;
    b. Caso contrário, um novo bloco é criado e a variável global que armazena o estado atual da heap é incrementada em (16 + x) bytes, onde x representa a quantidade de bytes requisitados.
2. Tendo-se o bloco em mão, seja ele criado ou encontrado, retorna-se para o procedimento principal o seu endereço deslocando de 16 bytes, ou seja, apontando para o endereço de dados.

2.4. Liberação de memória

A função encarregada de liberar memória é denominada memory_free. Esse método recebe um ponteiro que aponta para o endereço de dados de um bloco específico. Para indicar que esse bloco está agora livre, é suficiente decrementar o ponteiro em 16 bytes e atribuir o valor 0.