Alocação de memória com brk em assembly x86-64

Felipe Quaresma e Marcelo Schreiber

Novembro, 2023

1 Estratégias de Implementação

A implementação da alocação de memória utilizando a syscall `brk` e o algoritmo *first fit* foi conduzida de forma abstraída, focando em aspectos essenciais para o entendimento do código em assembly x86-64.

1.1 Setup Brk

Na função `setup_brk`, chama-se a syscall `brk` com 0 no argumento para retornar o final da *heap*. Depois disso, colocamos o resultado da syscall presente no registrador rax nas variáveis globais que guardam o brk incial e atual.

1.2 Dismiss Brk

Para a função `dismiss_brk`, optou-se por utilizar a syscall `brk` novamente, desta vez definindo o `brk` de volta para o valor inicial (`brk_inicial`). Após isso, também armazenamos o valor do registrador rax na variável global responsável por armazenar o brk atual, já que o alteramos.

1.3 Memory Alloc

A estratégia de alocação de memória (`memory_alloc`) segue o algoritmo *first fit*. Decidiu-se utilizar um loop para percorrer os blocos livres disponíveis no heap. Quando um bloco adequado é encontrado, ele é marcado como ocupado e seu endereço é retornado. Se não é encontrado aloca-se um novo com a utilização do `brk`.

É importante ressaltar que dentro do loop sempre é feita uma comparação entre o brk inicial e atual, já que se os dois forem iguais, estamos em um local que, a partir dali, a heap está vazia (ou seja, não tem mais blocos para percorrer). Nesse caso, alocamos um novo bloco.

Também verificamos, quando encontramos um bloco disponível, se a subtração do tamanho encontrado pelo tamanho desejado é maior ou igual a 17, para que possamos adicionar ou não novos registradores e, consequentemente, um novo bloco.

1.4 Memory Free

Para a função `memory_free`, verificamos, primeiramente, algumas condições baseadas em segurança, já que, dependendo do endereço enviado como argumento, podemos ter algumas complicações.

Então, primeiramente, verificamos se o endereço passado por argumento está dentro dos limites da heap. Depois, verificamos se, subtraindo 16 dele, ou seja, o que seria necessário para acessar os registradores do bloco, saímos ou não da heap.

Com essas verificações corretas, adicionamos 16 novamente e trocamos o registrador de valor livre para 0 e retornamos 1. Para possíveis erros, a função retorna 0.