UNIVERSIDADE FEDERAL
DE OURO PRETO

TRABALHO PRÁTICO III - SISTEMAS OPERACIONAIS

ENYA LUÍSA GOMES DOS SANTOS
19.2.4201

Relatório apresentado por exigência da disciplina BCC264 - SISTEMAS OPERACIONAIS, da Universidade Federal de Ouro Preto.
Professor: CARLOS FREDERICO MARCELO DA CUNHA CAVALCANTI

OURO PRETO - MG
2021

Tabela de relação de memória do algoritmo:

	Heap	Stack	Global.bss	Global.data	Por quê?
hypercounter			x		Pois a variável foi declarada no escopo global e foi inicializada com o valor 0.
counter			x		Pois a variável foi declarada no escopo global, porém não foi inicializada.
*idk2	-	-	-	-	*Não encontrei essa variável no algoritmo.*
Cook1		x			A variável se encontra no escopo local da função e foi declarada automaticamente, logo a memória será liberada automaticamente.
clk1		x			A variável se encontra no escopo local da função e foi declarada automaticamente, logo a memória será liberada automaticamente.
cook2->c		x			A variável se encontra no escopo local da classe e foi declarada automaticamente, logo a memória será liberada automaticamente.
*arr em clk1	x				Se trata de uma alocação dinâmica, em tese, a memória permanece alocada

					até que haja uma liberação de memória “manual”, como por exemplo um delete.
`*arr em clkk2`	x				Se trata de uma alocação dinâmica, em tese, a memória permanece alocada até que haja uma liberação de memória “manual”, como por exemplo um delete.

Gerenciamento de memória

1. Fixed-size blocks:

A alocação de blocos de tamanho fixo, também chamada de alocação de pool de memória, são usadas para alocar blocos do mesmo tamanho através de uma lista livre. As funções de alocação e de liberação são mais eficientes do que os pools de tamanho variável, mas são naturalmente limitadas a apenas ser capazes de alocar blocos de um tamanho determinado.

2. Buddy blocks:

“[...] é um algoritmo de alocação de memória que divide a memória em partições para tentar satisfazer uma solicitação de memória da forma mais adequada possível. Este sistema divide a memória ao meio para tentar dar um melhor ajuste.”

Os esquemas de partição estática sofrem da limitação de ter um número fixo de processos ativos e o uso de espaço também pode não ser ideal. O sistema buddy é um algoritmo de alocação e gerenciamento de memória que gerencia a memória em dois incrementos.

Suponha que o tamanho da memória seja 2U, suponha que um tamanho de S seja necessário.

Se 2U-1 <S <= 2U: Alocar todo o bloco

Senão: Divida recursivamente o bloco igualmente e teste a condição a cada vez, quando ela for satisfeita, aloque o bloco e saia do loop.

O sistema também mantém o registro de todos os blocos não alocados cada e pode mesclar esses blocos de tamanhos diferentes para fazer um grande pedaço.

Em suma, todo o espaço de memória disponível para alocação é inicialmente tratado como um único bloco cujo tamanho é uma potência de 2. Quando a primeira solicitação é feita, se seu tamanho for maior que a metade do inicial , todo o bloco é alocado. Caso contrário, o bloco é dividido em dois companheiros iguais. Se o tamanho da solicitação for maior do que a metade de um dos amigos e, em seguida, atribua um a ele. De outra forma, um dos amigos é dividido ao meio novamente. Este método continua até que o menor bloco maior ou igual ao tamanho da solicitação é encontrado e alocado a ele.

> “Nesse sistema, a memória é alocada em vários pools de memória em vez de apenas um, onde cada pool representa blocos de memória com uma certa potência de dois em tamanho, ou blocos de alguma outra progressão de tamanho conveniente. Todos os blocos de um determinado tamanho são mantidos em uma lista ou árvore vinculada classificada, todos os novos blocos que são formados durante a alocação são adicionados aos seus respectivos pools de memória para uso posterior. Se for solicitado um tamanho menor do que o disponível, o menor tamanho disponível será selecionado e dividido. Uma das partes resultantes é selecionada e o processo se repete até que a solicitação seja concluída. Quando um bloco é alocado, o alocador começa com o menor bloco suficientemente grande para evitar quebrar blocos desnecessariamente. Quando um bloco é liberado, ele é comparado ao seu parceiro. Se ambos estiverem livres, eles serão combinados e colocados na lista de blocos de amigos de tamanho maior.”

3. Slab Allocation:

A ideia básica por trás do alocador slab é ter caches de objetos comumente usados mantidos em um estado inicializado disponível para uso pelo kernel. Sem um alocador baseado em objeto, o kernel gastará muito do seu tempo alocando,

inicializando e liberando o mesmo objeto. O alocador slab visa armazenar em cache o objeto liberado para que a estrutura básica seja preservada entre os usos.

> “Este mecanismo de alocação de memória pré-aloca pedaços de memória adequados para ajustar objetos de um certo tipo ou tamanho. [3] Esses blocos são chamados de caches e o alocador só precisa manter o controle de uma lista de slots de cache livres. A construção de um objeto usará qualquer um dos slots de cache livres e a destruição de um objeto adicionará um slot de volta à lista de slots de cache livres. Essa técnica alivia a fragmentação da memória e é eficiente, pois não há necessidade de procurar uma parte adequada da memória, pois qualquer slot aberto será suficiente.”

4. Stack Allocation:

Uma pilha é uma área especial da memória do computador que armazena variáveis temporárias criadas por uma função. Na pilha, as variáveis são declaradas, armazenadas e inicializadas durante o tempo de execução.

É uma memória de armazenamento temporário. Quando a tarefa de computação for concluída, a memória da variável será automaticamente apagada. A seção de pilha contém principalmente métodos, variáveis locais e variáveis de referência. A pilha é frequentemente usada para armazenar variáveis de comprimento fixo local para as funções atualmente ativas.

Alinhamento de memória

Alinhamento de memória é um método de organização de dados em memória para que ele possa ser recuperado o mais rápido possível.

1. Globalmente:

Existe enquanto o programa rodar e está sempre acessível a todo o escopo desse.

Para alocar memória alinhada globalmente usando C ++, use-se o `alignas()` é usado para o alinhamento da memória.

2. Na Pilha:

Está associada ao uso local, pois é somente no escopo local que ela é conhecida, acessada e existe, e é liberada automaticamente quando a função principal é concluída.

Assim como no globalmente, aqui também se usa o **`alignas()`** porém o que muda eh o local de declaração da memória.

3. Dinamicamente:

Está associada a heap de memória, ou seja, toda a memória livre disponível no dispositivo. Usa-se a alocação de memória para reservar a memória necessária e sempre é necessário a liberação dessa memória para evitar leak e essa não é liberada de forma automática.

Utiliza-se o **`align_alloc()`** para alocação dinâmica, que fornece uma função menos desajeitada e mais portátil de implementação. Eh semelhante ao malloc() mas com um adicional parâmetro de alinhamento..