Trabalho de Sistemas Operacionais
Alocador Eficiente de Memória

Andreia Oliveira,Andrei Figueiredo

I. INTRODUÇÃO

Funções de alocação dinâmica de memória já foram implementadas considerando as arquiteturas multicores. A alocação de memória nestas arquiteturas é uma desafio já que os processos usam de forma compartilhada os barramentos o DMA(Direct memory access).C Entre elas o PTMalloc e o TCMalloc.O objetivo desse trabalho foi implementar um alocador de memória com um desempenho melhor que o Malloc da linguagem C.

II. DESENVOLVIMENTO

O myMalloc utiliza o conceito de blocos e lista de memória, foi levado em consideração o uso de mapa de memória, mas levando em conta que o mapa de memória não tem a possibilidade de aumentar seu tamanho durante a execução, ele foi descartado.

O algoritmo para decidir que bloco usar é o FIRST FIT por ser o mais facil de se implementar e o mais rápido, aconteceu a tentativa de usar o BEST FIT, mas resultou em várias falhas e devido ao tempo restante foi necessário usar o FIRST FIT.

Uma das pricipais vantagens do mymalloc é que ele é threadsafe. Outra idéia que o mymalloc usa é criar duas listas, uma lista será responsável pela memória solicitada pelo sbrk() e outra lista pela memória livre.

Para agilizar o percorrimento pela freelist, sempre quando um bloco é adicionado a freeLista ele é ordenado, visto que sempre que é preciso alocar memória é necessário percorrer a freelist.

Para aproveitar o máximo de memória que é requisitada pela função sbrk(), sempre quando possivel a memória que sobra de uma alocação é acrescentada a outro bloco livre pela função verifyFusion(), com isso é minimizada a memória ociosa.

Para mostrar a fragmentação de memória foi criada a função show_alloc_mem(), que não está funcionando, ela simplesmente percorre a memoryList e imprime o ponteiro, o offset, e o tamanho do bloco, dos blocos alocados.

verifyFusion: Ele ocorre em duas situações, caso seja alocada uma memória e sobre espaço no bloco que foi alocada, ou quando adicionamos um bloco ao freeList.É usada duas formulas para realizar essa fusão ambas fudem um bloco ao outro, mas cada uma tem uma formula diferente:

sizeDoBloco = size requisitado pelo usário que o bloco deve ter

tamanhoDoBloco = tamanho que a struct ocupa na memoria

1.(begin && begin->prox) = endereço + sizeDoBlocoBegin + tamanhoDoBlocoBegin;

2.(block->magic == FREE_MAGIC) = sizeDoBloco + sizeDoBlocoBlock + tamanhoDoBlocoBlock;

Na situação onde (block->magic == FREE_MAGIC) o bloco que vem em seguida não esta livre então é necessário tomar cuidado com os ponteiros.

freeList: Lista com todos os blocos livres.

memoryList: Lista com todos os blocos.

Thread Safety: é um conceito de programação de computadores aplicável no contexto de programas multi-thread. Um pedaço de código é dito thread-safe se ele apenas manipula estruturas de dados compartilhadas de uma forma que garanta uma execução segura através de várias threads ao mesmo tempo

O mymallocteste foi pensado em explorar ao maximo as vatagens do mymalloc, ele é um programa multithread para aproveitar o threadsafety do mymalloc. Ele aloca muita memória para mostrar a vantagem de se usar um lista de memória e ultiliza a biblioteca App.h para calcular o tempo de parede.

III. RESULTADOS

Para executarmos os testes,seguimos os seguintes parâmetros:Número de Threads,Tamanho,Contador e Loop.

Onde os testes tem os seguintes parâmentros:

Entrada 1:. Thread=2,Tamanho=5,Contador=5,Loop=5
Entrada 2.Thread=4,Tamanho=10,Contador=10,Loop=10
Entrada 3.Thread=8,Tamanho=20,Contador=20,Loop=10

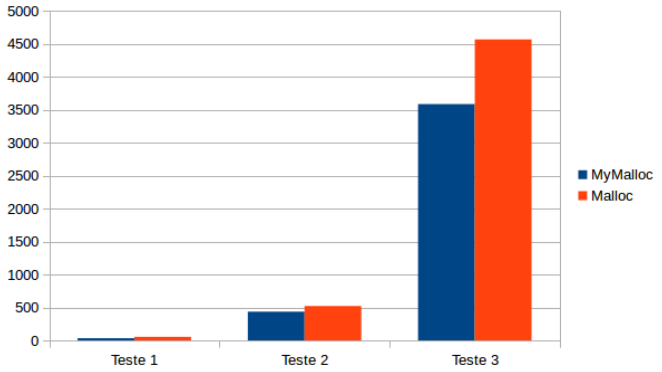

Figura 1: Média de cada entrada executada

Testes	Mymalloc	Malloc	Speedup
1	32.608002	54.006004	0.60
2	36.299004	55.661003	0.65
3	32.235001	55.722004	0.57
4	33.010002	54.341003	0.60
5	33.563000	51.915001	0.64

Tabela 1: Entrada 1

Teste	Mymalloc	Malloc	Speedup
1	434.495972	528.849976	0.82
2	435.442993	515.763000	0.84
3	434.786011	518.996948	0.83
4	435.303009	523.715942	0.83
5	436.885010	520.039978	0.84

Tabela 2:Entrada 2

Teste	Mymalloc	Malloc	Speedup
1	3586.485107	4563.424805	0.78
2	3587.895020	4582.951172	0.78
3	3580.927979	4568.217773	0.78
4	3595.906006	4562.604004	0.78
5	3578.345947	4561.900879	0.78

Tabela 3: Entrada 3

IV. Trabalhos Relacionados

Ao pesquisar sobre trabalhos relacionados foram identificados artigos muito interessante sobre a área estudada.Dentre eles,o artigo "Understanding the heap by braking it: A case study of the heap as a persistent data structure through non-traditional exploitation techniques"foca na implementação de gerenciamento de memória dinâmica da biblioteca GNU C , particularmente ptmalloc2 e mostra métodos para fugir certas checagens na biblioteca juntamente com métodos novos para obtenção do controle.

Resultado:A heap deve persistir e ser dividida. Threading fornece uma forma interessante para organizar código em uma sequencia vantajosa.

Outro artigo,"Moltly Lock-Free Malloc", tem como objetivo implementar um Malloc com baixa latencia, otima escabilidade, baixa fragmentação, aumentar localidade e generalidade.

Resultado:Conclusão foi introduzido um Multi-Processor reinicializável em regiões críticas e demonstrou a sua utilidade através do desenvolvimento de um alocador malloc escalável. Apesar de usar pilhas comuns o alocador pode geralmente operar sem bloqueio. O alocador tem excelentes características de execução , especialmente em sistemas multi-threading.

V. Conclusão

É possivel diminuir o tempo de alocação com estratégias basicas de gerencia de memória, entretanto a aplicação dessa estratégias pode ser bem dificil, pois se deve ter muito cuidado com a memória e os ponteiros. Utilizando uma estrutura simples de lista de memória e algumas estratégias de gerenciamento de memória o mymalloc conseguiu ganhar do malloc em todos os testes, existe uma grande possibilidade de caso seja testado acesso a memória o mymalloc também seria superior ao malloc. Uma das melhoras que poderiam ser implementadas seriam o uso de BET FIT ou WORST FIT, para diminuir a fragmentação e o uso da estrutura da hashtable para melhorar o acesso a memória.

VI. Referências

[1] Wikipedia,Thread Safety.
[2] Stack Overflow, "How does malloc work in a multithreaded environment?",Wladimir Palant.
[3] Stack Overflow, "Malloc and freeing memory between threads in C",Craig Sparks
[4] TLDP, "Memory Management",David A Rusling.
[5] The Linux Programming Interface, Michael Kerrisk.