Garbage Collector

Guilherme Aguiar Silva Milanez.

October 5, 2020

1 Introdução

Atualmente, os programas em sistemas modernos, utilizam a memória, disponível, de forma segmentada. Os dois segmentos principais são denominados Heap e Stack. Em ambos, a memória é dividida em frames. Na região denominada Stack cada frame é arquitetado de forma estática em tempo de compilação sendo que é necessário saber o tamanho do objeto que será inserido em tal área, enquanto que no Heap, o manejo das divisões é feito dinamicamente e não havendo a necessidade de se saber o tamanho dos objetos que serão inseridos nessa região da memória. A gerência dos arquivos na Stack é realizada pelo próprio sistema, enquanto que no Heap tal manejo é de encargo do desenvolvedor, essa responsabilidade leva a dois erros comuns denominados Memory Leak e Dangling References, que são definidos respectivamente quando, um espaço é requerido por um procedimento contudo tal procedimento passa a não necessitar desse espaço, e tal região fica inutilizada, já o segundo ocorre quando referências de ponteiros,existentes, apontam para espaços que não são objetos válidos, isso pode ocasionar situações em que um ponteiro pode ter acesso a um espaço, antes restrito ao procedimento, e obter assim informações incoerentes e até prejudiciais para o constructo do algoritmo.

2 Definição

Tendo esse contexto em mente, um procedimento que visa reduzir tais erros de manejo de memória no Heap é o Garbage Collector (GC). O principal objetivo deste recurso encontra-se na facilidade de gerenciar memória de programas, de forma automática. Após um conjunto de segmentos da memória serem alocados, e por algum motivo, não forem utilizados durante a execução do programa, essa rotina irá recuperar esses segmentos para que os mesmos possam ser aproveitados por outros procedimentos.

3 Paralelismo

endo esse contexto em mente, um procedimento que visa reduzir tais erros de manejo de memória no Heap é o Garbage Collector (GC). O principal objetivo deste recurso encontra-se na facilidade de gerenciar memória de programas, de forma automática. Após um conjunto de segmentos da memória serem alocados, e por algum motivo, não forem utilizados durante a execução do programa, essa rotina irá recuperar esses segmentos para que os mesmos possam ser aproveitados por outros procedimentos. Em seu princípio, o procedimento de coleta de lixo (GC) era desenvolvido para atuar em sistemas de processamento único (Uniprocessors), contudo à medida que a necessidade, por mais poder de processamento, se intensificou, houve o surgimento de sistemas contendo multiprocessadores capazes de realizar tarefas e procedimento em paralelo. Atualmente, tendo em vista que, os sistemas são, majoritariamente, equipados com multiprocessadores, surge a necessidade da adequação de um GC apto para trabalhos em paralelo. Em um dos artigos pesquisados, os autores propuseram uma arquitetura paralela de garbage Collector para compartilhamento de memória em multiprocessadores. A arquitetura proposta, baseia-se, na técnica "Stop-The-World", técnica na qual o coletor pausa as threads das aplicações durante a coleta dos dados prevenindo que outras aplicações acessem essa memória concorrente. A técnica é implementada em um multiprocessador que oferta várias formas de balanceamento de carga de dados. Os coletores paralelos, propostos no trabalho, utilizam essa estrutura para equilibrar o trabalho de varredura de dos ponteiros Root, usando um particionamento estático, e também para equilibrar o trabalho de rastrear

o gráfico do objeto tratado, usando um método de balanceamento de carga dinâmico chamado roubo de trabalho. Podendo, por conseguinte, obter uma melhora significativa da performance do Garbage Collector em nível de paralelismo.

4 Referência

[1] Picone, Nicholas. Garbage Collection: An Overview of Dynamic Data Management. Diss. 2010.

[2] https://www.ime.usp.br/ song/mac5710/slides/04gc.pdf

[3] Wilson, Paul R. "Uniprocessor garbage collection techniques." International Workshop on Memory Management. Springer, Berlin, Heidelberg, 1992.

[4] Flood, Christine H., et al. "Parallel Garbage Collection for Shared Memory Multiprocessors." Java Virtual Machine Research and Technology Symposium. 2001.

[5] Gidra, Lokesh, et al. "A study of the scalability of stop-the-world garbage collectors on multicores." ACM SIGPLAN Notices 48.4 (2013): 229-240.