Universidade Federal do Rio de Janeiro

Instituto de Matemática - Departamento de Ciência da Computação

Simulador de Gerenciamento de Memoria

Incluinto um gerenciamento de processos Round-Robin com feedback

Professor(a): Valéria Bastos

Lucas Rampazzo – 116015608

Joyce Brum – 116051654

Matheus Gouvêa – 113170726

Introdução

Este relatório explica a elaboração e funcionamento de um gerenciador de memória funcionando em conjunto com um gerenciador de processos implementado em C.

O gerenciador de processos implementa a estratégia de um escalonador Round-Robin com feedback de duas filas (alta e baixa prioridade). Dessa forma, processos que consomem mais CPU, perdem prioridade de acesso da mesma, enquanto que processos que fazem muitas requisições de I/O acabam por ganhar prioridade, evitando monopólio do processador.

Uma vez que um processo é criado, ele começará a requisitar espaço na memória para sua execução, e nessa hora, o gerenciador de memória entra em ação, adicionando ou tirando páginas de processos da mémoria, conforme o espaço disponível e a demanda dos processos.

Toda vez que for feita uma requisição de alocação de páginas de um processo na memória, o gerenciador de memória deve analisar a tabela de páginas, e caso ela esteja totalmente cheia, deve fazer uma decisão sobre qual páginas retirar da tabela, para atender a esse novo pedido. Essa decisão é feita pela estratégia LRU, que retira da tabela, as páginas do processo que foram referenciadas menos recentemente.

Estado da Arte

A memória é um dos principais recursos utilizados na execução de um programa, e como vimos em aula, é fundamental que o gerenciador de memória, atenda a todos os seus requisitos fundamentais que são: realocação, proteção, compartilhamento, organização lógica e organização física. (Stallings, 2012)

Nesse trabalho, exploramos a tarefa de realocação, que trata, justamente, de administrar de maneira eficiente o espaço de memória, e no nosso caso, essa tarefa envolve um conceito conhecido como Memória Virtual. A Memória Virtual, é um método de alocação de memória, onde a memória secundaria pode ser usada como se fosse parte da memória principal. Esse é um método utilizado para resolver problemas como o caso de um programa ser maior do que o tamanho da memória principal, nesse caso o programa não caberia na memória e não poderia ser carregado. Para organizar esse espaço de memória virtual, é usado um esquema conhecido por Paginação, onde os processos, ao invés de serem inteiramente carregados, são divididos em páginas - blocos de tamanho fixo de memória -, dessa forma, o sistema operacional pode carregar partes do processo na memória. (Ibidem, 2012)

Este trabalho então, baseado nos conceitos abordados, simula um gerenciador de memória em funcionamento, que recebe as requisiçõe do processo e administra a tabela de páginas virtuais, fazendo com que os processos possam executar sem nenhum problema.

Para implementação do simulador algumas premissas foram assumidas, elas estão especificadas abaixo:

- **Memória:**

  - Memória principal de 64 frames

  - Tamanho máximo da tabela de página igual a 64

- **Gerenciador de Memória:**

  - Working Set Limit de todos os processos igual a 4

  - Gerenciador com estratégia de realocação LRU

- **Processos:**

  - Requisições de novas páginas por processo a cada 3 segundos

  - Tempo máximo de chegada igual a 60

- **Gerenciador de processos:**

  - Geração de novos processos a cada 3 unidades de tempo

  - Gerenciador de processos Round Robin com feedback de duas filas

  - Quantum do Round Robin igual a 4

  - Número máximo de processos é 50

Metodologia

Neste tópico vamos falar sobre como foi feito o processo de implementação do simulador de memória e integração com o gerenciador de processos.

Bibliotecas

Primeiro definimos as bibliotecas descritas a seguir:

- **Variables.h:**
  Biblioteca que define as variáveis globais para os tipos de IO, gerenciador de memória, as váriaveis usadas na implementação do LRU (as sctructs No e GerenciadorPaginas), da FIFO do gerenciador de processos e das filas de prioridades.

- **Processos.h:**
  Define a função que gera o número de páginas de um processo, uma função que gera todas as tuplas de (IO, tempo) de um processo, uma função pra criar o PCB de um novo processo, uma função que incrementa o tempo de espera de um processo caso esteja na fila de prontos, utilizada para controlar a fila de prioridade e uma função pra exibir os resultados.

- **Fifo.h:**
  Define os métodos que usam a struct FIFO (defina em variables).

- **Lru.h:**
  Define os processos que usam as structs relacionadas a LRU

- **Gerenciadorfilas.h:**
  Define os métodos que iniciam as filas de alta e de baixa prioriadade e as filas de IO, e que as gerenciam.

- **Gerenciadormemoria.h:**
  Define os métodos básicos de ações que são executadas na memória virtual, como swap in/out, alocação de frames de memória e verificação de tempo de refencia na memória para execução da estratégia LRU.

Gerenciador de Processos:

Antes de falar sobre o Gerenciador de Memória, precisamos primeiramente dar uma breve explicação sobre o Gerenciador de Processos e seu funcionamento.

Para o funcionamento do Simulador, as premissas assumidas foram definidas de maneira a poder ser facilmente alteráveis em variables.h, como o limite máximo de processos criados e o quantum do Round Robin. Os tempos de serviço dos I/O's de cada processo foi criado randomicamente, mas de maneira a garantir que cada tempo de execução é único (não existirão dois ou mais I/O's executando ao mesmo tempo).

Temos 3 tipos de IO: O de disco, de impressora e o de fita.
O de disco retorna para a fila de baixa prioridade e o tempo de duração é de 6 unidades tempo. Como essa operação envia o processo para fila de baixa prioridade, decidiu-se que seu tempo de acesso deveria ser baixo.
O de impressora, retorna para a fila de alta prioridade e tem tempo de duração igual a 15 unidades de tempo.
O de fita magnética, retorna para a fila de alta prioridade e tem tempo de duração igual a 10 unidades de tempo.

Implementação

A implementação do Gerenciador de Memória, foi feita, baseada nas características do Gerenciador de Processos.

A função main do Gerenciador de Memória, consiste em um loop, em que, a cada iteração, o gerenciador incrementa em um instante de tempo, e em cada instante de tempo, o estado atual do gerenciador é analisado. Nesse instante também, ela checa a fila de processos bloqueados, pra verificar se algum processo vai sair do estado de bloqueado para o estado de pronto, e atualiza a fila.

Depois de gerenciar a fila de prontos, a função irá verificar se há processos a serem criados, ou seja, se há processos para adicionar a fila de prontos, criando assim, novos processos.

Depois dos passos anteriores, o método "processador()" é executado. Esse método tenta escalonar um processo, ou seja, tenta escolher um processo pra ser executado. Primeiro, ele tem que verificar se esse processo já terminou. Essa verificação é necessária porque pode acontecer o caso de que o processo já esteja exacutando e terminar. Se o processo terminou, ele imprime as informações do processo, o gerenciador retira todas a páginas do processo da memória, pelo metodo swap-out, atualiza as informações de tempo de término, adiciona em uma fila de processos finalizados, e então tenta escalonar outro

processo. Pode também ocorrer o caso em que o processo tenha completado sua execução, assim sendo, o método faz a preempção do processo escalona outro. Se o processo não for nem interrompido e nem finalizado, o método ira executar o Gerenciador de Memória. Nessa parte, o método vai verificar se o processo em questão, faz referencia a alguma página, se fizer, ele vai olhar se a página referenciada está na memória princiapal. Se estiver, o processo continua normalmente, e se a página não estiver na memória principal, ela precisará ser carregada na memória e o processo é bloqueado.

Depois de fazer a parte de gerência de memória, o método processador() fará uma chamada ao método executarprocessoProcesso(). Esse método primeiramente, incrementa o tempo executado do processo, porque independente de ter sido bloqueado ou não, ele ainda consumiu tempo no processador. Depois ele verifica se o processo vai fazer uma chamada de IO, caso contrário ele executa, retorna, e continua a execuação do loop.

Depois de executar o método processador(), o próximo passo do loop é atualizar o tempo de espera dos processos que estão prontos na fila de baixa prioridade, para isso, faz uma chamada ao método atualizarTempoEsperaProcessosReady(), porque quando os processos que estão na fila de baixa prioridade por muito tempo tem chances de subir a fila de alta prioridade.

Depois de executar todas os métodos anteriores, o próximo passo do loop é imprimir todas as filas e terminar a iteração.

Como dito anteriormente, o Gerenciador de Memória é chamado dentro do método processador(), ele primeiro verifica se o tempo executado do processo é múltiplo de 3, testando se é pra fazer uma requisição de memória, já que as requisições de memórias são feitas a cada 3 unidades de tempo de execução, se for necessário fazer a requisição, ele verifica se a página está na memória principal, caso a página esteja, ele atualizada a fila LRU do processo, ou seja, posiciona o índice da página para o fim da fila do LRU, e se ela não estiver na memória principal, ele chama o método loadPage(), esse método verifica se a LRU está cheia, ou seja, se o wsl já foi atingido na memória. Se o wsl foi atingido, o método retira uma das próprias páginas do processo da memória e põe outra no lugar, se o wsl ainda não foi atingido, ele procura um frame disponível pra ocupar a página e aí, pode acontecer um caso de swap-out. O método percorre toda a memória, que foi implementada em um vetor de inteiros em que cada posição pode ser 1 ou -1, sendo 1 o caso do frame estar livre, e -1 o caso dele estar ocupado. Se for encontrado um espaço de memória livre, esse espaço é ocupado, e é retornado o seu índice, se não for encontrado um espaço livre, o método retorna -1, indicando que a memória está cheia. Então o método alocarFrame(), faz uma chamada ao swap-out, que seleciona um processo pra remover da memória. O critério de seleção é, analisar os processos bloqueados, identificar o que vai ficar mais tempo bloqueado e seleciona-lo. Se não houver processos bloqueados ele vai selecionar o último processa da fila de baixa prioridade, que é o que vai ser executado por último. Se não houver nenhum processo na fila de baixa prioridade, então o método vai olhar a fila de alta prioridade e selecionar o último. A ideia é priorizar o processo que vai ficar mais tempo no processador.

O método swap swapOutProcess(), atribui -1 a todos os frames pertencentes ao processo que foi retirado, sinalizando que eles agora estão livres, e atualiza a tabela de paginas sinalizando que aquelas páginas não estão mais na memória, porém a fila LRU continua com a informação das páginas que ela tinha, porque quando o processo faz swap-in, ele vai ter que restaurar todas essas páginas.

O swap-in é feito no gerenciamento de memória, porque mesmo antes dele verificar se o processo faz alguma referência a mémoria, ele precisa verificar se o processo precisa fazer swap-in, se o processo precisar, ele vai procurar frames disponíveis, para poder restaurar todas as páginas do processo.

Sobre os estados ready suspended e blocked suspended, quando um processo que estava ready suspended é escalonado, suas páginas voltam para a memória, essa verificação é feita pelo método verificaSeFazSwapIn(), ele ve se o processo está com o status de ready suspended. Quando um processo que estava em blocked suspended é desbloqueado, ele vai para o status ready suspended.

Referências

STALLINGS, William. Operating systems: internals and design principles. Boston: Prentice Hall, 2012.

Process State – Wikipedia https://en.wikipedia.org/wiki/Process_state