MAC0422 – Sistemas Operacionais – 2s2016

EP3

Simulador para gerência de espaço livre e para substituição de páginas

Augusto Cesar Monteiro Silva - 8941234
Lucas Helfstein Rocha - 8802426

Implementação

Simulador

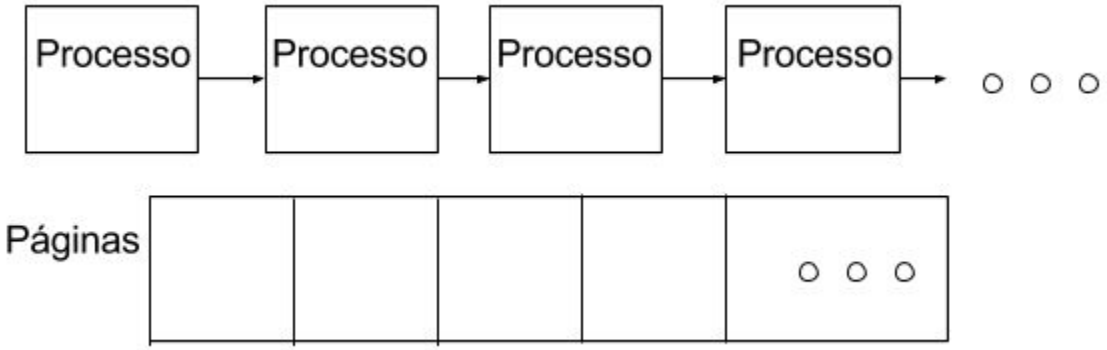
Para o simulador, usamos as seguintes classes:

- **Processos** (que possui informações sobre os processos do arquivo de trace, i. e., o PID do processo, uma lista com os endereços de memória que o processo irá acessar, uma lista com o tempo que o processo irá acessar uma memória, e o nome dele)

- **Páginas** (que possui todas as informações de uma página, i.e., o bit R, o seu lugar na tabela (se é a primeira, a segunda, etc), o bit de presente/ausente)

Simulador

Com essas classes, o simulador cria uma lista de processos (inicialmente vazia, representando os processos que estão sendo executados) e um vetor de páginas(representando todas as páginas da memória virtual). Temos então:



Simulador

O simulador então, entra num loop e funciona da seguinte maneira:

- Pega a próxima linha do arquivo de trace e verifica o t0
- Se t0 for menor que todos os ti's dos processos que estão na lista ou a lista estiver vazia (i.e., o t0 seria o próximo tempo em que ocorreria um evento), então o simulador coloca o processo t0 na lista de processos que estão executando, associa a esta linha um PID, e cria o objeto Processo associado a ele.
- Se existir um ti dos processos que seja menor que o t0 do próximo, o simulador fará os acessos a memória da seguinte forma:
    - Se este for o último tempo do processo, o simulador elimina o processo do bitmap, da lista de processos, do arquivo .vir e das estruturas que são usadas para a paginação.
    - Se for um acesso à memória, o simulador verificará se a página em que a memória acessada se encontra está presente na memória física (através do vetor de páginas). Se não, o simulador recorre ao algoritmo de paginação que foi escolhido
- Antes de realizar o loop novamente, o simulador ordena a lista dos processos que estão executando de acordo com o próximo ti do processo (assim, fica mais fácil de verificar qual o próximo tempo em que ocorrerá um evento)

Simulador

Para cada algoritmo de paginação, usamos uma estrutura de dados diferente, especificada na parte de paginação.

Gerenciamento de espaço livre

First Fit

- Coloca no primeiro lugar que couber o processo
- Percorre o bitmap e para no primeiro buraco que conseguir colocar o processo

Next Fit

- Percorre da mesma forma que o first fit, com uma particularidade
- Uma variável global mantém sempre a última posição que uma busca parou
- Uma nova busca começa sempre de onde a última parou
- Para evitar segfault --> *mod*(variável)

Best Fit

- Busca o menor buraco possível no bitmap, que caiba o processo
- Inicia uma variável *menor* com o tamanho do bitmap + 1
- Percorre toda a memória, e quando encontra buracos menores que *menor* troca e armazena onde começa esse buraco

Worst Fit

- Busca o maior buraco possível no bitmap, que caiba o processo
- Inicia uma variável *maior* com zero
- Percorre toda a memória, e quando encontra buracos maiores que *maior* troca e armazena onde começa esse buraco

Substituição de página

Optimal

- Para a realização do algoritmo Optimal, usamos um vetor de inteiros que representa o “tempo futuro”, i. e., o próximo tempo em que aquela página será acessada.
- Quando a página é colocada na memória física, o simulador percorre a lista de tempos do processo que está na página e coloca neste vetor o próximo tempo em que o processo irá acessar um endereço de memória que está nesta página
- Sempre que for ocorrer um pagefault, remove a página que tem o maior “tempo futuro”

Second-Chance

- Para a realização do Second-Chance, usamos uma fila de páginas. Toda vez que um processo acessa uma página e esta é colocada na memória física, a página é colocada nessa fila.
- Quando ocorre um pagefault, olhamos para a primeira página da fila. Se o bit R for 0, retiramos esta página. Se não for zero, colocamos a página no final da fila e zeramos o bit R
- Quando encontramos um bit R que vale 0, retiramos esta página da memória física

Clock

- Para a realização do Clock, usamos um vetor de páginas. Toda vez que um processo acessa uma página e esta é colocada na memória física, a página é colocada nesse vetor
- Quando ocorre um pagefault, o algoritmo se assemelha ao Second-Chance, porém, ao contrário de colocar a página no final da fila, o algoritmo percorre o vetor usando o módulo do tamanho do vetor, percorrendo-o assim de forma parecida com uma lista circular.

Least Recently Used

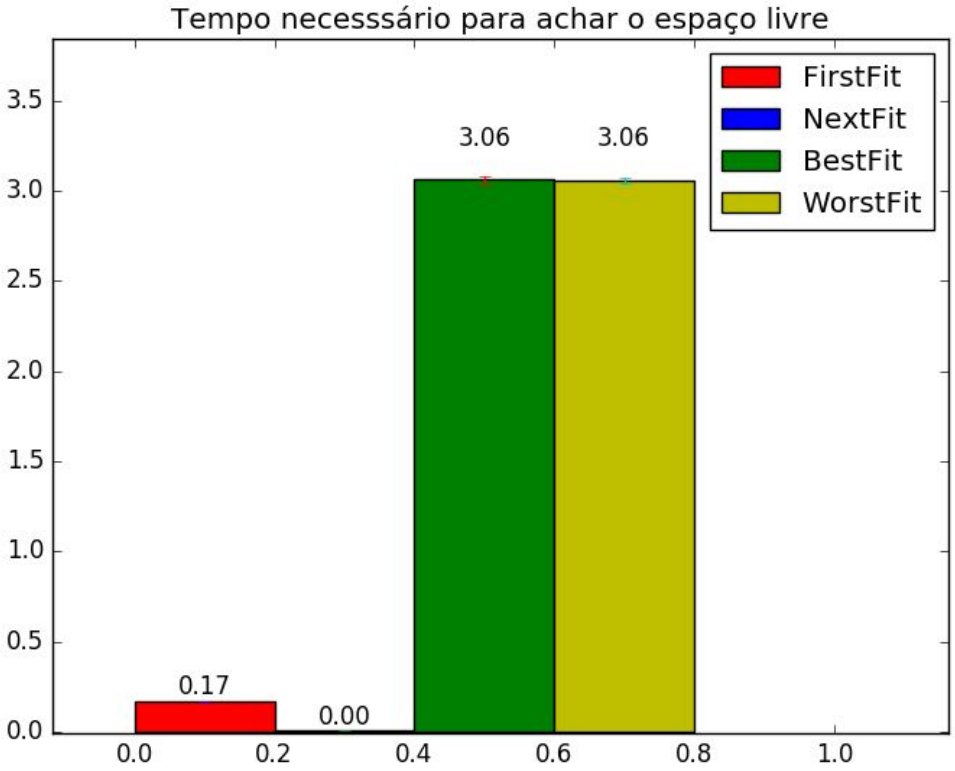
- Para a realização do LRU, usamos um vetor de inteiros que representa todas as páginas que estão na memória física
- Sempre que passa um certo intervalo de tempo (usamos de exemplo duas unidades de tempo) adicionamos em cada posição do vetor 2^8 * (bitR), simulando assim a adição do bit R no algarismo mais significativo de um contador binário (um contador de tamanho 8)
- Depois de adicionar este valor, zeramos o bit R
- Quando ocorre um pagefault, tiramos a página que possua o maior contador

Resultados

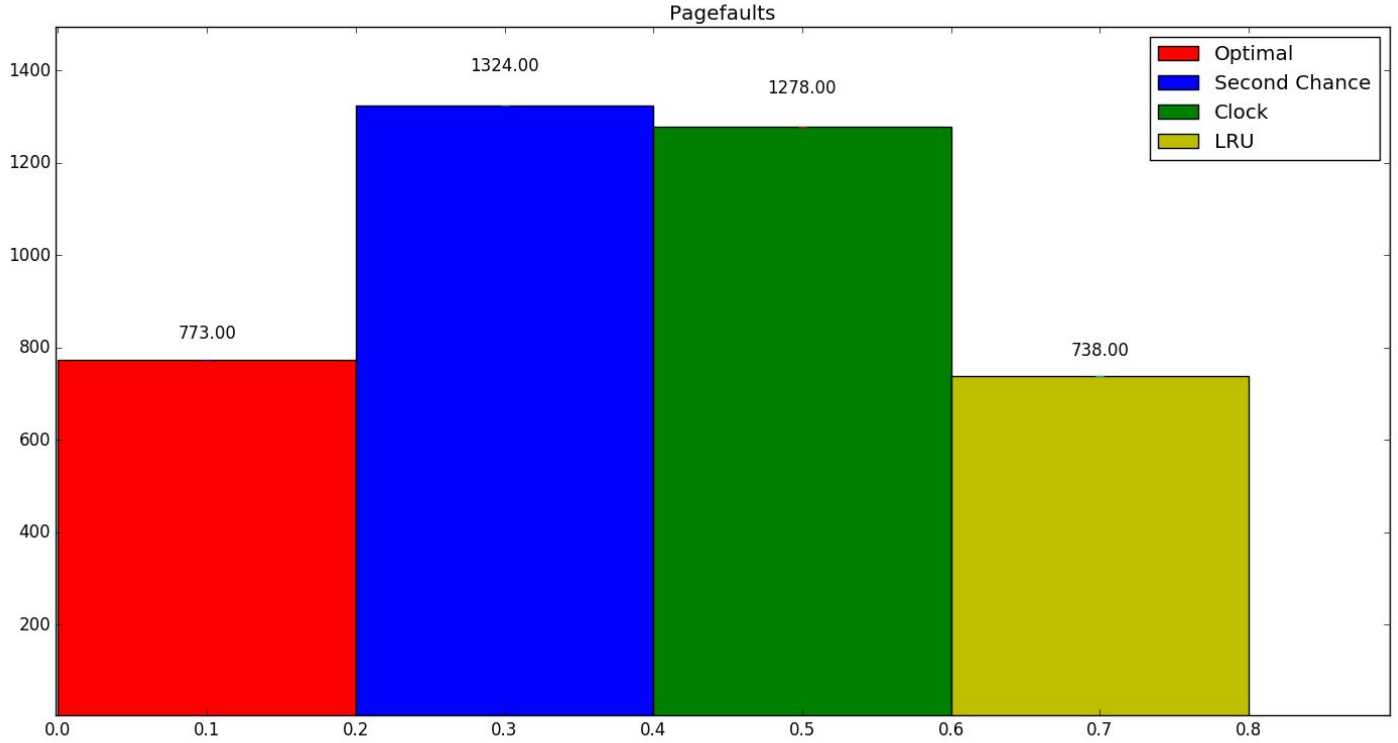
Sobre os testes realizados

- Consideramos um único arquivo de trace, com 1000 processos, 1000 de total de memória física e 100000 de memória virtual, 5 como unidade de alocação e 10 para o tamanho da página
- Executamos 30 vezes cada algoritmo
- Para os algoritmos de gerenciamento de memória, medimos o tempo em *ms*
- A máquina utilizada para testes tinha um processador intel i3, com 4gb de RAM

Gerenciamento de espaço livre



Substituição de página



Comentário sobre os resultados

- Para o gerenciamento de espaço livre, o gráfico tem o comportamento esperado. *Best Fit* e *Worst Fit* tem exatamente o mesmo tempo pois ambos percorrem toda a memória para decidir onde colocar um processo. *Next Fit* acaba se saindo melhor que o *First Fit* pois sempre faz menos comparações para decidir onde colocar o processo
- A grande diferença entre os dois primeiros e os dois últimos se dá pois o tamanho da memória que usamos é crucial para *Best Fit* e *Worst Fit*
- Sobre a substituição de páginas, era esperado que o *Optimal* tivesse menos pagefaults que o *LRU*, mas não foi o que aconteceu, provavelmente por causa do arquivo trace que utilizamos
- *Second-Chance* e o *Clock* tem mais pagefaults que os anteriores, isto era o esperado