EP3 - MAC0422 - 2015
main, file, directory, regular, command e block

Renato Lui Geh e Ricardo Fonseca

Linha de input

Assim como nos últimos EPs, há uma string `cmd_line` que é a linha de input do usuário. Em seguida dividimos `cmd` em tokens.

Isso é feito na função `Tokenize`. Os tokens são armazenados numa tabela, sendo que o primeiro elemento equivale ao próprio comando, e não o primeiro argumento do comando.

Prompt e Comando

Foi usado `readline` e `history` para o prompt, assim como foi feito no EP1. Com o comando do usuário mais seus argumentos, passamos a tabela com tais valores para a função `CommandToFunction` do módulo `utils`, que realizará, se possível, a função equivalente.

Atributos

A classe abstrata `file` é a representação de um arquivo em nosso programa. Ela possui os seguintes atributos:

`string name_`: Nome do arquivo

`time_t _create_`: Tempo de criação do arquivo

`time_t _modify_`: Tempo da última modificação do arquivo

`time_t _access_`: Tempo do último acesso ao arquivo

Métodos

Além disso, file possui um método para mudar seu nome (`Rename()`), um para que calcula quantos blocos o arquivo ocupa (`long int Block()`) e 3 métodos que atualizam os tempos de criação, modificação e último acesso:

`RefreshCreationTime()`,

`RefreshModifiedTime()` e

`RefreshAccessedTime()`, respectivamente.

Existem também duas funções abstratas, `long int Size()` e `bool IsDirectory()`, que retornam o tamanho em bytes do arquivo e se essa classe na verdade é um diretório.

Operadores

Por último, damos override nos operadores `<`, `>` e `==`, para comparar arquivos. Escolhemos fazer as comparações lexicográficas, para depois facilitar a impressão em ordem alfabética.

MORTE

MORTEMORTEMORTEMORTEMORTEMORTEMORTEMORTEMORT
MORTEMORTEMORTEMORTEMORTEMORTEMORTEMOR-
TEMORTEMORTE
MORTEMORTEMORTEMORTEMORTEMORTEMORTEMOR-
TEMORTEMORTE
MORTEMORTEMORTEMORTEMORTEMORTEMORTEMOR-
TEMORTEMORTE
MORTEMORTEMORTEMORTEMORTEMORTEMORTEMOR-
TEMORTEMORTE
MORTEMORTEMORTEMORTEMORTEMORTEMORTEMOR-
TEMORTEMORTE
MORTEMORTEMORTEMORTEMORTEMORTEMORTEMOR-
TEMORTEMORTE
MORTEMORTEMORTEMORTEMORTEMORTEMORTEMOR-
TEMORTEMORTE
MORTEMORTEMORTEMORTEMORTEMORTE

mem_node e size_node

Para manipular a memória virtual, utilizamos duas structs diferentes, uma representando um bloco na memória virtual e outro que é apenas usado para o algoritmo Quick Fit

`mem_node`: Possui 6 campos: (`char t`) para o tipo do bloco, (`int i`) para a posição início da memória do bloco, (`int s`) para tamanho do bloco, (`int pid`) para o identificador do processo utilizando o bloco, (`mem_node *n`) e (`mem_node *p`) como ponteiros para os próximos nós da lista.

`size_node`: Lista de listas de tamanhos de blocos para o algoritmo Quick Fit. Possui 4 campos. (`int s`) para tamanho dos blocos da lista que o nó aponta, (`mem_node *f`) como ponteiro para uma lista ligada sem cabeça com os blocos livres de tamanho `s`, (`size_node *n`) e (`size_node *p`) como ponteiros para os próximos nós da lista.

Algoritmos

Para ser mais fácil de se escolher qual gerenciador usar, criamos a variável `manager`, que é um ponteiro para função. Os argumentos de linha de comando são lidos e a função é atribuída a `manager` em seguida.

O gerenciador pelo método Quick Fit utiliza uma lista de listas, que é criado apenas no caso dele ser escolhido ao rodar o programa.

Listas Ligadas

Em todos os gerenciadores implementados foram usadas a lista com a cabeça `v_mem_h`. Para a memória física total utiliza-se `t_mem_h`

`v_mem_h`: guarda todos os blocos de memória livres ou ocupados na memória virtual. O algoritmo Quick Fit utiliza de forma um pouco diferente essa lista.

`t_mem_h`: guarda todos os blocos de memória ocupando a memória física total.

First Fit (FF)

Este gerenciador usa diretamente a lista da memória virtual. Quando o `t_secs` chega no instante `t0` de um processo, se há algum espaço livre que ele caiba, o processo é atribuído àquela parte da memória virtual. O bloco de memória é mudado para `P`.

O processo permanece na memória até terminar.

Next Fit (NF)

Este gerenciador também usa diretamente a lista da memória virtual. Quando o `t_secs` chega no instante `t0` de um processo, se há algum espaço livre que caiba o processo, o processo é atribuído àquela parte da memória virtual. Porém usamos um apontador de nó especial `v_last` que marca a última posição vista na lista, dessa forma para futuros usos da função, ela começa do último bloco criado.

Quick Fit (QF)

Ao escolher o algoritmo Quick Fit, o dicionário é criado dinamicamente por nosso algoritmo, que faz o seguinte:

- Pega o valor disponível (inicialmente toda a memória virtual disponível).
- Divide em 4 esse espaço, e aproxima até o maior multiplo de 2 menor que esse valor.
- Se o resultado for maior que o limite inferior (definido por nós como 16, de acordo com o tamanho das páginas), separa 3/4 do tamanho total recebido pela função como espaço livre nas listas (para cada 1/3 é criado um bloco de espaço livre desse tamanho próximo de 1/4 do tamanho total). Em seguida chama a função recursivamente para o restante do espaço.
- Caso o resultado seja igual ou menor que o limite inferior, separa o maior numero de intervalos com tamanho limite inferior como espaço livre na lista do quick fit.

QF - Continuação

Quando o `t_secs` chega no instante `t0` de um processo, ele procura o menor bloco livre que seja maior ou igual que o tamanho necessário do processo (percorre a lista de size_node), e depois marca como ocupado esse bloco, e põe no começo da lista `v_mem_h` esse processo (essa lista ligada não possue ordenação para processos, apenas tem todos os blocos que estão sendo ocupados por processos.

ff_nf_free

Tanto para o gerenciador de memória FF quanto NF utilizamos a mesma função para quando um processo termina. De forma simples ela transforma o bloco como livre, e com a ajuda das listas duplamente ligadas, verifica se o bloco estava entre algum outro bloco livre, eficientemente unindo-os em um só bloco pronto para ocupar outro processo.

qf_free

Para o gerenciador de memória QF utilizamos a função `qf_free` para quando um processo termina. Ela transforma o bloco do processo como livre, e utilizando nossa `lista de listas`, disponibiliza como livre o bloco na sua respectiva lista (a que possue outros blocos com seu mesmo tamanho).

Paginação

Para paginação, rodamos cada processo e se no tempo t_i houver um acesso, vemos se a memória a ser acessada está na memória física. Se não estiver, page fault e usamos um dos algoritmos de substituição de página. Senão não fazemos nada, já que estamos apenas simulando e não realmente lendo a memória.

Not Recently Used Page (NRU)

Foi usado um vetor `virt_refs::internal` para representar cada página da memória virtual. Nesse vetor estão os dados de acesso de cada página. Dado um elemento e_i, se $e_i = -1$, então ele não foi acessado recentemente. Senão, se $e_i = 0$, então ele foi acessado neste ciclo. Senão, entao $e_i > 0$ e portanto ele foi acessado e_i segundos atrás. Se e_i for maior ou igual ao limite `INTERRUPT_DT` que representa o tempo decorrido após cada *reset*, então $e_i = -1$. A função `nru_repl` procura por $e_i, ..., e_j$ que sejam, juntos, do tamanho requisitado e que tenham menor peso. Definimos peso como:

$w_i = 1$, se $e_i \geq 0$

$w_i = 0$, se $e_i < 0$

Portanto, queremos $min(\sum_{k=i}^{j} w_k)$.

First-In, First-Out (FIFO)

Neste algoritmo de substituição de página, apenas temos uma fila page_queue e toda vez que ocorre um *page fault*, pegam-se as *n* primeiras páginas ocupadas (e portanto mais "velhas") tal que a soma dos tamanhos delas sejam a menor soma possível e que sejam também maior ou igual ao tamanho requerido.

Second-Chance Page (SC)

Muito parecido com nosso FIFO, o algoritmo SC vai usar uma fila page_queue, porém ao ocorrer uma page fault, antes de retirar automaticamente a página do topo (as mais "velhas"), ele verifica o elemento e_i da pagina. Caso ele for >= 0, faz ele virar -1 e põe a página no fim da fila. Caso o contrário apenas remove que nem o algoritmo FIFO.

Observações

Nosso algoritmo Quick Fit divide a memória virtual até tamanhos maiores ou iguais que o limite inferior, o que gera um problema de desperdício de memória, pois se a memória não for múltiplo dele (16 no caso), é possível que até 15 bytes sejam desperdiçados e nunca utilizados, mas comparado com a memória virtual disponível não é de grande impacto esse valor.