MAC422 - EP3

Bruno Ferrero ; Rodrigo Alves Souza ; Taís Pinheiro

Problema

- Simular a gerência de memória utilizando diferentes algoritmos de uso de espaço livre e algoritmos de substituição de páginas;
- O espaço da **memória virtual** será manipulado por meio dos **algoritmos de gerência de espaço** livre;
- A política de acesso à **memória física** será feita por meio dos **algoritmos de substituição de páginas**.

Solução: ideia geral da simulação

Implementação feita em Python 2.7

1. Carrega um *trace*;
2. Cria lista de execução com `t0` e ação;
3. Ordena a lista em ordem crescente de `t0`;
4. Em um *loop* com um *clock* e um contador iterando na lista de execução executa as ações.

Solução: iterando na lista de execuções

```
if execucao[1] == 'COMPACTAR':
    print 't: ' + str(execucao[0]) + ' COMPACTAR'
    compactar()
elif execucao[1] == 'REMOVER':
    # Aqui vamos tratar o caso de remover um processo
    mem_virtual.remover_processo(execucao[3],mem_fisica)
elif execucao[1] == 'ALOCAR':
    # Chegou um novo processo: manda pra memoria virtual
    print "t: " + str(execucao[0]) + ' ' + execucao[1] + ' para ' + execucao[3].nome
    # disparando o cronometro
    t_i = time.time()
    if espaco == 1:
        mem_virtual.best_fit(execucao[3])
    elif espaco == 2:
        mem_virtual.worst_fit(execucao[3])
    elif espaco == 3:
        mem_virtual.quick_fit(execucao[3])
    #carrega processo na tabela de pagina (mem virtual)
    mem_virtual.set_pagina_tabela (execucao[3])
    # para o cronometro e soma o tempo
    t_f = time.time()
    timetotal += (t_f - t_i)
elif execucao[1] == 'ACESSO':
    # disparando o cronometro
    t_i = time.time()
    # executa ACESSO a memoria
    executa(execucao,substitui)
    # para o cronometro e soma o tempo
    t_f = time.time()
    timetotal += (t_f - t_i)
count += 1
if count == len(listaExecucao):
    print 'Fim da Simulacao em t: ' + str(clock)
    print 'Estado da Memoria Virtual em t: ' + str(clock)
    mem_virtual.dump()
    print 'Estado da Memoria Fisica em t: ' + str(clock)
    mem_fisica.dump()
    break
execucao = listaExecucao[count]
```

```
t: 1 ALOCAR para P0
t: 1 ACESSO de P0 em: 1
t: 2 ACESSO de P0 em: 2
t: 2 ALOCAR para P1
t: 2 ACESSO de P1 em: 1
t: 3 ACESSO de P0 em: 3
t: 3 ACESSO de P1 em: 2
t: 3 COMPACTAR
t: 4 ACESSO de P1 em: 4
t: 4 ALOCAR para P2
t: 4 ACESSO de P2 em: 1
t: 5 REMOVER P1 com inicio em p: 8
t: 5 ACESSO de P2 em: 2
t: 6 ACESSO de P2 em: 3
t: 7 ACESSO de P2 em: 4
t: 7 REMOVER P2 com inicio em p: 16
t: 10 REMOVER P0 com inicio em p: 0
```

Ações durante a simulação

```
while True:
  if clock == execucao[0]:
     if execucao[1] == 'COMPACTAR': ...
     elif execucao[1] == 'REMOVER': ...
     elif execucao[1] == 'ALOCAR': ...
       if espaco == 1:

mem_virtual.best_fit(execucao[3])
       elif espaco == 2:
          mem_virtual.worst_fit(execucao[3])
       elif espaco == 3:
          mem_virtual.quick_fit(execucao[3])
     elif execucao[1] == 'ACESSO':
           acessa(execucao,substitui)
```

`COMPACTAR`: compacta memória virtual e física;

`REMOVER`: remove um processo terminado (em tf) das memórias, da tabela de página e libera o espaço na lista ligada de espaços livres/ocupados;

`ALOCAR`: reserva um espaço contíguo de memória (virtual) que comporte o processo que chegou (algoritmos de gerência de espaço);

`ACESSO`: acessa as posições de memória que vão sendo solicitadas. O acesso é feito por páginas via algoritmos de substituição de páginas

Mais sobre o ALOCAR

- A alocação de espaço na memória física leva em consideração tanto a unidade de locação quanto o tamanho da página;

Exemplo: PID = 2; Tam = 8 bytes ; UA=3; Pg = 6;

Serão alocadas **duas** páginas e marcadas **três** unidades de locação

2 2 2 2 2 2 **|** 2 2 2 -1 -1 -1 **|** -1 -1 -1 -1 -1 -1 -1 ...

- Os algoritmos são implementados usando listas ligadas

Mais sobre o ALOCAR

best-fit

Faz uma busca linear

- Varre a lista completamente em busca do menor espaço disponível que pode adequar o processo
- Na alocação considera o tamanho da página e o tamanho do processo

worst-fit

Faz uma busca linear

- Análogo ao best-fit, varre a lista por completo em busca do maior espaço disponível capaz de alocar o processo

quick-fit

Manipula mais listas

Cria-se listas "extras" com possíveis tamanhos mais acessados

Mais sobre o ACESSO

SE: a memória física tem espaço livre para página, a página com o novo acesso entra nesse espaço;

SENÃO: Pagefault; uma página é removida de acordo com o algoritmo de substituição de página selecionado;

Mais sobre o ACESSO

FIFO

Variável tAcesso

- Possui uma variável tAcesso;
- Guarda o clock do último acesso;
- Busca na tabela de páginas a página com o acesso mais antigo;

LRUv2

Matriz de nPagXnPag

- Marca linha (com 1) referente a pagina acessada
- Marca coluna (com 0) referente a pagina acessada
- Remove a pagina com menor valor inteiro (bin->int)

LRUv4

Variável countLRUv4

```
def set_countLRUv4(self,k):
    self.countLRUv4 += 1
    self.countLRUv4 =  self.countLRUv4%k
```

Modularização do código

ep3.py

- Exibe toda a iteração com o usuário, recebendo os dados de entrada.

execute.py

- Processa o arquivo trace criando uma lista de ações que devem ser executadas na ordem cronológica.

linkedlist.py

- Gerencia o espaço disponível da memória através de listas ligadas.

memory.py

- Gerencia os status das memórias física e virtual, atualizando os arquivos necessários a cada chegada, acesso ou término de um processo

Modularização do código

paging.py

- Gerencia as tabelas de páginas e se necessário realiza as operações de paginação através dos algoritmos.

processo.py

- Responsável por criar a estrutura básica de cada processo e armazenar nesta estrutura os dados e características do processo em questão.

run_tests.sh

- Faz 30 execuções de um arquivo trace para todas as combinacoes de algoritmos de paginacao e de espaco livre.

Testes

Arquivo utilizado para os testes:

Conteúdo: 9 processos, total : 40, virtual: 100, s: 2 e p:4

1 16 7 P0 1 2 2 2 3 3 4 4 5 5 6 7

1 17 8 P1 1 2 4 3 5 4 6 5 7 6

2 4 9 P2 6 2 5 3 1 4

3 COMPACTAR

4 16 14 P3 1 5 2 6 3 8 5 8 5 9 7 10 1 11 0 12

7 20 15 P4 0 7 2 7 5 8 11 8 13 9 14 9

11 15 3 P5 2 11 2 12 0 13 1 14

11 20 3 P6 2 11 2 13 0 19 1 20

13 30 9 P7 0 13 2 14 1 16 4 17 6 18 6 19 7 20 8 21 8 22 8 27

14 COMPACTAR

15 30 18 P8 0 16 2 17 1 18 4 19 6 21 6 22 6 23 10 24 15 24 16 27 17 29

Execução via terminal interativo ou *batch mode*

$ python ep3.py

[ep3]: espaco 1

[ep3]: substitui 3

[ep3:] carrega trace.txt

[ep3:] executa 5

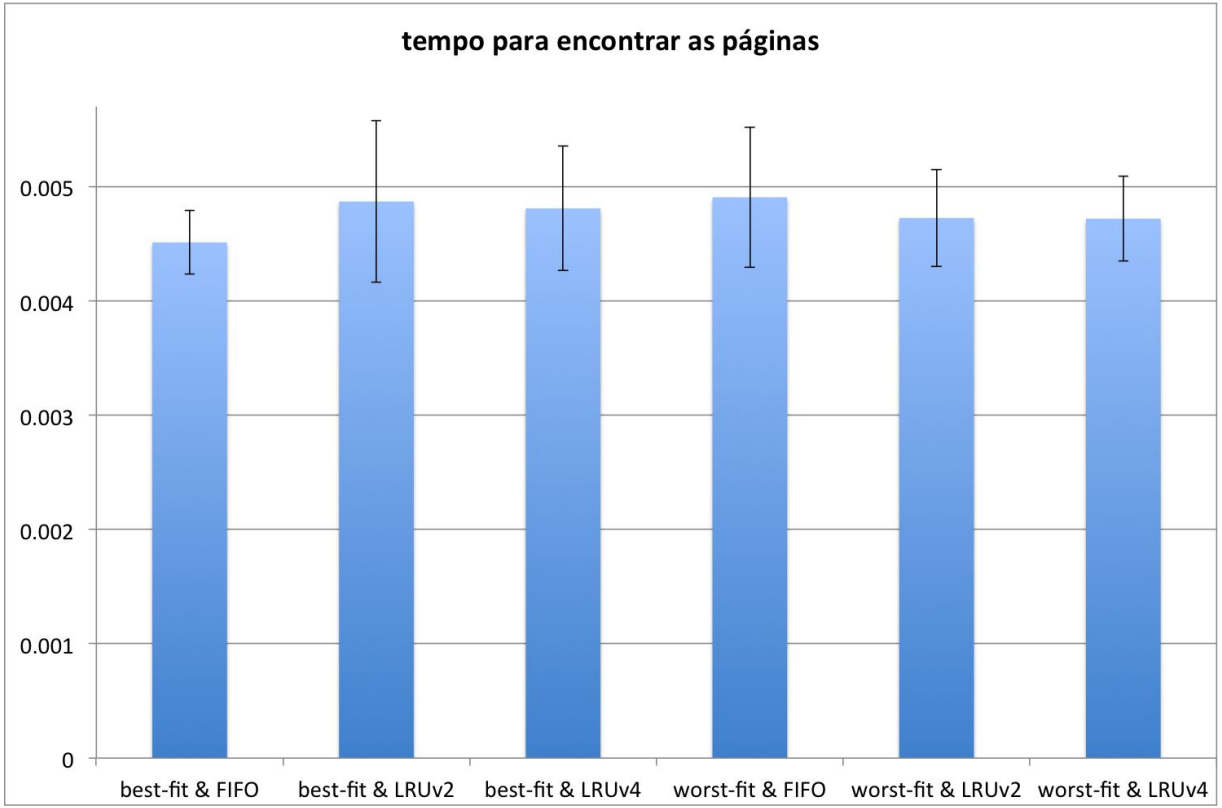
 # batch mode para gerar os resultados

$ python ep3.py ARQUIVO ESPACO SUBSTITUI

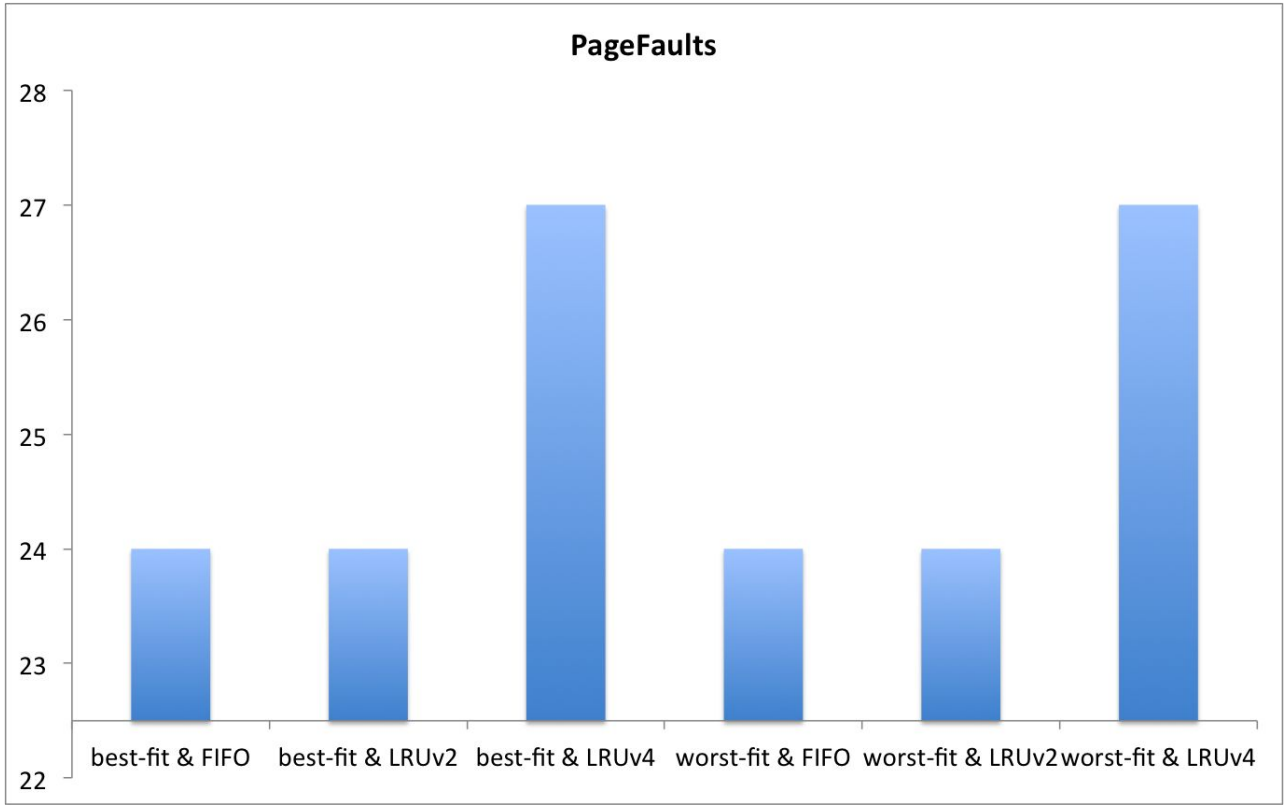
Execução: saída do script que gera os resultados

```
> chuva :ep3 $ ./run_tests.sh trace1.txt
best-fit & First In, First Out
media | std
0.00112181 0.000139746
Pagefault:  5
best-fit & LRUv2
media | std
0.00126387 0.000276424
Pagefault:  5
best-fit & LRUv4
media | std
0.00121386 0.000253529
Pagefault:  5
worst-fit & First In, First Out
media | std
0.00109519 0.000136653
Pagefault:  5
worst-fit & LRUv2
media | std
0.00115078 0.000177338
Pagefault:  5
worst-fit & LRUv4
media | std
0.00112385 0.000161002
Pagefault:  5
```

Resultados


tempo para encontrar as páginas
0.005
0.004
0.003
0.002
0.001
0
best-fit & FIFO
best-fit & LRUv2
best-fit & LRUv4
worst-fit & FIFO
worst-fit & LRUv2
worst-fit & LRUv4

Resultados


PageFaults
28
27
26
25
24
23
22
best-fit & FIFO
best-fit & LRUv2
best-fit & LRUv4
worst-fit & FIFO
worst-fit & LRUv2
worst-fit & LRUv4

Fim