

Universidade Federal do Rio Grande do Sul



Departamento de Informática Aplicada  
Sistemas Operacionais I

Felipe de Souza Lahti - 170715  
Germano de Mello Andersson - 137719

Documentação do simulador  
LRU com segunda chance

Prof. Dr. Sérgio Luis Cechin

Porto Alegre, 14 de Junho de 2012

# INTRODUÇÃO

O simulador LRU com segunda chance tem por objetivo simular a execução do algoritmo de substituição LRU com segunda chance a partir da execução de comandos recebidos fornecendo ao final da execução informações estatísticas tais como acesso, números de page faults, número de substituições realizadas para cada página de cada processo.

## Usando o simulador

Para utilização do simulador basta executar o comando 'lrusimul', na pasta bin, seguido do arquivo que contém com os comandos. O arquivo de log com as informações estatísticas serão salvos em "perf/log.txt". Exemplo: /bin/lrusimul testes/t1.txt

## AMBIENTE DE DESENVOLVIMENTO

### Plataforma

Trabalhamos em ambientes distintos:

#### Ambiente1

Processador: Intel Core i5-2410M @2.30GHz (2 cores, 4 threads com HT)

GNU/Linux: Ubuntu 11.10

Kernel: 3.0.0-12

GCC: 4.6.1

Rodado em ambiente virtualizado utilizando: VirtualBox 4.1.10

#### Ambiente2

Processador: Intel Core2 Duo T8300, 2 cores com suporte HT

GNU/Linux: Debian 6.0.3 (squeeze)

Kernel: 2.6.32-5-686

GCC: 4.4.5

## LRUSIMUL

### Estruturas de Dados

a) page\_struct: representa uma página. Contém número da página, informação se está na memória ou no swap (são as aceitas as constantes SWAP e MEM), bits RB (bit de referência) e MB (bit de modificação), número de vezes em que página foi lida e escrita, número de page faults e o número de substituições que a página sofreu. Se a página estiver na memória também contém ponteiros sobre a página anterior e a próxima para representar uma FIFO duplamente encadeada.

b) proc\_struct: estrutura utilizada para representar um processo no sistema. Contém o pid do processo, o número de páginas e a tabela de páginas.

c) proc\_struct\_list: representa uma lista de processos (proc\_struct).

d) mem\_actions\_struct: representa uma ação a ser executada pelo simulador. Contém a ação e ate dois parâmetros. O identificador da ação pode ser as seguintes constantes: MEMSIZE, PROCSIZE, READ, WRITE, ENDPROC.

e) mem\_struct: representa a estrutura da LRU com segunda chance. Contém um inteiro com o tamanho dos quadros, quantidade de frames utilizados. Para representar a tabela usamos uma fila

duplamente encadeada de `page_struct`. Por questões de otimização temos um ponteiro para o final da fila além de um ponteiro para o início da fila para as operações em que damos uma “segunda chance” para a página.

## Tabela de páginas

Cada processo tem uma tabela com informações sobre todas as suas páginas. Cada página é representada pela estrutura `page_struct` descrita acima.

## Precedência da segunda chance

Caso seja necessário a substituição de uma página é obedecida a seguinte precedência considerando os bits de referencia (RB) e de modificação (MB):

- 1) RB = 0 e MB = 0

Melhor escolha pois a página não esta sendo referenciada e nem foi escrita.

- 2) RB = 0 e MB = 1

Tem a desvantagem de precisar fazer I/O para salvar a pagina antes de substituí-la.

- 3) RB = 1 e MB = 0

Provavelmente a página sera lida novamente pois seu bit de referência esta ligado.

- 4) RB = 1 e MB = 1

Pior caso, alem de poder ser usada novamente requer I/O para salvar a página.

## Funcoes internas do simulador

**mem\_actions\_struct\* file\_2\_memaction ( FILE \*system\_config );**

Parser dos comandos recebidos no arquivo. Retorna uma lista de ações.

**void system\_run(mem\_actions\_struct \*system);**

Recebe a lista de ações e as executa na ordem passada.

**void execute\_action(mem\_actions\_struct \*action);**

Simula uma ação.

**void lru\_2nd\_choice(page\_struct \*page);**

Realiza o processo de colocação da página na memória. Se necessário, executa o algoritmo de LRU com segunda chance para escolher qual página será substituída para inserção desta nova.

**void move\_to\_swap(page\_struct \*page);**

Move uma página para o swap. Após aplicar o algoritmo de escolha da vítima a função `lru_2nd_choice` chama `move_to_swap` com a página a ser movida para o swap.

**void move\_to\_mem\_tail(page\_struct \*candidate);**

Move uma página para o final da fila, pois ela recebeu uma segunda chance.

**void memsize\_action(int size);**

Inicializa o tamanho da memória. Deve ser o primeiro comando na lista de comandos.

**void procsz\_action(int pid, int size);**

Cria um processo com “size” páginas.

**void read\_action(int page, int pid);**

Lê a página “page” do processo identificado por “pid”. Chama **lru\_2nd\_choice** para inserção na memória caso a página esteja no SWAP.

**void write\_action(int page, int pid);**

Escreve na página “page” do processo identificado por “pid”. Chama **lru\_2nd\_choice** para inserção na memória caso a página esteja no SWAP.

**void endproc\_action(int pid);**

Libera todas as páginas utilizadas que estão na memória.

**void reset\_page(page\_struct \*page, int page\_number);**

Inicializa uma página e zera as estatísticas.

**void print\_procs\_stats();**

Imprime as estatísticas na tela e no arquivo perf/log.txt.

## Repositório

Utilizamos o repositório público do google para hospedagem do nosso projeto:

<http://code.google.com/p/lrusimul/>

## TESTES

Utilizamos a ferramenta gdb (GNU Debugger) para depuração do programa, durante as fases de desenvolvimento. Para teste das funcionalidades do programa, criamos dois conjuntos de testes: o primeiro visando apontar falhas relacionadas a especificação e o segundo visando a utilização propriamente dita do sistema. Abaixo citamos os testes desenvolvidos e uma breve explicação do seu objetivo:

**t1.txt** – Testa uma substituição de página entre dois processos com 5 páginas e uma memória com 3 páginas(quadros).

**t2.txt** – Testa precedência de uma página modificada.

**t3.txt** – Testa erro de acesso a página inválida.

Outros testes com dados gerados aleatoriamente.

Para execução dos testes

```
>cd $lrusimul_dir; make install; bin/lrusimul testes/t[$testnumber].txt
```

## DECISÕES TÉCNICAS / DIFICULDADES

Uma das dificuldades foi encontrar uma boa IDE que facilite a localização de erros com ponteiros em C, tenha autocomplete para aumentar a produtividade, infelizmente, C puro pode não ser uma linguagem muito produtiva quando se trabalha com ponteiros onde enganos são facilmente cometidos sem o auxílio de uma IDE melhor e com verificação de tipos mais forte.