Simulador de Gerência de Memória - Trabalho prático 3 - ENTREGA : 14/06/2012

1. Objetivo

O objetivo deste trabalho é o desenvolvimento de um simulador de gerência de memória virtual que implementa uma política de substituição de páginas do tipo LRU com a variação "segunda chance". Deve ser utilizado a estratégia global para a alocação de quadros (não é necessário verificar a qual processo a página pertence).

2. Especificação do comportamento do simulador

O simulador receberá como entrada um arquivo texto com a configuração do sistema de memória e as operações realizadas pela memória virtual. Ao final da execução, o simulador deverá gravar um outro arquivo texto com os resultados da simulação.

O arquivo de entrada é composto por um conjunto de linhas. Cada linha contém uma ação a ser executada pelo simulador. Essas ações podem ser comandos de configuração do simulador ou a descrição de eventos a serem simulados. A tabela I fornece a lista de ações que o simulador deve ser capaz de realizar e seus respectivos parâmetros.

Ação	Parâmetros/Tipos de dados	Descrição
MEMSIZE	size(inteiro)	Define o tamanho da memória física (RAM) em $size$ quadros.
PROCSIZE	id(inteiro) size(inteiro)	Cria processo identificado por id com $size$ páginas. Considera-se que essa criação coloca **todas** as páginas do processo no $swap$ e não na memória RAM.
READ	page(inteiro) id(inteiro)	Realiza a leitura de uma posição de memória da página $page$ do processo id. Se a página não estiver na memória, ela deve ser carregada. Se não houver quadros livres, o algoritmo LRU deve ser acionado para providenciar a substituição de uma outra página e a carga da página solicitada.
WRITE	page(inteiro) id(inteiro)	Realiza a escrita em uma posição de memória na página $page$ do processo id. Se a página não estiver na memória, ela deve ser carregada. Se não houver quadros livres, o algoritmo LRU deve ser acionado para providenciar a substituição de uma outra página e a carga da página solicitada.
ENDPROC	id (inteiro)	Termina a execução do processo id. Todos os quadros alocados a este processo devem ser liberados.

TABLE 1 – Ações e eventos do simulador

Observe que, nas linhas de comandos do arquivo de entrada, podem existir até três campos : uma ação (comando) e um máximo de dois parâmetros. Todos os campos estarão separados por um ou mais espaços em branco ou TABs. Por exemplo, abaixo está a ação de leitura da página 5 do processo 8 :

```
READ 5 8
```

Note que as ações de leitura e escrita indicam - por processo - apenas as páginas que estão sendo acessadas, não tendo a informação específica do endereço de memória, pois, do ponto de vista da memória virtual, só é importante saber quais as páginas estão sendo referenciadas (lembre-se do conceito de *string de referência*). Ainda, a ação MEMSIZE do arquivo de entrada fornece a quantidade de quadros da RAM física. A indicação de quando, e quais processos serão criados, seus respectivos tamanhos, em termos de quantidade de páginas, é fornecida pela ação PROCSIZE. A simulação termina quando for lida e executada a última linha válida do arquivo de entrada.

O arquivo de saída deve apresentar as estatísticas da simulação realizada. Essas estatísticas devem ser organizadas por processos e, dentro de cada processo, devem estar listadas as estatísticas de cada página com as seguintes métricas :

- Página : Número de identificação da página cujas estatísticas serão listadas ;
- Acessos(RW) : Quantidade de acessos de leitura e/ou de escrita feitas nesta página.
- NroPageFault : Quantidade de vezes que uma leitura ou escrita provocaram falta de página nesta página (observe que toda página acessada provocará pelo menos uma falta de página : quando ela é lida do $swap$ para a memória pela primeira vez).
- NroSubst : Quantidade de vezes que a página foi eleita como "vítima" para ser substituída por outra, caso não existam mais quadros disponíveis na RAM.

O arquivo executável do simulador deve obrigatoriamente se chamar $lrusimul$. O arquivo de entrada poderá ter um nome a sua escolha, mas deverá ser do tipo ".txt". Dessa forma, o simulador será chamado com a seguinte linha de comando (onde foi escolhido o nome $myconfig.txt$ para o arquivo de entrada) :

```
% lrusimul myconfig.txt
```

Como resultado da simulação, deve ser gerado um arquivo de saída chamado $log.txt$, o qual deve ser armazenado no diretório *perf*. Na seção 6 é dada a estrutura de diretórios a ser seguida.

Considera-se que no momento da simulação da criação de um processo de usuário (ação PROCSIZE), o programa executável correspondente é lido e a imagem do processo inteiro (código, dados e pilha) é definida e gravada na área de $swap$. Dessa forma, supõe-se que após o comando PROCSIZE o processo passará a existir e todas as suas páginas estarão residentes no $swap$. PORTANTO, NÃO FAZ PARTE DO ESCOPO DO TRABALHO vocês se preocuparem com a implementação da área de $swap$ nem como o programa executável transforma-se em uma imagem de processo. BASTA vocês imaginarem que existe uma área fictícia onde já estão todas as páginas pertencentes ao espaço de endereçamento lógico de um processo que foi criado.

3. Exemplo de Utilização do Simulador

Para melhor compreender o que é necessário ser feito neste trabalho e o comportamento esperado do simulador, segue um exemplo de sua utilização. Inicialmente é necessário definir um arquivo de entrada. Abaixo está o conteúdo parcial de um destes arquivos :

```
MEMSIZE  100
PROCSIZE 1 30
PROCSIZE 2 35
PROCSIZE 3 40
READ     12 1
READ     12 1
READ     30 2
WRITE    39 3
PROCSIZE 4 10
...
```

No exemplo, está sendo definido uma memória RAM com 100 quadros (páginas físicas). A seguir, são criados três processos P_1, P_2 e P_3 com, respectivamente, 30, 35 e 40 páginas. Na criação de processos, considera-se que todas as páginas foram postas na área de $swap$. O processo 1 inicia solicitando duas leituras a página 12. O primeiro acesso provocará uma falta de página que deverá ser tratado adequadamente (localizar um quadro livre, alocar e atribuir-lhe a página 12 do processo 1). O segundo acesso à página 12 pelo processo 1 transcorrerá normalmente, pois a página 12 já está carregada em memória. Após, ocorre uma falta de página provocado pelo acesso de leitura na pagina 30 do processo 2, e outra falta de páginas provocada pela escrita na página 39 pelo processo 3. O processo 4 é então criado com 10 páginas.

A ação de configuração MEMSIZE é sempre a primeira do arquivo de configuração. Sua ausência deve provocar um erro de execução. Sequências de ações READ e WRITE só podem referenciar processos que já foram criados por PROCSIZE e que ainda não foram concluídos (ação ENDPROC). Se essas ações ocorrerem fazendo referência a um processo não existente ou já concluído, também deverá gerar um erro de execução. As ações READ e WRITE podem ser intercaladas por qualquer quantidade de ações PROCSIZE e ENDPROC.

Durante a execução do simulador deve ser gerado o arquivo de saída $log.txt$ com as estatísticas do sistema. Abaixo é apresentada uma visão parcial do formato (obrigatório) deste arquivo.

```
PROCESSO 1
Página Acessos(R/W) NroPageFault NroSubst
0         15                1          1
12        36                1          0
13        20                2          2
19        10                3          2
20        4                 1          0

PROCESSO 2
Página Acessos(R/W) NroPageFault NroSubst
0         1                 1          1
24        31                1          0
25        15                3          2
30        18                2          1
...
```

4. Material suplementar de apoio

O algoritmo LRU com a variação segunda chance está descrito nos livros listados na bibliografia da disciplina e apresentado em aula.

Dica : para a leitura das linhas do arquivo de entrada sugere-se a leitura completa de cada uma delas e a posterior separação em seus campos componentes. Para isso, pode-se usar funções das bibliotecas padrões do "C" tais como $atoi$, $strstr$ e $strtok$ (e suas similares). Para auxiliá-lo faça o uso do comando man e dos oráculos da Internet (google, yahoo, bing,...).

5. *Road map* para a implementação

Alguns comentários sobre a tarefa. O ponto fundamental do trabalho é a implementação de um algoritmo LRU com a variação segunda chance. Portanto, **primeiro**, é necessário definir uma estrutura de dados para encadear de forma FIFO as páginas que estão sendo carregadas em memória e os ponteiros para analisar o bit de referência e o bit de modificação (sujo ou $dirty$). **Segundo**, quando um processo for criado, deve-se alocar uma estrutura para fazer o papel de sua tabela de páginas. Nessa tabela deverá haver, pelo menos, a informação se a página está na memória ou em $swap$, e os bits de referência e de modificação. **Terceiro**, implementar o LRU com segunda chance. **Por fim**, será necessário fazer com que o programa receba um arquivo de entrada via linha de comando e gere um arquivo $log.txt$ com as estatísticas.

IMPORTANTE : O programa deve obrigatoriamente ser implementado em C (não em C++) e executar em ambientes GNU/Linux. O trabalho pode ser desenvolvido em DUPLA (dupla significa "dois alunos").

6. Entregáveis

Deverá ser entregue, via moodle, um arquivo *tar.gz* (**sem** arquivos *rar* ou similares ! !), cujo nome deve ser o número de cartão UFRGS de um dos membros do grupo. O *tar* deve conter os fontes (arquivos.c), os arquivos de inclusão (arquivos .h), arquivo de $makefile$, conjunto de testes elaborados pelo grupo (arquivos de configuração), além da documentação do programa. É obrigatório obedecer a seguinte estrutura de diretórios :

```
\lrusimul
    |----makefile  (arquivo de makefile)
    |----src       (diretório onde estarão os fontes do simulador)
    |----include   (diretório onde estarão os arquivos .h do simulador)
    |----bin       (diretório onde será gerado o executável do simulador - lrusimul)
    |----testes    (diretório onde estarão arquivos de teste elaborados pelo grupo)
    |----perf      (diretório onde será gravado o arquivo de estatísticas - log.txt)
    +----doc       (arquivo com o relatório do programa em formato PDF)
```

ATENÇÃO :

(A) Faz parte da avaliação a obediência RÍGIDA a estes padrões de entrega.

(B) Obedecer imperativamente o formato de cada linha. O string que indica a *ação* deve ser exatamente aquele fornecido na tabela I e seus parâmetros são números inteiros.

(C) O programa executável deve ser chamado $lrusimul$ e gerar um arquivo de estatísticas com o nome $log.txt$ (gravado no diretório $perf$).

7. Critérios de avaliação

O trabalho será avaliado da seguinte forma :

- **1 ponto :** uso das melhores práticas de programação : clareza e organização do código, programação modular, makefiles, arquivos de inclusão bem feitos (sem código C dentro de um *include* ! !) e comentários "inteligentes". Obediência a especificação.
- **2 pontos :** documentação. A documentação corresponde a responder, no mínimo, ao questionário fornecido abaixo.
- **1 ponto :** elaboração dos programas de teste (quantidade e qualidade !).
- **6 pontos :** funcionamento do programa de acordo com a especificação. Para seu teste serão empregados programas padrão desenvolvidos pelo professor e pelos programas de teste elaborados pelo grupo.

Questionário base para documentação

1. Nome dos componentes do grupo e número do cartão.
2. Descrição da plataforma utilizada para desenvolvimento. Qual o tipo de processador (número de cores, com ou sem suporte HT) ? Qual a distribuição GNU/Linux utilizada e a versão do núcleo ? Qual a versão do gcc ? Se o trabalho foi feito ou não em ambientes virtualizados ? Em caso afirmativo, qual a máquina virtual utilizada (versão) ?
3. Descrever a estrutura de dados empregada para encadear as páginas no LRU para permitir a análise delas como se fosse o movimento de um ponteiro de relógio (algoritmo segunda chance).
4. Descrever a estrutura da tabela de páginas.
5. Explicitar, em função dos bits de referência e modificação, qual foi a ordem de preferência para escolher uma página vítima (modificada e acessada ; acessada e não modificada ; não acessada e modificada ; não acessada e não modificada). Explique o porque, as vantagens e as desvantagens dessa escolha. Utilize, também, os seus casos de teste para justificar a resposta.
6. Descrever o que funciona e o que NÃO está funcionando no simulador desenvolvido. Em caso de não funcionamento, dizer qual é a sua visão do porquê deste não funcionamento.
7. Qual a metodologia utilizada para testar o simulador ? Isto é, quais foram os passos (e arquivos de teste) efetuados para testar o simulador ? Foi utilizado um *debugger* ? Qual ?
8. Quais as principais dificuldades encontradas e quais as soluções empregadas para contorná-las.

8. Data de entrega e avisos gerais— LEIA com MUITA ATENÇÃO ! ! !

1. O trabalho pode ser feito em DUPLAS (de dois ! duplas com mais de dois terão sua nota final dividida pelo número de participantes do grupo)
2. O trabalho deverá ser entregue até as 23 :59 :00 horas do dia 14 de JUNHO de 2012 via moodle. Entregar um arquivo *tar.gz* conforme descrito na seção 6.
3. Trabalhos entregues atrasados serão penalizados com descontos na sua nota máxima : Trabalhos entregues até 21 de junho de 2012 (23 :59 :00 horas) passam a valer 80% da nota máxima ; trabalhos entregues até 28 de junho de 2012 (23 :59 :00 horas) passa a valer 60% da nota máxima. Expirado o atraso máximo (em 28/06/2012), nenhum trabalho será mais aceito.
4. O professor da disciplina se reserva, caso necessário, o direito de solicitar uma demonstração do programa com a presença de todo o grupo. A nota final será baseada nos parâmetros acima e na arguição sobre questões de projeto e de implementação feitas ao(s) aluno(s).