Sistemas Operacionais

TP2: simulação de um sistema de memória virtual

Trabalho individual ou em dupla.

Data de entrega: no moodle.

(Não serão aceitos trabalhos fora do prazo)

Este trabalho tem por objetivo fazer com que os alunos tenham contato com o tipo de código usado em simuladores e apliquem os conceitos de memória virtual vistos em sala de aula.

O problema

Neste trabalho você deve implementar um simulador de memória virtual. Assim, dentro do simulador você deverá implementar uma réplica das estruturas de um mecanismo de gerência de memória virtual.

Princípio geral

Seu programa deverá ser implementado em C (não serão aceitos recursos de C++, como classes da biblioteca padrão ou de outras fontes).

O simulador receberá como entrada um arquivo que conterá a sequência de endereços de memória acessados por um programa real (na verdade, apenas um pedacinho da sequência total de acessos de um programa). Esses endereços estarão escritos como números hexadecimais, seguidos por uma letra R ou W, para indicar se o acesso foi de leitura ou escrita. Ao iniciar o programa, será definido o tamanho da memória (em quadros) para aquele programa e qual o algoritmo de substituição de páginas a ser utilizado. O programa deve, então, processar cada acesso à memória para atualizar os bits de controle de cada quadro, detectar falhas de páginas (*page faults*) e simular o processo de carga e substituição de páginas. Durante todo esse processo, estatísticas devem ser coletadas, para gerar um relatório curto ao final da execução.

Forma de operação

O programa, que deverá se chamar `tp2virtual`, deverá ser iniciado com quatro argumentos:

```
tp2virtual lru arquivo.log 4 128
```

Esse argumentos representam, pela ordem:

1. o algoritmo de substituição a ser usado (`lru`, `2a` — segunda chance —, `fifo` e `random`);
2. o arquivo contendo a sequência de endereços de memória acessados (`arquivo.log`, nesse exemplo);

3. o tamanho de cada página/quadro de memória, em kilobytes — faixa de valores razoáveis: de 2 a 64;

4. o tamanho total da memória física disponível para o processo, também em kilobytes — faixa de valores razoáveis: de 128 a 16384 (16 MB).

Formato da saída

Ao final da simulação, quando a sequência de acessos à memória terminar, o programa deve gerar um pequeno relatório, contendo:

- a configuração utilizada (definida pelos quatro parâmetros);
- o número total de acessos à memória contidos no arquivo;
- o número de *page faults* (páginas lidas);
- o número de páginas "sujas" que tiveram que ser escritas de volta no disco (lembrando-se que páginas sujas que existam no final da execução não precisam ser escritas).

Um exemplo de saída poderia ser da forma (valores completamente fictícios):

```
prompt> tp2virtual lru arquivo.log 4 128
Executando o simulador...
Arquivo de entrada: arquivo.log
Tamanho da memoria: 128 KB
Tamanho das páginas: 4 KB
Tecnica de reposicao: lru
Paginas lidas: 520
Paginas escritas: 352
```

Formato do arquivo de entrada

Como mencionado anteriormente, cada linha contém um endereço de memória acessado, seguido das letras R ou W, indicando um acesso de leitura ou escrita, respectivamente. Por exemplo, as linhas a seguir foram retiradas de um dos arquivos utilizados:

```
0785db58 W
000652d8 R
0005df58 W
000652e0 R
0785db50 W
000652e0 R
31308800 R
00062368 R
```

Os arquivos fornecidos representam o registro de todas as operações de acesso à memória observadas durante a execução real de alguns programas considerados significativos de classes de programas reais:

- **compilador.log** compilador.zip: execução de um compilador, que normalmente utiliza um grande número de estruturas de dados internas complexas;
- **matriz.log** matriz.zip: um programa científico que utiliza cálculos matriciais relativamente simples, mas sobre grandes matrizes e vetores;
- **compressor.log** compressor.zip: um programa de compressão de arquivos, que usa estruturas de dados mais simples;
- **simulador.log** simulador.zip: um simulador de partículas, que executa cálculos complexos sobre estruturas relativamente simples.

Todos os arquivos estão compactados (.zip) e devem ser descompactados antes de serem usados. A leitura do arquivo pode ser feita com o comando `scanf()`, como no trecho a seguir:

```
unsigned addr;
char rw;
    ...
fscanf(file,"%x %c",&addr,&rw);
```

Determinação da página a partir do endereço

Como visto em sala, para se identificar a página associada a um endereço, basta descartar os s bits menos significativos do endereço. Se a página fosse fixada em 4 KB, s seria sempre 12. Entretanto, para se implementar um tamanho de página variável, s deve ser calculado a cada execução. Para simplificar, vamos assumir que o tamanho da página será sempre fornecido como uma potência de 2 (não é necessário checar). O código para se determinar s pode ser:

```
unsigned s, page_size, tmp;

/* Derivar o valor de s: */
tmp = page_size;
s = 0;
while (tmp>1) {
    tmp = tmp>>1;
    s++;
}
```

Nesse caso, para um endereço `addr`, a página pode ser determinada simplesmente fazendo-se `page = addr >> s;`

Implementação da tabela de páginas

Como os endereços são de 32 bits nos arquivos fornecidos, para páginas de 2 KB (as menores que precisam ser consideradas), podemos ter até 21 bits de identificação de página, isto é, uma tabela de página de mais de dois milhões de entradas. Se cada entrada da tabela tiver um inteiro para identificar a página

física, teremos um vetor de 8 MB. Não será muito eficiente, mas para fins do simulador, essa organização é aceitável. Vocês não devem se preocupar em montar uma tabela de páginas hierárquica, como seria realmente implementada na prática; isso geraria uma complexidade adicional que não seria muito útil. Vocês podem também implementar uma tabela de páginas reversa, ou uma tabela por *hash*, caso se sintam inspirados.

A estrutura de dados para representar cada quadro físico deve conter campos para registrar atributos como página referenciada, instante do último acesso, página alterada, etc. (os detalhes são parte da implementação e vão depender da forma como vocês implementarem cada algoritmo).

Implementação dos algoritmos de reposição

Os detalhes de outras estruturas de dados que podem vir a ser usadas para os algoritmos são de livre escolha dos implementadores. Vocês devem documentar no relatório como cada algoritmo foi implementado e certificar-se de que o desempenho final do simulador não seja demasiado lento (preferivelmente na ordem de segundos, não dezenas de minutos).

Vocês podem utilizar um contador que inicie em zero e seja incrementado a cada acesso à memória, como a forma de manter o tempo de cada acesso/leitura/escrita, quando necessário.

Para a implementação da política aleatória, as funções `random()` e `srandom()` podem ser usadas para se controlar o gerador de números aleatórios. Para se gerar um número entre 0 e n, a expressão `(random() % n)` é suficiente. Note-se que enquanto houver páginas vazias na memória elas devem ser preenchidas; apenas quando a memória estiver completamente ocupada uma página aleatória deve ser substituída.

Verificação do funcionamento do programa

Apesar da versão a ser entregue precisar gerar apenas o relatório final descrito anteriormente, recomenda-se fortemente que o programa tenha um modo "depuração" onde ele escreva uma linha (ou mais) descrevendo o que é feito a cada acesso à memória. Assim, utilizando-se um arquivo de teste reduzido (e possivelmente escrito especialmente para testar cada caso de operação) você pode acompanhar a operação do programa passo-a-passo. Para isso, é permitido prever um quinto parâmetro, opcional, que seja definido quando se deseje que o programa tenha esse comportamento com saída detalhada. A forma desse parâmetro é de escolha dos desenvolvedores. Pode ser uma palavra, como `debug`, pode ser qualquer coisa (apenas para configurar um quinto argumento), ou pode ser um número, que pode ser usado para especificar um grau de detalhamento (números maiores geram mensagens de depuração mais detalhadas, por exemplo).

O que deve ser entregue

Você deve entregar no moodle um arquivo .zip ou .tgz contendo o(s) arquivo(s) contendo o código fonte do programa (.c e .h), um `Makefile` e um relatório sobre o seu trabalho. **Não inclua os arquivos de teste no processo de entrega!**

O relatório deve conter:

- Um resumo do projeto: alguns parágrafos que descrevam a estrutura geral do seu código e todas as estruturas importantes.
- Decisões de projeto: descreva como você lidou com quaisquer ambiguidades na especificação. Por exemplo, para este projeto, explicar como seu interpretador lida com linhas que não têm comandos, apenas manipulação de arquivos.
- Uma análise do desempenho dos algoritmos de substituição de páginas para os vários programas utilizados.

Essa análise de desempenho é uma parte importante do trabalho e será responsável por uma fração significativa da nota (+/- 40%). Em diversos momentos precisamos comparar algoritmos, determinar o que esperar em diferentes condições. Seu relatório deve avaliar o comportamento dos algoritmos de reposição de página para os quatro programas, em duas situações: quando o tamanho da memória cresce, com páginas de 4 KB, e quando o tamanho da memória fica constante, mas o tamanho das páginas varia de 2 KB a 64 KB (em potências de 2).

Finalmente, embora você possa desenvolver o seu código em qualquer sistema que quiser, certifique-se que ele execute corretamente com o sistema operacional Linux (Ubuntu).

Considerações finais

1. **Dúvidas:** usem o moodle (minha.ufmg).
2. Comece a fazer o trabalho logo, pois apesar do programa final ser relativamente pequeno, o tempo não é muito e o prazo de entrega não vai ficar maior do que ele é hoje (independente de que dia é hoje).
3. Vão valer pontos clareza, qualidade do código e da documentação e, obviamente, a execução correta da chamada do sistema com programas de teste. A participação nos fóruns de forma positiva também será considerada.

Última alteração: 24 de maio de 2023