Relatório do Projeto 2: Simulador de Memória Virtual

Disciplina: Sistemas Operacionais **Professor:** Lucas Figueiredo **Data:** 6/11/2025

Integrantes do Grupo

- João Vitor de Araújo Trindade - 10403255

---

1. Instruções de Compilação e Execução

1.1 Compilação

Para compilar o projeto, execute o seguinte comando no diretório do projeto:

```
gcc -o simulador simulador.c
```

Este comando irá gerar o executável chamado `simulador` a partir do arquivo fonte `simulador.c`.

1.2 Execução

O simulador recebe três argumentos: o algoritmo de substituição, o arquivo de configuração e o arquivo de acessos.

Sintaxe geral:

```
./simulador <algoritmo> <arquivo_config> <arquivo_acessos>
```

Exemplo com FIFO:

```
./simulador fifo tests/config_1.txt tests/acessos_1.txt
```

Exemplo com Clock:

```
./simulador clock tests/config_1.txt tests/acessos_1.txt
```

Parâmetros: - `<algoritmo>`: “fifo” para algoritmo FIFO ou “clock” para algoritmo Clock - `<arquivo_config>`: arquivo contendo configurações do sistema (número de frames, tamanho da página, etc.) - `<arquivo_acessos>`: arquivo contendo a sequência de acessos à memória a serem simulados

---

2. Decisões de Design

2.1 Estruturas de Dados

Descreva as estruturas de dados que você escolheu para representar:

Tabela de Páginas: - Qual estrutura usou? (array, lista, hash map, etc.) A estrutura usada foi array - Quais informações armazena para cada página? Para cada pagina guardamos, o frame no qual essa pagina esta carregada, o valid_bit (se a pagina esta alocada ou nao na memoria fisica), e o r_bit (para verificar o quão recentemente foi acessada). - Como organizou para múltiplos processos? Utilizei uma struct para definir cada pagina, e construi um array dessa struct, nesse array temos todas paginas de um processo (array dinamico, alocado com malloc). E então criei um array com esses arrays de struct, tendo um array de arrays para cada processo. Assim na nossa table temos que `table[process][page]`. - **Justificativa:** Por que escolheu essa abordagem? Achei a forma mais simples de conseguir implementar, sem precisar lidar com estruturas de dados mais complexas, dessa forma tudo que preciso fazer é alocar memória dinamicamente. Além desse método ser um dos melhores em questão de performance.

Frames Físicos: - Como representou os frames da memória física? Representei como uma struct Frame que tem as seguintes informações: se o frame esta cheio, o tamanho total do frame, a posição em que deve ser acessada agora no frame, e um array com todos os espacos do frame. - Quais informações armazena para cada frame? Armazeno a pagina e o processo alocados ali. - Como rastreia frames livres vs ocupados? Utilizo de um array, e vou alocando do elemento 0 ate size-1, mantenho sempre o endereco da proxima posição a visitar dentro de curr_pos, então não volto onde já esta ocupado a não ser que o array inteiro já esteja ocupado. - **Justificativa:** Por que escolheu essa abordagem? Acreditei ser a mais simples de se implementar e bastante funcional, tanto que o desenvolvimento de todos os algoritimos foi bem simples, devido a esse método ja lever em consideração a ordem de chegada dos elementos.

Estrutura para FIFO: - Como mantém a ordem de chegada das páginas? Mantém a ordem através do frame->curr_pos, que aponta sempre para a posição da próxima página a ser substituída. A ordem é mantida pelo fato de preenchermos os frames sequencialmente e utilizarmos uma lógica circular. - Como identifica a página mais antiga? A página mais antiga está sempre na posição indicada por frame->curr_pos, já que preenchemos sequencialmente e utilizamos uma abordagem circular (quando chega no final, volta para o início). - **Justificativa:** Por que escolheu essa abordagem? É simples, não requer estruturas de dados adicionais como filas ou listas. O próprio array de frames já mantém a ordem de chegada através do ponteiro circular.

Estrutura para Clock: - Como implementou o ponteiro circular? Utilizo o mesmo frame->curr_pos do FIFO, mas agora o ponteiro percorre os frames procurando páginas com R-bit = 0. Quando chega no final do array (frame->size), volta para a posição 0. - Como armazena e atualiza os R-bits? Os R-bits estão armazenados na tabela de páginas (table[process][page].r_bit). São setados para 1 a cada acesso à página e zerados quando o algoritmo Clock passa por eles durante a busca por uma vítima. - **Justificativa:** Por que escolheu essa abordagem? Reutiliza a mesma estrutura do FIFO, apenas mudando a lógica

de seleção da vítima. Os R-bits na tabela de páginas facilitam a verificação e atualização durante os acessos.

2.2 Organização do Código

Descreva como organizou seu código:

- Quantos arquivos/módulos criou? Criei apenas um arquivo: simulador.c. Optei por manter tudo em um único módulo pois o projeto é de escopo limitado e facilita a compilação.
- Qual a responsabilidade de cada arquivo/módulo? O simulador.c é responsável por toda a lógica do simulador: parsing dos arquivos, gerenciamento de memória, algoritmos de substituição e controle da simulação.
- Quais são as principais funções e o que cada uma faz?

```
simulador.c
  main() - coordena execução, lê argumentos, abre arquivos e chama funções principais
  parse_config() - processa arquivo de configuração e extrai parâmetros do sistema
  parse_acess() - processa arquivo de acessos e extrai sequência de acessos à memória
  create_table_processes() - cria e inicializa tabelas de páginas para todos os processos
  translate_virtual_adress() - loop principal da simulação, traduz endereços e gerencia ace
  check_table() - verifica se página está na memória (HIT, MISS ou MISS_MEMORY_FULL)
  hit() - processa acesso quando página já está na memória
  miss() - processa page fault quando há frame livre disponível
  fifo() - implementa algoritmo FIFO para seleção de vítima
  clock() - implementa algoritmo Clock para seleção de vítima
  free_table_processes() - libera memória alocada para as tabelas de páginas
```

2.3 Algoritmo FIFO

Explique **como** implementou a lógica FIFO:

- Como mantém o controle da ordem de chegada? Utilizo um ponteiro circular (frame->curr_pos) que sempre aponta para o próximo frame a ser utilizado. Durante a fase de preenchimento inicial da memória, este ponteiro avança sequencialmente. Quando a memória fica cheia, o ponteiro continua avançando circularmente, garantindo que sempre aponte para a página mais antiga.
- Como seleciona a página vítima? A página vítima é sempre aquela localizada no frame apontado por frame->curr_pos. Como este ponteiro avança sequencialmente e de forma circular, ele sempre aponta para a página que chegou primeiro (mais antiga) entre todas as páginas presentes na memória.
- Quais passos executa ao substituir uma página? Primeiro, identifica qual processo e página estão atualmente no frame indicado por curr_pos. Em seguida, atualiza a tabela de páginas da vítima (zera valid_bit, r_bit

e coloca frame para -1). Depois, aloca a nova página no mesmo frame, atualizando sua entrada na tabela de páginas (seta valid_bit e r_bit para 1, e altera o frame para a posicao do frame atual). Por fim, atualiza as informações do frame com o novo processo e página, e avança o ponteiro curr_pos para a próxima posição (retornando para 0 se ultrapassar o tamanho da memória).

Não cole código aqui. Explique a lógica em linguagem natural.

2.4 Algoritmo Clock

Explique **como** implementou a lógica Clock:

- Como gerencia o ponteiro circular? Utilizo o mesmo ponteiro circular do FIFO (frame->curr_pos), mas com comportamento diferente. O ponteiro percorre os frames sequencialmente procurando uma vítima adequada. Quando chega no final do array (frame->size), automaticamente retorna para a posição 0, criando um movimento circular contínuo.
- Como implementou a "segunda chance"? A segunda chance é implementada através de um loop que verifica o R-bit de cada página. Se a página no frame atual tem R-bit = 1, zero o bit e avanço para o próximo frame. Se encontro uma página com R-bit = 0, ela é selecionada como vítima e substituída imediatamente.
- Como trata o caso onde todas as páginas têm R=1? O algoritmo continua percorrendo todos os frames, zerando os R-bits de todas as páginas até dar uma volta completa. Na segunda passada, todas as páginas terão R-bit = 0, então a primeira página encontrada será escolhida como vítima. Isso garante que sempre haverá uma vítima disponível.
- Como garante que o R-bit é setado em todo acesso? Em todo acesso à memória (seja HIT ou durante a alocação de uma nova página), seto o R-bit para 1 na função hit() e nas funções miss() e clock(). Isso acontece independente do algoritmo utilizado, garantindo que páginas recentemente acessadas tenham seu bit de referência marcado.

Não cole código aqui. Explique a lógica em linguagem natural.

2.5 Tratamento de Page Fault

Explique como seu código distingue e trata os dois cenários:

Cenário 1: Frame livre disponível - Como identifica que há frame livre? Utilizo a flag frame->full que indica se a memória física está completamente ocupada. Quando esta flag é 0 (false), significa que ainda há frames livres disponíveis. Além disso, frame->curr_pos aponta para o próximo frame livre a ser utilizado.

- Quais passos executa para alocar a página? Primeiro, atualiza a tabela de páginas setando frame, valid_bit e r_bit para os valores corretos. Em seguida, armazena no frame as informações do processo e página alocados. Depois, avança o ponteiro curr_pos para a próxima posição livre. Se o ponteiro alcançar o final da memória (frame->size), marca a flag full como 1 e reseta curr_pos para 0, indicando que a memória agora está cheia.

Cenário 2: Memória cheia (substituição) - Como identifica que a memória está cheia? Verifico a flag frame->full. Quando ela está setada para 1, significa que todos os frames estão ocupados e será necessário fazer substituição de páginas para alocar uma nova página.

- Como decide qual algoritmo usar (FIFO vs Clock)? A decisão é feita através do parâmetro algorithm passado para a função translate_virtual_adress. Este valor é definido na main() baseado no argumento da linha de comando (argv[1]). Se for "fifo", usa FIFO; se for "clock", usa Clock.
- Quais passos executa para substituir uma página? Chama a função apropriada (fifo() ou clock()) passando todos os parâmetros necessários. Ambas as funções seguem a mesma lógica geral, identificam a página vítima, atualizam a tabela de páginas da vítima (zerando valid_bit e r_bit), alocam a nova página no frame liberado, atualizam a tabela da nova página e avançam o ponteiro curr_pos conforme a política do algoritmo.

---

3. Análise Comparativa FIFO vs Clock

3.1 Resultados dos Testes

Preencha a tabela abaixo com os resultados de pelo menos 3 testes diferentes:

Descrição do Teste	Total de Acessos	Page Faults FIFO	Page Faults Clock	Diferença
Teste 1 - Básico	8	5	5	0
Teste 2 - Memória Pequena	10	10	10	0
Teste 3 - Simples	7	4	4	0
Teste 5 - Teste Próprio	150	78	77	1

3.2 Análise

Com base nos resultados acima, responda:

1. **Qual algoritmo teve melhor desempenho (menos page faults)?**

Clock teve melhor desempenho, com 77 page faults contra 78 do FIFO no teste maior. Nos testes menores, ambos tiveram resultados iguais.

2. **Por que você acha que isso aconteceu?** Considere:
   - Como cada algoritmo escolhe a vítima
   - O papel do R-bit no Clock
   - O padrão de acesso dos testes

Clock teve melhor resultado porque usa o R-bit para dar segunda chance a páginas recentemente acessadas, evitando substituir páginas que podem ser reutilizadas em breve. FIFO substitui sempre a página mais antiga, independente de uso recente.

3. **Em que situações Clock é melhor que FIFO?**
   - Dê exemplos de padrões de acesso onde Clock se beneficia

Clock é melhor em padrões com localidade temporal: loops que acessam as mesmas páginas repetidamente, processamento de matrizes em múltiplas passadas, programas que reutilizam estruturas de dados.

4. **Houve casos onde FIFO e Clock tiveram o mesmo resultado?**
   - Por que isso aconteceu?

Sim, nos testes 1, 2 e 3. Isso aconteceu porque os testes eram pequenos e com acessos principalmente sequenciais, sem oportunidade para mostrar a vantagem do mecanismo de segunda chance.

5. **Qual algoritmo você escolheria para um sistema real e por quê?**

Clock, porque aproxima melhor o comportamento LRU com baixo overhead, adapta-se a diferentes padrões de acesso e é amplamente usado em sistemas reais.

---

4. Desafios e Aprendizados

4.1 Maior Desafio Técnico

O maior desafio técnico enfrentado foi na implementação do parsing dos dados, onde cometi alguns erros de alocação de memória:

- **Qual foi o problema?** No início da implementação, tive dificuldades para fazer o parsing correto dos arquivos de configuração e acessos. Cometi alguns erros de alocação de memória, especialmente ao tentar alocar espaço para os arrays dinâmicos que armazenam os dados dos processos e acessos. Além disso esqueci de iniciar alguns elementos, o que também estava dando problema.

- **Como identifiquei o problema?** O programa estava apresentando comportamentos estranhos, às vezes funcionando e às vezes crashando. Através de debugging e testes com diferentes arquivos de entrada, percebi que o problema estava relacionado ao gerenciamento de memória durante o parsing.
- **Como resolvi?** Demorei um pouco para resolver, mas consegui corrigir verificando cuidadosamente cada malloc() e garantindo que estava alocando o tamanho correto para cada estrutura. Também implementei verificações de erro para cada alocação, garantindo que o programa termine graciosamente se não conseguir alocar memória.
- **O que aprendi com isso?** Aprendi a importância de ser mais cuidadoso com o gerenciamento de memória em C, sempre verificando se as alocações foram bem-sucedidas e calculando corretamente o tamanho necessário. Também percebi como o debugging sistemático é fundamental para identificar problemas de memória.

4.2 Principal Aprendizado

O principal aprendizado sobre gerenciamento de memória que tive com este projeto foi entender bem a função da tabela de páginas para tradução do endereço virtual para o físico:

- **O que não entendia bem antes e agora entendo?** Antes do projeto, a função da tabela de páginas estava meio confusa para mim. Agora compreendo claramente como ela serve como uma ponte entre o espaço de endereçamento virtual (que o processo vê) e o espaço físico (onde os dados realmente estão armazenados). Implementar a tradução me mostrou como cada página virtual é mapeada para um frame físico específico.
- **Como este projeto mudou minha compreensão de memória virtual?** O projeto me fez perceber como a memória virtual é uma abstração poderosa que permite aos processos "enxergarem" um espaço de memória contíguo, mesmo quando suas páginas estão espalhadas na memória física. A tabela de páginas é o mecanismo que mantém essa ilusão funcionando, guardando onde cada página virtual realmente está localizada.
- **Que conceito das aulas ficou mais claro após a implementação?** O conceito de tradução de endereços ficou muito mais claro. Implementar o cálculo do número da página (endereço / tamanho_página) e do deslocamento (endereço % tamanho_página) me mostrou na prática como um endereço virtual é decomposto e depois reconstruído usando as informações da tabela de páginas.

---

5. Vídeo de Demonstração

Link do vídeo: [https://drive.google.com/file/d/1DhXwRLpdJEx3o-OKMCl_V5TZnCUfsBHQ/view?usp=sharing]

Conteúdo do vídeo:

Confirme que o vídeo contém:

- ☒ Demonstração da compilação do projeto
- ☒ Execução do simulador com algoritmo FIFO
- ☒ Execução do simulador com algoritmo Clock
- ☒ Explicação da saída produzida
- ☒ Comparação dos resultados FIFO vs Clock
- ☒ Breve explicação de uma decisão de design importante

Checklist de Entrega

Antes de submeter, verifique:

- ☒ Código compila sem erros conforme instruções da seção 1.1
- ☒ Simulador funciona corretamente com FIFO
- ☒ Simulador funciona corretamente com Clock
- ☒ Formato de saída segue EXATAMENTE a especificação do ENUNCIADO.md
- ☒ Testamos com os casos fornecidos em tests/
- ☒ Todas as seções deste relatório foram preenchidas
- ☒ Análise comparativa foi realizada com dados reais
- ☒ Vídeo de demonstração foi gravado e link está funcionando
- ☒ Todos os integrantes participaram e concordam com a submissão

---

Referências

Liste aqui quaisquer referências que utilizaram para auxiliar na implementação (livros, artigos, sites, **links para conversas com IAs.**)

Utilizei do Claude Code para me auxiliar a documentar e comentar as funções do meu codigo.
