Universidade Federal de Lavras
PPGCC
PCC508 – Sistemas Operacionais

Tópico 4 Lista Avaliativa

Douglas Aquino T. Mendes
12 de novembro de 2024

Sumário

1 Introdução

Este documento tem como objetivo apresentar o desenvolvimento das atividades avaliativas para o tópico 4 da disciplina de Sistemas Operacionais, focando na implementação de códigos em linguagem C. Serão apresentadas as questões, a resolução, os códigos desenvolvidos, seguidos de uma explicação sobre sua lógica de funcionamento.

2 Questões

2.1 Memória virtual

Pergunta: 1) Explique como funciona a memória virtual, juntamente com a paginação.

Resposta:A memória virtual é um mecanismo de gerenciamento de memória que permite que os sistemas operacionais utilizem mais memória do que a fisicamente disponível no hardware (RAM). Ela faz isso mapeando endereços virtuais usados pelos programas em endereços físicos na RAM. Quando um programa tenta acessar um endereço de memória que não está na RAM, o sistema operacional interrompe o processo e carrega os dados necessários da memória secundária (geralmente o disco rígido ou SSD) para a RAM. Esse processo é chamado de page fault. **A paginação** é uma técnica de implementação da memória virtual que divide tanto a memória virtual quanto a memória física em blocos de tamanho fixo chamados páginas e molduras de página (ou frames). Cada página virtual de um processo pode ser mapeada para uma moldura de página na RAM [1], [2].

- A memória virtual de um processo é dividida em páginas de tamanho fixo.
- Cada processo tem sua própria tabela de páginas, que mapeia as páginas virtuais para as molduras físicas na RAM. Essa tabela contém informações sobre onde uma página virtual está armazenada na memória física.
- Quando um programa acessa um endereço de memória virtual, a Unidade de Gestão de Memória (MMU) traduz esse endereço em um endereço físico usando a tabela de páginas.
- Se a página que está sendo acessada não está na RAM, ocorre um page fault. O sistema operacional pausa a execução do processo, busca a página necessária no disco e a carrega em uma moldura livre da RAM. Se não houver molduras livres, uma página existente é removida (muitas vezes usando algoritmos como LRU – Least Recently Used), e a nova página é carregada.

2.2 Tradução de um endereço virtual

Pergunta: 2) Explique em detalhes como funciona a tradução de um endereço virtual para o endereço físico em um sistema com memória virtual e paginação. Use um exemplo e considere que o endereço virtual tem mais bits que o endereço físico. Isso é possível? Explique também qual a função da MMU.

Resposta:A memória virtual é dividida em páginas de tamanho fixo, e a memória física é dividida em molduras de página de mesmo tamanho. A tradução do endereço virtual para o endereço físico envolve o uso de uma tabela de páginas que mapeia cada página virtual para uma moldura física.

Um endereço virtual pode ser dividido em duas partes, Número da página virtual e Deslocamento (Indica a posição específica dentro da página). Um endereço físico também é dividido em duas partes, Número da moldura física e Deslocamento. Considerando um exemplo onde o endereço virtual tem 16 bits, o endereço físico tem 14 bits e o tamanho de cada página é de 1 KB, então o deslocamento dentro da página tem 10 bits, os bits restantes do endereço virtual (16 - 10 = 6 bits) representam o número da página virtual. Da mesma forma, os bits restantes do endereço físico (14 - 10 = 4 bits) representam o número da moldura física [1], [3], [4].

Exemplo de Tradução de Endereço

Considere um endereço virtual de 16 bits: 0011 1010 0101 1100.

- **Número da página virtual (VPN):** Os 6 bits mais significativos → 001110.
- **Deslocamento:** Os 10 bits menos significativos → 101011100.

Para traduzir esse endereço, a MMU faz o seguinte:

- **Consulta a tabela de páginas** usando a VPN 001110 para encontrar a moldura de página correspondente.
- Substitui a VPN pelo PFN obtido na tabela de páginas, mantendo o deslocamento inalterado.
- Suponha que a tabela de páginas indique que a VPN "001110"corresponde à PFN "0101". Então, o endereço físico resultante seria **0101** 1010 1110 1100.

A MMU é o componente de hardware responsável por:

- Converter endereços virtuais em endereços físicos em tempo real.
- Verificar permissões (leitura, escrita, execução) para garantir que os processos não acessem memória de forma indevida.
- Se a MMU não encontrar uma entrada válida na tabela de páginas (indicando que a página não está na RAM), ela aciona um page fault e chama o sistema operacional para carregar a página da memória secundária.

2.3 Tabela de páginas

Pergunta: 3) Explique a estrutura de uma tabela de páginas.

Resposta: A tabela de páginas é uma estrutura de dados que implementam memória virtual e paginação. Sua função é mapear endereços virtuais (utilizados pelos processos) para endereços físicos (localizados na RAM). A tabela de páginas é composta por entradas, e cada entrada da tabela de páginas (Page Table Entry - PTE) contém informações sobre o mapeamento de uma página virtual para uma moldura física. Alguns componentes comuns de uma entrada de Tabela de Páginas são[1], [5]:

- Número da moldura física.
- Bit de presença.
- Bits de controle de acesso.
- Bit de modificado.
- Bit de referenciado.

2.4 TLB

Pergunta: 4) Explique o que é, para que serve e como funciona uma TLB (translation lookaside buffer).

Resposta: A TLB é uma pequena, mas muito rápida, cache que armazena pares de mapeamentos de endereços virtuais e físicos. Como as operações de busca e tradução de endereços em tabelas de páginas na RAM podem ser demoradas, a TLB melhora o desempenho armazenando as traduções mais recentes. Se a MMU encontrar o mapeamento desejado na TLB, a tradução ocorre instantaneamente. A TLB é usada para Aumentar a velocidade de tradução de endereços, diminuindo o tempo de acesso à memória e evitando acessos repetidos à tabela de páginas armazenada na RAM [1], [6].

3 Desenvolver um programa

3.1 Compartilhar uma área de memória criada com a chamada mmap

Enunciado: Deve ser desenvolvido um programa que cria dois processos filhos. Esses processos devem compartilhar uma área de memória criada com a chamada mmap. Vamos agora utilizar essa área como um buffer compartilhado. Um dos processos será um produtor e deve gerar uma letra aleatória em cada intervalo de tempo (configurável) e adicionar ao buffer. O outro processo deve consumir esses itens (letras), uma a cada intervalo de tempo (configurável) e imprimir o item na tela. Não esquecer o problema da condição de corrida. O processo pai não executa nenhuma tarefa, fica bloqueado esperando os processos filhos.

3.2 Código

```c
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <time.h>
#include <semaphore.h>
#include <fcntl.h>

#define BUFFER_SIZE 10 // Tamanho do buffer compartilhado

typedef struct
{
    char buffer[BUFFER_SIZE];
    int in;
    int out;
} shared_memory;

int main()
{

    shared_memory *shared = mmap(NULL, sizeof(shared_memory),
                                 PROT_READ | PROT_WRITE, MAP_SHARED |
                                     MAP_ANONYMOUS, -1, 0);
    if (shared == MAP_FAILED)
    {
        perror("mmap");
        exit(1);
    }

    shared->in = 0;
    shared->out = 0;

    sem_t *mutex = sem_open("/mute", O_CREAT | O_EXCL, 0644, 1);

    if (mutex == SEM_FAILED)
    {
        perror("sem_open");
```

```
40          exit(1);
41      }
42
43      pid_t producer_pid = fork();
44      if (producer_pid == 0)
45      {
46
47          srand(time(NULL));
48          while (1)
49          {
50              sleep(2);
51
52              char letter = 'A' + (rand() % 26);
53
54              sem_wait(mutex);
55
56              shared->buffer[shared->in] = letter;
57              printf("Produtor: letra %c adicionada na entrada %d\n", letter,
                    shared->in);
58              shared->in = (shared->in + 1) % BUFFER_SIZE;
59
60              sem_post(mutex);
61          }
62      }
63
64      pid_t consumer_pid = fork();
65      if (consumer_pid == 0)
66      {
67
68          while (1)
69          {
70              sleep(3);
71
72              sem_wait(mutex);
73
74              char letter = shared->buffer[shared->out];
75              printf("Consumidor: letra %c consumida da entrada %d\n", letter,
                    shared->out);
76              shared->out = (shared->out + 1) % BUFFER_SIZE;
77
78              sem_post(mutex);
79          }
80      }
81
82      // Processo pai
83      wait(NULL);
84      wait(NULL);
85
86      // Limpeza
87      sem_unlink("/empty");
88      sem_unlink("/full");
89      sem_unlink("/mutex");
90      sem_close(mutex);
91      munmap(shared, sizeof(shared_memory));
```

```
92
93      return 0;
94  }
```

3.3 Testes e Resultados

Como resultado da execução do código exibido na subceção 3.2, obtivemos a saída ilustrada na figura 1. É importante ressaltar que o programa pode apresentar um problema, pois não há verificação de produzir apenas quando há espaço, ou seja, o buffer pode acabar sendo subscrito pela função de produzir, antes da função de consumir.

```
douglas@Virtuoso:~/Documentos/Projetos/Disciplinas/Mestrado_SO/Topic4/Avaliativa/Codes$ ./atvi
Produtor: letra A adicionada na entrada 0
Consumidor: letra A consumida da entrada 0
Produtor: letra D adicionada na entrada 1
Consumidor: letra D consumida da entrada 1
Produtor: letra X adicionada na entrada 2
Produtor: letra N adicionada na entrada 3
Consumidor: letra X consumida da entrada 2
Produtor: letra K adicionada na entrada 4
Consumidor: letra N consumida da entrada 3
Produtor: letra O adicionada na entrada 5
Produtor: letra W adicionada na entrada 6
Consumidor: letra K consumida da entrada 4
Produtor: letra A adicionada na entrada 7
Consumidor: letra O consumida da entrada 5
Produtor: letra E adicionada na entrada 8
Produtor: letra H adicionada na entrada 9
Consumidor: letra W consumida da entrada 6
Produtor: letra L adicionada na entrada 0
Consumidor: letra A consumida da entrada 7
Produtor: letra T adicionada na entrada 1
Produtor: letra Q adicionada na entrada 2
Consumidor: letra E consumida da entrada 8
Produtor: letra U adicionada na entrada 3
Consumidor: letra H consumida da entrada 9
Produtor: letra J adicionada na entrada 4
```

Figura 1: Resultado da execução do programa

Referências

[1] A. S. Tanenbaum e H. Bos, *SISTEMAS OPERACIONAIS MODERNOS*, 4ª ed. Boston: Pearson, 2021.

[2] Escola LBK. "O que é Memória Virtual." Acesso em: 12 nov. 2024. (2024), endereço: https://escolalbk.com.br/glossario/o-que-e-memoria-virtual/.

[3] Resumos LEIC-A. "Memória Virtual." Acesso em: 12 nov. 2024. (), endereço: https://resumos.leic.pt/oc/memoria-virtual/.

[4] R. Gomes, *Aula 18 - Sistemas Operacionais: Memória Virtual*, Acesso em: 12 nov. 2024, 2024. endereço: http://www.inf.ufes.br/~rgomes/so_fichiers/aula18.pdf.

[5] P. C. A. M. Lima, *Gerenciamento de Memória*, Acesso em: 12 nov. 2024, 2012. endereço: https://thewaysx.wordpress.com/wp-content/uploads/2012/11/aula131.pdf.

[6] TechTarget. "Translation Lookaside Buffer (TLB)." Acesso em: 12 nov. 2024. (), endereço: https://www.techtarget.com/whatis/definition/translation-look-aside-buffer-TLB.