UNIVERSIDADE FEDERAL DE VIÇOSA
Sistemas de Informação

Stefani Kaline Leonel Dias 4702

IMPLEMENTAÇÃO DE UM GERENCIADOR DE MEMÓRIA

SIN 351 - Sistemas Operacionais

Rio Paranaíba-MG
2021

SUMÁRIO

Objetivo

Projeto proposto pelo professor Rodrigo Moreira na disciplina de Sistemas Operacionais - SIN351 da *Universidade Federal de Viçosa - Campus Rio Paranaíba (UFV-CRP)*. Seu objetivo é o desenvolvimento de algoritmos de substituição de páginas, complementando assim o Gerenciador de Memória disponibilizado pelo mesmo e por fim comparar a eficiência destes algoritmos após dez execuções distintas.

Algoritmos de Substituição de Página

- FIFO (*First in, First Out*): O algoritmo parte do princípio que páginas que foram recentemente usadas deverão ser consultadas novamente em breve e devem permanecer na memória. Desta maneira, segundo Tanenbaum (2003), o sistema operacional mantém uma fila com páginas atuais na memória, organizadas da forma que as mais antigas fiquem no início e consequentemente páginas que chegaram recentemente no fim da mesma. Quando ocorre uma falta de página, a página presente no início da fila é removida;

```
int fifo(int8_t** page_table, int num_pages, int prev_page,
         int fifo_frm, int num_frames, int clock) {
    for (int page = 0; page < num_pages; page++) {//percorre a tabela de páginas a procura da página que tem o endereço físico
        if (page_table[page][PT_FRAMEID] == fifo_frm) {//correspondente a fifo_frm
            return page;
        }
    }
}
```

Figura 1 - Algoritmo FIFO

- FIFO + Bit R (Segunda Chance): Trata-se de uma adaptação do algoritmo FIFO que impede que páginas intensamente usadas sejam descartadas. Neste algoritmo o bit R da primeira página é verificado, se o mesmo for zero a página é removida, se não, o bit R é limpo, seu tempo de carregamento é alterado de forma que parece que a mesma tenha acabado de chegar a memória e por fim a página é colocada no fim da fila e a busca prossegue.

```
int second_chance(int8_t** page_table, int num_pages, int prev_page,
                  int fifo_frm, int num_frames, int clock) {
    for (int frame = 0;frame < num_frames; frame++) {
            for (int page = 0; page < num_pages; page++) {
                if (page_table[page][PT_FRAMEID] == frame) {
                    if (page_table[page][PT_REFERENCE_BIT] == 1) {
                        page_table[page][PT_REFERENCE_BIT] = 0; //seta com 0 o bit de referência dando uma segunda chance a página
                        prev_page = page; //coloca a página no fim da fila
                        break;
                    } else {
                        return page; //retorna a página encontrada
                    }
                }
            }
    }
    //Caso todas as páginas tenham sido referenciadas o algoritmo de Segunda Chance tornasse FIFO puro
    return fifo(page_table, num_pages,prev_page, fifo_frm, num_frames, clock);
}
```

Figura 2 - Algoritmo FIFO + bit R (Segunda Chance)

- NRU (Não usada recentemente): Considera a existência dos bits de status *R* e *M* associados a cada página. O bit *R* informa se a página foi referenciada e o bit *M* por sua vez, indica se esta foi modificada. O sistema operacional então divide as páginas em quatro categorias:

  Classe 0: páginas não referenciadas e não modificadas;
  Classe 1: páginas não referenciadas e modificadas;
  Classe 2: páginas referenciadas e não modificadas;
  Classe 3: páginas referenciadas e modificadas;

Quando ocorre uma falta de página, o algoritmo remove uma página aleatória de sua classe mais baixa.

```
int nru(int8_t** page_table, int num_pages, int prev_page,
        int fifo_frm, int num_frames, int clock) {
    //Busca páginas da Classe 0
    for (int page = 0; page < num_pages; page++) {
        if ((page_table[page][PT_REFERENCE_BIT] == 0) && (page_table[page][PT_DIRTY] == 0)) {
            if (page_table[page][PT_MAPPED] != 0){
                return page;
                }
        }
    }
    //Busca páginas da Classe 1
    for (int page = 0; page < num_pages; page++) {
        if ((page_table[page][PT_REFERENCE_BIT] == 0) && (page_table[page][PT_DIRTY] == 1)) {
            if (page_table[page][PT_MAPPED] != 0 ){
                return page;
                }
        }
    }
    //Busca páginas da Classe 2
    for (int page = 0; page < num_pages; page++) {
        if ((page_table[page][PT_REFERENCE_BIT] == 1) && (page_table[page][PT_DIRTY] == 0)) {
            if (page_table[page][PT_MAPPED] != 0) {
                return page;
            }
        }
    }

     //Busca páginas da Classe 3
     for (int page = 0; page < num_pages; page++) {
         if ((page_table[page][PT_REFERENCE_BIT] == 1) && (page_table[page][PT_DIRTY] == 1)) {
             if (page_table[page][PT_MAPPED] != 0) {
                 return page;
             }
         }
     }

}
```

Figura 3 - Algoritmo NRU

- *Aging* (Envelhecimento): Trata-se de uma adaptação do algoritmo NFU (Not Recently Used), o qual para cada página existe um contador que controla a frequência com que a página é acessada. Nesse contexto, a cada interrupção do relógio o bit *R* é adicionado ao contador e durante uma falta de página, a página que possuir o menor contador é removida. O problema deste algoritmo é que o contador não é resetado, dessa forma páginas que foram intensamente usadas no início podem possuir um contador maior que páginas que foram acessadas recentemente.
  Para contornar este problema no algoritmo Aging os contadores são movidos um bit à direita antes que o bit *R* seja acrescentado. Além disso, o bit *R* é adicionado ao bit mais à esquerda.

```
int aging(int8_t** page_table, int num_pages, int prev_page,
          int fifo_frm, int num_frames, int clock) {
    int small, page_aux, i;
    for (page_aux = 0; page_aux < num_pages; page_aux++) { //localiza a primeira página que possui um endereco fisico presente
        if (page_table[page_aux][PT_MAPPED] != 0) {
            small = page_aux;
            break;
        }

    }
    for(i = (page_aux + 1); i<num_pages; i++) {//localiza a página com o menor valor no contador comecando da primeira página ocupada
         if (page_table[i][PT_MAPPED] != 0) {
             if (page_table[small][PT_AGING_COUNTER] > page_table[i][PT_AGING_COUNTER]) {
                small = i;
             }
         }
    }
    return small;
}
```

Figura 4 - Algoritmo Aging

Execuções e Resultados

Para realizar as dez execuções distintas de cada algoritmo, o arquivo “anomaly.dat” foi alterado. A seguir é mostrado a estrutura de memória e referência usada em cada execução:

1°	2°	3°	4°	5°	6°	7°	8°	9°	10°
10 3	10 3	10 3	10 3	10 3	10 3	10 3	10 3	10 3	10 3
3 w	3 w	0 w	5 w	1 r	0 r	0 r	0 r	0 r	3 w
2 w	2 w	2 w	2 w	1 r	2 w	2 r	2 r	2 r	2 r
1 w	2 w	1 r	8 w	8 r	1 r	1 r	1 r	1 r	1 r
0 w	0 r	2 r	4 r	6 w	2 r	2 r	2 w	2 w	2 w
3 w	3 r	3 r	3 r	3 r	3 r	2 r	2 r	2 r	2 r
2 w	2 r	6 r	2 r	2 r	6 r	3 r	8 r	8 r	1 r
4 w	1 w	1 w	0 w	0 w	1 w	6 r	6 w	6 w	3 w
3 w	3 w	3 w	0 w	0 w	5 w	1 r	1 r	1 r	6 r
2 w	2 w	2 w	2 w	2 w	2 w	5 r	5 w	5 w	5 w
1 w	1 w	1 w	5 w	5 w	1 w	2 r	9 r	9 r	1 r
0 w	7 w	7 r	7 w	7 r	0 r	1 r	1 r	1 r	1 r
4 w	4 w	4 r	4 r	4 r	4 r	0 r	0 r	4 r	0 r

Tabela 1 - Estrutura de Memória e Referência por execução

Ao final das execuções foi obtida a seguinte tabela de faltas de página, assim como a média da falta de página de cada algoritmo:

Execuções	FIFO	Segunda Chance	NRU	Aging	Random
1	9	9	7	7	8
2	8	9	6	6	8
3	9	9	9	10	9
4	10	10	10	10	10
5	9	9	10	10	9

6	10	9	9	11	10
7	9	8	10	10	8
8	9	8	10	10	9
9	9	8	10	10	8
10	6	6	7	7	6
Média	8,8	8,5	8,8	9,1	8,5

Tabela 2 - Tabela de faltas de página

Conclusão

Ao analisar a tabela 2 é possível concluir que o algoritmo FIFO + bit R(Segunda Chance) e *Random* obtiveram os melhores resultados, ambos com uma média de falta de páginas igual a 8,5. No entanto, foi percebido que os algoritmos obtiveram um desempenho próximo entre si, sendo a variação de falta de página entre o pior algoritmo, *Aging*, e os melhores,citados acima, de apenas 0,6 faltas de página.

Bibliografia

TANENBAUM, A. S. , Sistemas Operacionais Modernos. Segunda Edição, Prentice Hall, 2003