# SO1

## Época normal 2020/2021

1- Espera ativa, quando usada para o problema da secção crítica, consiste num processo a verificar continuamente uma condição que, quando for verdadeira, permitirá a entrada do processo na região crítica.

Apesar desta solução resolver o problema, não deve ser usada pois consome recursos desnecessariamente, em alternativa, devem-se usar outros métodos de sincronização (e.g. semáforos, mutexes, condition variables)

2- Copy-on-write é o mecanismo que faz com que, após um fork(), as páginas de memória do pai não sejam imediatamente copiadas, inicialmente, o processo pai e o filho partilham as mesmas páginas, apenas após ser detetada uma escrita numa página partilhada é que esta é copiada.

Deste modo, a criação de processos é otimizada pois não são copiadas partes de memória desnecessariamente que nunca vão ser alteradas.

Este benifício verifica-se na chamada de exec() após fork(), uma vez que sem CoW, as páginas seriam inteiramente copiadas inutilmente, já que elas são substituídas após a chamada de exec().

3- Como há apenas 1 entrada na page table por cada endereço física, é poupado espaço em sistemas com grandes espaços de memória virtual e pequenos espaços de memória física, no entanto, a velocidade de procura pelo endereço correto na PT pode ser diminuída. Mesmo usando uma hash table, o processo torna-se mais complexo e podem sempre haver colisões.

4- Sim, para haver thrashing, é necessário gastar mais tempo a resolver conflitos de paginação do que a executar os programas, ou seja, não está diretamente relacionado com a quantidade de processos em execução. Thrashing pode acontecer com poucos processos se, por exemplo, o tamanho dos working-sets for superior ao tamanho da memória disponível

5.1 $8KB = 2^{13} B \Rightarrow offset = \log_2(2^{13}) = 13$

$PTE = 64 \text{ bits} = 2^3 B$

$\#entradas = \frac{2^{13}}{2^3} = 2^{10} \Rightarrow p2 = \log_2(2^{10}) = 10$

Pois uma tabela de segundo nível tem de ter 8KB (e cada entrada ocupa 8 bytes)

$< p_1, p_2, offset >$

$\downarrow \quad \downarrow \quad \downarrow$

$9 \quad 10 \quad 13$

Há 2 entradas de 8 bytes

5.2 $PA \times PTE = 2^9 \cdot 2^3 = 2^{12}$ B

5.3 Ocupado pelas tabelas de 2º nível:

Há $2^9$ entradas de 1º nível, ou seja, há $2^9$ tabelas de 2º nível, como cada tabela de 2º nível ocupa 1 página, então precisamos de $2^9 = 512$ páginas

Ocupado pela tabela de 1º nível:

Como calculado anteriormente, ocupa $2^{12}$ B = 4 KB = $\frac{1}{2}$ pag.

Então precisamos de 512 + 1 páginas no total

6.1. Max: [9, 2, 9]

Available = Max - $\sum$ InUseColumns = [0, 0, 2]

| Need - Allocated | | | |
|---|---|---|---|
| 2 | 1 | 2 | P1-2 |
| 1 | 2 | 6 | P2-4 |
| 6 | 0 | 1 | P3-5 |
| 0 | 0 | 1 | P4-1 |
| 2 | 2 | 1 | P5-3 |

Available $\xrightarrow{P4}$ [2, 1, 3] $\xrightarrow{P1}$ [2, 2, 3] $\xrightarrow{P5}$ [4, 2, 6] $\xrightarrow{P2}$ $\xrightarrow{P2}$ [6, 2, 6] $\xrightarrow{P3}$ [9, 2, 9]

Um safe state é caracterizado pela existência de uma safe sequence, que existe e é: P4 → P1 → P5 → P2 → P3

6.2. Primeiro, assume-se que o pedido é aceite, e corre-se o algoritmo com a matriz de recursos alocados + [0, 0, 1] no processo 4 e - [0, 0, 1] no vetor de recursos disponíveis.

| Need - Allocated | | |
|---|---|---|
| 2 | 1 | 2 |
| 1 | 2 | 6 |
| 6 | 0 | 1 |
| 0 | 0 | 0 |
| 2 | 2 | 1 |

Available $\xrightarrow{P4}$ [2, 1, 2]

↑ Igual ao 6.1 a partir daqui, já que os recursos do P4 são libertados

Safe sequence igual a 6.1, o gestor de recursos aceita o pedido.

7. M ≡ Modify bit ativo

| | PF | PF | PF | PF | SO PF | | | PF | PF | > Σ 2 SO 7 PF |
|---|---|---|---|---|---|---|---|---|---|---|
| FIFO | W1 | W3 | R2 | R3 | R8 | W5 | R3 | R8 | R1 | W6 |
| F0 | 1M | 1M | 1M | 1M | 1M | 5M | 5M | 5M | 5M | 5M |
| F1 | | 3M | 3M | 3M | 3M | 3M | 3M | 3M | 1 | 1 |
| F2 | | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 6M |
| F3 | | | | | 8 | 8 | 8 | 8 | 8 | 8 |

| | PF | PF | PF | PF | SO PF | | | PF | SO PF | > Σ 2 SO 7 PF |
|---|---|---|---|---|---|---|---|---|---|---|
| LRU | W1 | W3 | R2 | R3 | R8 | W5 | R3 | R8 | R1 | W6 |
| F0 | 1M | 1M | 1M | 1M | 1M | 5M | 5M | 5M | 5M | 1 |
| F1 | | 3M | 3M | 3M | 3M | 3M | 3M | 3M | 3M | 3M |
| F2 | | | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
| F3 | | | | | 8 | 8 | 8 | 8 | 8 | 8 |

u ≡ use bit = 0
c ≡ clock position

| | PF | PF | PF | PF | PF | | | PF | SO PF | > Σ 1 SO 7 PF |
|---|---|---|---|---|---|---|---|---|---|---|
| CLOCK | W1 | W3 | R2 | R3 | R8 | W5 | R3 | R8 | R1 | W6 |
| F0 | 1M | 1M | 1M | 1M | c1M | 5M | 5M | 5M | 5M | 5Mu |
| F1 | c | 3M | 3M | 3M | 3M | c3Mu | c3M | c3M | 3Mu | 6M |
| F2 | | c | 2 | 2 | 2 | 2u | 2u | 2u | 1 | c1 |
| F3 | | | c | c | 8 | 8u | 8u | 8 | c8 | 8u |

8.1 Bloco - 4 KB
Ponteiro - 4B
Inode
- 8 ponteiros diretos
- 1 ponteiro indireto
- 1 ponteiro duplamente indireto

8 diretos, cada um aponta para bloco: $8 . 4KB = 32kB = 2^{15} B$

1 indireto, aponta para um bloco de ponteiros, cada ponteiro tem 4B, então esse bloco tem $\frac{4KB}{4B} = \frac{2^{15}}{2^5} = 2^{10}$ ponteiros, ou seja, referencia $2^{10} . 2^{12} = 2^{22} B$

1 duplamente indireto aponta para bloco de ponteiros indiretos, ou seja, referencia $2^{10} . 2^{22} = 2^{32} B$

O tamanho máximo de um ficheiro é $(2^{15} + 2^{22} + 2^{32})B$

8.2 4 Blocos:
- Superbloco
- Bloco da diretoria root
- Bloco da diretoria usr
- Bloco do ficheiro teste.pdf

8.3 O byte 32720 está no bloco $\left\lceil \frac{32720}{2^{12}} \right\rceil = 9$

Para aceder ao bloco 9, precisamos de aceder ao bloco apontado pelo ponteiro indireto e de seguida o bloco específico que contém o byte em questão, ou seja, precisamos de ler 2 blocos após o inode.