109590004 呂育瑋 作業系統 HW3 hand-written part

7.6

(a) 可以，鎖死的原因通常是系統資源不足，而增加可用資源可以解決這個問題。

(b) 不可以，減少可用資源可能會導致系統中某些進程無法獲得所需資源而陷入鎖死狀態。

(c) 不可以，增加一個進程的最大需求量會使進程要求的資源更多，可能導致系統資源不夠分配給該進程，導致鎖死狀態。

(d) 可以，減少一個進程的最大需求量會使系統資源更好地分配，因為進程要求的資源比原本來的小，系統資源能更有彈性的分配。

(e) 如果系統資源能被分配給新進程，且系統沒有進入不安全狀態，是可行的。

(f) 可以移除不必要的進程增加系統資源，對其他進程的分配更有彈性。

7.12

(a)

系統狀態為safe，進程順序為：P2 → P4 → P0 → P1 → P3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Need | | | | Available | | | |
| Resource type | A | B | C | D | A | B | C | D | A | B | C | D | A | B | C | D |
| P0 | 3 | 0 | 1 | 4 | 5 | 1 | 1 | 7 | 2 | 1 | 0 | 3 | 0 | 3 | 0 | 1 |
| P1 | 2 | 2 | 1 | 0 | 3 | 2 | 1 | 1 | 1 | 0 | 0 | 1 | 3 | 4 | 2 | 2 |
| P2 | 3 | 1 | 2 | 1 | 3 | 3 | 2 | 1 | 0 | 2 | 0 | 0 | 7 | 6 | 3 | 4 |
| P3 | 0 | 5 | 1 | 0 | 4 | 6 | 1 | 2 | 4 | 1 | 0 | 2 | 10 | 6 | 4 | 9 |
| P4 | 4 | 2 | 1 | 2 | 6 | 3 | 2 | 5 | 2 | 1 | 1 | 3 | 12 | 8 | 5 | 9 |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 12 | 13 | 6 | 9 |

(b)

系統狀態為safe，進程順序為：P1 → P2 → P3 → P4 → P0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Need | | | | Available | | | |
| Resource type | A | B | C | D | A | B | C | D | A | B | C | D | A | B | C | D |
| P0 | 3 | 0 | 1 | 4 | 5 | 1 | 1 | 7 | 2 | 1 | 0 | 3 | 1 | 0 | 0 | 2 |
| P1 | 2 | 2 | 1 | 0 | 3 | 2 | 1 | 1 | 1 | 0 | 0 | 1 | 3 | 2 | 1 | 2 |
| P2 | 3 | 1 | 2 | 1 | 3 | 3 | 2 | 1 | 0 | 2 | 0 | 0 | 6 | 3 | 3 | 3 |
| P3 | 0 | 5 | 1 | 0 | 4 | 6 | 1 | 2 | 4 | 1 | 0 | 2 | 6 | 8 | 4 | 3 |
| P4 | 4 | 2 | 1 | 2 | 6 | 3 | 2 | 5 | 2 | 1 | 1 | 3 | 10 | 10 | 5 | 5 |
|  |  |  |  |  |  |  |  |  |  |  |  |  | 13 | 10 | 6 | 9 |

7.15

Process NorthboundFarmer {

lock(bridge\_lock);

while southbound\_passing > 0 do

wait(cond\_south, mutex);

end while

++northbound\_passing;

unlock(bridge\_lock);

// passing the bridge

lock(bridge\_lock);

--northbound\_passing;

unlock(bridge\_lock);

broadcast(cond\_north);

}

Process SouthboundFarmer {

lock(bridge\_lock);

while northbound\_passing > 0 do

wait(cond\_north, mutex);

end while

++southbound\_passing;

unlock(bridge\_lock);

// passing the bridge

lock(bridge\_lock);

--sourthbound\_passing;

unlock(bridge\_lock);

broadcast(cond\_south);

}

Mutex bridge\_lock

Condition cond\_north

Condition cond\_south

Integer southbound\_passing = 0

Integer northbound\_passing = 0

8.5

(a)

連續內存分配：容易產生外部碎片，因為空閒與已分配區域相互交替且大小不一，當沒有連續

的空閒區域可用時，將無法滿足大型進程的內存需求。

純分段和純分頁：不容易產生外部碎片，在內存中每個段或頁單獨分配一個空間，這樣空閒與

已分配區域就不會相互交替。

(b)

連續內存分配：容易產生內部碎片，因為必須將進程分配到連續的物理內存區域中，當進程大

小不是物理內存區域大小的倍數時，可能會浪費部分空間。

純分段和純分頁：不容易產生內部碎片，因為每段或每頁的大小都是固定的，並且進程將只使

用它需要的段或頁，而不是整個連續內存區域。

(c)

連續內存分配：可以很容易地實現代碼共享，因為相同的代碼可以映射到不同的進程中的相同

位置，並且可以通過修改某些頁表實現共享。

純分段和純分頁：需要額外的支持和操作來實現代碼共享。在純分段中，可以通過額外的段表

實現共享；在純分頁中，可以通過將相同的頁映射到不同進程的相同位置來實現共享。

8.9

–Compare paging with segmentation with respect to how much memory the address translation structures require to convert virtual addresses to physical addresses.

8.13

–The BTV operating system has a 21 bit virtual address, yet on certain embedded devices, it has only a 16 bit physical address. It also has a 2KB page size. How many entries are there in each of the following?

– (a) A conventional, single level page table

– (b) An inverted page table

9.6

–Assume that we have a demand paged memory.

• The page table is held in registers.

• It takes 8 milliseconds to service a page fault if an empty frame is available or if the replaced page is not modified and 20 milliseconds if the replaced page is modified.

• Memory access time is 100 nanoseconds.

–Assume that the page to be replaced is modified 70 percent of the time.

–What is the maximum acceptable page fault rate for an effective access time of no more than 200 nanoseconds?

9.8

Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1.

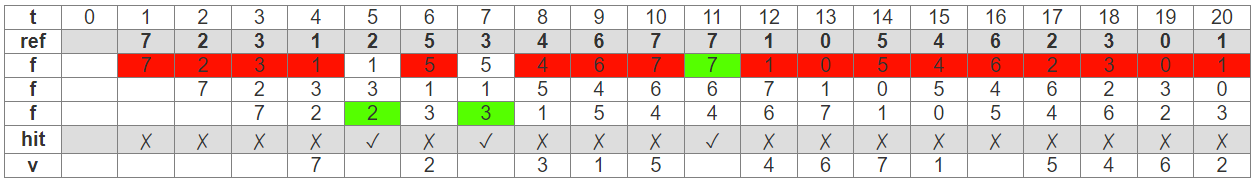
Assume demand paging with three frames, how many page faults would occur for the following replacement algorithms?

1. LRU replacement

18

1. FIFO replacement

17 page faults



Request 7 page fault! current frame:7

Request 2 page fault! current frame:7 2

Request 3 page fault! current frame:7 2 3

Request 1 page fault! current frame:1 2 3

Request 2 Memory resident.

Request 5 page fault! current frame:1 5 3

Request 3 Memory resident.

Request 4 page fault! current frame:1 5 4

Request 6 page fault! current frame:6 5 4

Request 7 page fault! current frame:6 7 4

Request 7 Memory resident.

Request 1 page fault! current frame:6 7 1

Request 0 page fault! current frame:0 7 1

Request 5 page fault! current frame:0 5 1

Request 4 page fault! current frame:0 5 4

Request 6 page fault! current frame:6 5 4

Request 2 page fault! current frame:6 2 4

Request 3 page fault! current frame:6 2 3

Request 0 page fault! current frame:0 2 3

Request 1 page fault! current frame:0 1 3

[3 frames] Page fault 17 with FIFO algorithm.

1. Optimal replacement

13

=== FIFO algorithm ===

=== Optimal algorithm ===

Request 7

page fault!

current frame:7

Request 2

page fault!

current frame:7 2

Request 3

page fault!

current frame:7 2 3

Request 1

page fault!

current frame:1 2 3

Request 2

Memory resident.

Request 5

page fault!

current frame:1 5 3

Request 3

Memory resident.

Request 4

page fault!

current frame:1 5 4

Request 6

page fault!

current frame:1 5 6

Request 7

page fault!

current frame:1 5 7

Request 7

Memory resident.

Request 1

Memory resident.

Request 0

page fault!

current frame:1 5 0

Request 5

Memory resident.

Request 4

page fault!

current frame:1 4 0

Request 6

page fault!

current frame:1 6 0

Request 2

page fault!

current frame:1 2 0

Request 3

page fault!

current frame:1 3 0

Request 0

Memory resident.

Request 1

Memory resident.

[3 frames] Page fault 13 with Optimal algorithm.

=== LRU algorithm ===

Request 7

page fault!

current frame:7

Request 2

page fault!

current frame:7 2

Request 3

page fault!

current frame:7 2 3

Request 1

page fault!

current frame:1 2 3

Request 2

Memory resident.

Request 5

page fault!

current frame:1 2 5

Request 3

page fault!

current frame:3 2 5

Request 4

page fault!

current frame:3 4 5

Request 6

page fault!

current frame:3 4 6

Request 7

page fault!

current frame:7 4 6

Request 7

Memory resident.

Request 1

page fault!

current frame:7 1 6

Request 0

page fault!

current frame:7 1 0

Request 5

page fault!

current frame:5 1 0

Request 4

page fault!

current frame:5 4 0

Request 6

page fault!

current frame:5 4 6

Request 2

page fault!

current frame:2 4 6

Request 3

page fault!

current frame:2 3 6

Request 0

page fault!

current frame:2 3 0

Request 1

page fault!

current frame:1 3 0

[3 frames] Page fault 18 with LRU algorithm.

9.17

A page replacement algorithm should minimize the number of page faults. We can achieve this minimization by distributing heavily used pages evenly over all of memory, rather than having them compete for a small number of page frames. We can associate with each page frame a counter of the number of pages associated with that frame. Then, to replace a page, we can search for the page frame with the smallest counter.

– (a) Define a page replacement algorithm using this basic idea. Specifically address these problems:

• ( i ) What is the initial value of the

• (ii) When are counters increased?

• (iii) When are counters decreased?

• (iv) How is the page to be replaced selected?

– (b) How many page faults occur for your algorithm for the following reference string with four page frames?

• 1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2.

– (c) What is the minimum number of page faults for an optimal page replacement strategy for the reference string in part(b) with four page frames?

9.19

What is the cause of thrashing ? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?