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) 某些情況可以，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；反之不會影響。

(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) 可以移除已釋放資源的進程，增加系統可用資源，對其他進程的分配更有彈性。

7.12

(a)

進程順序為：P2 → P1 → P3 → unsafe，系統狀態為unsafe，跑完P3後資源D沒有任何進程

的資源D小於Available 資源D。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Need | | | | Available | | | |
|  | 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 | 5 | 6 | 3 | 2 |
| P3 | 0 | 5 | 1 | 0 | 4 | 6 | 1 | 2 | 4 | 1 | 0 | 2 |  |  |  |  |
| P4 | 4 | 2 | 1 | 2 | 6 | 3 | 2 | 5 | 2 | 1 | 1 | 3 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

(b)

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Allocation | | | | Max | | | | Need | | | | Available | | | |
|  | 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.

Paging需要比較多記憶體空間，而segmentation需要較少。 Segmentation需要兩個registers(1%)，Paging需要entry(1%)。

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

1. LRU replacement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Page fault | | | 18 | |
| ref | 7 | 2 | 3 | 1 | | 2 | 5 | 3 | 4 | 6 | 7 | 7 | 1 | 0 | 5 | 4 | 6 | 2 | 3 | 0 | 1 |
| f1 | 7 | 7 | 7 | 1 | | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |
| f2 |  | 2 | 2 | 2 | | 2 | 2 | 5 | 5 | 5 | 7 | 7 | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
| f3 |  |  | 3 | 3 | | 3 | 5 | 3 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |

1. FIFO replacement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Page fault | | | 17 | |
| ref | 7 | 2 | 3 | 1 | | 2 | 5 | 3 | 4 | 6 | 7 | 7 | 1 | 0 | 5 | 4 | 6 | 2 | 3 | 0 | 1 |
| f1 | 7 | 7 | 7 | 1 | | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 0 | 0 | 0 | 6 | 6 | 6 | 0 | 0 |
| f2 |  | 2 | 2 | 2 | | 2 | 5 | 5 | 5 | 5 | 7 | 7 | 7 | 7 | 5 | 5 | 5 | 2 | 2 | 2 | 1 |
| f3 |  |  | 3 | 3 | | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 4 | 4 | 4 | 3 | 3 | 3 |

1. Optimal replacement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Page fault | | | 13 | |
| ref | 7 | 2 | 3 | 1 | | 2 | 5 | 3 | 4 | 6 | 7 | 7 | 1 | 0 | 5 | 4 | 6 | 2 | 3 | 0 | 1 |
| f1 | 7 | 7 | 7 | 1 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| f2 |  | 2 | 2 | 2 | | 2 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 4 | 6 | 2 | 3 | 3 | 3 |
| f3 |  |  | 3 | 3 | | 3 | 3 | 3 | 4 | 6 | 7 | 7 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

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 counters?

• (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?

繁忙交換（thrashing）是指系統在處理器和存儲器之間不斷進行無意義的頁面交換操作，導致系統效能急劇下降的情況。它通常發生在當系統所分配的物理內存不足以容納當前活躍進程所需的所有頁面時。

當系統檢測到繁忙交換發生時，通常使用以下指標來進行檢測：

缺頁率（Page Fault Rate）：缺頁率是指在一段時間內發生的頁面錯誤中的比例。當缺頁率過高時，表示系統正在不斷處理頁面錯誤，這可能是繁忙交換的指標之一。

CPU利用率（CPU Utilization）：當系統處於繁忙交換狀態時，CPU將花費大部分時間在頁面錯誤處理上，而不是實際執行進程。因此，CPU利用率的下降也可能是檢測繁忙交換的指標之一。

一旦系統檢測到繁忙交換，它可以採取以下措施來消除這個問題：

增加物理內存：增加系統可用的物理內存可以提供更多的空間來容納活躍進程的頁面，從而減少頁面錯誤和繁忙交換的發生。

優化頁面置換算法：系統可以使用更有效的頁面置換算法，例如LRU（最近最少使用）或LFU（最不常用）來選擇被置換的頁面。這樣可以減少頁面錯誤的數量，從而減輕繁忙交換的影響。

調整進程數量：系統可以通過調整並發運行的進程數量，限制活躍進程的數量，從而減少頁面錯誤和繁忙交換的發生。

優化程序內存使用：優化進程的內存使用方式，例如減少內存碎片化、釋放不需要的資源等，可以減少對物理內存的需求，減輕繁忙交換的壓力。

提高磁盤性能：繁忙交換通常涉及頻繁的磁盤I/O操作，因此提高磁盤性能（例如使用更快速的硬碟、使用快取技術等）可以減少頁面錯誤的處理時間，緩解繁忙交換問題。

增加頁面文件大小：將頁面文件的大小增加到能夠容納更多頁面的範圍，可以減少頁面錯誤和繁忙交換的頻率。

綜合上述措施，系統可以通過增加物理內存、優化頁面置換算法、調整進程數量、優化程序內存使用、提高磁盤性能和增加頁面文件大小等來消除繁忙交換問題。實際採取哪些措施取決於具體的系統配置和需求，並可能需要進行詳細的性能分析和優化。