Ch 10.3 - 10.4

10.3 Copy-on-Write

- fork() 시스템 호출을 통해 프로세스를 생성할 때에는 **페이지 공유와 비슷한 기법으로 첫 demand paging조차 생략하는 것이 가능**하다. 이 기법을 통해 프로세스 생성 시간을 더 줄일 수 있다.
- 기존에는 fork()를 하면 부모 프로세스의 페이지들을 자식 프로세스에 복사한 후 자식 프로세스의 주소 공간을 구성해 주는 방식으로 이루어졌는데, 대부분의 자식 프로세스들은 주소 공간이 마련되자마자 exec() 시스템 호출을 하기 때문에 부모로부터 모든 페이지들을 복사해오는 것이 unnecessary한 경우가 많다. **copy-on-write** 방식은 자식 프로세스가 시작할 때 부모의 페이지를 당분간 함께 공유하다가, **두 프로세스 중 하나가 특정 페이지에 다가 쓰려고 하면 그 페이지의 복사본을 만들어서 그 복사본에 쓰는 방식**이다.

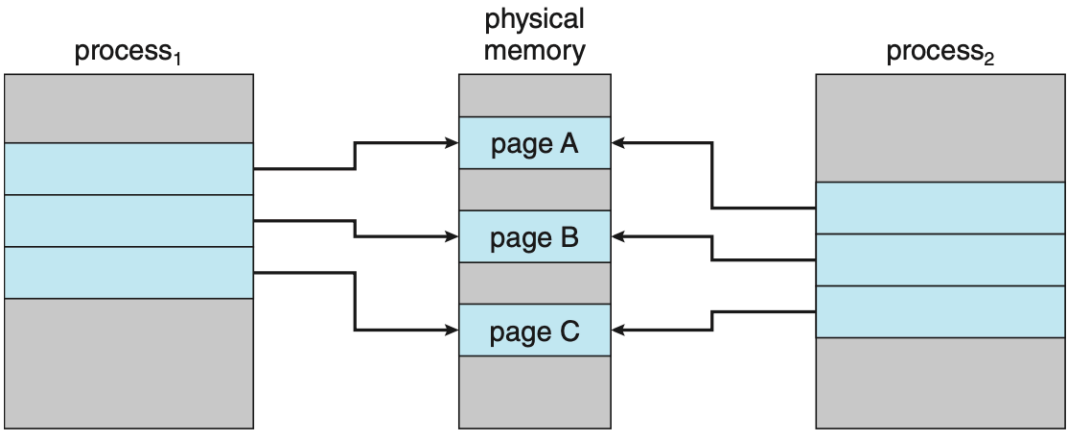


Figure 10.7 Before process 1 modifies page C.

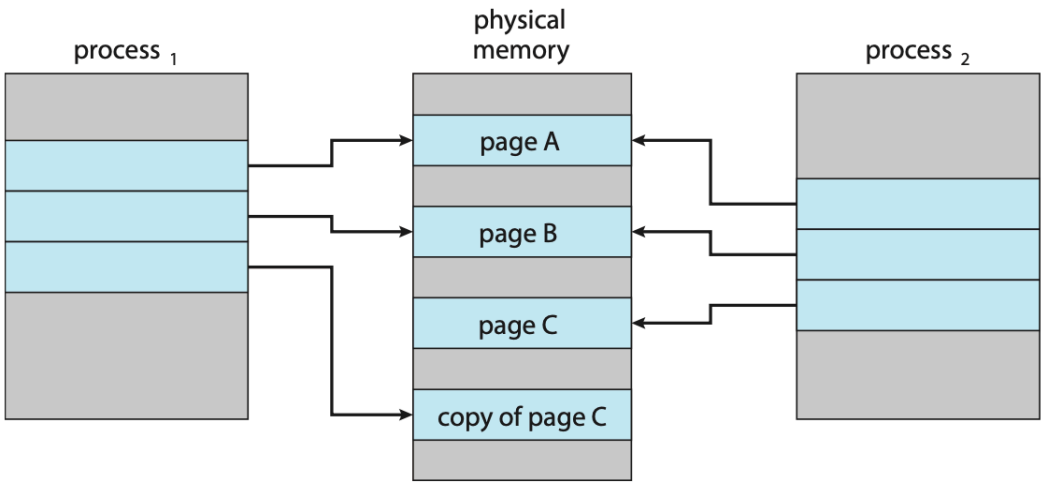


Figure 10.8 After process 1 modifies page C.

- UNIX의 몇가지 버전에서는 **vfork() (virtual memory fork)**라는 것을 제공하는데, vfork()는 fork()와는

다르게 부모 프로세스는 잠시 중단(suspend)되고 자식 프로세스가 부모의 주소 공간을 사용하게 된다. vfork()는 copy-on-write를 사용하지 않으므로 **자식 프로세스가 부모 주소 공간의 페이지를 수정하게 되면 변경된 페이지가 재실행 시 부모 프로세스에 그대로 보여진다.** 따라서 vfork()를 사용할 때에는 자식 프로세스가 부모 주소 공간 페이지들에 변경을 가하지 않도록 주의해야 한다. **vfork()는 자식 프로세스가 만들어지자마자 exec()를 호출하는 경우를 위해 마련된 것**이다.

10.4 Page Replacement

- page fault가 발생했을 때, 만약 메모리에 해당 페이지를 올릴 공간이 남아있다면 문제 없지만, **메모리 공간이 모두 찼을 경우**에는 메모리에 있는 페이지의 일부를 디스크로 다시 내보내고 해당 페이지를 메모리에 올려야한다. 이때 **메모리에 있는 페이지 중 어떤 페이지를 메모리 밖으로 내보낼 것인지 결정**하는 방법이 page replacement이다.

10.4.1 Basic Page Replacement

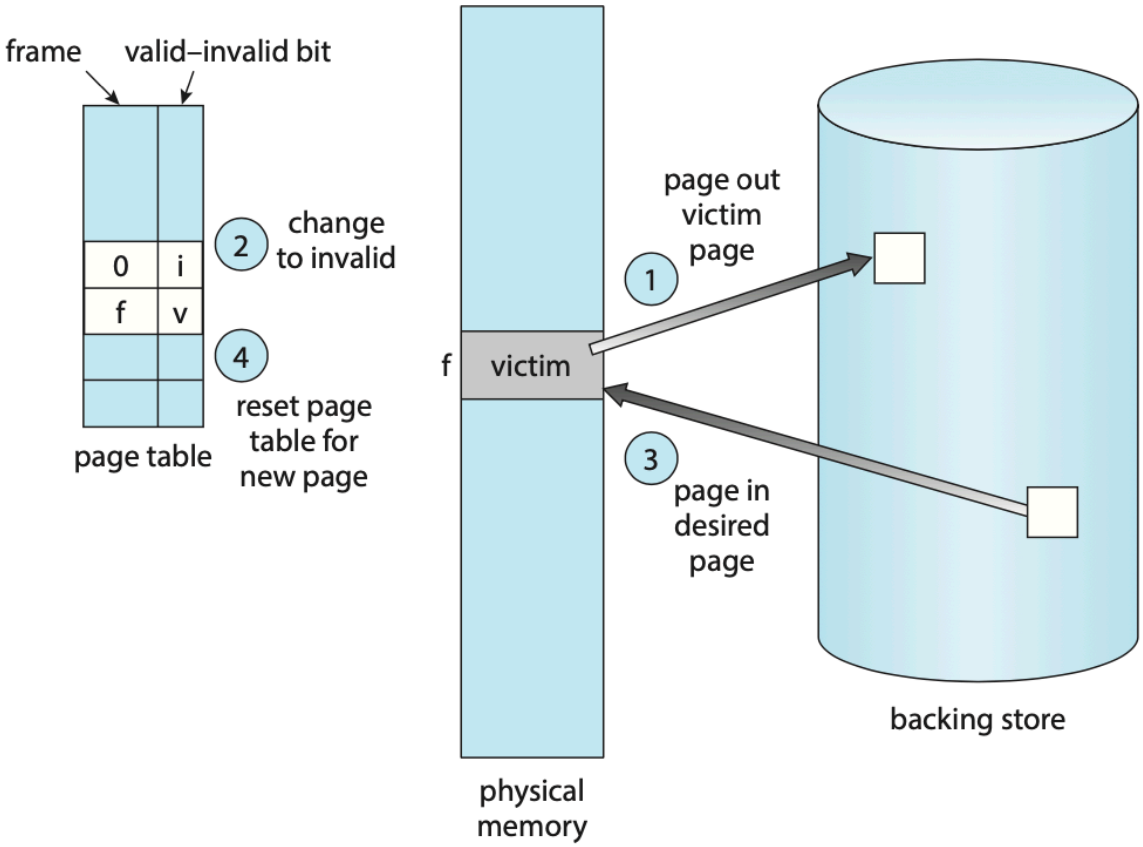


Figure 10.10 Page replacement.

1. 필요한 페이지의 위치를 secondary storage에서 찾는다
2. free frame이 있는지 확인한다.
    1. 있다면, 그것을 사용한다.
    2. 없다면, **page-replacement algorithm** 중 하나를 사용해서 메모리에서 내보내질 **victim frame**을 선정한다.
    3. 필요하다면 victim frame을 secondary storage에 write한 후에, 변경사항을 반영하기 위해 page

table과 frame table를 갱신하다.

3. page fault가 일어난 지점부터 다시 프로세스를 실행한다.

- 위의 과정을 보면, page replacement가 일어날 때마다 2번의 page transfer(나가는 페이지인 page-out과 들어오는 페이지인 page-in)이 일어난다는 것을 볼 수 있다. 이는 page-fault를 해결하는데에 시간이 두배로 늘어난다는 뜻으로 이를 더 빠르게 하기 위해 **modify bit(dirty bit)**이라는 것을 사용한다. 모든 page와 frame은 modify bit을 가지고 있으며 해당 page가 수정이 되었다면 1, 수정된 적이 없다면 0으로 set되어 있다. **만약 victim page의 modify bit이 1이라면 이는 해당 페이지가 메모리에 올려진 후에 수정된 적이 있다는 뜻**이기 때문에 secondary storage에 해당 페이지를 write해야한다. 만약 modify bit이 0이라면 수정된 적이 없다는 뜻이므로 secondary storage를 업데이트하는 데에 시간을 쓸 필요가 없다는 것이다.

- 다양한 page-replacement algorithm들이 존재하지만, **가장 낮은 page-fault rate을 가진 알고리즘을 선택**하는 것이 중요하다. 알고리즘들의 page-fault rate을 계산하기 위해 사용하는 임의의 page number sequence를 **reference string**이라고 부른다. **아래의 알고리즘들을 설명 할 때 우리는 아래와 같은 reference string을 가지고 예시를 들 것이며, 메모리에는 frame이 3개 있다고 가정 할 것이다.**

```
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
```

10.4.2 FIFO Page Replacement

- **가장 먼저 메모리에 적재된 페이지를 먼저 내보내는 방법**이다.

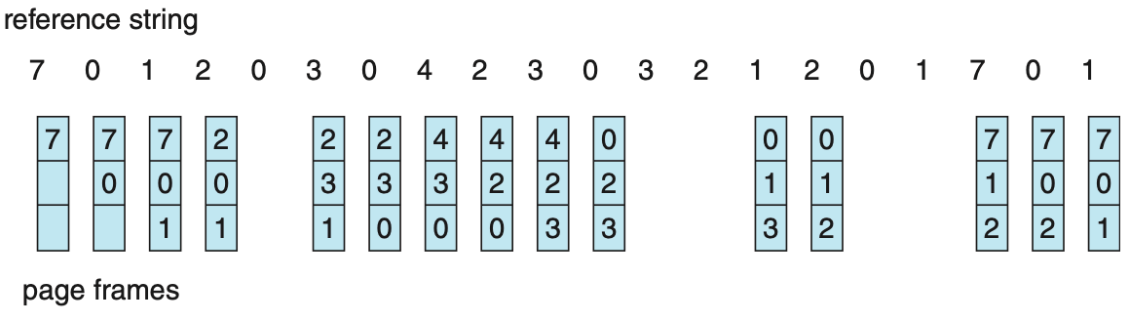
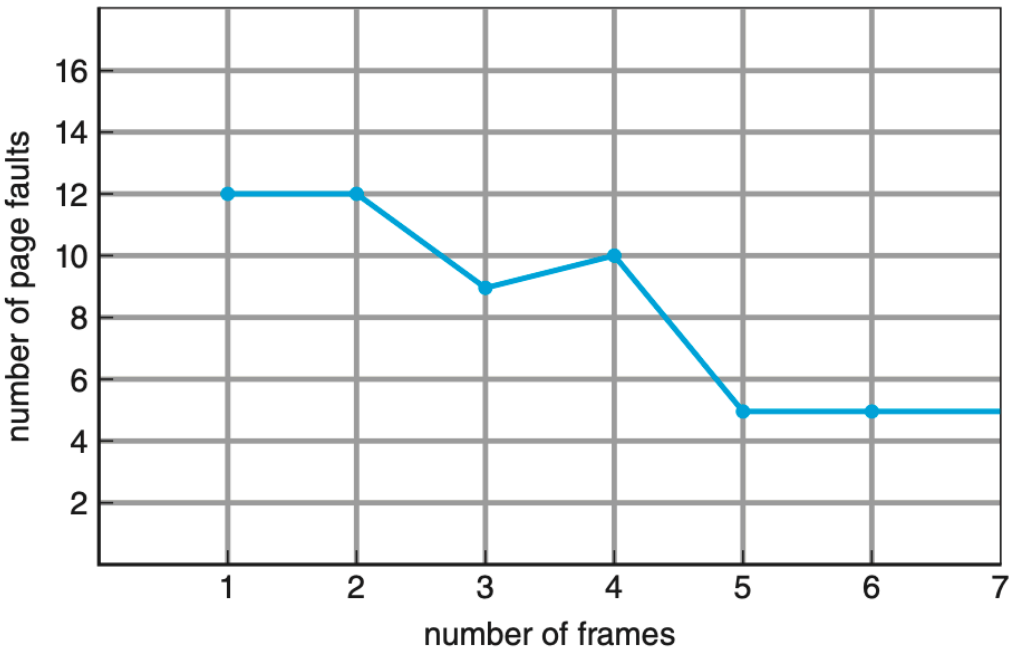


Figure 10.12 FIFO page-replacement algorithm.

  - 예시 reference string에 FIFO page-replacement algorithm을 적용한 결과 **총 15번의 page-fault**가 일어난다.
- 이 알고리즘은 단순하다는 장점이 있지만, 대신에 퍼포먼스가 항상 좋지는 않으며, 특히 Belady's anomaly라는 현상이 생길 수 있다. 보통은 메모리에 있는 frame의 개수가 늘어날 수록 page-fault rate가 줄어들지만, **Belady's anomaly는 frame 수가 늘어났지만 page-fault rate가 줄어드는 대신 오히려 늘어나는 경우**를 말한다.
  - `1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5`라는 reference string에 FIFO algorithm을 적용할 때, frame이 3개 있을 때는 page-fault가 9번 밖에 안 일어나지만 frame이 4개 있을 때는 page-fault가 10으로 늘어나는 현상을 관찰할 수 있다.



10.4.3 Optimal Page Replacement

- OPT 방식으로 불리며, **미래를 보고 앞으로 가장 사용 안될 페이지를 교체하는 방법**이다.

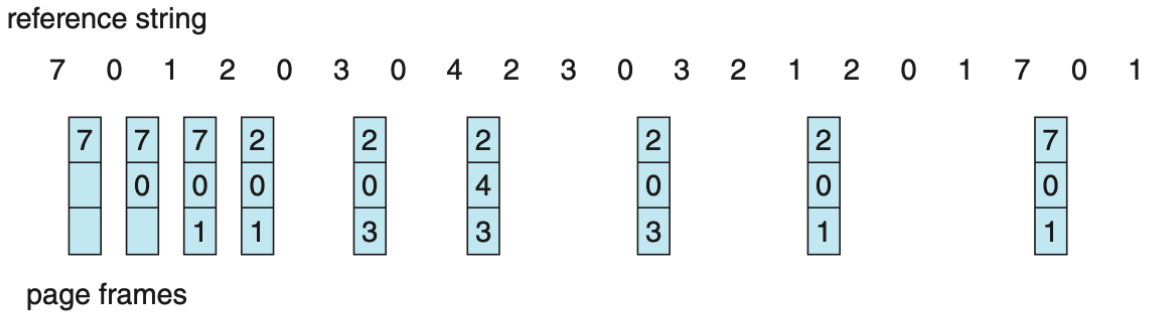


Figure 10.14 Optimal page-replacement algorithm.

  - 예시 reference string에 optimal page-replacement algorithm을 적용한 결과 **총 9번의 page-fault**가 일어난다.
- 하지만 현실에서는 앞으로 사용되지 않을 페이지에 대해 알 수가 없으므로 **비현실적인 방법**이다.

10.4.4 LRU(Least Recently Used) Page Replacement

- **최근에 가장 오랫동안 사용되지 않은 페이지를 교체하는 방법**이다. optimal page replacement는 미래를 보는 것이라면, LRU는 과거를 보는 것이다. OPT만큼 좋지는 않지만 현실적으로 가능한 방법이기 때문에 자주 사용되는 알고리즘이다.

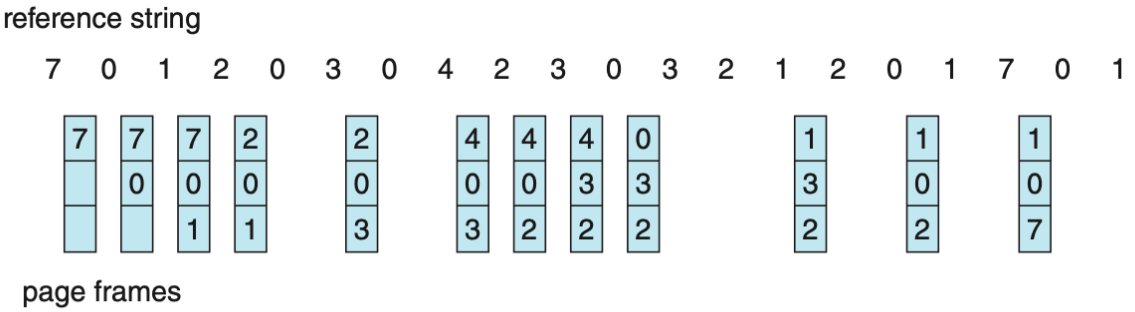


Figure 10.15 LRU page-replacement algorithm.

  - 예시 reference string에 LRU page-replacement algorithm을 적용한 결과 **총 12번의 page-fault**가 일어난다.
- **page들의 최근 사용 순서를 기록**하기 위해 하드웨어의 부분적인 지원이 필요한데, 크게 두 가지 방법을 사용한다.
  - **counters**: 각 page-table 항목에 time-of-use라는 필드를 만들고 CPU에 logical clock 또는 counter를 추가한다. 새로운 memory reference가 일어날 때마다 clock(counter)의 값을 증가 시키며, page reference가 일어날 때마다 page-table에 있는 해당 페이지의 time-of-use 필드에 현재 clock(counter) 값이 기입된다. Page replacement가 일어날 때는 time-of-use 값이 제일 작은 페이지가 vicitim page로 선정된다.
  - **stack**: page number들을 stack에 쌓아놓고 특정 페이지가 reference되면 해당 페이지는 stack의 가장 위로 옮겨진다. 따라서 가장 최근에 사용된 페이지는 stack의 가장 위에, 가장 오랫동안 사용되지 않은 페이지는 stack의 가장 아래에 놓이게 된다.
- OPT와 LRU 모두 Belady's anomaly가 일어나지 않는 stack algorithms에 해당한다.

10.4.5 LRU-Approximation Page Replacement

- 위에서 말했듯이 LRU를 구현하기 위해서는 하드웨어의 지원이 필요한데, 그 지원을 충분히 해줄 수 있는 컴퓨터 시스템들이 많지는 않다. 대신에, **reference bit**이라는 형태로 지원을 해주는 경우가 많은데, reference bit은 page table의 각 항목마다 하나씩 있으며, **해당 항목의 페이지가 참조(read or write)되는 경우에 1로 바뀐다.**
- 모든 reference bit은 초기에는 0으로 set되어 있으며, 프로세스가 시작된 후 참조되면 1로 바뀐다. 하지만 이렇게 하면 **페이지의 참조 여부는 알 수 있지만 참조 순서는 알 수 없기** 때문에 이를 해결하기 위해 몇가지 알고리즘이 존재한다.

10.4.5.1 Additional-Reference-Bits Algorithm

- reference bit을 추가해서 순서 정보를 얻는 알고리즘이다. 현재 memory에 올라가 있는 모든 페이지들에 대하여 8bit를 부여한 후, time period를 정해서(예를 들어 100ms) 한 period가 끝날 때마다 운영체제가 reference bit을 8bit 위에서 오른쪽으로 한칸씩 옮겨준다. 따라서 각 8bit shift register들은 지난 8번의 time period동안의 해당 페이지의 참조여부 정보가 있는 것이다.
  - 만약 shift register가 00000000이면 해당 페이지는 지난 8번의 time period동안 단 한번도 참조되지 않은 것이며, 11000100의 값을 가지는 페이지는 01110111을 가지는 페이지보다 더 최근에 사용되었음을 알 수 있다.

- shift register 값이 가장 작은 페이지가 LRU 페이지이며, 이 페이지가 victim page로 선정될 것이다. 만약 같은 값을 가지는 페이지가 여러개면, FIFO 알고리즘을 이용해서 그 중 하나를 골라서 replace하거나 다 replace하면 된다.

10.4.5.2 Second-Chance Algorithm

- FIFO replacement algorithm이랑 비슷하지만, **선정된 페이지를 replace하지 전에 그 페이지의 reference bit을 본다.** 만약 값이 0이라면 해당 페이지를 바로 replace하지만 값이 1이라면 해당 페이지에게 한번의 기회를 더 주고 다음 FIFO 페이지로 넘어간다. 두번째 기회가 주어진 페이지의 reference bit은 다시 0으로 초기화되고 도착시간은 현재 시간으로 업데이트된다. 따라서 두번째 기회가 주어진 페이지는 다른 페이지들이 모두 replace되거나 모두 두번째 기회가 주어지기 전까지는 replace되지 않을 것이다.
- 이 알고리즘을 구현하기 위한 한가지 방법은 circular queue를 사용하는 것이다.

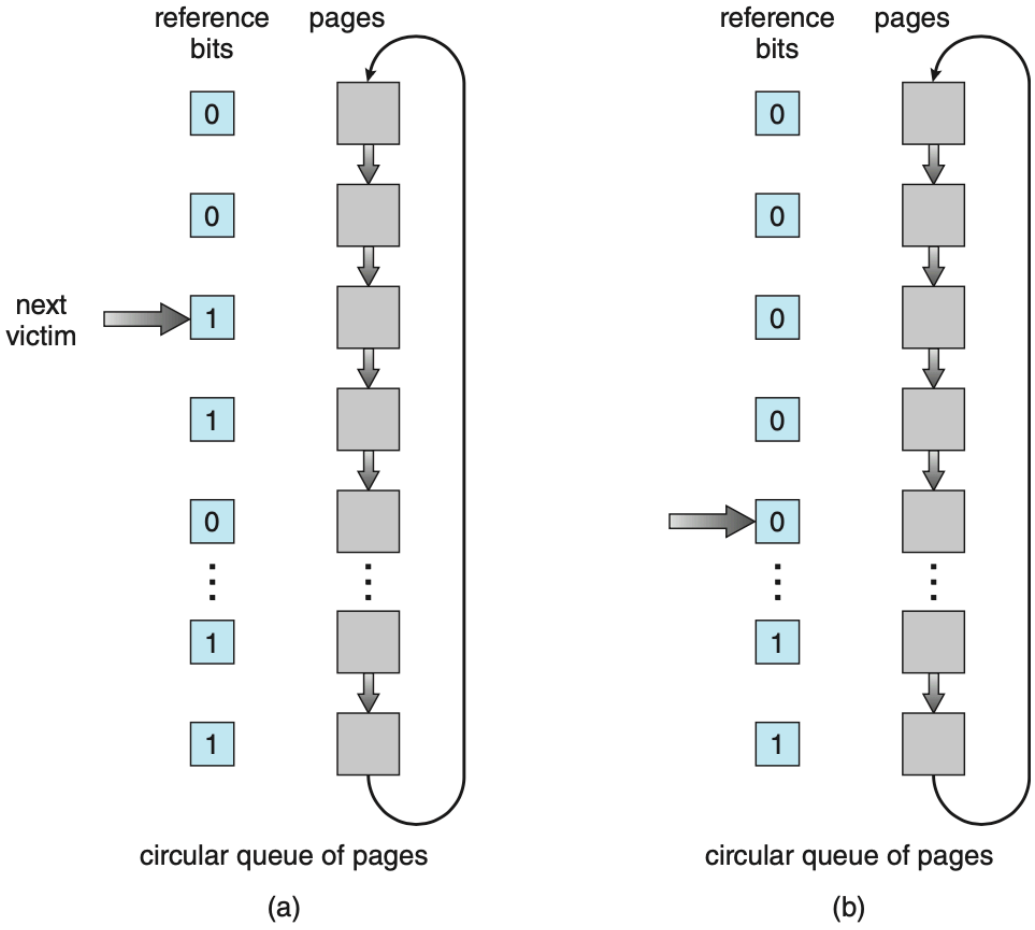


Figure 10.17 Second-chance (clock) page-replacement algorithm.

10.4.5.3 Enhanced Second-Chance Algorithm

- Second chance algorithm의 심화 버전으로, 이번에는 **reference bit과 modify bit 둘 다를 고려**한다. 각 페이지는 다음의 4가지 클라스 중 하나에 속하게 된다.
  - class 0 (0, 0): 참조되지 않음, 수정되지 않음
  - class 1 (0, 1): 참조되지 않음, 수정됨

    - class 2 (1, 0): 참조됨, 수정되지 않음
    - class 3 (1, 1): 참조됨, 수정됨
- 이 알고리즘에서는 second chance algorithm과 같이 원형큐를 돌면서 victim page를 선정하지만 이때는 **가장 낮은 class에 속하는 페이지가 victim page가 된다.** 이 알고리즘은 수정됐지만 참조되지 않은 페이지를 수정되지 않았지만 참조된 페이지보다 덜 중요하게 간주한다.

10.4.6 Counting-Based Page Replacement

- counter를 이용해서 각 페이지마다 reference가 몇번 만들어졌는지에 대한 정보를 저장해서 이를 이용하는 page replacement algorithm들도 존재한다.
    - **Least frequently used(LFU) page replacement algorithm**: 가장 낮은 count를 가진 페이지가 victim page로 선정된다. count가 높은 페이지들은 자주 사용되는 페이지들일 것이라는 가정을 하고 있다.
    - **Most frequently used(MFU) page replacement algorithm**: 가장 높은 count를 가진 페이지가 vicitim page로 선정된다. count가 낮은 페이지들은 최근에 메모리에 올려졌으며 따라서 아직 앞으로 많이 사용될 것이라는 가정을 하고 있다.

10.4.7 Page-Buffering Algorithms

- page-replacement algorithm에 더해서 추가적으로 실행되는 과정들이 몇 개 있다.
    - keeping a pool of free frames
        - page fault가 일어나서 victim frame이 선정되면, victim frame이 메모리에서 내보내지기 전에 필요한 페이지는 pool에서 하나의 free frame을 가져와서 거기에 쓰여진다. 이 방법을 사용하면 victim page가 메모리에서 내보내지는 것을 기다릴 필요없이 바로 프로세스를 재실행 할 수 있다. victim이 내보내지면, 그 frame은 free-frame pool에 추가된다.
    - list of modified pages
        - paging device가 바쁘지 않을 때, modified page 중 하나를 선택해서 secondary storage에 해당 데이터를 업데이트한다. 이렇게 하면 나중에 만약 이 페이지가 victim page로 선정되면 secondary storage에 수정된 내용을 업데이트하는데에 시간을 낭비할 필요가 없을 것이다.
    - keeping a pool of free frames but remembering which page was in each frame
        - free frame pool을 유지하지만 그 풀 속 각 프레임의 원래 임자 페이지가 누구였는지를 기억해 놓는 것이다. pool 속의 있는 프레임의 내용은 그것을 secondary storage에 썼다고 하더라도 수정되지 않았을 확률이 있으므로 거기에 저장되어 있던 페이지는 프레임이 사용되기 전까지는 다시 사용될 수 있기 때문이다. 이 경우 I/O는 전혀 필요하지 않으며 page fault가 일어날 때 우선 찾는 페이지가 아직도 pool에 훼손되지 않은 채로 남아 있는지 검사한 후 만약 없으면 그 때 free frame을 선정에서 거기에 필요한 페이지를 읽어들인다.

10.4.8 Applications and Page Replacement

- database나 data warehouse와 같은 몇몇 application들은 운영체제의 virtual memory를 통해 데이터를 접근하는 것보다 자신들만의 memory management 방법으로 데이터를 접근하는 것이 훨씬 더 빠르고 효율적인 경우가 있다. 이런 경우에는 운영체제가 이들에게 secondary storage의 일부를 file-system data structure 없이 접근할 수 있게 해주며, 이를 **raw disk**라고 부른다.