**Second Chance Page Replacement**

Assignment 4

**COMSATS University Islamabad**

Sahiwal Campus



**Usama Sarwar**

FA17-BS(CS)-090-B

**Ms Aniqa Khawar**

Operating System Concepts

December 09, 2019

Second Chance Page Replacement

Apart from LRU, OPT and FIFO page replacement policies, we also have the second chance/clock page replacement policy. In the Second Chance page replacement policy, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between consecutive considerations will not be replaced. The page replaced is the one that, when considered in a round robin matter, has not been accessed since its last consideration.

It can be implemented by adding a “second chance” bit to each memory frame-every time the frame is considered (due to a reference made to the page inside it), this bit is set to 1, which gives the page a second chance, as when we consider the candidate page for replacement, we replace the first one with this bit set to 0 (while zeroing out bits of the other pages we see in the process). Thus, a page with the “second chance” bit set to 1 is never replaced during the first consideration and will only be replaced if all the other pages deserve a second chance too!

**Example**

Let’s say the reference string is **0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4** and we have **3** frames. Let’s see how the algorithm proceeds by tracking the second chance bit and the pointer.

* Initially, all frames are empty so after first 3 passes they will be filled with {0, 4, 1} and the second chance array will be {0, 0, 0} as none has been referenced yet. Also, the pointer will cycle back to 0.
* **Pass-4:** Frame={0, 4, 1}, second\_chance = {0, 1, 0} [4 will get a second chance], pointer = 0 (No page needed to be updated so the candidate is still page in frame 0), pf = 3 (No increase in page fault number).
* **Pass-5:** Frame={2, 4, 1}, second\_chance= {0, 1, 0} [0 replaced; it’s second chance bit was 0, so it didn’t get a second chance], pointer=1 (updated), pf=4
* **Pass-6:** Frame={2, 4, 1}, second\_chance={0, 1, 0}, pointer=1, pf=4 (No change)
* **Pass-7:** Frame={2, 4, 3}, second\_chance= {0, 0, 0} [4 survived but it’s second chance bit became 0], pointer=0 (as element at index 2 was finally replaced), pf=5
* **Pass-8:** Frame={2, 4, 3}, second\_chance= {0, 1, 0} [4 referenced again], pointer=0, pf=5
* **Pass-9:** Frame={2, 4, 3}, second\_chance= {1, 1, 0} [2 referenced again], pointer=0, pf=5
* **Pass-10:** Frame={2, 4, 3}, second\_chance= {1, 1, 0}, pointer=0, pf=5 (no change)
* **Pass-11:** Frame={2, 4, 0}, second\_chance= {0, 0, 0}, pointer=0, pf=6 (2 and 4 got second chances)
* **Pass-12:** Frame={2, 4, 0}, second\_chance= {0, 1, 0}, pointer=0, pf=6 (4 will again get a second chance)
* **Pass-13:** Frame={1, 4, 0}, second\_chance= {0, 1, 0}, pointer=1, pf=7 (pointer updated, pf updated)
* **Page-14:** Frame={1, 4, 0}, second\_chance= {0, 1, 0}, pointer=1, pf=7 (No change)
* **Page-15:** Frame={1, 4, 2}, second\_chance= {0, 0, 0}, pointer=0, pf=8 (4 survived again due to 2nd chance!)
* **Page-16:** Frame={1, 4, 2}, second\_chance= {0, 1, 0}, pointer=0, pf=8 (2nd chance updated)
* **Page-17:** Frame={3, 4, 2}, second\_chance= {0, 1, 0}, pointer=1, pf=9 (pointer, pf updated)
* **Page-18:** Frame={3, 4, 2}, second\_chance= {0, 1, 0}, pointer=1, pf=9 (No change)

In this example, second chance algorithm does as well as the LRU method, which is much more expensive to implement in hardware.

**More Examples**

**Input:** 2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1

3

**Output:** 13

**Input:** 2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1

4

**Output:** 11

**Algorithm**  
Create an array **frames** to track the pages currently in memory and another Boolean array **second\_chance** to track whether that page has been accessed since it’s last replacement (that is if it deserves a second chance or not) and a variable **pointer** to track the target for replacement.

1. Start traversing the array **arr**. If the page already exists, simply set its corresponding element in **second\_chance** to true and return.
2. If the page doesn’t exist, check whether the space pointed to by **pointer** is empty (indicating cache isn’t full yet) – if so, we will put the element there and return, else we’ll traverse the array **arr** one by one (cyclically using the value of **pointer**), marking all corresponding **second\_chance** elements as false, till we find a one that’s already false. That is the most suitable page for replacement, so we do so and return.
3. Finally, we report the page fault count.

|  |
| --- |
| **CODE**  // Java program to find largest in an array  // without conditional/bitwise/ternary/ operators  // and without library functions.  import java.util.\*;  import java.io.\*;  class secondChance  {      public static void main(String args[])throws IOException      {          String reference\_string = "";          int frames = 0;            //Test 1:          reference\_string = "0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4";          frames = 3;            //Output is 9          printHitsAndFaults(reference\_string,frames);            //Test 2:          reference\_string = "2 5 10 1 2 2 6 9 1 2 10 2 6 1 2 1 6 9 5 1";          frames = 4;            //Output is 11          printHitsAndFaults(reference\_string,frames);        }        //If page found, updates the second chance bit to true      static boolean findAndUpdate(int x,int arr[],                         boolean second\_chance[],int frames)        {          int i;            for(i = 0; i < frames; i++)          {                if(arr[i] == x)              {                  //Mark that the page deserves a second chance                  second\_chance[i] = true;                    //Return 'true', that is there was a hit                  //and so there's no need to replace any page                  return true;              }          }            //Return 'false' so that a page for replacement is selected          //as he reuested page doesn't exist in memory          return false;        }          //Updates the page in memory and returns the pointer      static int replaceAndUpdate(int x,int arr[],                   boolean second\_chance[],int frames,int pointer)      {          while(true)          {                //We found the page to replace              if(!second\_chance[pointer])              {                  //Replace with new page                  arr[pointer] = x;                    //Return updated pointer                  return (pointer+1)%frames;              }                //Mark it 'false' as it got one chance              // and will be replaced next time unless accessed again              second\_chance[pointer] = false;                //Pointer is updated in round robin manner              pointer = (pointer+1)%frames;          }      }        static void printHitsAndFaults(String reference\_string,                                                    int frames)      {          int pointer,i,l,x,pf;            //initially we consider frame 0 is to be replaced          pointer = 0;            //number of page faults          pf = 0;            //Create a array to hold page numbers          int arr[] = new int[frames];            //No pages initially in frame,          //which is indicated by -1          Arrays.fill(arr,-1);            //Create second chance array.          //Can also be a byte array for optimizing memory          boolean second\_chance[] = new boolean[frames];            //Split the string into tokens,          //that is page numbers, based on space          String str[] = reference\_string.split(" ");            //get the length of array          l = str.length;            for(i = 0; i<l; i++)          {                x = Integer.parseInt(str[i]);                //Finds if there exists a need to replace              //any page at all              if(!findAndUpdate(x,arr,second\_chance,frames))              {                  //Selects and updates a victim page                  pointer = replaceAndUpdate(x,arr,                            second\_chance,frames,pointer);                    //Update page faults                  pf++;              }          }            System.out.println("Total page faults were "+pf);      }  } |

**Output:**

Total page faults were 9

Total page faults were 11