**Report**

**Cache**

***Abstract summary*** *– implementation of second chance algorithm. Simulate cache with default size of 10 (or passed number) pages block for faster access of data.*

**Implementation**

Every page Blocks represents as EntryBlock class that contains following attributes:

* **public boolean dirty\_bit;**
* **public boolean reference\_bit;**
* **public int frameID;**

Also, every EntryBlock goes along with arrayList of pages type of array of bytes:

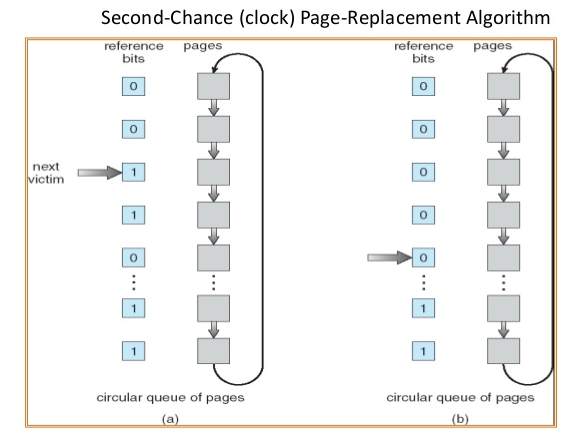
* **byte[] pages.**

**Description**

The process starts at the first element of EntryBlock and first element of pages. Every element of **EntryBlock refrence\_bit** is set to false with implying that nothing has been cached yet. When we start writing to the cache, the closet **EntryBlock** with false **refrence\_bit** will be occupied (cached) and set to **true.** After, the pointer will point to the next **EntryBlock** with **false refrence\_bit** to be set to **true** as well. The process will continue until all Entry Blocks are occupied.

When all EntryBlocks are cached (all refrence\_bit set to true), the process starts implying second chance algorithm. Closest **EntryBlock** will be reset to the false and move pointer to the next **EntryBlock** where it is repeating set **refrence\_bit to false**. When the pointer finds the nearest refrence**\_bit** as **false**, the program will re-cached current **EntryBlock** and set it to the **true.**

**Visual representation**



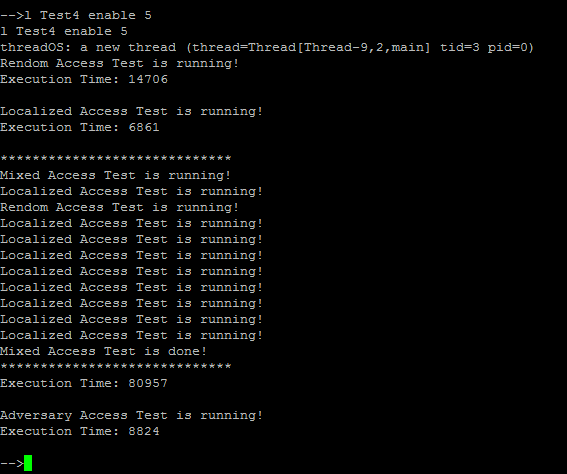
**Main function of Cache.java**

Two methods read () and write () of class Cache are using the second chance algorithm. The FAQ page discusses the read function. The report will cover more detail method write ().

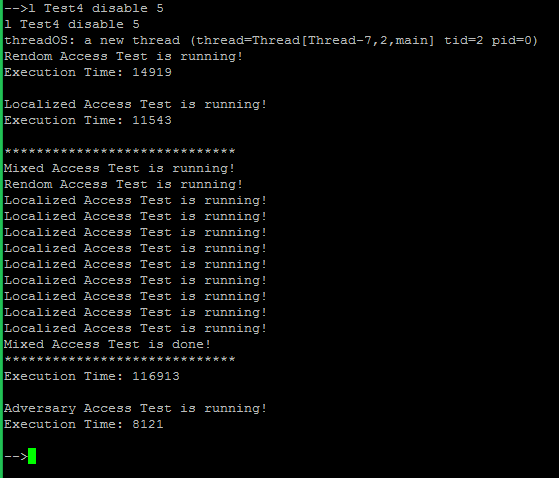
**Write ()**

Begin to loop the cache if it contains given **blockID**. IF so, set **refrence\_bit** and **dirty\_bit** to **true** and copy given buffer into cached buffer. If could not find given **blockID** in the cache, find possible open cache page or force second-change algorithm. Method **findOpenPage ()** takes care of it. After, the program uses private function **toCached ()** that caches the **blockID** with given buffer. Similar approach is implemented in read () function.

**Test4 enable**



**Test4 disable**



**Over all Result of testing**

Over all using cache gives better performance with localized access test. The number is **enable cache 6861** and **disable cache 11543**. While random access give relatively close performance. The number is **enable cache 14702** and **disable cache 14919.** Similarly foradversary test. The number is **enable cache 8824** and **disable cache 8121**. Mixed performance is much faster within enable cache. The number is **enable cache 80957** and **disable cache 116913.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Random | Localized | Mixed | adversary |
| enable | **14702** | **6861** | **11543** | **8824** |
| disable | **14919** | **11543** | **116913** | **8121** |