Computer Engineering
and Software Systems
Program

OPERATING SYSTEMS

A Simulation For Page Replacement Algorithms

Submitted by:

Peter Nabil Zaghloul
Sec(2) ID:16P8100
Mostafa Mahmoud El-Rosasy
Sec(2) ID:16P8113

Submitted to:

Prof. Dr. Gamal Abd El Shafi
Eng. Kareem Darweesh

TABLE OF CONTENTS

1. INTRODUCTION

This Project is a Simulation of six page replacement Algorithms, The Project was implemented using Python Programming language and is composed of eight functions, we also use one built in python Module the 'random' module which we use to generate random variables used inside the project.

2. IMPLEMENTATION

2.1 fifo Function

The fifo function implements the FIFO (first in first out) algorithm, this algorithm quite simply replaces the new page with the first page that entered the memory.
The function takes the reference string as a list, the number of pages and the number of frames.
It loops over the reference string and if the page already occupies a frame it moves on else it overwrites the first frame with the new page deleting the old frame and increments the page faults by one.
The function returns the integer value of the number of page faults.

2.2 lru Function

The lru function implements the LRU (least recently used) algorithm, this algorithm looks in the existing frames, and finds which one was used the furthest in the past, if it finds more than one it chooses between them by means of the FIFO algorithm.
The function keeps track of when every page has been put in the "frames" list using "recently" list.
The index i in "frames" represents the page at that index, and in "recently" it represents when this page has been added.
If a page is requested and it's already in the "frames" in index i the function will get the maximum value in the "recently" list and increment it and place it at index i in recently.
If a page is requested and it's not in the frames, the function will get the minimum value in recently (let it be at index j) and add the new page in frames at index j, then it will get the maximum value in recently and increment it and add it at j in recently.

2.3 lfu Function

The lfu function implements the LFU (least frequently used) algorithm, in this algorithm, when we need a page replacement we look on the reference string from

the start till the page we currently want to place in memory, and we replace it with the least used page.
The function keeps track of the number of times a page has been requested.
The index i in "frames" represents the page at that index, and in "frequency" it represents the number of times the page has been requested while it's in "frames".
When a page that exists in "frames" is requested, its corresponding index in "frequency" is incremented by one.
When a page that doesn't exist in "frames" is requested the the function gets the minimum value in "frequency" (let it be at index j) and add the new page at index j in "frames", then sets the value at index j in "frequency" to 1.
If there are more than one minimum value in "frequency" the least recently used page is replaced.

2.4 second_chance Function

The second_chance function implements the second chance algorithm, this algorithm adds a reference bit to each page in the queue, each page will have a reference bit when it is first allocated this bit is set to 0 when it's addressed again while in the table the bit changes to 1 , if we want to replace the first page that meets us with a 0 reference bit is removed and add the new page to the end of the queue , if we meet any page with 1 reference bit before we find a page with 0 reference bit we change the reference bit of that page to 0 and move it to the end of the queue (giving it a second chance).
The function creates an empty list memqueue, selectdict a dictionary for the reference bit.
Then, loops on the reference string.
If the page that is requested now already exists, it converts the used bit in the selectdict to 1.
If the memqueue is already full, it loops on the memqueue until it finds a page whose used bit is 0, it only checks the first element in the queue if it finds that the used bit is 1 it changes it to 0 and moves that element to the end of the list otherwise it replaces the first element in the queue with the selected element.
If the memqueue isn’t full it appends the selected element at the end.

2.5 enhanced_second_chance Function

The enhanced_second_chance function applies the Enhanced Second Chance algorithm, in this algorithm each page will have two bits added to them one used bit (same as second_chance) and the other is the modified bit this bit will be put randomly.

the priority for replacement is as follows:

1. (0, 0) neither recently used nor modified—best page to replace.
2. (0, 1) not recently used but modified—not quite as good, because the page will need to be written out before replacement.
3. (1, 0) recently used but clean—probably will be used again soon.
4. (1, 1) recently used and modified—probablywill be used again soon, and the page will be need to be written out to disk before it can be replaced.

The function creates 4 memqueues (one for each priority), a selectdict dictionary which will contain a list (used bit, modified bit).
Then, it loops over the reference string.
If the page selected exists in the queue00, it changes the reference bit to 1 and moves the page to queue10.
If the page selected exists in the queue01, it changes the reference bit to 1 and moves it to queue11.
If the page selected exits in queue10 or queue11 it does nothing.
If the selected page isn't in any queue and the length of the 4 queues is less than the frame_num variable (there's room for extra pages), then it adds the page to the appropriate queue according to the selectdict.
If there's no room in the memory, it will start the replacement :
look in the queue00 if it isn't empty it replaces the first element there with the new page.
If queue00 is empty, it checks queue01 , if it isn't empty it replaces the first page.
If queue01 is empty it checks for queue10, if it isn't empty it takes the first page and moves all the pages to queue00 and changes the select bit of each to 0.
If queue10 is empty it checks for queue11, if it isn't empty it takes the first page and moves all the pages to queue01 and changes the select bit of each to 0.
Then it loops again on the first and second queue which now won't be empty and makes the page replacement.

2.6 optimal Function

The optimal function implements the optimal algorithm, in this algorithm we will calculate the frequency of each page used from the reference string before we start the replacement, and when we need to replace a page we replace the least frequently used along the whole reference string.
The function creates a frequency dictionary and loops over the reference string to add it to the dictionary and it's corresponding frequency.
Then, it loops over the reference string and if the memory isn't yet filled it add the selected page to it.
If the memory has no empty frame, it creates loops over the frames and finds the one with the smallest frequency and this becomes the victim page and is replaced with the selected page.

2.7 printing Function

The printing function converts the list that represents the reference string to a string that could be readable to the user.

2.8 main Function

This is the function that runs the show.
Firstly, it generates random number representing the number of pages (0,99), random number representing the number of frames (1,20), and a random number representing the length of the reference string (10,50).
Secondly, it generates random reference string bounded to select within the range(1,number of pages) and it's length will be bounded by the randomly generated length of the reference string.
Thirdly the function prints to the screen the pages' number and frames' number and the reference string(using the printing function see sec 2.7).
Lastly, the function calls all the 6 function performing the 6 algorithms and prints the results of each iteration and page faults of each algorithm.

3. Assumptions

- The number of pages is between 0 and 99 (from the document).
- The number of frames is between 1 and 20 (from the document).
- The number of requests will not be less than 10 and not exceed 50, we used this assumption so that it would be readable to the user.

4. Usage

The program is run from the command line example : python3 project.py
The output of the program is a little bit long so we advise you to output to a file.
Example : python3 project.py > output.py

5. Test Cases

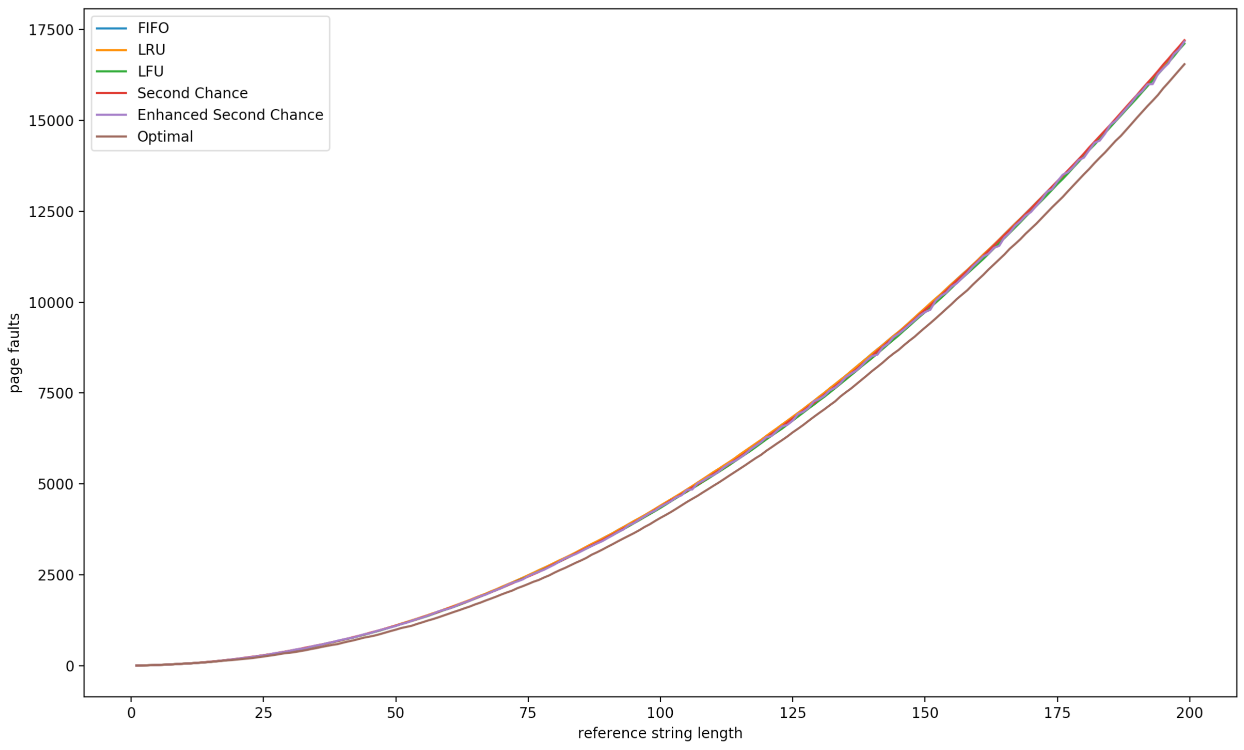
```
the page number  : 11
the frame number : 5
the reference string length : 15
the reference string 2-->2-->3-->4-->5-->6-->6-->3-->5-->7-->8

Using FIFO :
-1 MEANS AN EMPTY FRAME
the Memory : [2, -1, -1, -1, -1]
the Memory : [2, 3, -1, -1, -1]
the Memory : [2, 3, 4, -1, -1]
the Memory : [2, 3, 4, 5, -1]
the Memory : [2, 3, 4, 5, 6]
the Memory : [7, 3, 4, 5, 6]
the Memory : [7, 8, 4, 5, 6]
The Page Faults = 7

Using LRU :
-1 MEANS AN EMPTY FRAME
```

```
the Memory : [2, -1, -1, -1, -1]
the Memory : [2, 3, -1, -1, -1]
the Memory : [2, 3, 4, -1, -1]
the Memory : [2, 3, 4, 5, -1]
the Memory : [2, 3, 4, 5, 6]
the Memory : [7, 3, 4, 5, 6]
the Memory : [7, 3, 8, 5, 6]
The Page Faults = 7

Using LFU :
-1 MEANS AN EMPTY FRAME
the Memory : [2, -1, -1, -1, -1]
the Memory : [2, 3, -1, -1, -1]
the Memory : [2, 3, 4, -1, -1]
the Memory : [2, 3, 4, 5, -1]
the Memory : [2, 3, 4, 5, 6]
the Memory : [2, 3, 7, 5, 6]
the Memory : [2, 3, 8, 5, 6]
The Page Faults = 7

Using Second-Chance :

The used and modified bits of each page in the system :
{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2, 3]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2, 3, 4]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2, 3, 4, 5]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2, 3, 4, 5, 6]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 0, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2, 3, 4, 5, 6]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 1, 4: 0, 5: 0, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2, 3, 4, 5, 6]

The used and modified bits of each page in the system :
{1: 0, 2: 1, 3: 1, 4: 0, 5: 1, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [2, 3, 4, 5, 6]

The used and modified bits of each page in the system :
{1: 0, 2: 0, 3: 0, 4: 0, 5: 1, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [7, 5, 6, 2, 3]

The used and modified bits of each page in the system :
{1: 0, 2: 0, 3: 0, 4: 0, 5: 1, 6: 1, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0}
the Memory : [8, 5, 6, 2, 3]

The Page Faults = 7

Using enhanced Second Chance :

The used and modified bits of each page in the system :
{1: [0, 0], 2: [0, 0], 3: [0, 0], 4: [0, 1], 5: [0, 1], 6: [0, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : [2]
queue01 : []
queue10 : []
queue11 : []

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [0, 0], 4: [0, 1], 5: [0, 1], 6: [0, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : []
queue01 : []
queue10 : [2]
queue11 : []

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [0, 0], 4: [0, 1], 5: [0, 1], 6: [0, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
```

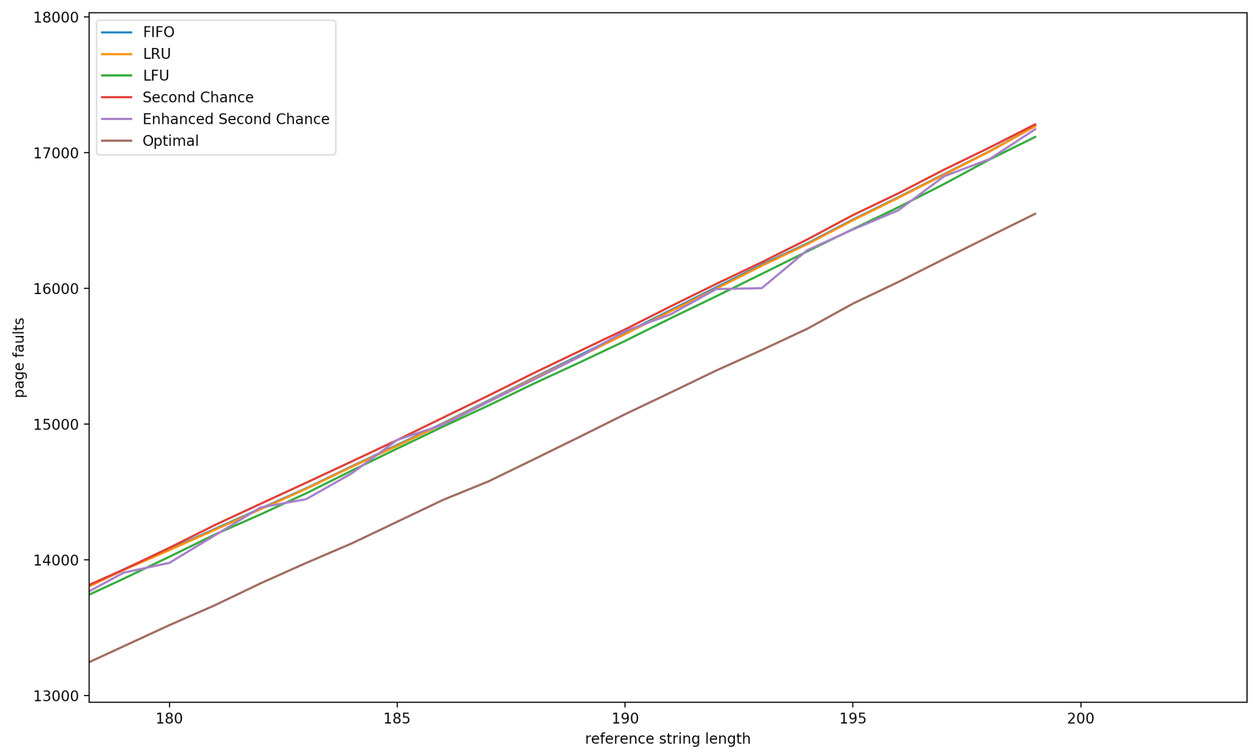
```
the Memory :
 queue00 : [3]
queue01 : []
queue10 : [2]
queue11 : []

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [0, 0], 4: [0, 1], 5: [0, 1], 6: [0, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : [3]
queue01 : [4]
queue10 : [2]
queue11 : []

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [0, 0], 4: [0, 1], 5: [0, 1], 6: [0, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : [3]
queue01 : [4, 5]
queue10 : [2]
queue11 : []

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [0, 0], 4: [0, 1], 5: [0, 1], 6: [0, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : [3]
queue01 : [4, 5, 6]
queue10 : [2]
queue11 : []

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [0, 0], 4: [0, 1], 5: [0, 1], 6: [1, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : [3]
queue01 : [4, 5]
queue10 : [2]
queue11 : [6]

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [1, 0], 4: [0, 1], 5: [0, 1], 6: [1, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : []
queue01 : [4, 5]
queue10 : [2, 3]
queue11 : [6]

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [1, 0], 4: [0, 1], 5: [1, 1], 6: [1, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : []
queue01 : [4]
queue10 : [2, 3]
queue11 : [6, 5]

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [1, 0], 4: [0, 1], 5: [1, 1], 6: [1, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}

the Memory :
 queue00 : [7]
queue01 : []
queue10 : [2, 3]
queue11 : [6, 5]

The used and modified bits of each page in the system :
{1: [0, 0], 2: [1, 0], 3: [1, 0], 4: [0, 1], 5: [1, 1], 6: [1, 1], 7: [0, 0], 8: [0, 0], 9: [0, 0], 10: [0, 0],
11: [0, 1]}
the Memory :
 queue00 : [8]
queue01 : []
queue10 : [2, 3]
queue11 : [6, 5]

The Page Faults = 7

Using Optimal :
the Memory : [2]
the Memory : [2, 3]
the Memory : [2, 3, 4]
the Memory : [2, 3, 4, 5]
the Memory : [2, 3, 4, 5, 6]
the Memory : [2, 3, 7, 5, 6]
the Memory : [2, 3, 8, 5, 6]
The Page Faults = 7
```

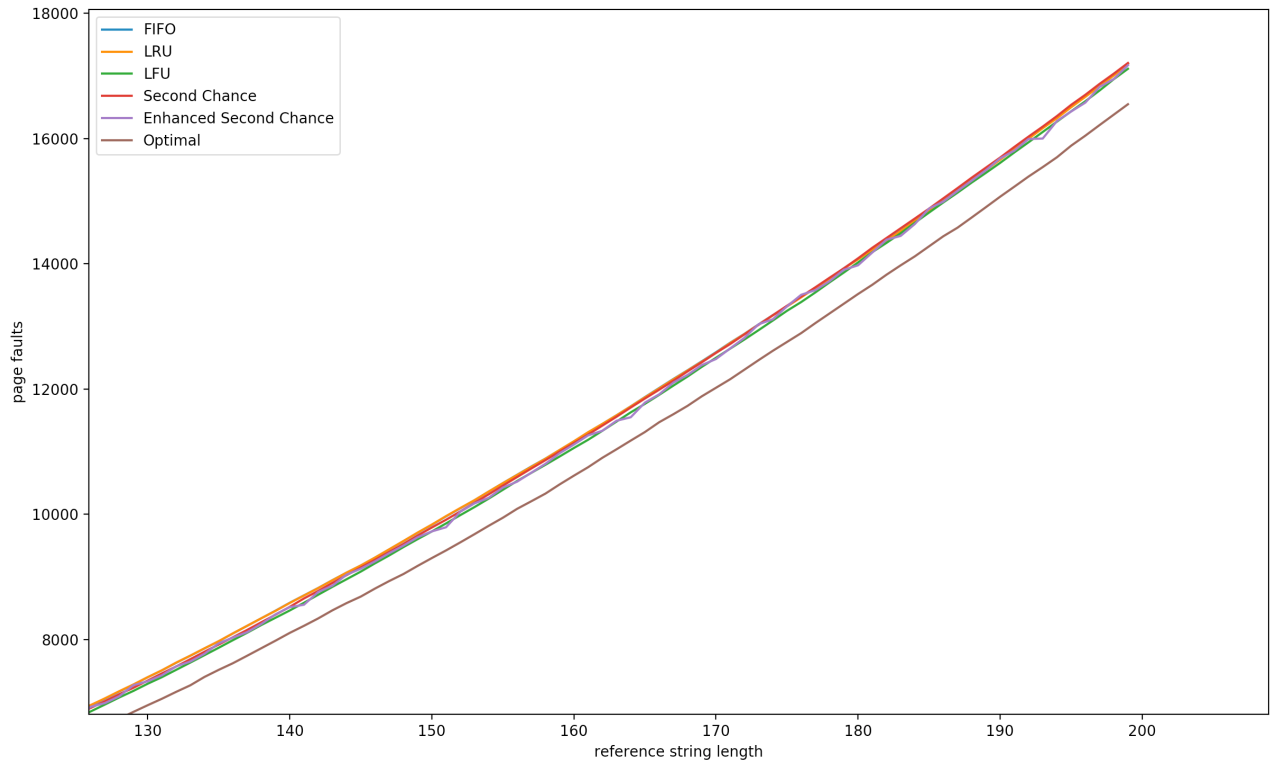
6. The Graphical Results

After implementing the algorithms we took the liberty to plot a graph using the Pylab module representing the difference in page faults when the length of the reference string increases, the code used is commented out in the file attached.
We fixed the number of pages to be 200 and the number of frames to be 55.
The results below show that the optimal algorithm deviates as we increase the length of the reference string, while all of the other algorithm don't show a significant change.
keep in mind that the reference string is still generated randomly we just controlled it's length.



.


FIFO
LRU
LFU
Second Chance
Enhanced Second Chance
Optimal
page faults
reference string length
18000
17000
16000
15000
14000
13000
180
185
190
195
200


FIFO
LRU
LFU
Second Chance
Enhanced Second Chance
Optimal
page faults
reference string length
18000
16000
14000
12000
10000
8000
130
140
150
160
170
180
190
200