

# Implementation Details and Source Code for OS Practical Report

---

## Source Code

### LRU

```
1  import java.util.HashSet;
2  import java.util.LinkedList;
3
4  public class LruMMU implements MMU {
5      private int frames;
6      private boolean debug;
7      private int diskReads;
8      private int diskWrites;
9      private int pageFaults;
10     private int pageHits;
11     private LinkedList<Integer> pageQueue;
12     private HashSet<Integer> memorySet;
13     //private HashSet<Integer> diskSet;
14     private HashSet<Integer> dirtyPages; // This will store pages in memory that
are written to.
15
16
17
18     public LruMMU(int frames) {
19         this.frames = frames;
20         this.pageQueue = new LinkedList<>();
21         this.memorySet = new HashSet<>();
22         //this.diskSet = new HashSet<>();
23         this.dirtyPages = new HashSet<>();
24         this.diskReads = 0;
25         this.diskWrites = 0;
26         this.pageFaults = 0;
27         this.debug = false;
28         this.pageHits = 0;
29     }
30
31     public void setDebug() {
32         this.debug = true;
33     }
34
35     public void resetDebug() {
36         this.debug = false;
37     }
38
39     public void readMemory(int page_number) {
```

```

40     if (!memorySet.contains(page_number)) {
41         debugPrint("Page fault" + page_number);
42         pageFaults++;
43         diskReads++;
44         if (memorySet.size() == frames) {
45             int evictedPage = pageQueue.removeFirst();
46             memorySet.remove(evictedPage);
47             //diskSet.add(evictedPage);
48             if (dirtyPages.contains(evictedPage)) {
49                 diskWrites++;
50                 debugPrint("disk write" + evictedPage);
51                 dirtyPages.remove(evictedPage);
52             }
53         }
54         pageQueue.addLast(page_number);
55         memorySet.add(page_number);
56     } else {
57         pageHits++;
58         pageQueue.remove(Integer.valueOf(page_number));
59         pageQueue.addLast(page_number);
60     }
61     debugPrint("reading" + page_number);
62 }
63
64 public void writeMemory(int page_number) {
65     if (!memorySet.contains(page_number)) {
66         debugPrint("Page fault" + page_number);
67         debugPrint("Writing" + page_number);
68         pageFaults++;
69         //dirtyPages.add(page_number); // Mark the page as dirty.
70         diskReads++;
71         dirtyPages.add(page_number);
72         if (memorySet.size() == frames) {
73             int evictedPage = pageQueue.removeFirst();
74             memorySet.remove(evictedPage);
75             if (dirtyPages.contains(evictedPage)) {
76                 diskWrites++;
77                 debugPrint("disk write" + evictedPage);
78                 dirtyPages.remove(evictedPage);
79             }
80         }
81         memorySet.add(page_number);
82         pageQueue.addLast(page_number);
83     } else {
84         pageHits++;
85         pageQueue.remove(Integer.valueOf(page_number));
86         dirtyPages.add(page_number); // Mark the page as dirty.
87         pageQueue.addLast(page_number);
88         debugPrint("writting" + page_number);

```

```

89     }
90 }
91
92 public int getTotalDiskReads() {
93     return diskReads;
94 }
95
96 public int getTotalDiskWrites() {
97     return this.diskWrites;
98 }
99
100 public int getTotalPageFaults() {
101     return pageFaults;
102 }
103 public int getPageHits() {
104     return this.pageHits;
105 }
106
107 protected void debugPrint(String message) {
108     if (debug) {
109         System.out.println(message);
110     }
111 }
112 }
113

```

## Clock

```

1  import java.util.HashMap;
2  import java.util.HashSet;
3
4  public class ClockMMU implements MMU {
5      private int frames;
6      private boolean debug;
7      private int diskReads;
8      private int diskWrites;
9      private int pageFaults;
10     private int pageHits;
11     private int clockHand;
12     private int insertIndex;
13     private HashMap<Integer, Integer> pageSet; // Store page and its use bit
14     private HashSet<Integer> dirtyPages;
15     private int[] pageTable;
16
17     public ClockMMU(int frames) {
18         this.frames = frames;
19         this.pageSet = new HashMap<>();
20         this.diskReads = 0;
21         this.diskWrites = 0;

```

```

22         this.pageFaults = 0;
23         this.debug = false;
24         this.dirtyPages = new HashSet<>();
25         this.clockHand = 0;
26         this.insertIndex = 0;
27         this.pageHits = 0;
28         pageTable = new int[frames];
29         for (int i = 0; i < frames; i++) {
30             pageTable[i] = -1;
31         }
32     }
33
34     public void setDebug() {
35         this.debug = true;
36     }
37
38     public void resetDebug() {
39         this.debug = false;
40     }
41
42     private void evictAndReplace(int page_number) {
43         while (true) {
44             int page = pageTable[clockHand];
45             if (pageSet.get(page) == 0) {
46                 // Check if the page is dirty before evicting
47                 if (dirtyPages.contains(page)) {
48                     diskWrites++;
49                     debugPrint("Disk write " + page);
50                     dirtyPages.remove(page);
51                 }
52                 // Evict the page
53                 pageSet.remove(page);
54                 pageTable[clockHand] = page_number;
55                 pageSet.put(page_number, 1);
56                 break;
57             } else {
58                 pageSet.put(page, 0);
59             }
60             clockHand = (clockHand + 1) % frames; // Move clock hand in a circular
manner
61         }
62     }
63
64     public void readMemory(int page_number) {
65         if (!pageSet.containsKey(page_number)) {
66             debugPrint("Page fault " + page_number);
67             pageFaults++;
68             diskReads++;
69             if (pageSet.size() == frames) {

```

```

70         evictAndReplace(page_number);
71     }
72     pageTable[clockHand] = page_number;
73     pageSet.put(page_number, 1);
74     clockHand = (clockHand + 1) % frames;
75 } else {
76     pageHits++;
77     pageSet.put(page_number, 1);
78 }
79 debugPrint("Reading " + page_number);
80
81 //System.out.println(diskReads + " " + diskWrites + " " + pageFaults);
82 }
83
84 public void writeMemory(int page_number) {
85     if (!pageSet.containsKey(page_number)) {
86         debugPrint("Page fault " + page_number);
87         pageFaults++;
88         diskReads++;
89         if (pageSet.size() == frames) {
90             evictAndReplace(page_number);
91         }
92         pageTable[clockHand] = page_number;
93         pageSet.put(page_number, 1);
94         clockHand = (clockHand + 1) % frames;
95     } else {
96         pageHits++;
97         pageSet.put(page_number, 1);
98     }
99     dirtyPages.add(page_number);
100    debugPrint("Writing " + page_number);
101
102    //System.out.println(diskReads + " " + diskWrites + " " + pageFaults);
103 }
104
105 public int getTotalDiskReads() {
106     return diskReads;
107 }
108
109 public int getTotalDiskWrites() {
110     return diskWrites;
111 }
112
113 public int getTotalPageFaults() {
114     return pageFaults;
115 }
116
117 @Override
118 public int getPageHits() {

```

```

119         return pageHits;
120     }
121
122     protected void debugPrint(String message) {
123         if (debug) {
124             System.out.println(message);
125         }
126     }
127 }
128

```

## Rand

```

1  import java.util.HashSet;
2  import java.util.Random;
3
4  /**
5   * MMU using random selection replacement strategy
6   */
7
8  public class RandMMU implements MMU {
9      private int frames;
10     private boolean debug;
11     private Random random;
12     private HashSet<Integer> pageSet; // To store currently loaded pages
13     private int diskReads;
14     private int diskWrites;
15     private int pageFaults;
16     private int pageHits;
17     private HashSet<Integer> dirtyPages;
18     public RandMMU(int frames) {
19         //todo
20         this.frames = frames;
21         this.random = new Random();
22         this.pageSet = new HashSet<>();
23         this.diskReads = 0;
24         this.diskWrites = 0;
25         this.pageFaults = 0;
26         this.pageHits = 0;
27         this.debug = false;
28         this.dirtyPages = new HashSet<>();
29     }
30
31     public void setDebug() {
32         //todo
33         this.debug = true;
34     }
35
36     public void resetDebug() {

```

```

37         //todo
38         this.debug = false;
39     }
40
41     public void readMemory(int page_number) {
42         //todo
43         if (!pageSet.contains(page_number)) {
44             debugPrint("Page fault at page " + page_number);
45             pageFaults++;
46             if (pageSet.size() == frames) {
47                 // Randomly select a page for eviction
48                 int evictedPage = (int) pageSet.toArray()
[random.nextInt(pageSet.size())];
49                 if (dirtyPages.contains(evictedPage)) {
50                     diskWrites++;
51                     debugPrint("Disk write " + evictedPage);
52                     dirtyPages.remove(evictedPage);
53                 }
54                 pageSet.remove(evictedPage);
55             }
56             pageSet.add(page_number);
57             diskReads++;
58         } else {
59             pageHits++;
60             debugPrint("Page " + page_number + " found in memory (read).");
61         }
62     }
63
64     public void writeMemory(int page_number) {
65         //todo
66         // This method can be similar to readMemory, but you also account for a
disk write when replacing a "dirty" page.
67         // For simplicity, let's assume every written page is dirty and leads to a
disk write when evicted.
68         dirtyPages.add(page_number);
69         if (!pageSet.contains(page_number)) {
70             pageFaults++;
71             debugPrint("Page fault" + page_number);
72             if (pageSet.size() == frames) {
73                 int evictedPage = (int) pageSet.toArray()
[random.nextInt(pageSet.size())];
74                 if (dirtyPages.contains(evictedPage)) {
75                     diskWrites++;
76                     debugPrint("Disk write " + evictedPage);
77                     dirtyPages.remove(evictedPage);
78                 }
79                 pageSet.remove(evictedPage);
80                 diskWrites++; // Assuming evicted page is dirty after a write
81             }

```

```

82         pageSet.add(page_number);
83         diskReads++;
84     } else {
85         pageHits++;
86     }
87
88 }
89
90 public int getTotalDiskReads() {
91     //todo
92     return diskReads;
93 }
94
95 public int getTotalDiskWrites() {
96     //todo
97     return diskWrites;
98 }
99
100 public int getTotalPageFaults() {
101     return pageFaults;
102 }
103
104 @Override
105 public int getPageHits() {
106     return pageHits;
107 }
108
109 protected void debugPrint(String message) {
110     if (debug) {
111         System.out.println(message);
112     }
113 }
114
115 }

```

## LFU

```

1  import java.util.HashMap;
2  import java.util.HashSet;
3  import java.util.PriorityQueue;
4
5  public class LFUMMU implements MMU {
6      private int frames;
7      private boolean debug;
8      private int diskReads;
9      private int diskWrites;
10     private int pageFaults;
11     private int pageHits;
12

```



```

13     private HashMap<Integer, Integer> frequencyMap; // To store frequency of
pages.
14     private PriorityQueue<Integer> leastFrequentlyUsedQueue;
15     private HashSet<Integer> inMemoryPages;
16     private HashSet<Integer> dirtyPages;
17
18     public LFUMMU(int frames) {
19         this.frames = frames;
20         this.debug = false;
21         this.diskReads = 0;
22         this.diskWrites = 0;
23         this.pageFaults = 0;
24         this.pageHits = 0;
25         this.frequencyMap = new HashMap<>();
26         this.inMemoryPages = new HashSet<>();
27         this.dirtyPages = new HashSet<>();
28         this.leastFrequentlyUsedQueue = new PriorityQueue<>(
29             (a, b) -> frequencyMap.get(a) - frequencyMap.get(b)
30         );
31     }
32
33     private void evictAndReplace(int pageNumber) {
34         int evictPage = leastFrequentlyUsedQueue.poll();
35         inMemoryPages.remove(evictPage);
36         frequencyMap.remove(evictPage);
37         if (dirtyPages.contains(evictPage)) {
38             diskWrites++;
39             dirtyPages.remove(evictPage);
40         }
41         inMemoryPages.add(pageNumber);
42         frequencyMap.put(pageNumber, 1);
43         leastFrequentlyUsedQueue.add(pageNumber);
44     }
45
46     public void readMemory(int pageNumber) {
47         if (!inMemoryPages.contains(pageNumber)) {
48             pageFaults++;
49             diskReads++;
50             if (inMemoryPages.size() == frames) {
51                 evictAndReplace(pageNumber);
52             } else {
53                 inMemoryPages.add(pageNumber);
54                 frequencyMap.put(pageNumber, 1);
55                 leastFrequentlyUsedQueue.add(pageNumber);
56             }
57         } else {
58             pageHits++;
59             frequencyMap.put(pageNumber, frequencyMap.get(pageNumber) + 1);

```

```

60         // We might need to update the priority queue since frequency has
changed
61         leastFrequentlyUsedQueue.remove(pageNumber);
62         leastFrequentlyUsedQueue.add(pageNumber);
63     }
64     debugPrint("Reading " + pageNumber);
65 }
66
67 public void writeMemory(int pageNumber) {
68     if (!inMemoryPages.contains(pageNumber)) {
69         pageFaults++;
70         diskReads++;
71         if (inMemoryPages.size() == frames) {
72             evictAndReplace(pageNumber);
73         } else {
74             inMemoryPages.add(pageNumber);
75             frequencyMap.put(pageNumber, 1);
76             leastFrequentlyUsedQueue.add(pageNumber);
77         }
78     } else {
79         pageHits++;
80         frequencyMap.put(pageNumber, frequencyMap.get(pageNumber) + 1);
81         // We might need to update the priority queue since frequency has
changed
82         leastFrequentlyUsedQueue.remove(pageNumber);
83         leastFrequentlyUsedQueue.add(pageNumber);
84     }
85     dirtyPages.add(pageNumber);
86     debugPrint("Writing " + pageNumber);
87 }
88
89 @Override
90 public void setDebug() {
91     this.debug = true;
92 }
93
94 @Override
95 public void resetDebug() {
96     this.debug = false;
97 }
98
99 public int getTotalDiskReads() {
100     return diskReads;
101 }
102
103 public int getTotalDiskWrites() {
104     return diskWrites;
105 }
106

```

```

107     public int getTotalPageFaults() {
108         return pageFaults;
109     }
110
111     @Override
112     public int getPageHits() {
113         return pageHits;
114     }
115
116     protected void debugPrint(String message) {
117         if (debug) {
118             System.out.println(message);
119         }
120     }
121 }
122

```

## FIFO

```

1  import java.util.HashSet;
2  import java.util.LinkedList;
3  import java.util.Queue;
4
5  public class FifoMMU implements MMU{
6      private int frames;
7      private boolean debug;
8      private int diskReads;
9      private int diskWrites;
10     private int pageFaults;
11     private int pageHits;
12     private Queue<Integer> pageQueue;
13     private HashSet<Integer> memorySet;
14     private HashSet<Integer> dirtyPages;
15
16     public FifoMMU(int frames) {
17         this.frames = frames;
18         this.pageQueue = new LinkedList<>();
19         this.memorySet = new HashSet<>();
20         this.dirtyPages = new HashSet<>();
21         this.diskReads = 0;
22         this.diskWrites = 0;
23         this.pageFaults = 0;
24         this.pageHits = 0;
25         this.debug = false;
26     }
27
28     @Override
29     public void readMemory(int page_number) {
30         if (!memorySet.contains(page_number)) {

```

```

31         debugPrint("Page fault          " + page_number);
32         pageFaults++;
33         diskReads++;
34         if (memorySet.size() == frames) {
35             int evictedPage = pageQueue.remove();
36             memorySet.remove(evictedPage);
37             if (dirtyPages.contains(evictedPage)) {
38                 diskWrites++;
39                 debugPrint("disk write          " + evictedPage);
40                 dirtyPages.remove(evictedPage);
41             }
42         }
43         pageQueue.add(page_number);
44         memorySet.add(page_number);
45     } else {
46         pageHits++;
47     }
48     debugPrint("reading          " + page_number);
49 }
50
51
52
53 @Override
54 public void writeMemory(int page_number) {
55     if (!memorySet.contains(page_number)) {
56         debugPrint("Page fault          " + page_number);
57         pageFaults++;
58         diskReads++;
59         if (memorySet.size() == frames) {
60             int evictedPage = pageQueue.remove();
61             memorySet.remove(evictedPage);
62             if (dirtyPages.contains(evictedPage)) {
63                 diskWrites++;
64                 debugPrint("disk write          " + evictedPage);
65                 dirtyPages.remove(evictedPage);
66             }
67         }
68         pageQueue.add(page_number);
69         memorySet.add(page_number);
70     } else {
71         pageHits++;
72     }
73     debugPrint("Writing          " + page_number);
74     dirtyPages.add(page_number);
75 }
76
77 @Override
78 public void setDebug() {
79     this.debug = true;

```

```

80     }
81
82     @Override
83     public void resetDebug() {
84         this.debug = false;
85     }
86
87     @Override
88     public int getTotalDiskReads() {
89         return diskReads;
90     }
91
92     @Override
93     public int getTotalDiskWrites() {
94         return diskWrites;
95     }
96
97     @Override
98     public int getTotalPageFaults() {
99         return pageFaults;
100    }
101
102    @Override
103    public int getPageHits() {
104        return pageHits;
105    }
106
107    private void debugPrint(String s) {
108        if (debug) {
109            System.out.println(s);
110        }
111    }
112 }
113

```

## ARC

```

1  import java.util.HashMap;
2  import java.util.LinkedHashSet;
3
4  public class ARCMMU implements MMU {
5      private int frames;
6      private boolean debug;
7      private int diskReads;
8      private int diskWrites;
9      private int pageFaults;
10     private int pageHits;
11
12     private LinkedHashSet<Integer> T1;

```

```

13     private LinkedHashSet<Integer> T2;
14     private int p; // Target size for T1
15
16     private HashMap<Integer, Boolean> dirtyPages;
17
18     public ARCMMU(int frames) {
19         this.frames = frames;
20         this.debug = false;
21         this.diskReads = 0;
22         this.diskWrites = 0;
23         this.pageFaults = 0;
24         this.pageHits = 0;
25         this.T1 = new LinkedHashSet<>();
26         this.T2 = new LinkedHashSet<>();
27         this.p = 0;
28         this.dirtyPages = new HashMap<>();
29     }
30
31     private void replace(int page) {
32         if (!T1.isEmpty() && (T1.size() > p || (!T2.contains(page) && T1.size() ==
33 p))) {
34             int last = T1.iterator().next();
35             T1.remove(last);
36             if (dirtyPages.containsKey(last)) {
37                 diskWrites++;
38                 dirtyPages.remove(last);
39             }
40         } else {
41             int last = T2.iterator().next();
42             T2.remove(last);
43             if (dirtyPages.containsKey(last)) {
44                 diskWrites++;
45                 dirtyPages.remove(last);
46             }
47         }
48
49     public void readMemory(int page) {
50         if (!T1.contains(page) && !T2.contains(page)) {
51             pageFaults++;
52             diskReads++;
53             if (T1.size() + T2.size() == frames) {
54                 replace(page);
55             }
56             T1.add(page);
57         } else if (T1.contains(page)) {
58             pageHits++;
59             T1.remove(page);
60             T2.add(page);

```

```

61         } else if (T2.contains(page)) {
62             pageHits++;
63             // Already in T2, just update it
64         }
65         if (T1.contains(page) && T2.contains(page)) {
66             if (T1.size() / (double) frames > p / (double) frames) {
67                 p++;
68             } else {
69                 p--;
70             }
71         }
72         debugPrint("Reading " + page);
73     }
74
75     public void writeMemory(int page) {
76         readMemory(page); // Similar logic, but marking page dirty
77         dirtyPages.put(page, true);
78         debugPrint("Writing " + page);
79     }
80
81     @Override
82     public void setDebug() {
83         this.debug = true;
84     }
85
86     @Override
87     public void resetDebug() {
88         this.debug = false;
89     }
90
91     public int getTotalDiskReads() {
92         return diskReads;
93     }
94
95     public int getTotalDiskWrites() {
96         return diskWrites;
97     }
98
99     public int getTotalPageFaults() {
100         return pageFaults;
101     }
102
103     @Override
104     public int getPageHits() {
105         return pageHits;
106     }
107
108     protected void debugPrint(String message) {
109         if (debug) {

```

```

110         System.out.println(message);
111     }
112 }
113 }
114

```

## MMU Interface

```

1  /**
2  * Interface for Memory Management Unit.
3  * The memory management unit should maintain the concept of a page table.
4  * As pages are read and written to, this changes the pages loaded into the
5  * the limited number of frames. The MMU keeps records, which will be used
6  * to analyse the performance of different replacement strategies implemented
7  * for the MMU.
8  */
9
10 public interface MMU {
11     public void readMemory(int page_number);
12     public void writeMemory(int page_number);
13
14     public void setDebug();
15     public void resetDebug();
16
17     public int getTotalDiskReads();
18     public int getTotalDiskWrites();
19     public int getTotalPageFaults();
20
21     public int getPageHits();
22 }

```

## Entry Program

```

1  import java.io.BufferedReader;
2  import java.io.FileReader;
3
4  public class Memsim {
5      public static void main(String[] args) {
6          int page_offset = 12;          // page is 2^12 = 4KB
7
8          int frames;
9          BufferedReader input = null;
10         MMU mmu = null;
11
12         /* read parameters */
13         //the file
14         try {
15             input = new BufferedReader(new FileReader(args[0]));
16         }

```



```

17         catch (java.io.FileNotFoundException e) {
18             System.out.println("Input '" + args[0] + "' could not be found");
19             System.out.println("Usage: java Memsim inputfile numberframes
replacementmode debugmode");
20             System.exit(-1);
21         }
22
23         //number of frames
24         frames = Integer.parseInt(args[1]);
25
26         //the replacement mode
27         if (args[2].equals("rand"))
28             mmu = new RandMMU(frames);
29         else if (args[2].equals("lru"))
30             mmu = new LruMMU(frames);
31         else if (args[2].equals("clock"))
32             mmu = new ClockMMU(frames);
33         else if (args[2].equals("fifo"))
34             mmu = new FifoMMU(frames);
35         else if (args[2].equals("lfu"))
36             mmu = new LFUMMU(frames);
37         else if (args[2].equals("arc"))
38             mmu = new ARCMMU(frames);
39         else {
40             System.out.println("Usage: java Memsim inputfile numberframes
replacementmode debugmode");
41             System.out.println("replacementmodes are [ rand | lru | esc ]");
42             System.exit(-1);
43         }
44
45         //debug mode?
46         if (args[3].equals("debug"))
47             mmu.setDebug();
48         else if (args[3].equals("quiet"))
49             mmu.resetDebug();
50         else {
51             System.out.println("Usage: java Memsim inputfile numberframes
replacementmode debugmode");
52             System.out.println("debugmode are [ debug | quiet ]");
53             System.exit(-1);
54         }
55
56         /* Process the traces from the file */
57         String traceLine;
58         String[] traceCmd;
59         long logical_address;
60         int page_number;
61         int no_events = 0;
62

```

```

63     try {
64         traceLine = input.readLine();
65         while (traceLine != null) {
66             traceCmd = traceLine.split(" ");
67
68             //convert from hexadecimal address from file, to appropriate page
number
69             logical_address = Long.parseLong(traceCmd[0],16);
70             page_number = (int) (logical_address >>> page_offset);
71
72             //process read or write
73             if (traceCmd[1].equals("R"))
74                 mmu.readMemory(page_number);
75             else if (traceCmd[1].equals("W"))
76                 mmu.writeMemory(page_number);
77             else {
78                 System.out.println("Badly formatted file. Error on line " +
(no_events+1));
79                 System.exit(-1);
80             }
81
82             no_events++;
83             traceLine = input.readLine();
84         }
85     }
86     catch (java.io.IOException e) {
87         System.out.println("Error reading input file");
88         System.exit(-1);
89     }
90     catch (NumberFormatException e) {
91         System.out.println("Memory address strange on line " + (no_events+1));
92         System.exit(-1);
93     }
94
95     /* Print results */
96     System.out.println("total memory frames: " + frames);
97     System.out.println("events in trace: " + no_events);
98     System.out.println("total disk reads: " + mmu.getTotalDiskReads());
99     System.out.println("total disk writes: " + mmu.getTotalDiskWrites());
100    System.out.printf("page fault rate: %.4f\n", ((double)
mmu.getTotalPageFaults())/no_events);
101    System.out.println("total page hits: " + mmu.getPageHits());
102    System.out.println("hit rate: " + ((double) mmu.getPageHits())/no_events);
103    //System.out.println("page fault rate: " + ((double)
mmu.getTotalPageFaults())/no_events);
104    }
105
106 }
107

```

## Results:

Using Machine Learning built-in library to fit the data. Because the way I use my simulator, I can only generate limited data, so I use Machine Learning technique to fit the model and get a smoother graph.

All the data used below is manually typed.

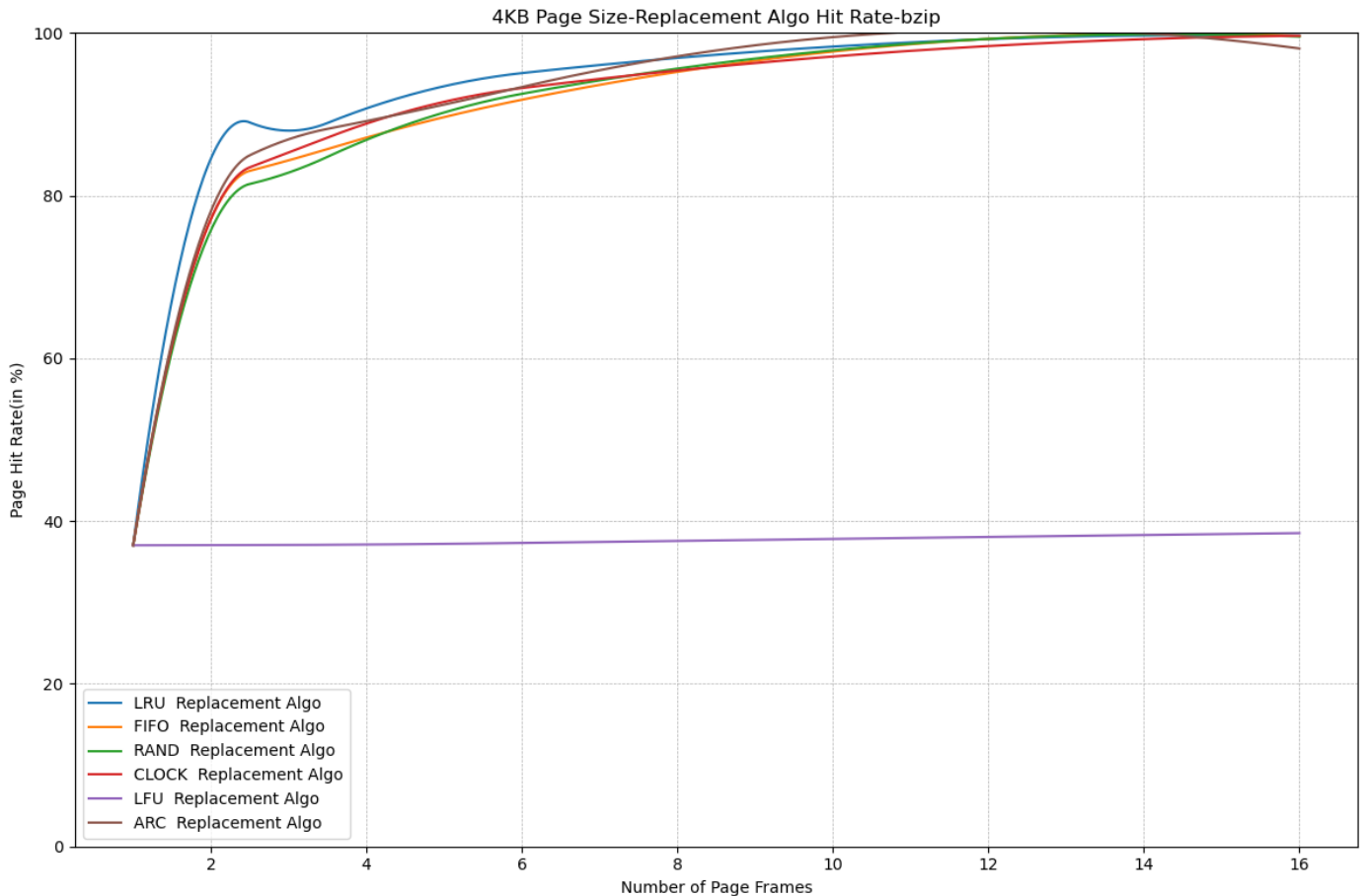
Page size = 4KB, bzip trace

```
1  #bzip
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from scipy.interpolate import make_interp_spline
5
6  # Data
7  datasets = {
8      "LRU": ([1, 2, 3, 4, 8, 16], [37.0263, 84.5571, 87.9896, 90.723, 96.9309,
99.6656]),
9      "FIFO": ([1, 2, 3, 4, 8, 16], [37.0263, 77.1162, 84.3426, 87.1399, 95.2172,
99.618]),
10     "RAND": ([1, 2, 3, 4, 8, 16], [37.0263, 75.758, 82.8353, 86.8604, 95.6176,
99.5504]),
11     "CLOCK": ([1, 2, 3, 4, 8, 16], [37.0263, 77.1162, 85.3077, 88.8558, 95.3836,
99.6532]),
12     "LFU": ([1, 2, 3, 4, 8, 16], [37.0263, 37.0488, 37.0635, 37.1146, 37.5611,
38.5195]),
13     "ARC": ([1, 2, 3, 4, 8, 16], [37.0263, 78.1058, 86.9301, 89.1687, 97.1275,
98.0932])
14 }
15
16 # Plotting
17 plt.figure(figsize=(12,8))
18
19 # Iterate through each dataset, interpolate, and plot
20 for label, (X, Y) in datasets.items():
21     X_new = np.linspace(min(X), max(X), 500)
22     spl = make_interp_spline(X, Y, k=2)
23     Y_new = spl(X_new)
24
25     # Scatter and line plot
26     #plt.scatter(X, Y, s=50, label=f'{label} Data Points') # s=50 for bigger
markers
27     plt.plot(X_new, Y_new, linestyle='-', label=f'{label} Replacement Algo')
28
29 # Axes, Title, Grid, and Legend
30 plt.xlabel('Number of Page Frames')
31 plt.ylabel('Page Hit Rate(in %)')
32 plt.title('4KB Page Size-Replacement Algo Hit Rate')
```

```

33 plt.grid(True, which='both', linestyle='--', linewidth=0.5)
34 plt.ylim(0, 100) # Adjusted the maximum value of y to 100
35 plt.legend(loc='lower left')
36 plt.tight_layout()
37 plt.show()
38

```



Page size = 4KB, gcc trace

```

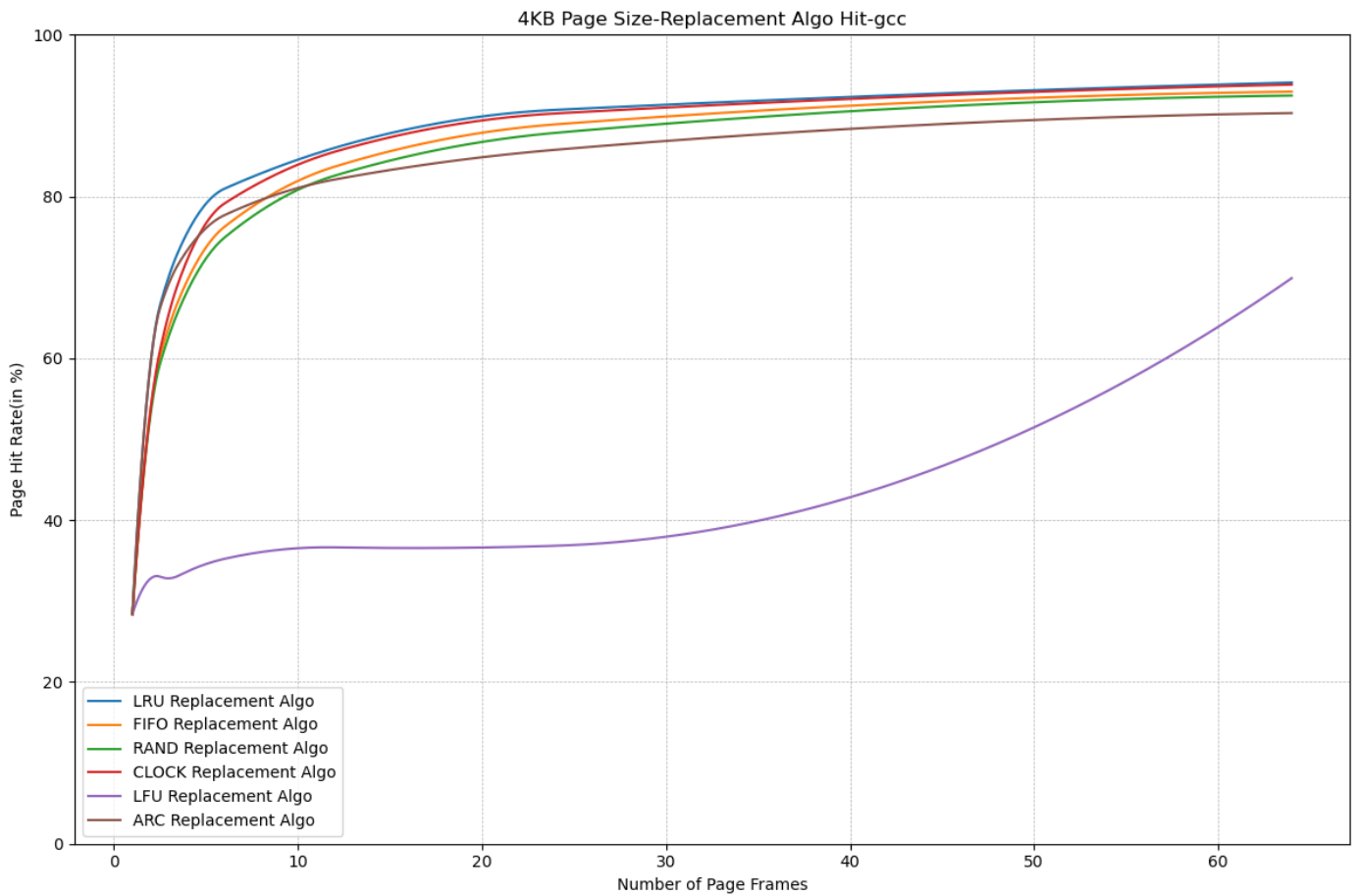
1 #gcc
2 import matplotlib.pyplot as plt
3 import numpy as np
4 from scipy.interpolate import make_interp_spline
5
6 # Data
7 datasets = {
8     "LRU": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 59.6045, 70.2867, 75.6191,
9         82.8814, 88.3396, 91.5599, 94.0911]),
10    "FIFO": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 53.9088, 64.0434, 69.714,
11        79.4632, 86.1461, 90.1933, 92.9658]),
12    "RAND": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 53.2855, 62.9108, 68.3930,
13        78.2784, 84.9899, 89.3481, 92.4822]),
14    "CLOCK": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 53.9088, 65.7005, 72.3032,
15        81.8144, 87.8318, 91.2314, 93.836]),
16

```

```

12     "LFU": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 32.8122, 32.8155, 33.6793,
36.0637, 36.5566, 38.6289, 69.9115]),
13     "ARC": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 59.6207, 69.3614, 73.42,
79.5941, 83.6352, 87.2037, 90.3199])
14 }
15
16 # Plotting
17 plt.figure(figsize=(12,8))
18
19 # Iterate through each dataset, interpolate, and plot
20 for label, (X, Y) in datasets.items():
21     X_new = np.linspace(min(X), max(X), 500)
22     spl = make_interp_spline(X, Y, k=2)
23     Y_new = spl(X_new)
24
25     # Scatter and line plot
26     #plt.scatter(X, Y, s=50, label=f'{label} Data Points') # s=50 for bigger
markers
27     plt.plot(X_new, Y_new, linestyle='-', label=f'{label} Replacement Algo')
28
29 # Axes, Title, Grid, and Legend
30 plt.xlabel('Number of Page Frames')
31 plt.ylabel('Page Hit Rate(in %)')
32 plt.title('4KB Page Size-Replacement Algo Hit-gcc')
33 plt.grid(True, which='both', linestyle='--', linewidth=0.5)
34 plt.ylim(0, 100) # Adjusted the maximum value of y to 100
35 plt.legend(loc='lower left')
36 plt.tight_layout()
37 plt.show()
38

```



```

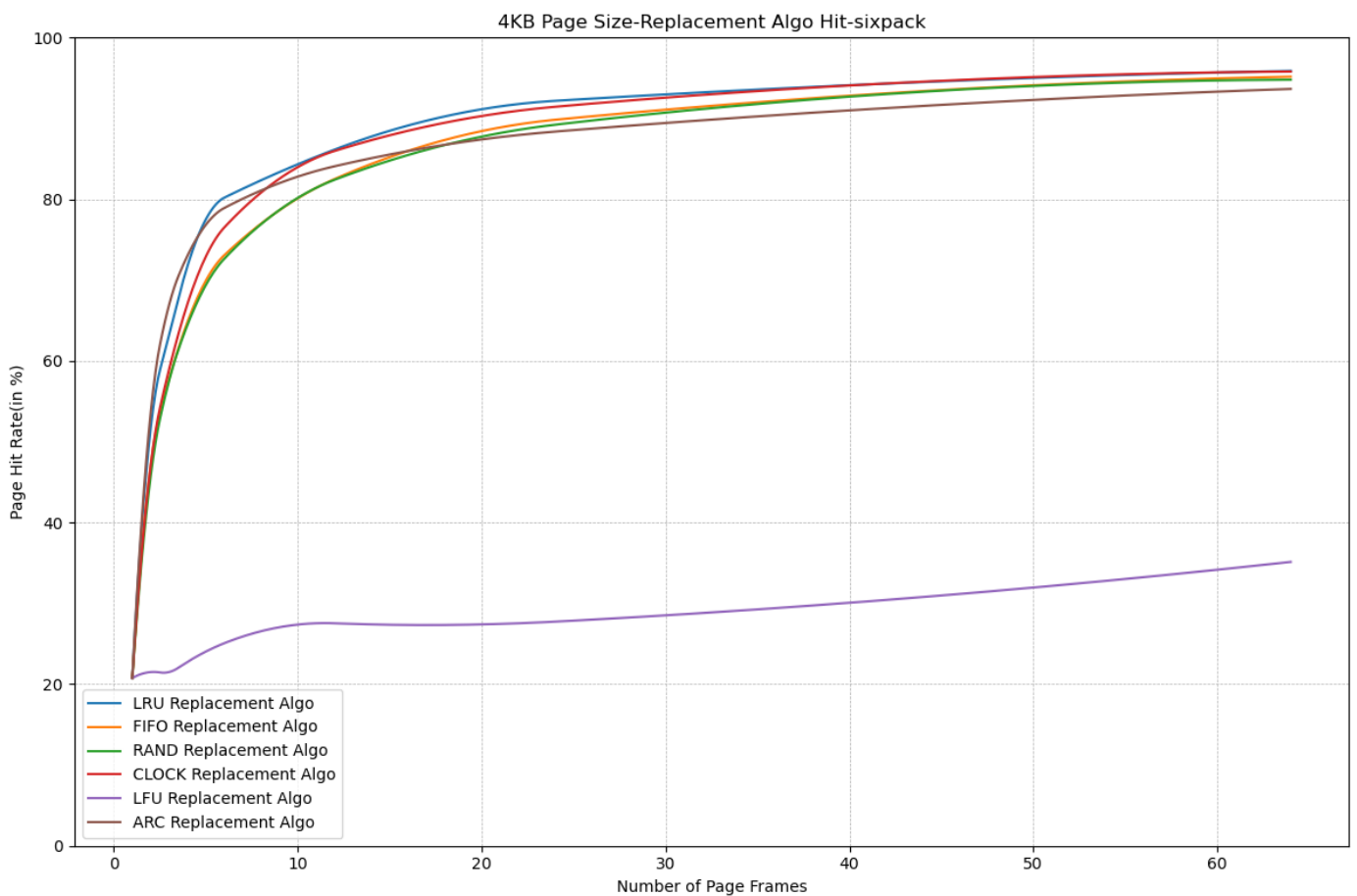
1  #sixpack
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from scipy.interpolate import make_interp_spline
5
6  # Data
7  datasets = {
8      "LRU": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 51.6839, 63.2442, 71.738,
9          82.3504, 89.1318, 93.2253, 95.8814]),
10     "FIFO": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 47.0763, 57.7565, 64.819,
11         76.9832, 85.9917, 91.4717, 95.1699]),
12     "RAND": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 45.7322, 57.5457, 64.4536,
13         76.9125, 85.4826, 91.1693, 94.7933]),
14     "CLOCK": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 47.0763, 58.9645, 67.0632,
15         80.824, 88.4794, 92.9151, 95.8051]),
16     "LFU": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 21.4984, 21.5024, 22.76,
17         26.5544, 27.3282, 28.8034, 35.1312]),
18     "ARC": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 53.6075, 66.9364, 73.0243,
19         81.084, 85.9731, 89.7762, 93.6508])
20 }
21
22 # Plotting
23 plt.figure(figsize=(12,8))
24
25 # Iterate through each dataset, interpolate, and plot

```

```

20 for label, (X, Y) in datasets.items():
21     X_new = np.linspace(min(X), max(X), 500)
22     spl = make_interp_spline(X, Y, k=2)
23     Y_new = spl(X_new)
24
25     # Scatter and line plot
26     #plt.scatter(X, Y, s=50, label=f'{label} Data Points') # s=50 for bigger
markers
27     plt.plot(X_new, Y_new, linestyle='-', label=f'{label} Replacement Algo')
28
29 # Axes, Title, Grid, and Legend
30 plt.xlabel('Number of Page Frames')
31 plt.ylabel('Page Hit Rate(in %)')
32 plt.title('4KB Page Size-Replacement Algo Hit-sixpack')
33 plt.grid(True, which='both', linestyle='--', linewidth=0.5)
34 plt.ylim(0, 100) # Adjusted the maximum value of y to 100
35 plt.legend(loc='lower left')
36 plt.tight_layout()
37 plt.show()
38

```



```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from scipy.interpolate import make_interp_spline
4

```

```

5 # Data
6 datasets = {
7     "LRU": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.529321,
8         0.608734, 0.653064, 0.714625, 0.828039, 0.951746, 0.978344]]),
9     "FIFO": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.458542,
10         0.535343, 0.580491, 0.669107, 0.785705, 0.918362, 0.969578]]),
11     "RAND": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.443155,
12         0.524444, 0.57203, 0.678965, 0.804642, 0.91622, 0.964585]]),
13     "CLOCK": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.458542,
14         0.564483, 0.619318, 0.706481, 0.808152, 0.946975, 0.977389]]),
15     "LFU": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.554351,
16         0.555903, 0.598963, 0.640912, 0.701694, 0.726324, 0.806565]]),
17     "ARC": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.53276,
18         0.591725, 0.63552, 0.763602, 0.858651, 0.873644, 0.903442]])
19 }
20
21 # Plotting
22 plt.figure(figsize=(12,8))
23
24 # Iterate through each dataset, interpolate, and plot
25 for label, (X, Y) in datasets.items():
26     X_new = np.linspace(min(X), max(X), 500)
27     spl = make_interp_spline(X, Y, k=2)
28     Y_new = spl(X_new)
29
30     # Scatter and line plot
31     #plt.scatter(X, Y, s=50, label=f'{label} Data Points') # s=50 for bigger
32     # markers
33     plt.plot(X_new, Y_new, linestyle='-', label=f'{label} Replacement Algo')
34
35 # Axes, Title, Grid, and Legend
36 plt.xlabel('Number of Page Frames')
37 plt.ylabel('Page Hit Rate(in %)')
38 plt.title('4KB Page Size-Replacement Algo swim')
39 plt.grid(True, which='both', linestyle='--', linewidth=0.5)
40 plt.ylim(0, 100) # Y values are already in percentage
41 plt.legend(loc='lower left')
42 plt.tight_layout()
43 plt.show()
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```



