Implementation Details and Source Code for OS Practical Report

Source Code

LRU

```java
import java.util.HashSet;
import java.util.LinkedList;

public class LruMMU implements MMU {
    private int frames;
    private boolean debug;
    private int diskReads;
    private int diskWrites;
    private int pageFaults;
    private int pageHits;
    private LinkedList<Integer> pageQueue;
    private HashSet<Integer> memorySet;
    //private HashSet<Integer> diskSet;
    private HashSet<Integer> dirtyPages;  // This will store pages in memory that are written to.



    public LruMMU(int frames) {
        this.frames = frames;
        this.pageQueue = new LinkedList<>();
        this.memorySet = new HashSet<>();
        //this.diskSet = new HashSet<>();
        this.dirtyPages = new HashSet<>();
        this.diskReads = 0;
        this.diskWrites = 0;
        this.pageFaults = 0;
        this.debug = false;
        this.pageHits = 0;
    }

    public void setDebug() {
        this.debug = true;
    }

    public void resetDebug() {
        this.debug = false;
    }

    public void readMemory(int page_number) {
        if (!memorySet.contains(page_number)) {
            debugPrint("Page fault        " + page_number);
            pageFaults++;
            diskReads++;
            if (memorySet.size() == frames) {
                int evictedPage = pageQueue.removeFirst();
                memorySet.remove(evictedPage);
                //diskSet.add(evictedPage);
                if (dirtyPages.contains(evictedPage)) {
                    diskWrites++;
                    debugPrint("disk write          " + evictedPage);
                    dirtyPages.remove(evictedPage);
                }
            }
            pageQueue.addLast(page_number);
            memorySet.add(page_number);
        } else {
            pageHits++;
            pageQueue.remove(Integer.valueOf(page_number));
            pageQueue.addLast(page_number);
        }
        debugPrint("reading           " + page_number);
    }

    public void writeMemory(int page_number) {
        if (!memorySet.contains(page_number)) {
            debugPrint("Page fault        " + page_number);
            debugPrint("Writing           " + page_number);
            pageFaults++;
            //dirtyPages.add(page_number);  // Mark the page as dirty.
            diskReads++;
            dirtyPages.add(page_number);
            if (memorySet.size() == frames) {
                int evictedPage = pageQueue.removeFirst();
                memorySet.remove(evictedPage);
                if (dirtyPages.contains(evictedPage)) {
                    diskWrites++;
                    debugPrint("disk write          " + evictedPage);
                    dirtyPages.remove(evictedPage);
                }
            }
            memorySet.add(page_number);
            pageQueue.addLast(page_number);
        } else {
            pageHits++;
            pageQueue.remove(Integer.valueOf(page_number));
            dirtyPages.add(page_number);  // Mark the page as dirty.
            pageQueue.addLast(page_number);
            debugPrint("writting          " + page_number);
        }
    }

    public int getTotalDiskReads() {
        return diskReads;
    }

    public int getTotalDiskWrites() {
        return this.diskWrites;
    }

    public int getTotalPageFaults() {
        return pageFaults;
    }
    public int getPageHits() {
        return this.pageHits;
    }

    protected void debugPrint(String message) {
        if (debug) {
            System.out.println(message);
        }
    }
}

```

Clock

```java
import java.util.HashMap;
import java.util.HashSet;

public class ClockMMU implements MMU {
    private int frames;
    private boolean debug;
    private int diskReads;
    private int diskWrites;
    private int pageFaults;
    private int pageHits;
    private int clockHand;
    private int insertIndex;
    private HashMap<Integer, Integer> pageSet; // Store page and its use bit
    private HashSet<Integer> dirtyPages;
    private int[] pageTable;

    public ClockMMU(int frames) {
        this.frames = frames;
        this.pageSet = new HashMap<>();
        this.diskReads = 0;
        this.diskWrites = 0;
        this.pageFaults = 0;
        this.debug = false;
        this.dirtyPages = new HashSet<>();
        this.clockHand = 0;
        this.insertIndex = 0;
        this.pageHits = 0;
        pageTable = new int[frames];
        for (int i = 0; i < frames; i++) {
            pageTable[i] = -1;
        }
    }

    public void setDebug() {
        this.debug = true;
    }

    public void resetDebug() {
        this.debug = false;
    }

    private void evictAndReplace(int page_number) {
        while (true) {
            int page = pageTable[clockHand];
            if (pageSet.get(page) == 0) {
                // Check if the page is dirty before evicting
                if (dirtyPages.contains(page)) {
                    diskWrites++;
                    debugPrint("Disk write " + page);
                    dirtyPages.remove(page);
                }
                // Evict the page
                pageSet.remove(page);
                pageTable[clockHand] = page_number;
                pageSet.put(page_number, 1);
                break;
            } else {
                pageSet.put(page, 0);
            }
            clockHand = (clockHand + 1) % frames; // Move clock hand in a circular manner
        }
    }

    public void readMemory(int page_number) {
        if (!pageSet.containsKey(page_number)) {
            debugPrint("Page fault " + page_number);
            pageFaults++;
            diskReads++;
            if (pageSet.size() == frames) {
                evictAndReplace(page_number);
            }
            pageTable[clockHand] = page_number;
            pageSet.put(page_number, 1);
            clockHand = (clockHand + 1) % frames;
        } else {
            pageHits++;
            pageSet.put(page_number, 1);
        }
        debugPrint("Reading " + page_number);

        //System.out.println(diskReads + " " + diskWrites + " " + pageFaults);
    }

    public void writeMemory(int page_number) {
        if (!pageSet.containsKey(page_number)) {
            debugPrint("Page fault " + page_number);
            pageFaults++;
            diskReads++;
            if (pageSet.size() == frames) {
                evictAndReplace(page_number);
            }
            pageTable[clockHand] = page_number;
            pageSet.put(page_number, 1);
            clockHand = (clockHand + 1) % frames;
        } else {
            pageHits++;
            pageSet.put(page_number, 1);
        }
        dirtyPages.add(page_number);
        debugPrint("Writing " + page_number);

        //System.out.println(diskReads + " " + diskWrites + " " + pageFaults);
    }

    public int getTotalDiskReads() {
        return diskReads;
    }

    public int getTotalDiskWrites() {
        return diskWrites;
    }

    public int getTotalPageFaults() {
        return pageFaults;
    }

    @Override
    public int getPageHits() {
```

```java
        return pageHits;
    }

    protected void debugPrint(String message) {
        if (debug) {
            System.out.println(message);
        }
    }
}

```

Rand

```java
import java.util.HashSet;
import java.util.Random;

/**
* MMU using random selection replacement strategy
*/

public class RandMMU implements MMU {
    private int frames;
    private boolean debug;
    private Random random;
    private HashSet<Integer> pageSet; // To store currently loaded pages
    private int diskReads;
    private int diskWrites;
    private int pageFaults;
    private int pageHits;
    private HashSet<Integer> dirtyPages;
    public RandMMU(int frames) {
        //todo
        this.frames = frames;
        this.random = new Random();
        this.pageSet = new HashSet<>();
        this.diskReads = 0;
        this.diskWrites = 0;
        this.pageFaults = 0;
        this.pageHits = 0;
        this.debug = false;
        this.dirtyPages = new HashSet<>();
    }

    public void setDebug() {
        //todo
        this.debug = true;
    }

    public void resetDebug() {
        //todo
        this.debug = false;
    }

    public void readMemory(int page_number) {
        //todo
        if (!pageSet.contains(page_number)) {
            debugPrint("Page fault at page " + page_number);
            pageFaults++;
            if (pageSet.size() == frames) {
                // Randomly select a page for eviction
                int evictedPage = (int) pageSet.toArray()[random.nextInt(pageSet.size())];
                if (dirtyPages.contains(evictedPage)) {
                    diskWrites++;
                    debugPrint("Disk write " + evictedPage);
                    dirtyPages.remove(evictedPage);
                }
                pageSet.remove(evictedPage);
            }
            pageSet.add(page_number);
            diskReads++;
        } else {
            pageHits++;
            debugPrint("Page " + page_number + " found in memory (read).");
        }
    }

    public void writeMemory(int page_number) {
        //todo
        // This method can be similar to readMemory, but you also account for a disk write when replacing a "dirty" page.
        // For simplicity, let's assume every written page is dirty and leads to a disk write when evicted.
        dirtyPages.add(page_number);
        if (!pageSet.contains(page_number)) {
            pageFaults++;
            debugPrint("Page fault" + page_number);
            if (pageSet.size() == frames) {
                int evictedPage = (int) pageSet.toArray()[random.nextInt(pageSet.size())];
                if (dirtyPages.contains(evictedPage)) {
                    diskWrites++;
                    debugPrint("Disk write " + evictedPage);
                    dirtyPages.remove(evictedPage);
                }
                pageSet.remove(evictedPage);
                diskWrites++;  // Assuming evicted page is dirty after a write
            }
            pageSet.add(page_number);
            diskReads++;
        } else {
            pageHits++;
        }

    }

    public int getTotalDiskReads() {
        //todo
        return diskReads;
    }

    public int getTotalDiskWrites() {
        //todo
        return diskWrites;
    }

    public int getTotalPageFaults() {
        return pageFaults;
    }

    @Override
    public int getPageHits() {
        return pageHits;
    }

    protected void debugPrint(String message) {
        if (debug) {
            System.out.println(message);
        }
    }

}
```

LFU

```java
import java.util.HashMap;
import java.util.HashSet;
import java.util.PriorityQueue;

public class LFUMMU implements MMU {
    private int frames;
    private boolean debug;
    private int diskReads;
    private int diskWrites;
    private int pageFaults;
    private int pageHits;

    private HashMap<Integer, Integer> frequencyMap;  // To store frequency of pages.
    private PriorityQueue<Integer> leastFrequentlyUsedQueue;
    private HashSet<Integer> inMemoryPages;
    private HashSet<Integer> dirtyPages;

    public LFUMMU(int frames) {
        this.frames = frames;
        this.debug = false;
        this.diskReads = 0;
        this.diskWrites = 0;
        this.pageFaults = 0;
        this.pageHits = 0;
        this.frequencyMap = new HashMap<>();
        this.inMemoryPages = new HashSet<>();
        this.dirtyPages = new HashSet<>();
        this.leastFrequentlyUsedQueue = new PriorityQueue<>(
                (a, b) -> frequencyMap.get(a) - frequencyMap.get(b)
        );
    }

    private void evictAndReplace(int pageNumber) {
        int evictPage = leastFrequentlyUsedQueue.poll();
        inMemoryPages.remove(evictPage);
        frequencyMap.remove(evictPage);
        if (dirtyPages.contains(evictPage)) {
            diskWrites++;
            dirtyPages.remove(evictPage);
        }
        inMemoryPages.add(pageNumber);
        frequencyMap.put(pageNumber, 1);
        leastFrequentlyUsedQueue.add(pageNumber);
    }

    public void readMemory(int pageNumber) {
        if (!inMemoryPages.contains(pageNumber)) {
            pageFaults++;
            diskReads++;
            if (inMemoryPages.size() == frames) {
                evictAndReplace(pageNumber);
            } else {
                inMemoryPages.add(pageNumber);
                frequencyMap.put(pageNumber, 1);
                leastFrequentlyUsedQueue.add(pageNumber);
            }
        } else {
            pageHits++;
            frequencyMap.put(pageNumber, frequencyMap.get(pageNumber) + 1);
            // We might need to update the priority queue since frequency has changed
            leastFrequentlyUsedQueue.remove(pageNumber);
            leastFrequentlyUsedQueue.add(pageNumber);
        }
        debugPrint("Reading " + pageNumber);
    }

    public void writeMemory(int pageNumber) {
        if (!inMemoryPages.contains(pageNumber)) {
            pageFaults++;
            diskReads++;
            if (inMemoryPages.size() == frames) {
                evictAndReplace(pageNumber);
            } else {
                inMemoryPages.add(pageNumber);
                frequencyMap.put(pageNumber, 1);
                leastFrequentlyUsedQueue.add(pageNumber);
            }
        } else {
            pageHits++;
            frequencyMap.put(pageNumber, frequencyMap.get(pageNumber) + 1);
            // We might need to update the priority queue since frequency has changed
            leastFrequentlyUsedQueue.remove(pageNumber);
            leastFrequentlyUsedQueue.add(pageNumber);
        }
        dirtyPages.add(pageNumber);
        debugPrint("Writing " + pageNumber);
    }

    @Override
    public void setDebug() {
        this.debug = true;
    }

    @Override
    public void resetDebug() {
        this.debug = false;
    }

    public int getTotalDiskReads() {
        return diskReads;
    }

    public int getTotalDiskWrites() {
        return diskWrites;
    }

```

```java
    public int getTotalPageFaults() {
        return pageFaults;
    }

    @Override
    public int getPageHits() {
        return pageHits;
    }

    protected void debugPrint(String message) {
        if (debug) {
            System.out.println(message);
        }
    }
}

```

FIFO

```java
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;

public class FifoMMU implements MMU{
    private int frames;
    private boolean debug;
    private int diskReads;
    private int diskWrites;
    private int pageFaults;
    private int pageHits;
    private Queue<Integer> pageQueue;
    private HashSet<Integer> memorySet;
    private HashSet<Integer> dirtyPages;

    public FifoMMU(int frames) {
        this.frames = frames;
        this.pageQueue = new LinkedList<>();
        this.memorySet = new HashSet<>();
        this.dirtyPages = new HashSet<>();
        this.diskReads = 0;
        this.diskWrites = 0;
        this.pageFaults = 0;
        this.pageHits = 0;
        this.debug = false;
    }

    @Override
    public void readMemory(int page_number) {
        if (!memorySet.contains(page_number)) {
            debugPrint("Page fault        " + page_number);
            pageFaults++;
            diskReads++;
            if (memorySet.size() == frames) {
                int evictedPage = pageQueue.remove();
                memorySet.remove(evictedPage);
                if (dirtyPages.contains(evictedPage)) {
                    diskWrites++;
                    debugPrint("disk write          " + evictedPage);
                    dirtyPages.remove(evictedPage);
                }
            }
            pageQueue.add(page_number);
            memorySet.add(page_number);
        } else {
            pageHits++;
        }
        debugPrint("reading           " + page_number);
    }



    @Override
    public void writeMemory(int page_number) {
        if (!memorySet.contains(page_number)) {
            debugPrint("Page fault        " + page_number);
            pageFaults++;
            diskReads++;
            if (memorySet.size() == frames) {
                int evictedPage = pageQueue.remove();
                memorySet.remove(evictedPage);
                if (dirtyPages.contains(evictedPage)) {
                    diskWrites++;
                    debugPrint("disk write          " + evictedPage);
                    dirtyPages.remove(evictedPage);
                }
            }
            pageQueue.add(page_number);
            memorySet.add(page_number);
        } else {
            pageHits++;
        }
        debugPrint("Writing           " + page_number);
        dirtyPages.add(page_number);
    }

    @Override
    public void setDebug() {
        this.debug = true;
    }

    @Override
    public void resetDebug() {
        this.debug = false;
    }

    @Override
    public int getTotalDiskReads() {
        return diskReads;
    }

    @Override
    public int getTotalDiskWrites() {
        return diskWrites;
    }

    @Override
    public int getTotalPageFaults() {
        return pageFaults;
    }

    @Override
    public int getPageHits() {
        return pageHits;
    }

    private void debugPrint(String s) {
        if (debug) {
            System.out.println(s);
        }
    }
}

```

ARC

```
import java.util.HashMap;
import java.util.LinkedHashSet;

public class ARCMMU implements MMU {
    private int frames;
    private boolean debug;
    private int diskReads;
    private int diskWrites;
    private int pageFaults;
    private int pageHits;

    private LinkedHashSet<Integer> T1;
```

```java
    private LinkedHashSet<Integer> T2;
    private int p;  // Target size for T1

    private HashMap<Integer, Boolean> dirtyPages;

    public ARCMMU(int frames) {
        this.frames = frames;
        this.debug = false;
        this.diskReads = 0;
        this.diskWrites = 0;
        this.pageFaults = 0;
        this.pageHits = 0;
        this.T1 = new LinkedHashSet<>();
        this.T2 = new LinkedHashSet<>();
        this.p = 0;
        this.dirtyPages = new HashMap<>();
    }

    private void replace(int page) {
        if (!T1.isEmpty() && (T1.size() > p || (!T2.contains(page) && T1.size() == p))) {
            int last = T1.iterator().next();
            T1.remove(last);
            if (dirtyPages.containsKey(last)) {
                diskWrites++;
                dirtyPages.remove(last);
            }
        } else {
            int last = T2.iterator().next();
            T2.remove(last);
            if (dirtyPages.containsKey(last)) {
                diskWrites++;
                dirtyPages.remove(last);
            }
        }
    }

    public void readMemory(int page) {
        if (!T1.contains(page) && !T2.contains(page)) {
            pageFaults++;
            diskReads++;
            if (T1.size() + T2.size() == frames) {
                replace(page);
            }
            T1.add(page);
        } else if (T1.contains(page)) {
            pageHits++;
            T1.remove(page);
            T2.add(page);
        } else if (T2.contains(page)) {
            pageHits++;
            // Already in T2, just update it
        }
        if (T1.contains(page) && T2.contains(page)) {
            if (T1.size() / (double) frames > p / (double) frames) {
                p++;
            } else {
                p--;
            }
        }
        debugPrint("Reading " + page);
    }

    public void writeMemory(int page) {
        readMemory(page);  // Similar logic, but marking page dirty
        dirtyPages.put(page, true);
        debugPrint("Writing " + page);
    }

    @Override
    public void setDebug() {
        this.debug = true;
    }

    @Override
    public void resetDebug() {
        this.debug = false;
    }

    public int getTotalDiskReads() {
        return diskReads;
    }

    public int getTotalDiskWrites() {
        return diskWrites;
    }

    public int getTotalPageFaults() {
        return pageFaults;
    }

    @Override
    public int getPageHits() {
        return pageHits;
    }

    protected void debugPrint(String message) {
        if (debug) {
            System.out.println(message);
        }
    }
}

```

MMU Interface

```
/**
* Interface for Memory Management Unit.
* The memory management unit should maintain the concept of a page table.
* As pages are read and written to, this changes the pages loaded into the
* the limited number of frames. The MMU keeps records, which will be used
* to analyse the performance of different replacement stratergies implemented
* for the MMU.
*/

public interface MMU {
    public void readMemory(int page_number);
    public void writeMemory(int page_number);

    public void setDebug();
    public void resetDebug();

    public int getTotalDiskReads();
    public int getTotalDiskWrites();
    public int getTotalPageFaults();

    public int getPageHits();
}
```

Entry Program

```
import java.io.BufferedReader;
import java.io.FileReader;

public class Memsim {
    public static void main(String[] args) {
        int page_offset = 12;           // page is 2^12 = 4KB

        int frames;
        BufferedReader input = null;
        MMU mmu = null;

        /* read parameters */
        //the file
        try {
           input = new BufferedReader(new FileReader(args[0]));
        }
```

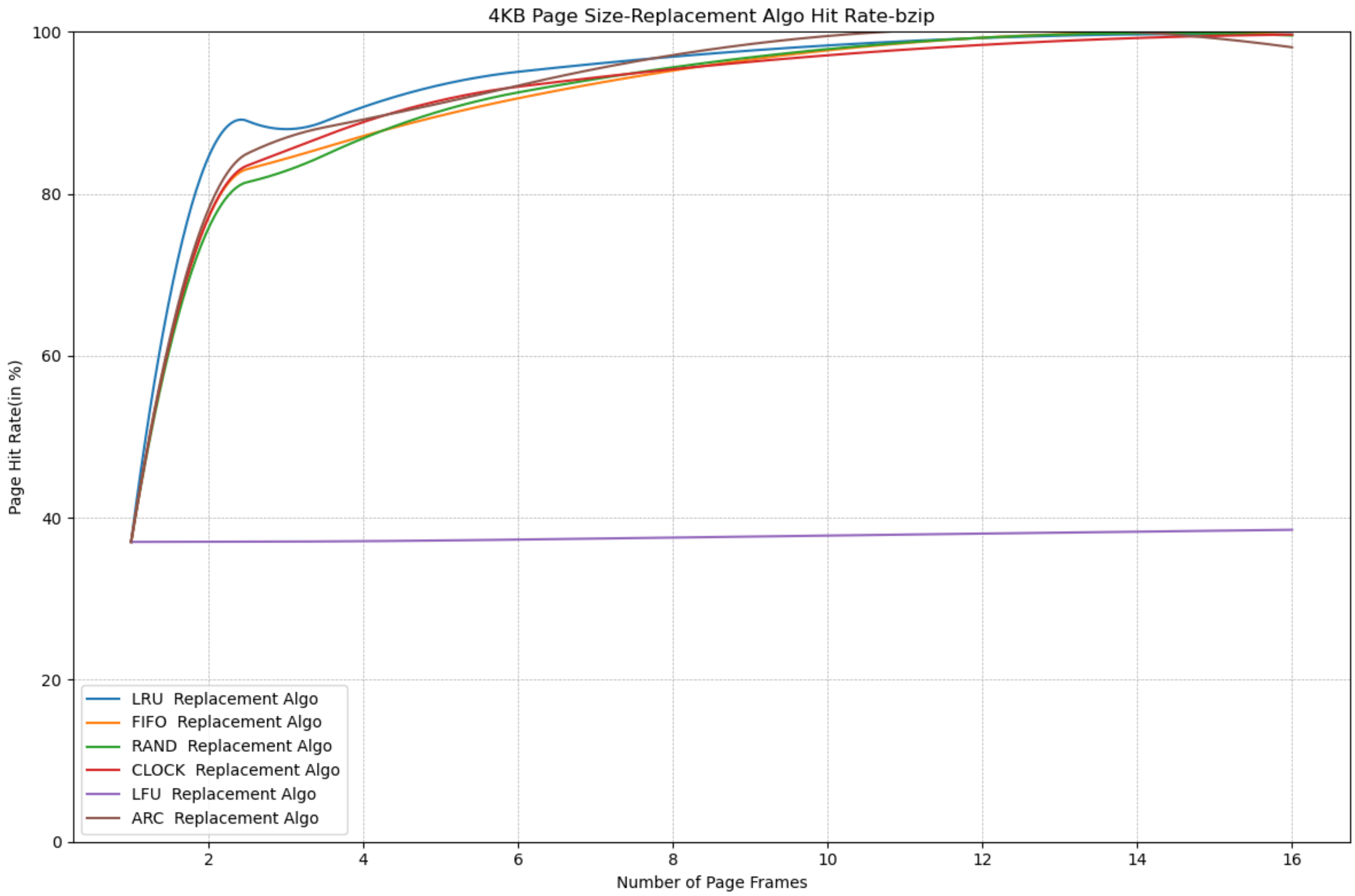
```java
        catch (java.io.FileNotFoundException e) {
            System.out.println("Input '" + args[0] + "' could not be found");
            System.out.println("Usage: java Memsim inputfile numberframes replacementmode debugmode");
            System.exit(-1);
        }

        //number of frames
        frames = Integer.parseInt(args[1]);

        //the replacement mode
        if (args[2].equals("rand"))
            mmu = new RandMMU(frames);
        else if (args[2].equals("lru"))
            mmu = new LruMMU(frames);
        else if (args[2].equals("clock"))
            mmu = new ClockMMU(frames);
        else if (args[2].equals("fifo"))
            mmu = new FifoMMU(frames);
        else if (args[2].equals("lfu"))
            mmu = new LFUMMU(frames);
        else if (args[2].equals("arc"))
            mmu = new ARCMMU(frames);
        else {
            System.out.println("Usage: java Memsim inputfile numberframes replacementmode debugmode");
            System.out.println("replacementmodes are [ rand | lru | esc ]");
            System.exit(-1);
        }

        //debug mode?
        if (args[3].equals("debug"))
            mmu.setDebug();
        else if (args[3].equals("quiet"))
            mmu.resetDebug();
        else {
            System.out.println("Usage: java Memsim inputfile numberframes replacementmode debugmode");
            System.out.println("debugmode are [ debug | quiet ]");
            System.exit(-1);
        }

        /* Process the traces from the file */
        String traceLine;
        String[] traceCmd;
        long logical_address;
        int page_number;
        int no_events = 0;

        try {
            traceLine = input.readLine();
            while (traceLine != null) {
                traceCmd = traceLine.split(" ");

                //convert from hexadecimal address from file, to appropriate page number
                logical_address = Long.parseLong(traceCmd[0],16);
                page_number = (int) (logical_address >>> page_offset);

                //process read or write
                if (traceCmd[1].equals("R"))
                    mmu.readMemory(page_number);
                else if (traceCmd[1].equals("W"))
                    mmu.writeMemory(page_number);
                else {
                    System.out.println("Badly formatted file. Error on line " + (no_events+1));
                    System.exit(-1);
                }

                no_events++;
                traceLine = input.readLine();
            }
        }
        catch (java.io.IOException e) {
            System.out.println("Error reading input file");
            System.exit(-1);
        }
        catch (NumberFormatException e) {
            System.out.println("Memory address strange on line " + (no_events+1));
            System.exit(-1);
        }

        /* Print results */
        System.out.println("total memory frames: " + frames);
        System.out.println("events in trace: " + no_events);
        System.out.println("total disk reads: " + mmu.getTotalDiskReads());
        System.out.println("total disk writes: " + mmu.getTotalDiskWrites());
        System.out.printf("page fault rate: %.4f\n", ((double) mmu.getTotalPageFaults())/no_events);
        System.out.println("total page hits: " + mmu.getPageHits());
        System.out.println("hit rate: " + ((double) mmu.getPageHits())/no_events);
        //System.out.println("page fault rate: " + ((double) mmu.getTotalPageFaults())/no_events);
    }

}

```

Results:

> Using Machine Learning built-in libriray 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.

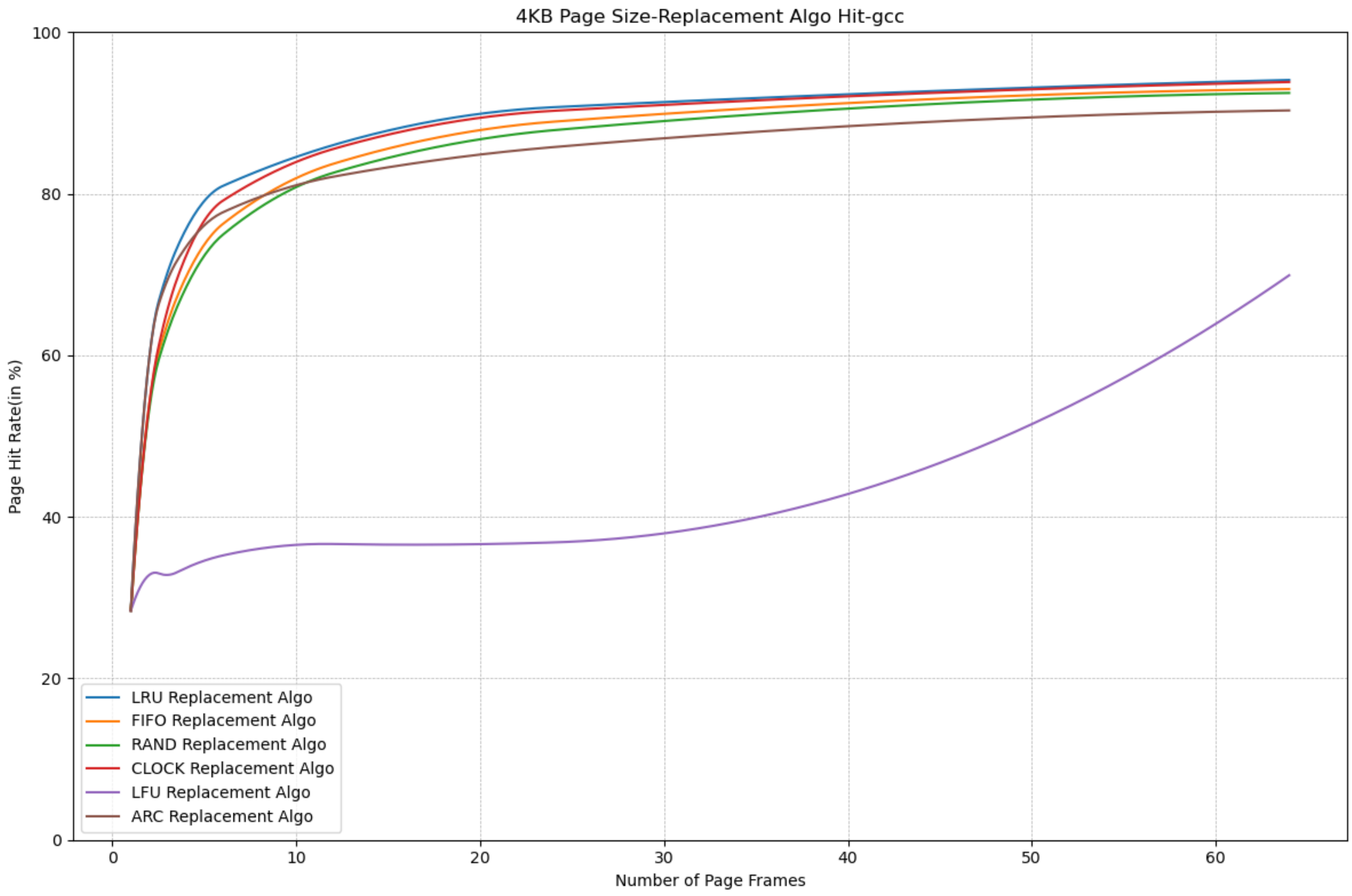
Page size = 4KB, bzip trace

```python
#bzip
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import make_interp_spline

# Data
datasets = {
    "LRU": ([1, 2, 3, 4, 8, 16], [37.0263, 84.5571, 87.9896, 90.723, 96.9309, 99.6656]),
    "FIFO": ([1, 2, 3, 4, 8, 16], [37.0263, 77.1162, 84.3426, 87.1399, 95.2172, 99.618]),
    "RAND": ([1, 2, 3, 4, 8, 16], [37.0263, 75.758, 82.8353, 86.8604, 95.6176, 99.5504]),
    "CLOCK": ([1, 2, 3, 4, 8, 16], [37.0263, 77.1162, 85.3077, 88.8558, 95.3836, 99.6532]),
    "LFU": ([1, 2, 3, 4, 8, 16], [37.0263, 37.0488, 37.0635, 37.1146, 37.5611, 38.5195]),
    "ARC": ([1, 2, 3, 4, 8, 16], [37.0263, 78.1058, 86.9301, 89.1687, 97.1275, 98.0932])
}

# Plotting
plt.figure(figsize=(12,8))

# Iterate through each dataset, interpolate, and plot
for label, (X, Y) in datasets.items():
    X_new = np.linspace(min(X), max(X), 500)
    spl = make_interp_spline(X, Y, k=2)
    Y_new = spl(X_new)

    # Scatter and line plot
    #plt.scatter(X, Y, s=50, label=f'{label} Data Points')  # s=50 for bigger markers
    plt.plot(X_new, Y_new, linestyle='-', label=f'{label}  Replacement Algo')

# Axes, Title, Grid, and Legend
plt.xlabel('Number of Page Frames')
plt.ylabel('Page Hit Rate(in %)')
plt.title('4KB Page Size-Replacement Algo Hit Rate')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.ylim(0, 100)  # Adjusted the maximum value of y to 100
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()

```



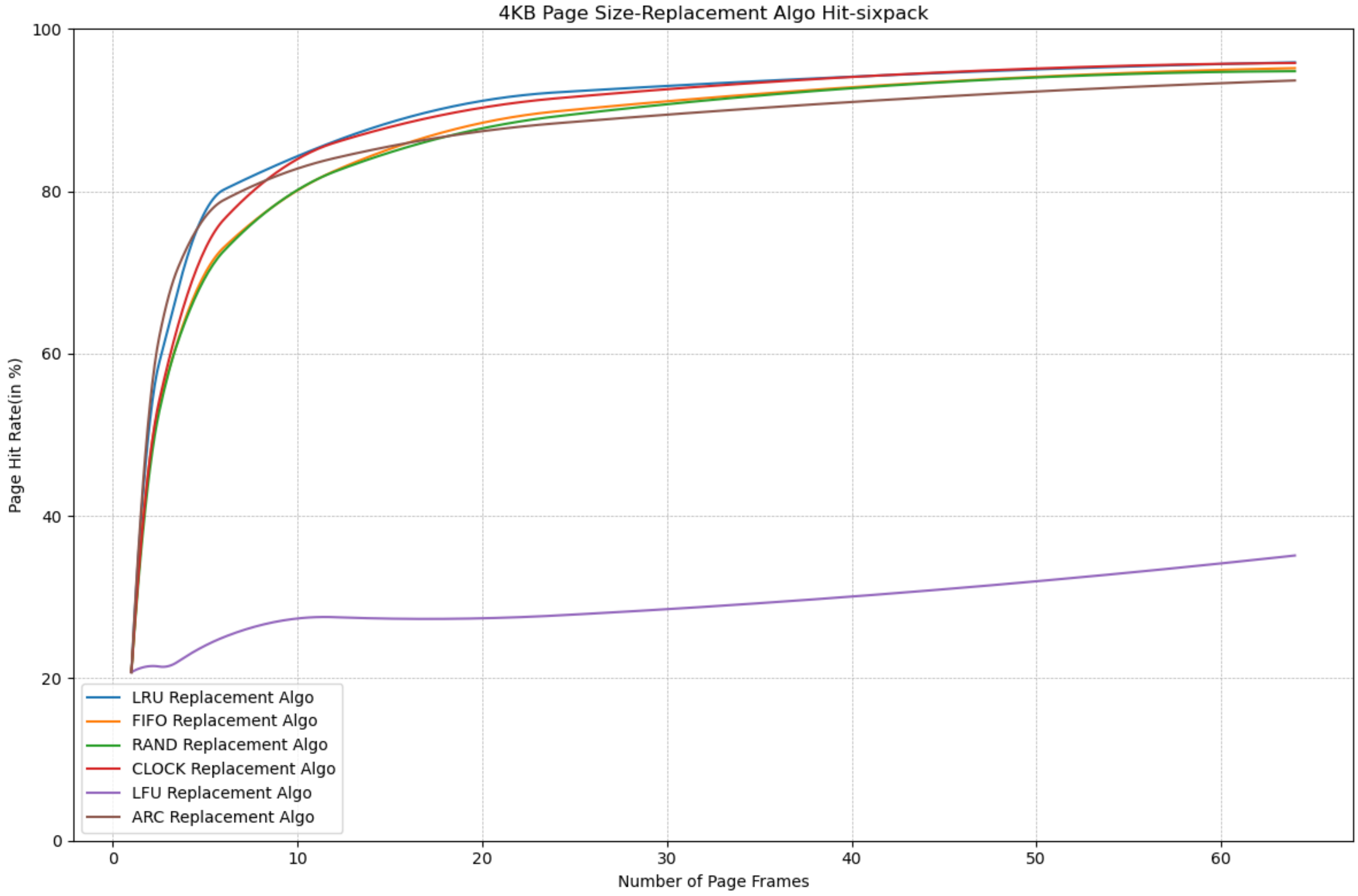
Page size = 4KB, gcc trace

```
#gcc
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import make_interp_spline

# Data
datasets = {
    "LRU": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 59.6045, 70.2867, 75.6191, 82.8814, 88.3396, 91.5599, 94.0911]),
    "FIFO": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 53.9088, 64.0434, 69.714, 79.4632, 86.1461, 90.1933, 92.9658]),
    "RAND": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 53.2855, 62.9108, 68.3930, 78.2784, 84.9899, 89.3481, 92.4822]),
    "CLOCK": ([1, 2, 3, 4, 8, 16, 32, 64], [28.3894, 53.9088, 65.7005, 72.3032, 81.8144, 87.8318, 91.2314, 93.836]),
    "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]),
    "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])
}

# Plotting
plt.figure(figsize=(12,8))

# Iterate through each dataset, interpolate, and plot
for label, (X, Y) in datasets.items():
    X_new = np.linspace(min(X), max(X), 500)
    spl = make_interp_spline(X, Y, k=2)
    Y_new = spl(X_new)

    # Scatter and line plot
    #plt.scatter(X, Y, s=50, label=f'{label} Data Points')  # s=50 for bigger markers
    plt.plot(X_new, Y_new, linestyle='-', label=f'{label} Replacement Algo')

# Axes, Title, Grid, and Legend
plt.xlabel('Number of Page Frames')
plt.ylabel('Page Hit Rate(in %)')
plt.title('4KB Page Size-Replacement Algo Hit-gcc')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.ylim(0, 100)  # Adjusted the maximum value of y to 100
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()

```



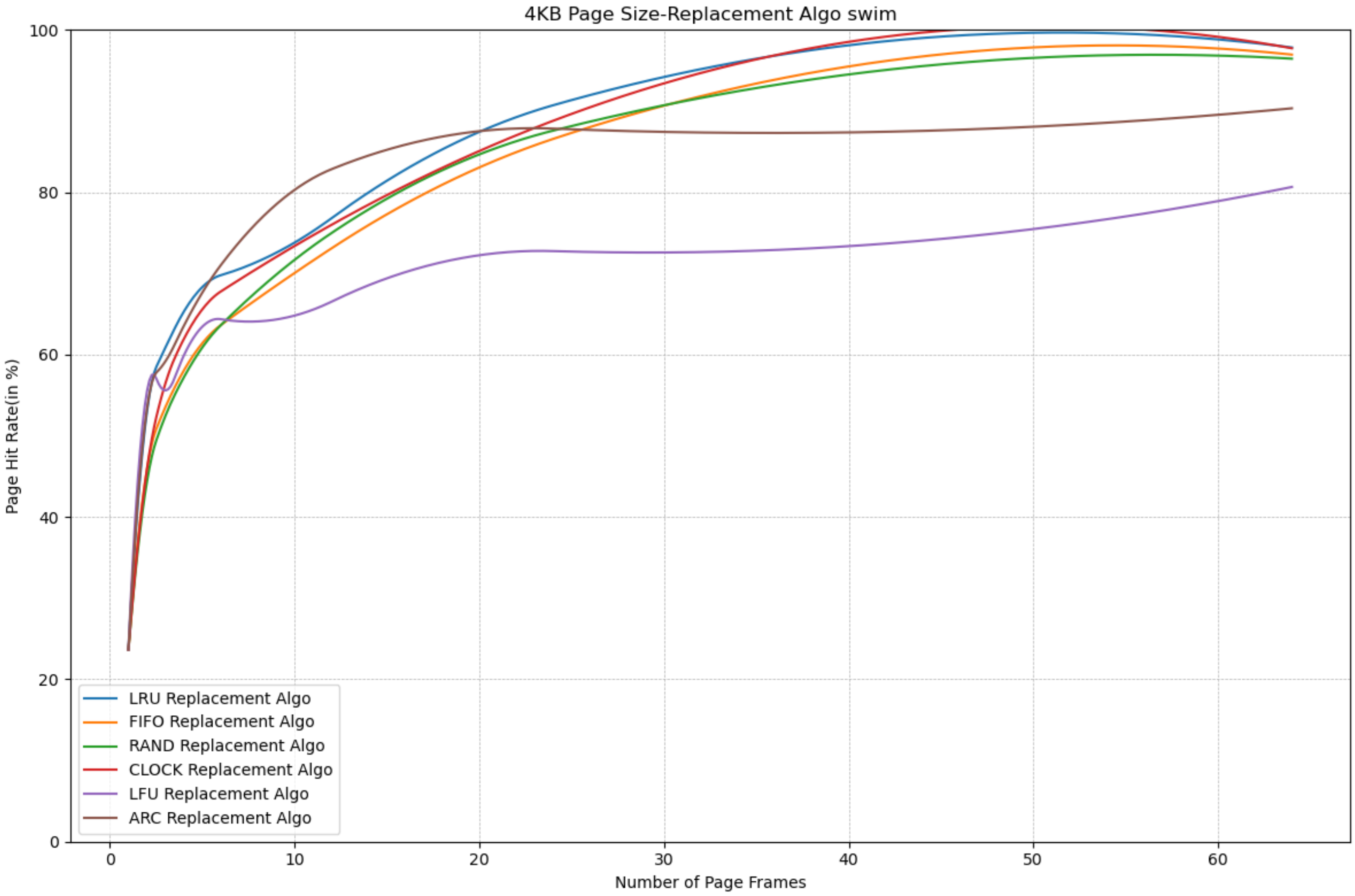
```
#sixpack
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import make_interp_spline

# Data
datasets = {
    "LRU": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 51.6839, 63.2442, 71.738, 82.3504, 89.1318, 93.2253, 95.8814]),
    "FIFO": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 47.0763, 57.7565, 64.819, 76.9832, 85.9917, 91.4717, 95.1699]),
    "RAND": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 45.7322, 57.5457, 64.4536, 76.9125, 85.4826, 91.1693, 94.7933]),
    "CLOCK": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 47.0763, 58.9645, 67.0632, 80.824, 88.4794, 92.9151, 95.8051]),
    "LFU": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 21.4984, 21.5024, 22.76, 26.5544, 27.3282, 28.8034, 35.1312]),
    "ARC": ([1, 2, 3, 4, 8, 16, 32, 64], [20.7621, 53.6075, 66.9364, 73.0243, 81.084, 85.9731, 89.7762, 93.6508])
}

# Plotting
plt.figure(figsize=(12,8))

# Iterate through each dataset, interpolate, and plot
```

```
for label, (X, Y) in datasets.items():
    X_new = np.linspace(min(X), max(X), 500)
    spl = make_interp_spline(X, Y, k=2)
    Y_new = spl(X_new)

    # Scatter and line plot
    #plt.scatter(X, Y, s=50, label=f'{label} Data Points')  # s=50 for bigger markers
    plt.plot(X_new, Y_new, linestyle='-', label=f'{label} Replacement Algo')

# Axes, Title, Grid, and Legend
plt.xlabel('Number of Page Frames')
plt.ylabel('Page Hit Rate(in %)')
plt.title('4KB Page Size-Replacement Algo Hit-sixpack')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.ylim(0, 100)  # Adjusted the maximum value of y to 100
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()

```



```
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import make_interp_spline

```

```
# Data
datasets = {
    "LRU": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.529321, 0.608734, 0.653064, 0.714625, 0.828039, 0.951746, 0.978344]]),
    "FIFO": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.458542, 0.535343, 0.580491, 0.669107, 0.785705, 0.918362, 0.969578]]),
    "RAND": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.443155, 0.524444, 0.57203, 0.678965, 0.804642, 0.91622, 0.964585]]),
    "CLOCK": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.458542, 0.564483, 0.619318, 0.706481, 0.808152, 0.946975, 0.977389]]),
    "LFU": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.554351, 0.555903, 0.598963, 0.640912, 0.701694, 0.726324, 0.806565]]),
    "ARC": ([1, 2, 3, 4, 8, 16, 32, 64], [i * 100 for i in [0.236556, 0.53276, 0.591725, 0.63552, 0.763602, 0.858651, 0.873644, 0.903442]])
}

# Plotting
plt.figure(figsize=(12,8))

# Iterate through each dataset, interpolate, and plot
for label, (X, Y) in datasets.items():
    X_new = np.linspace(min(X), max(X), 500)
    spl = make_interp_spline(X, Y, k=2)
    Y_new = spl(X_new)

    # Scatter and line plot
    #plt.scatter(X, Y, s=50, label=f'{label} Data Points')  # s=50 for bigger markers
    plt.plot(X_new, Y_new, linestyle='-', label=f'{label} Replacement Algo')

# Axes, Title, Grid, and Legend
plt.xlabel('Number of Page Frames')
plt.ylabel('Page Hit Rate(in %)')
plt.title('4KB Page Size-Replacement Algo swim')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.ylim(0, 100)  # Y values are already in percentage
plt.legend(loc='lower left')
plt.tight_layout()
plt.show()

```


4KB Page Size-Replacement Algo swim
Page Hit Rate(in %)
Number of Page Frames
0
20
40
60
80
100
0
10
20
30
40
50
60
LRU Replacement Algo
FIFO Replacement Algo
RAND Replacement Algo
CLOCK Replacement Algo
LFU Replacement Algo
ARC Replacement Algo