GTU Department of Computer Engineering
CSE 312 - Spring 2023
Homework 2 Report

Emirkan Burak Yılmaz
1901042659

Contents

1 Part 1

1.1 Page Table Structure

The structure of the page table is designed in a way that given page replacement algorithms can work on the same structure. To achieve this, two data structure is used to keep the pages and handle the virtual to physical page mapping. The std::unordered_map is used to keep the mapping between physical page number and virtual page number. The std::list is used to keep the actual page table entries. The management of the page list enables to use different page replacement algorithm. Front of the list contains the least recently used page and this list can be treated as circular link list to implement WSClock algorithm.

```
You, now | 1 author (You)
class InvertedPageTable {
public:
    enum PageReplacementAlgo {SC, LRU, WSClock};

    InvertedPageTable(int capacity, int frameSize, PageReplacementAlgo replacement, unsigned int tau = 4);

    ~InvertedPageTable();

    long map(long virtualAddr, enum AccessMode mode);

    std::list<PTE *>::iterator findEntry(int virtualFrameID);

    int getSize();

    bool isFull();

    void printTable();

    bool addPage(PTE *entry);

    std::list<PTE *> getPageList();

    PTE *removePage(int virtualFrameID);

    PTE *evictPage(std::vector<PTE *> &modifiedPages);

    PTE *evictPage_SC();

    PTE *evictPage_LRU();

    PTE *evictPage_WSClock(std::vector<PTE *> &modifiedPages);

    void clearReferenceBit();

private:
    int capacity;
    int frameSize;
    unsigned long int sysclock;
    const unsigned int tau;
    PageReplacementAlgo replacement;
    std::unordered_map<int, int> pageTable;
    std::list<PTE *> pageList;
};
```

1.2 Page Table Entry (PTE)

PTE class contains information's related such as mapping between the virtual page number and physical page number and the modified/referenced bits. In addition to these the last reference time is kept tracking with the time field.

```
You, 4 hours ago | 1 author (You)
class PTE {
public:
    PTE(int virtualFrameID, int physicalFrameID);

    ~PTE();

    void reset(int virtualFrameID, int physicalFrameID);

    void setVirtualFrameID(int virtualFrameID);

    void setPhysicalFrameID(int physicalFrameID);

    void setReferenced(bool referenced);

    void setModified(bool modified);

    void setTime(unsigned long int time);

    bool getReferenced();

    bool getModified();

    int getPhysicalFrameID();

    int getVirtualFrameID();

    unsigned long int getTime();

private:
    bool referenced;
    bool modified;
    bool presentAbsent;
    int virtualFrameID;
    int physicalFrameID;
    unsigned long int time;
};
```

1.3 Page Replacement Algorithms

1.3.1 LRU

As it's mentioned earlier, the least recently used page can be found and removed easily by removing the front entry of the page list.

```
PTE *InvertedPageTable::evictPage_LRU()
{
    PTE *leastRecentPage = pageList.front();
    return removePage(leastRecentPage->getVirtualFrameID());
}
```

1.3.2 SC

Fort the implementation of the SC algorithm the page list is treated as circular link list and whole list traversed until finding the first unreferenced page.

```
PTE *InvertedPageTable::evictPage_SC()
{
    while (true) {
        PTE *oldestPage = pageList.front();
        /* Check the reference bit of the page */
        if (oldestPage->getReferenced()) {
            /* Page has been referenced, reset the reference bit */
            oldestPage->setReferenced(false);
            /* Move the page to the end of the list */
            pageList.pop_front();
            pageList.push_back(oldestPage);
        }
        else {
            /* Page can be evicted */
            return removePage(oldestPage->getVirtualFrameID());
        }
    }
}
```

1.3.3 WSClock

For the implementation of the SC algorithm the page list is treated as circular link list and a counter named sysclock is used. To evict a page, the page list traversed and the first page which last reference time is less than tau (constant class field) is removed. During the traverse, the refence and modified bits of the pages are cleared, and the required disk write operations are implemented.

```
PTE *InvertedPageTable::evictPage_WSClock(std::vector<PTE *> &modifiedPages)
{
    while (true) {
        PTE *currentPage = pageList.front();

        if (currentPage->getModified() || currentPage->getReferenced()) {
            /* Set the current clock time and set the R and M bit to 0 */
            if (currentPage->getModified()) {
                /* Clear the modified bit, and indicate that this page should write back to the disk */
                currentPage->setModified(false);
                modifiedPages.push_back(currentPage);
            }

            currentPage->setReferenced(false);

            currentPage->setTime(sysclock);

            /* Move the page to the end of the list (circular linked list for clock algorithm) */
            pageList.pop_front();
            pageList.push_back(currentPage);
        }
        else if (sysclock - currentPage->getTime() > tau) {
            /* check if the page's age to see if it's older than the tau */
            /* Page can be evicted */
            return removePage(currentPage->getVirtualFrameID());
        }
        /* Witch each condition check, the clok and arrow (front) are advanced */
        ++sysclock;
    }
}
```

2 Part 2

2.1 Memory Management

The physical memory is represented as integer array and kept in MemoryManagement class. It's the abstraction of physical memory and disk. The memory management is handling with the page table

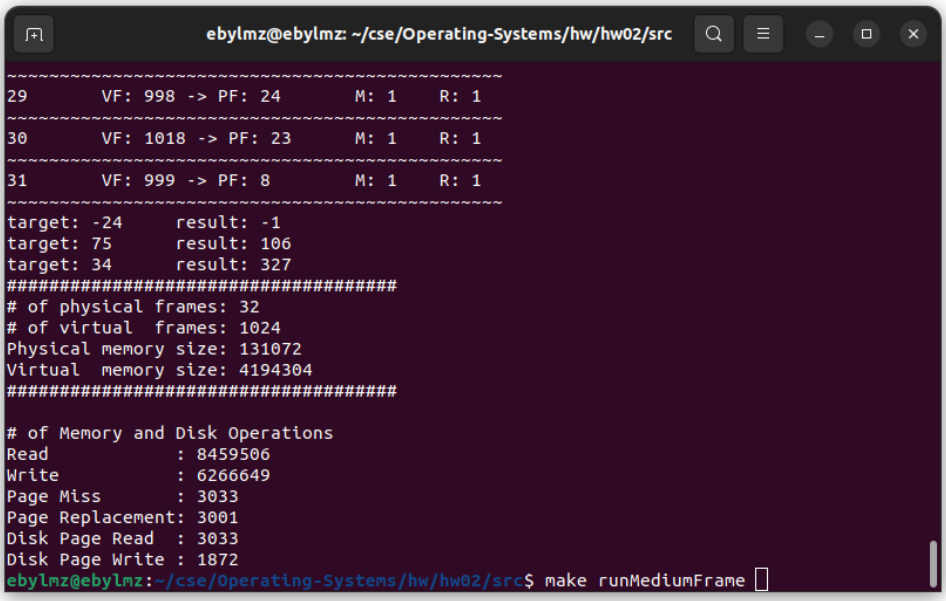
and disk file by read write operations. To create a physical memory with the virtual address space the parameters such as size of the page frame and the number of physical and virtual frames are needs to be given. The MemoryManagent class has public functions read and write for the usage of the memory.

```
You, 4 hours ago | 1 author (You)
class MemoryManagement
{
public:
    MemoryManagement(
        unsigned long int frameSize,
        unsigned long int numPhysicalFrames,
        unsigned long int numVirtualFrames,
        InvertedPageTable::PageReplacementAlgo replacement,
        const char *diskFileName);

    ~MemoryManagement();

    int getVirtualMemSize();

    int getPhysicalMemSize();

    int getFrameSize();

    int read(long virtualAddr);

    int write(long virtualAddr, int data);

    void printStats();

private:
    unsigned long int frameSize;
    unsigned long int numVirtualFrames;
    unsigned long int numPhysicalFrames;
    const char *diskFileName;
    int *physicalMem;
    InvertedPageTable pageTable;
    pthread_mutex_t mutex;
    MemStats stats;

    long access(long virtualAddr, AccessMode mode);

    long diskPosition(int virtualFrameID);

    long memoryPosition(int physicalFrameID);

    int getFrameID(long addr);

    int getFrameOffset(long addr);

    void initDisk();

    void readDisk(PTE *entry);

    void writeDisk(PTE *entry);
};
```

2.2 Simulation

The virtual memory simulation covers all the virtual space by running simulation function on different virtual memory portions. First the virtual memory is filled randomly by using write function from the MemoryManagement class. After that each portion of the memory is given to a new thread with the

simulation function. With this way simulation function can be executed on different portion of the virtual memory in parallel. Each simulation function makes matrix-vector and vector-transpose multiplications in parallel, and the resulting matrix and vectors are mutually summed, and the two-search algorithm linear and binary searches is applied on the resulting array. The execution statistics are given below.

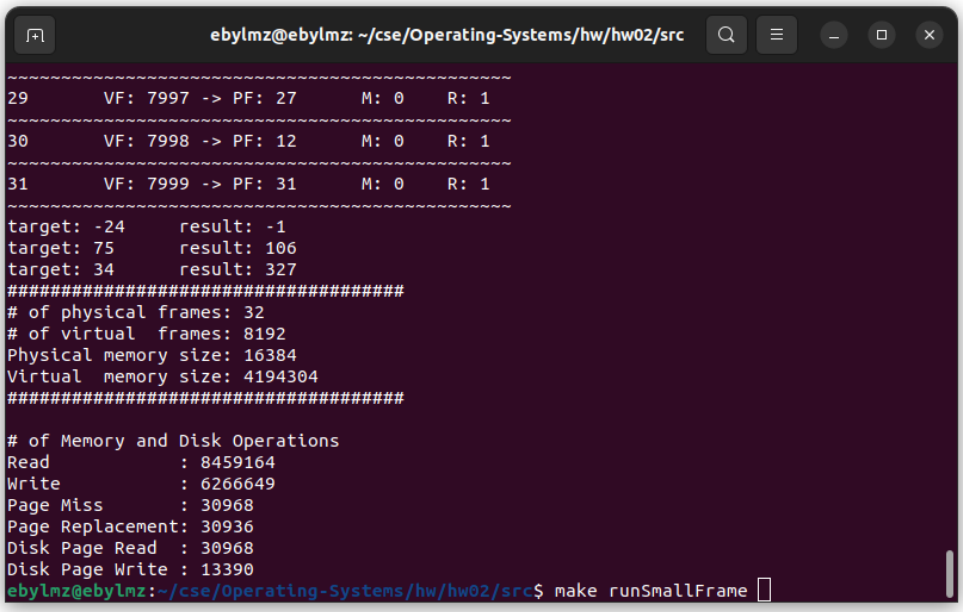


3 Part 3

3.1 Best Frame Size

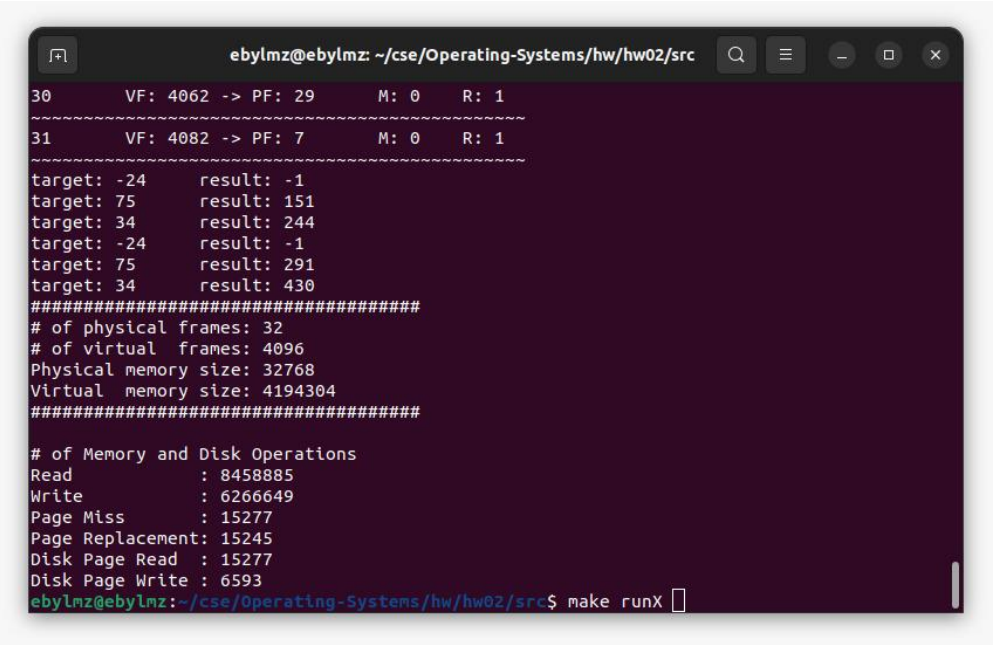
3.1.1 Small Frame (512MB)

Using small page frame is requires many pages replacement due to the frequent page misses.

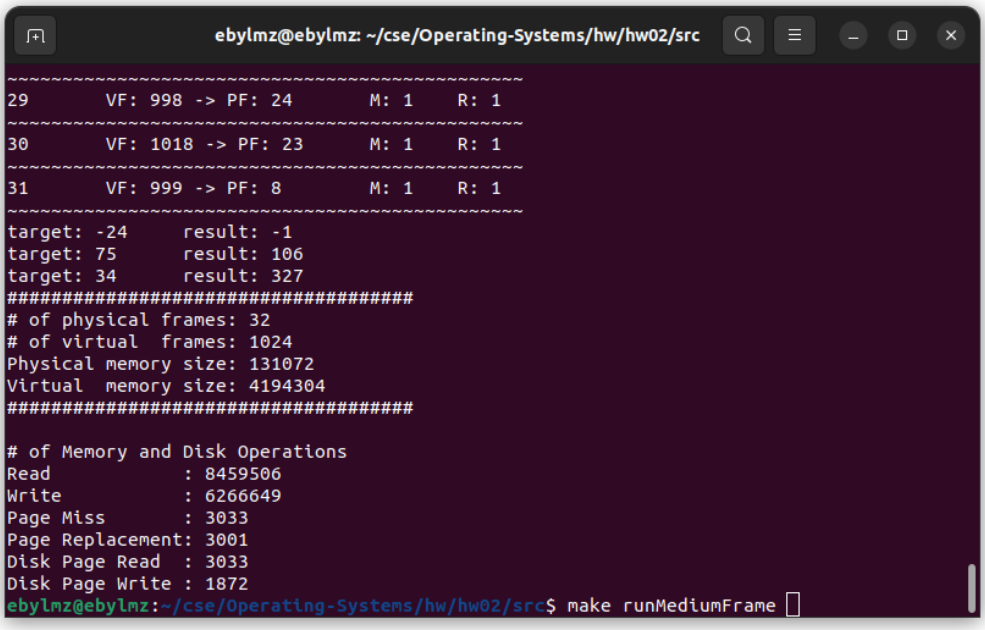


3.1.2 Medium Size Frame (1KB-4KB)

When we select our frame size around 1KB to 4 KB, the number of page miss becomes less compared to small page frame.



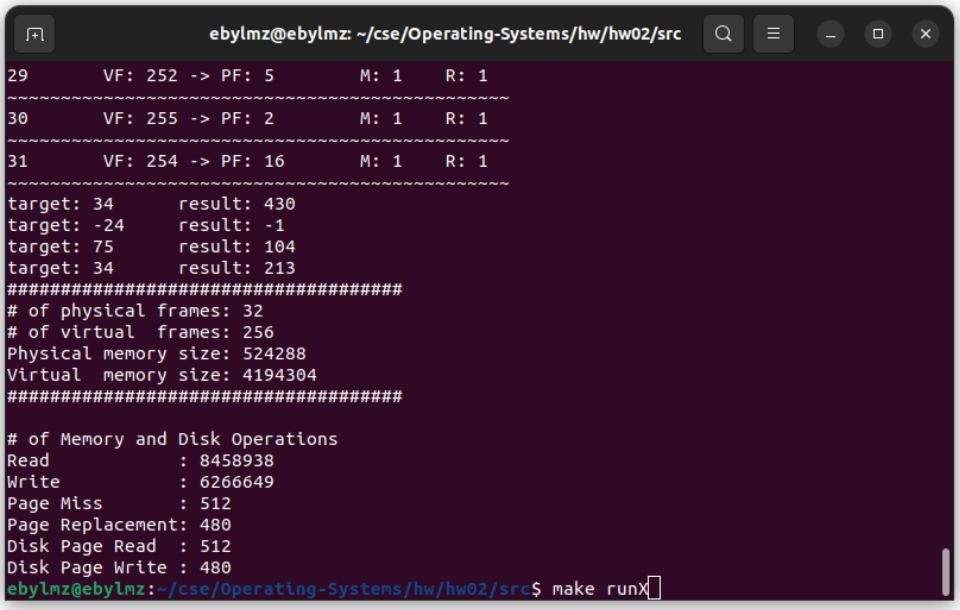
1KB Page Frame



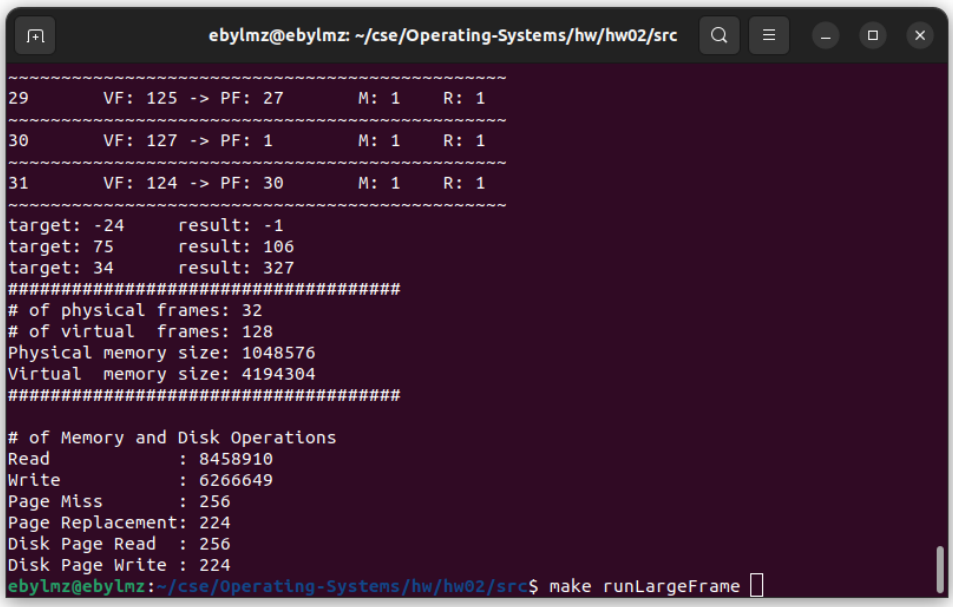
4KB Page Frame

3.1.3 Large Frame (16KB - 32KB)

Using larger frame resulted in having even smaller number of page miss.



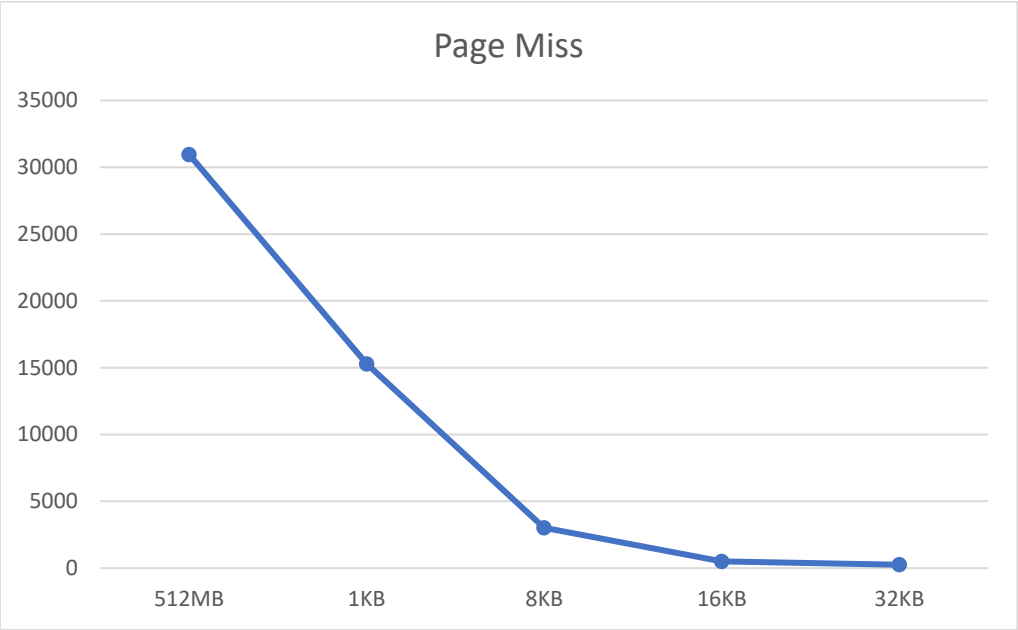
16KB Frame



32KB Page Frame

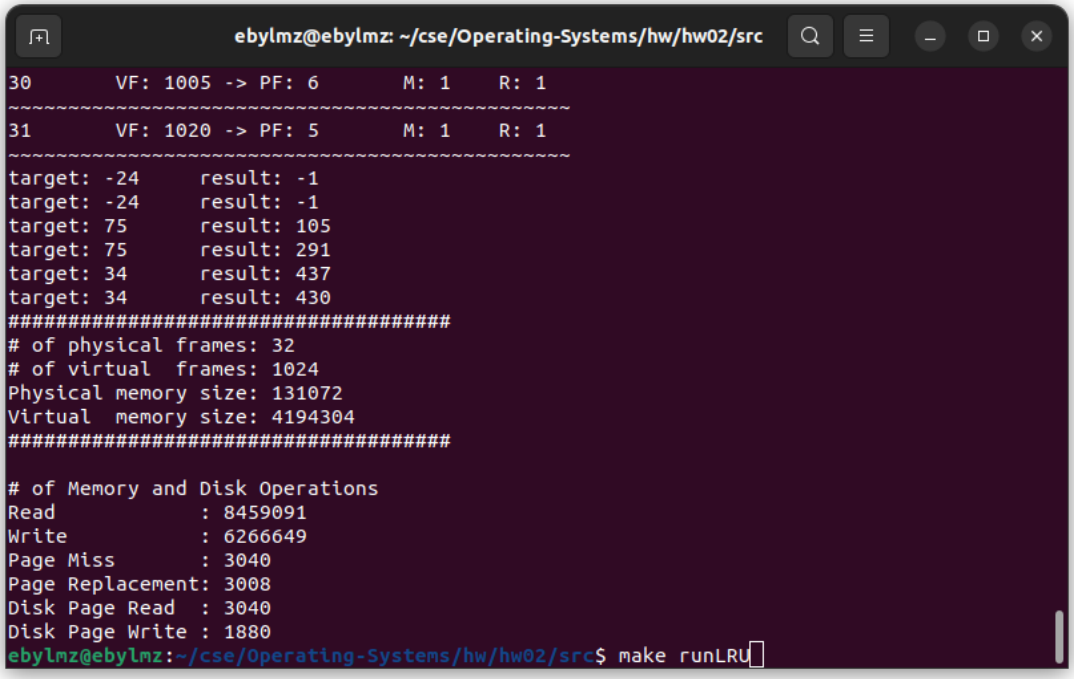
3.1.4 Conclusion

In summary, in our single process simulation, the number of page misses decreases as the size of the page frame is increases. However, with larger page frames, there is a higher chance of internal fragmentation. When a smaller amount of data is stored in a large page frame, the remaining space within the page frame is wasted. This can lead to inefficient memory utilization, especially if most of the allocated memory does not fully utilize the page frames.

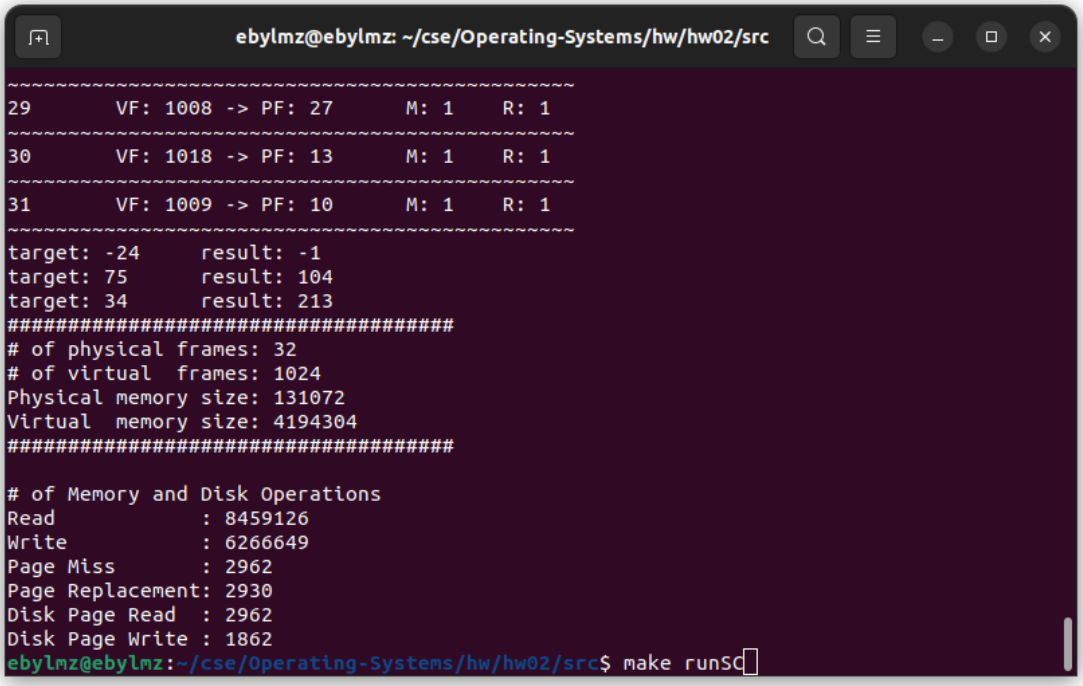


3.2 Best Page Replacement Algorithms

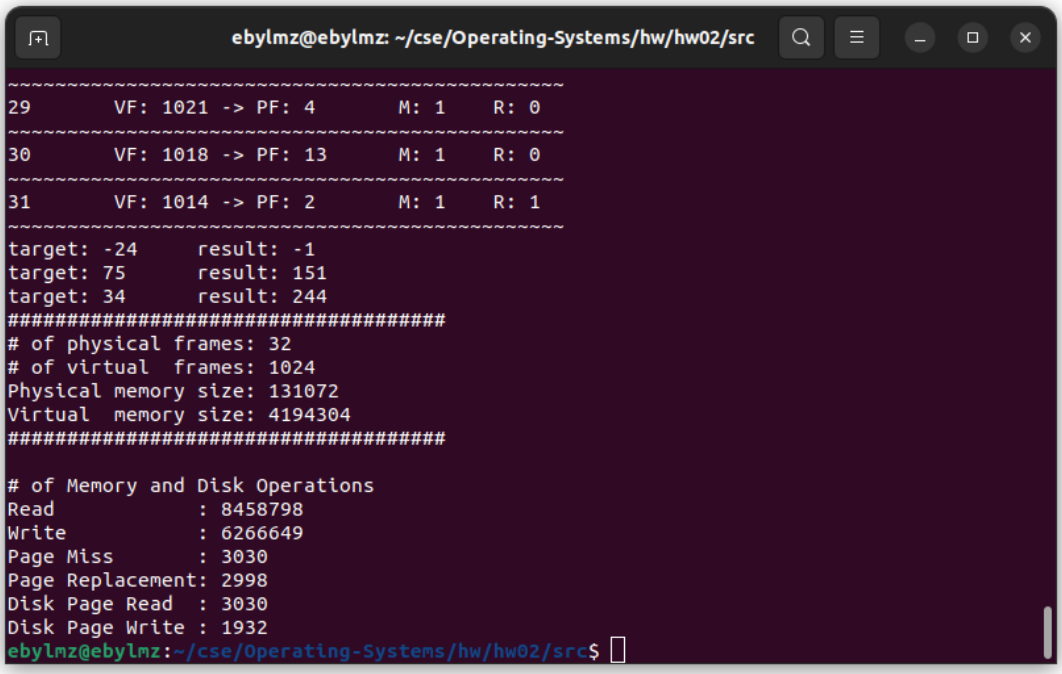
3.2.1 LRU



3.2.2 SC



3.2.3 WSClock



3.2.4 Conclusion

According to simulation results Second Change page replacement algorithm resulted in least number of pages miss and page replacement.