

Memory-Simulator

Xavier Guinto Rios A01366106

Gustavo Santamaría A01362484

Sergio Domínguez Cordero A01365942

OBJECTIVE

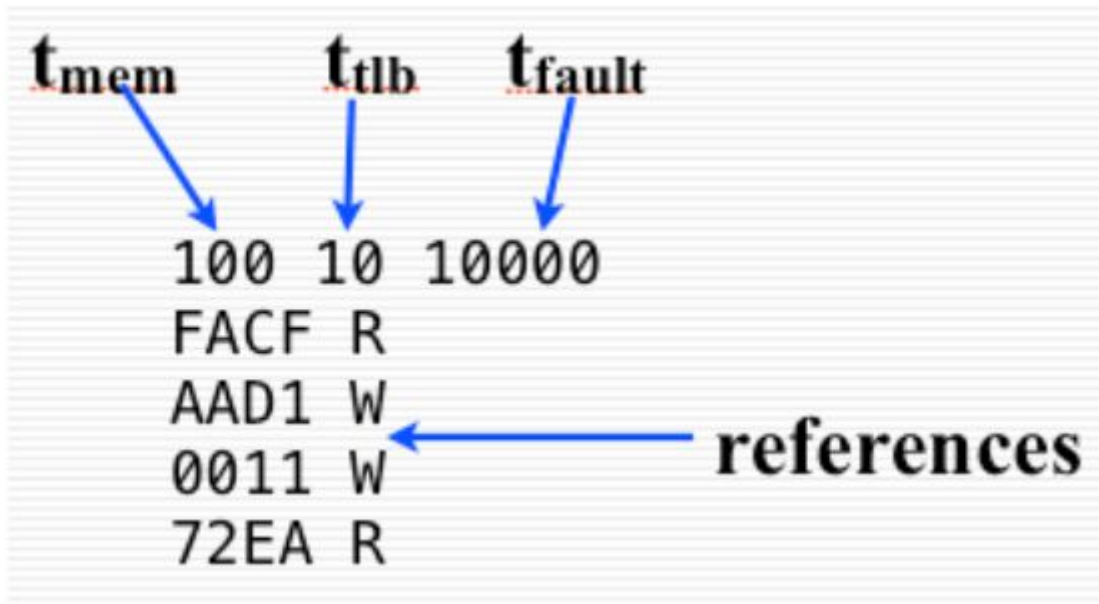
The goal of this project is to implement a simulator for virtual memory manager. The simulator reads a file with memory references, called a memory trace, and the proceeds to simulate the memory access in a system that consists of a small **Translation LookAhead Buffer (TLB)**. The simulator reports the total number of memory frames used, and the *average access time* for the entire trace. To verify you read all events in the trace it is recommended that you also keep track of the total number of events(memory accesses) in the trace.

GOALS

The implementation of the virtual memory simulator will test your comprehension of the interactions between elements in a simple memory hierarchy, as well as verify that the student is capable of using basic data structures, file management, and general programming methodologies.

PROBLEM STATEMENT

We are required to implement a *virtual memory simulator* for a system with 16-bit virtual addresses (64Kbytes total) and 8Kbytes of physical memory. Furthermore, assume that the pages and frames are 512-bytes each. To reduce the access time the system has an 8-entry TLB. The TLB uses the Least Recently Used (LRU) replacement policy to evict entries. To simplify your implementation you can use a single-level page table for the virtual to physical address translation. Given the previous conditions your system will have 128 entries in the page table and a total of 16 frames in memory. Our simulator must keep track of what pages are loaded into memory. As it processes each memory event from the trace, it should check if the reference is in the TLB and also check if the corresponding page is in memory or not. If not, it should choose a frame to remove from memory. Of course, if the frame to be replaced is dirty, it must be saved to disk. Finally, the new frame is loaded into memory from disk, and the page table is updated. Our program will take as input a text file with an **arbitrary** number of memory references. The first line in the file has the main memory access time (**tmem**), the TLB access time (**ttlb**), and the page fault penalty (tfault). All times are in nano seconds. After the first line, each line will have a 16-bit address followed by the access type (read or write). As an example consider the following four memory references:



Note that our program should read three positive integers followed by a newline character and then for each line we should read a 16-bit address followed by a space and then the access type which is either **R** (read) or **W** (write). The trace file that will be provided for us has 1,000,000 virtual memory references and for each reference you must perform the translation from virtual to physical memory and depending on whether the reference is in the TLB or the page table we will keep track of the average access time for each reference. Note that when you are evicting an entry in the TLB or from a frame in memory you need to check if the entry is dirty (i.e. the reference was a **write**). If it's **dirty** you need to simulate a memory **write** (writes the dirty page into disk), followed by a memory **read** (reads the new page from disk into memory). Note that the TLB has a subset of the memory references contained in the frame table, simplifying your coding significantly.

EXPECTED OUTPUTS

The simulator reports the TLB hit rate, the page faults in the form of disk reads and writes, total number of memory frames used, and the average access time for the entire trace. Note that the final trace is over 1,000,000 entries long so you will be provided with a smaller trace for testing purposes.

PROPOSED SOLUTION

Taking knowledge of the topics of the class, We use some part of code from the previous project (Scheduler) to skip the comments at the moment of reading the file for the test. The txt file does not contain the character '#' that is the one that we skip but to take order we implement it. In the project we use some condition at the moment of reading the file, for example when the file does not exist and in case that the file exist the program verify two more condition that are 'W' for write and 'R' for read.

We make use of arrays because it make more easy to manage the information that we required. First we define the size of the global constant for the logical address, physical address, offset, TLB and the number of parameters we receive. The we get the lowest LRU in the TLB also we check if the number of parameters is correct in case that this is true. We set the values from the tables (page table, frame table, TLB) to neutrals.

Then we start to make the calculation needed for the purpose of this project taking the previous info with the respective calculations. For every process finish we reset all the value to move to the next one until all the process were read. At the final we print all the information recopilated in the process (Number of events, Page in, Page out, Average time, TLB hit ratio).

How To Build Executable Of The Program

Using a Linux environment of **64-bits**, to build the executable of the program is required the ise of the terminal, also called *command line*.

The prerequisites are:

1. Install C/C++ compiler and related tools

Though these are already installed as gcc (GNU Compiler Collection), it is not bad to update it and verify it is installed.

You can use the following commands to check the updates:

- \$ sudo apt-get update
- \$ sudo apt-get install build-essential manpages-dev

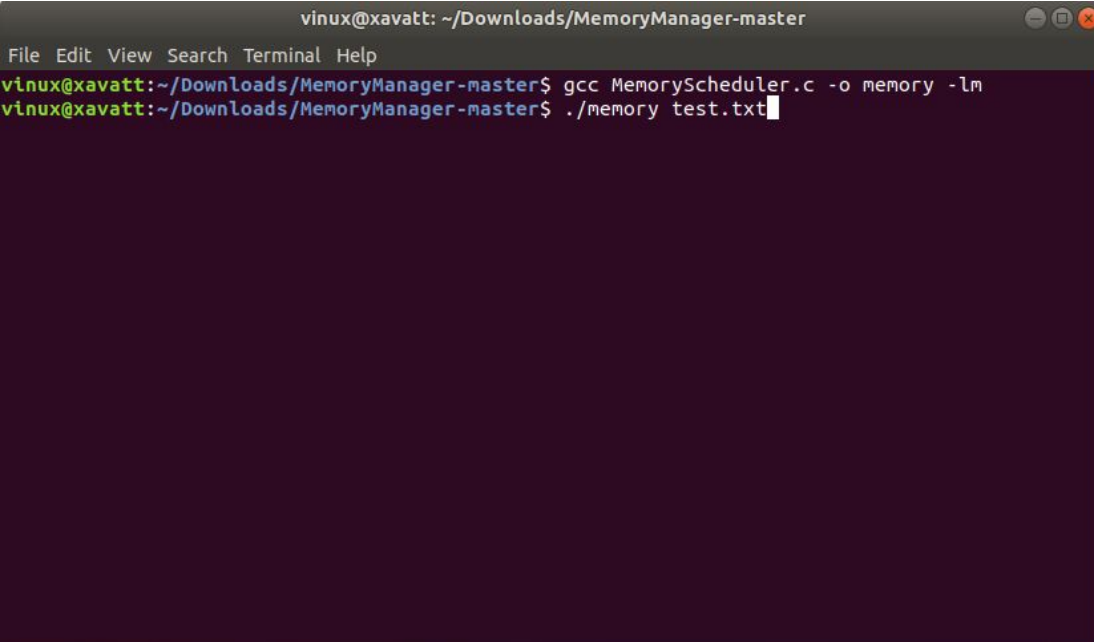
Finally, **To compile** the executable Schedler the following command is required:

- gcc MemoryScheduler.c -o memory -lm

To run the executable just created, the next command is applied:

- Directory/folder\$./memory test.txt

Here are a ScreenShoot to make it more visible.



```
vinux@xavatt: ~/Downloads/MemoryManager-master
File Edit View Search Terminal Help
vinux@xavatt:~/Downloads/MemoryManager-master$ gcc MemoryScheduler.c -o memory -lm
vinux@xavatt:~/Downloads/MemoryManager-master$ ./memory test.txt
```