OPERATING SYSTEM

Hw2 Report

Elifnur Kabalcı
1801042617

Structure Definitions:

```
struct PageTableEntry{
    int referenced;
    int present;
    int modified;
    int frameIndex;
    int LRUcounter;
    unsigned long accessTime;
};
typedef PageTableEntry PTE;
```

I designed a struct structure to hold data in Page Table. Here I have discussed the bit's that are kept in a page. In addition to the normal page definition, I used the LRU counter. This is one of the ones requested to be done in the homework pdf.

1.referenced: This bit is used in memory management and indicates the last reference of a page in the page table. This bit is set (marked 1) when a page memory is accessed or modified. This helps keep track of the last time the page was used and can be used to track how long pages are kept in memory.

2.present: This bit indicates whether the page exists in memory. If the page is in memory, this bit is set (marked 1). If the page is not in memory, this bit remains zero. Whether the page resides in memory is important for memory management operations. For example, when the page receives a memory request, it is checked whether the page is in memory. If the page is not in memory, a page swap is performed and the page is brought into memory.

3.modified: This bit indicates whether the content of the page in memory has been changed. This bit is set (marked 1) if the page has been modified in memory. This bit remains zero if the page has not been changed in memory. Whether the page is modified or not is important for page management and memory management operations. If the page has been modified, the changes may need to be written back.

4.frameIndex: This field specifies the location of the page in memory. The page table is used to show which frame each page is captured. Frame index is usually used as the index of the page table.

5.LRUcounter: This field is used by the Least Recently Used algorithm in the page table. The LRU algorithm tracks when the pages in memory were last used, and determines the least used page and replaces it with another page. LRUcounter is a counter used to keep track of when the page was last used.

6.accessTime: This field holds the last access time of the page. By tracking the last access time of each page in memory, it can be determined how often the pages are used. This information can be useful for memory management algorithms.

```
struct Statistics{
    int read;
    int write;
    int miss;
    int replacement;
    int diskWrite;
    int diskRead;
    vector<vector<int>> workingSet;
};
typedef Statistics statistic;
```

Number of reads: This value represents the number of read operations of a thread in memory. It keeps track of the number of read operations performed to access data in memory.

Number of writes : This value represents the number of writes in memory of a thread. It keeps track of the number of writes to data in memory.

Number of page misses: This value represents the number of pages missing from a thread's page table. When a page is requested, a page miss occurs if that page does not exist in memory. This value keeps track of the number of missing pages in memory.

Number of page replacements : This value represents the number of times a thread has replaced pages in memory. When a page is missing from memory, another page is removed from memory and replaced with a new page. This process is called page replacement. This value keeps track of the number of page replacements.

Number of disk page writes: This value represents the number of page writes made by a thread to the disk. In cases where pages in memory are changed, the original contents of the modified pages are written onto the disc. This value keeps track of the number of page writes to the disc.

Number of disk page reads : This value represents the number of page operations a thread has read from disk. When a missing page is requested from memory, that page is read from disk and brought into memory. This value keeps track of the number of pages read from the disk.

Estimated working set function w for each thread as a table : This value represents the working set size of each thread. A working set is the set of pages that a thread is actively using in memory at a given time. This table provides an important metric for memory management by tracking the size of each thread's working set. The estimated working set size can be used to optimize memory allocation and page management.

```
struct Parameter{
    int frameSize;
    int numPhysical;
    int numVirtual;
    int pageTablePrintInt;
    char *pageReplacement;
    char *tableType;
    char *diskFileName;
};
typedef Parameter par;
```

1.frameSize: This variable represents the size of a memory frame in memory. Each frame in memory is the units in which the pages are stored. Frame size is usually expressed in bytes.

2.numPhysical: This variable represents the total number of memory frames available in physical memory. Physical memory refers to the number of actual physical memory units.

3.numVirtual: This variable represents the total number of pages in virtual memory. Virtual memory is a level of abstraction that holds the image of programs in memory. Each page in virtual memory is mapped to real physical memory frames.

4.pageTablePrintInt: This variable represents the range used to print the page table. For example, if this value is set to 10, a page table will be printed every 10 pages.

5.pageReplacement: This variable is a string of characters representing the page replacement algorithm. The page replacement algorithm determines how pages in memory are managed. It can contain a value such as "LRU".

6.tableType: This variable is a string of characters representing the page table type. A page table is a data structure that holds how pages in virtual memory are mapped to frames in physical memory. For example, it may contain a value such as "Inverted Page Table" or "Hierarchical Page Table".

7.diskFileName: This variable is a string of characters representing the name of a file on disk. This file shows the source of the pages on disk. When a missing page is requested from the memory, this file is read from the disk and the page is brought to the memory.

```
struct helper{ // for global variables
    int LRUcounter = 0;
    int WSCcounter = 0;
    int MemoryAccesscounter = 0;
};
typedef helper help;
```

1. LRUcounter: This variable is a counter used for the Least Recently Used page replacement algorithm. The LRU algorithm tracks the last time the pages in memory were accessed and determines the least used page and performs page replacement. LRUcounter is used to update the last used time of each page and to support the page replacement algorithm.

2.WSCcounter: This variable is a counter used for the Working Set Clock page switching algorithm. The Working Set Clock algorithm tracks the last access time of pages in memory and their referencing by the processor. WSCcounter is used to monitor the referencing status of pages and to support the page replacement algorithm.

3.MemoryAccesscounter: This variable represents the total number of memory accesses. Each memory access refers to an operation such as reading or writing a page. MemoryAccesscounter can be used to monitor memory usage and perform performance analysis by counting total memory accesses. This value can be an important metric for memory management and optimization.

```
vector<vector<int>> physicalM; // memory definitons
vector<vector<int>> virtualM;
vector<int> pageFrame;
```

Physical Memory: Refers to the actual physical memory units in the computer system. These memory units are memory modules with hardware known as RAM. Physical memory represents the area of memory where programs and operating system run, data is stored and processed.

Virtual Memory: It is a memory management technique used in the computer system. This technique provides an area of addressable memory that is not limited by the physical memory size. Virtual memory is used to deal with memory demands that exceed the actual physical memory size. Each address in virtual memory is mapped to a page frame in real physical memory.

```
// calling structres globally
par user;
statistic stat[6]; // for every thread
PTE* pte; // page table entries
help helper;
```

These are defined with typedefs for easy use of structs.

```
// thread declarations
pthread_t threads[6]; // linear, binary, fill, matrix multiplication, vector multiplication, array summation
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

I have defined 6 threads. I defined a field for 3 calculations, 2 searching and 1 vector filling thread I requested from us. I made the mutex definition global so that I can access it from anywhere. I aimed to enable threads to be opened and closed easily.

Project Creation Steps:

1) First of all, I provided the controls and storage of the parameters received from the user. I implemented this in (assign_entries) method. I checked the number of parameters in this method. I then assigned it to the variables I defined in the Struct structures. I used Struct elements here, I thought it was more object oriented than global. Then I checked whether the page Replacement type is included in the project. In our project, second change, lru and wsclock algorithms are working. Then I checked whether the numerically received values are greater than zero and compared the virtual and physical memory with each other. Then I reassigned these values as powers of 2 as shown in the assignment pdf.

2) I assigned the values given as struct and not defined first. The reason for the number 6 here is because I am using 6 threads. I did this in the (initials_structures) method.

3) I called the print method and printed the virtual memory before the page replacement was applied. Here I implemented the data extraction with the get method.

4) In the get method, I kept the row and column values for accessing the page table. I divided the index I into framesize for the row. For the column, I got the mod of the index and framesize. Here I printed the page table to the screen (with the print_pagetable) method. Then I started mutex I and changed the reference bit at the pte position in question and increased the lru counter I. After initializing the duration values, I found the hit-miss values (hit-miss). If the present bit at the sent location is 1, it indicates a miss, if not a hit. If it is hit, the physical memory equivalent of the coordinates sent to the sent value is returned. If there is a miss, a new disk element is created (getPageUnit), initial declarations are made and it is returned.

5) In the hit-mis method, if there is a miss, the generated page is sent to the page replacement algorithms. Calling the page replacement method of the type received from the user.

6) After these are finished, the program returns to the 696th state in main. Here it runs thread operations(thread_op).

7) In the thread operations method, creation and join are done for each thread. The for loops work by dividing them in half to add wait time. In this section, the thread_func method is called while creating a thread. In this method, task distribution is made for each thread. At the top of the project, the definition was made according to the order in the field where the threads were first defined.

8) Here, there is linear search in the first place. Here I set a random number and asked it to search for this number in virtual memory.

9) there is binary search in the second place. Here, I gave a random number. And I wanted him to find it. However, here, I wrote a method that divides the virtual memory sent in accordance with the binary seach algorithm and includes it in the process (binary).

10) There is a fill in the third row. A dedicated thread to save the data from the fill dat file to the vector.

11) In the fourth row, there is a method that provides matrix multiplication. Here I kept 2d vector. In order to do this multiplication, I searched for 2d vector among the ones I used before and did this with physical and virtual memories.

12) In the fifth row, there is the vector product. Here I have multiplied using a 1-dimensional vector. Here I used the pageFrame array to test it.

13) Sixth and lastly, I did array addition in the Summation part. There was no array in my previous uses for this. I also took the read and write elements of the stat array I used for statistics and used them.

14) Since I don't have enough time to apply the inverted part in this assignment, I only get this value, but I can't process it.

15) There are some methods that start with num at the beginning of the code. These are the incrementers of the variables kept for statistics.