CSE 312/504 – Operating Systems - Spring 2020

Final Exam Project Report

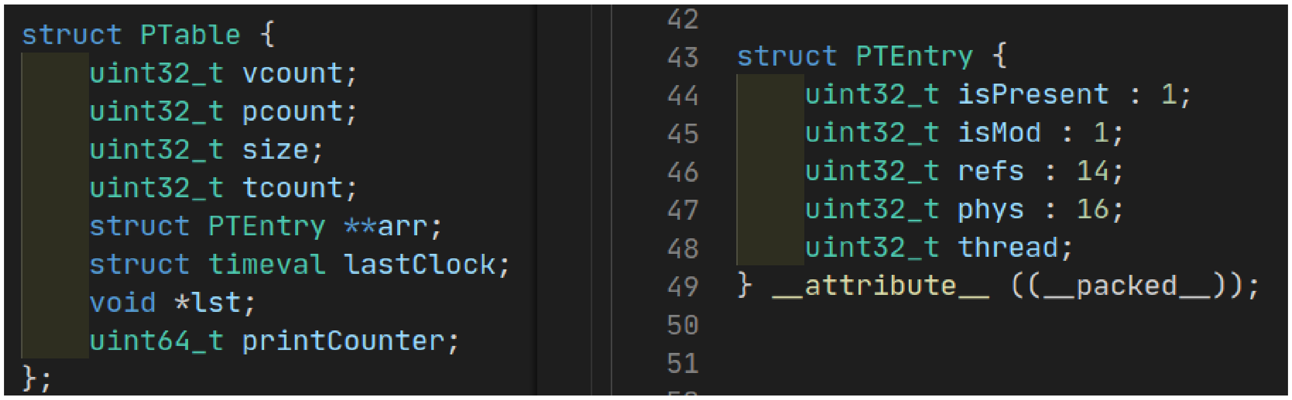
Student Name: Ozan Şelte

Student Number: 161044061

General Structure:

The language that used in this project is C with C++ list and vector. The virtual memory is represented as a binary file and the physical memory is a C array. Thread statistics stored by a dynamic structure array and the start configurations from console arguments stored by global structures. There is not any disk map, Figure 3-28(a) has been used for project. Virtual memory file size is constant.

To obtain syncronisation, a global mutex(pthread_mutex_t) used. With this mutex, page replacements done safely. In general, main thread fills the virtual memory, creates four new threads, waits all of them and finally, checks all of the numbers. In sorting algorithms, Index Sort has not been implemented because I do not know what that is and on the internet, I could not find the context. Anything other than index sort is working correctly according to my tests. There is not any memory leak, not closed file descriptor or not joined threads end of the program.


```c
struct PTable {
	uint32_t vcount;
	uint32_t pcount;
	uint32_t size;
	uint32_t tcount;
	struct PTEntry **arr;
	struct timeval lastClock;
	void *lst;
	uint64_t printCounter;
};
```

```c
struct PTEntry {
	uint32_t isPresent : 1;
	uint32_t isMod : 1;
	uint32_t refs : 14;
	uint32_t phys : 16;
	uint32_t thread;
} __attribute__ ((__packed__));
```

Page Table:

The page table contains the number of pages in the virtual memory(vcount), the number of pages in the physical memory(pcount), size of a page as integers(size) and the size of threads that worked until that time(tcount). These variables are mostly constant during the runtime. arr is the page entry array. There is a clock variable to simulate some reference jobs and lst is a dynamic variable according to the algorithm. std::vector for WSClock and std::list for FIFO or Second Chance.

In page table entry structure, isPresent and isMod is using in every algorithm but refs is an array. Homework wanted us a single page table structure for all of the algorithms so some algorithms using only one bit of refs and some(LRU, WSClock) using all. phys is the page id in physical memory(if is present) and finally the thread is an id to know which thread is the owner of the page. Threads information is in another structure array.

Thread Statistics Structure:

A statistics structure holds a name and seven different numbers. A global structure array keeps this data. id of the structure is thread id, and it’s name is in the structure as a char array. One of these numbers is active page number in physical memory and the others are the numbers which wanted in the homework. Page misses is generally a smaller number than page replacements, because at the start page is empty, page miss occurs but page replacement not.

Functions and Explanations:

void setLastTime();	Sets time variable to current time.
void checkClock();	Checks the time, if it passes the time limit does the interrupt jobs.
void initTable();	Initializes the page table.
void initVirtualMemory();	Initializes the virtual memory, opens and truncates the file.
void initPhysicalMemory();	Initializes the physical memory. Allocates space for the array.
void waitAlgos();	Waits child threads with pthread_join.
void freeResources();	Frees all allocates resources, closes the virtual memory file.
void fillAll();	Fills all of the virtual memory with random numbers. Uses set function only.
void checkSorting();	Checker function. Checks every 1/4 part of the virtual array.
void createThreads();	Creates 4 new thread for sorting algorithms, gives everyone a index number because index sort has not implemented. There is two merge sorts now.
void *bubbleMain(void *data);	Thread main for bubble sort.
int quickPartition(int low, int high, char *name);	Quick sort helper function.
void quickSort(int low, int high, char *name);	Recursive quick sort function.
void *quickMain(void *data);	Thread main for quick sort.
void mergeTwo(int low, int mid, int high, char *name);	Merge sort helper function.
void mergeSort(int low, int high, char *name);	Recursive merge sort function.
void *mergeMain(void *data);	Thread main for merge sort.
uint32_t physicalFrame(uint32_t vFrame, int64_t tid);	Returns physical page index for a virtual page. If it is not in physical memory, it searches an empty page, if it is also not be found, it calls a pagefault, then writes new virtual page to the destination.
uint32_t pageFault(uint32_t vFrame, int64_t tid);	Finds most unimportant page and edits its page entry. Calls the known algorithm to find a page.
void setPage(uint32_t vFrame, uint32_t pFrame, int64_t tid);	Sets the virtual page to given physical memory location. Edits its page entry.
int64_t findEmpty();	Tries to find an empty page location in physical memory.
int64_t findThread(const char *tName);	Tries to find a thread with given name.
uint32_t countTid(int64_t tid);	Returns how many page is active on physical memory for that thread.
uint32_t countNames();	Counts how many threads worked until that time. If there is not any empty slot in structure array, reallocates it.

uint32_t countActiveThreads();	Counts how many thread has an active page un physical memory.
int64_t biggestTid();	Returns the most occupying thread's id. It is needed for local allocation.
uint32_t physicalAddress(unsigned int index, const char *tName, uint8_t mod);	Returns the physical address for given index, it does all nesessary jobs like page fault, replacement etc.
void printTable();	Prints page table.
void printStats();	Prints statistics for all threads.
void set(unsigned int index, int value, const char *tName);	Sets the given integer to the virtual memory.
int get(unsigned int index, const char *tName);	Returns the number in the given location.
uint32_t nruAlgorithm(int64_t tid);	NRU algorithm for page fault.
uint32_t fifoAlgorithm(int64_t tid);	FIFO algorithm for page fault.
uint32_t scAlgorithm(int64_t tid);	Second Chance algorithm for page fault.
uint32_t lruAlgorithm(int64_t tid);	LRU algorithm for page fault.
uint32_t wscAlgorithm(int64_t tid);	WSClock algorithm for page fault.

General Flow

When a get(or set) called, they lock the mutex first, then program calls the physicalAddress function to identify the actual index for physical memory array. When they got the right address they save the number in it(or write to it) than unlock the mutex. physicalAddress function parses the index to virtual page and address in page parts. Calculates the physical frame index, writes thread name to the statistics structure and calls helper functions according to the replacement algorithm. checkClock for NRU, LRU and WSClock and queue pushing for FIFO and Second Chance. After these operations, it program edits the page entry. If print table needed, it also prints it. Then, returns the physical address.

If the thread name is new, the structure in the stats array is initialized. Then physicalFrame function is calling. If the wanted page in already in physical memory, it returns. If not it checks there is an empty page in physical memory or not. If GLOBAL is selected, it takes the free page but in LOCAL, it checks the count for itself. It it has the maximum, it calls page fault. When there is not an empty page in physical memory(or in local allocation it has to do page fault), it does a page fault. If local allocation is selected but the new thread is came to the memory, it must take a page from another running thread. For this reason, program finds the thread which keeps most of the pages, and takes from it to the current thread(pageTable.cpp:385&392).

Not Recently Used Algorithm

If NRU is selected, after each access to the memory, program checks the clock. If after the last check the given time(pageTable.h:9) has been passed, if shifts the refs. For NRU, it means the clearing of R bit. It uses first bit of refs variable as R(pageTable.cpp:675).

Least Recently Used Algorithm

If LRU is selected, after each access to the memory, program checks the clock. If after the last check the given time(pageTable.h:9) has been passed, if shifts the refs. For LRU, it means the aging process of NFU. It uses the value of refs and isModified(pageTable.cpp:753).

FIFO Algorithm

If FIFO is selected, after every memory access program adds current page to end of the list(pageTable.cpp:588). Removes the selected page, and skips other threads pages for LOCAL allocation. FIFO uses std::list<uint32_t> which in void* in Page Table.

Second Chance Algorithm

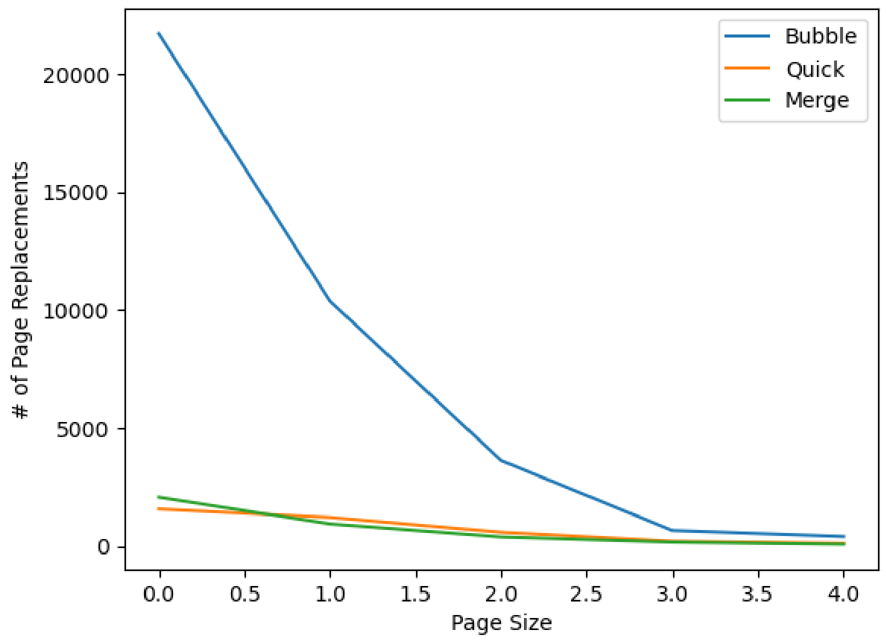
If SC is selected, after every memory access program adds current page to end of the list(pageTable.cpp:588). Removes the selected page, and skips other threads pages for LOCAL allocation. It uses the first bit of refs as R bit(pageTable.cpp:728). SC uses std::list<uint32_t> which in void* in Page Table.

WS Clock Algorithm

If WSClock is selected, after each access to the memory, program checks the clock. If after the last check the given time(pageTable.h:9) has been passed, if shifts the refs. For WSClock, it means the clock check for Clock algorithm. It uses the value of refs(pageTable.cpp:780). WSClock uses std::vector<uint32_t> which in void* in Page Table. The first element is the pointer of the algorithm, the latest page(pageTable.cpp:769).

Part 3 Statistics

My program runs slowly for given numbers so I reduced them. I used perf(a profiling tool) and cannot be fasten. For every sort algorithm I wrote, when the page size increases, pare replacement decrease. I run the program for all algorithms and all policies, and then calculated the geometric mean. The graph is includes that values.


Bubble
Quick
Merge
# of Page Replacements
Page Size