Project 2 Architectural Document

Yuteng Zhou (ID:827819800)
Department of Electrical and Computer Engineering, Worcester Polytechnic Institute

I. A List of What is Included in My Design

In this project, all the tests from test2a to test2g has been successfully tested. In my design, besides files provided by professor, I have added four header files and four C source files into the design. They are queue_function.h, queue_function.c, message_function.h, message_function.c, yuteng.h, yuteng.c, memory_function.h, memory_function.c. Since the first six files have already been explained in phase 1, here I only explain memory_function.h and memory_function.c. Of course, some modifications have been done to existing functions and some existing structs such as OSCreateProcess() function and struct PCB, these modifications will also be explained in this section. Below is a list of what's included in these files, more details can be seen in my code.

First of all, to give a brief impression of my file system. The file association is plotted in figure 1.

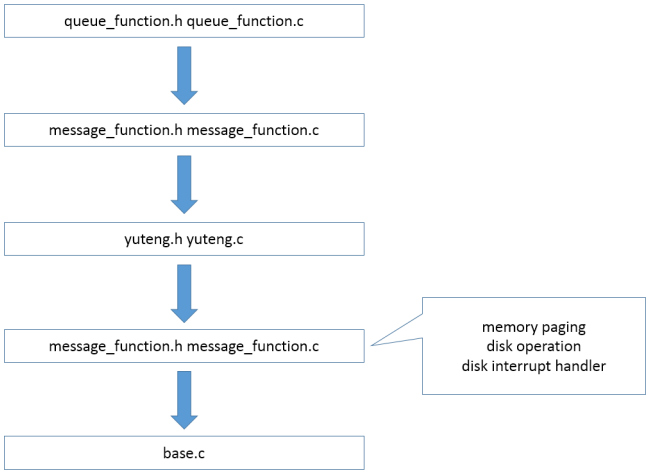

Figure 1. Files Overview

A. Modifications in queue_function.h

Originally in this file, struct PCB is declared. Now the PCB struct contains additional items, they are one page table and one shadow page table with length 1024, 1024 is the maximum logical pages our system supports. Also, PCB struct contains items like disk id, disk sector, one pointer to disk data to be read or written, and one flag indicating read disk operation or write disk operation. Eight disk waiting queues are also declared. The architecture of disk queue is shown in figure 2, in our system, there are 8 disks, for each individual disk, operation (either read or write) is unique, that is when disk 1 is handling request by process 1 as shown in figure 2, process 3 is also trying to use the disk 1, then it has to wait for process 1 to finish operating the disk. The same behavior applies to other disks. Disk operation on different disks can be performed at the same time, that is, in figure 2, while disk 1 is handling process 1's request, disk 2 can also be running which is started by process 0 shown in this figure.

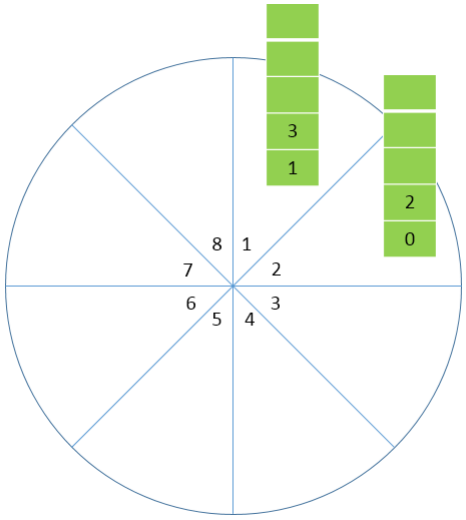

Figure 2. Disk Queue

B. Modifications in yuteng.c

In this file, OSCreateProcess() function is realized, since now PCB contains more items, in the OSCreateProcess() function, these newly added items need to be initialized too. Each entry in both page table and shadow page table is to be initialized to 0 indicating these pages are invalid at the very beginning.

C. memory_function.h

It contains declarations for memory paging, page replacement function when all 64 physical frames in this system have been used, a declaration for struct PAGE, which keeps record of physical page utilization in this system. The PAGE struct records process ID, logical address, and physical frame number taken by that process's logical address, it also contains pointer to another PAGE struct. Besides, it has declarations for SVC calls disk read and disk write.

D. memory_function.c

In this file, all the memory and disk related functions are to be realized. It contains memory_mapping() function, which

returns a valid physical frame number, if all physical frame has been occupied, then it will call the page_replacement() function to exchange page contents with disk. Also, it contains disk_interrupt() function, when a disk interrupt is triggered, meaning current disk finishes its required operation no matter it's a read or write, then the disk_interrupt routine will pop out the process and push it back to ready queue, then it will check it there is no more process waiting to use the disk, if there is at least one process, the corresponding disk will be started right away. The memory_function.c file also contains print memory function.

II. High Level Design

In phase 2, we are primarily focused on memory access and disk access, basically four functions, memory read, memory write, disk read and disk write. These four functions are to be explained in details in following subsections.

A. MEM_READ/MEM_WRITE

In this system, both MEM_READ and MEM_WRITE functions go to MemoryCommon function. Initially each page is set to invalid, so it will cause program goes to fault handler, then in the fault handler, I set up the memory_mapping function to assign one physical frame in the system to the logical address defined by user. The function also returns the physical frame number. The flow is shown in figure 3.

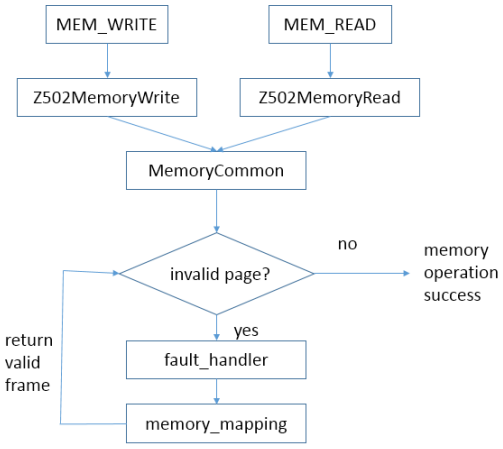

Figure 3. MEM_READ/MEM_WRITE

B. Disk Read

Disk read is one SVC call, and it tries to read data from a sector of one disk. The flow of disk read is shown in figure 4. It first stores disk operation information to current PCB's disk related items, like disk id, disk sector, disk operation (read or write), pointer to disk data. Then current PCB needs to be removed from ready queue and been pushed into disk waiting queue for the target disk. After then, I need to check whether the target disk is free, if so, it means current PCB is the only process using the disk, so I can start the disk right away, and then reschedule. Otherwise, reschedule the process without starting the disk, since disk is currently busy operating other process's request.

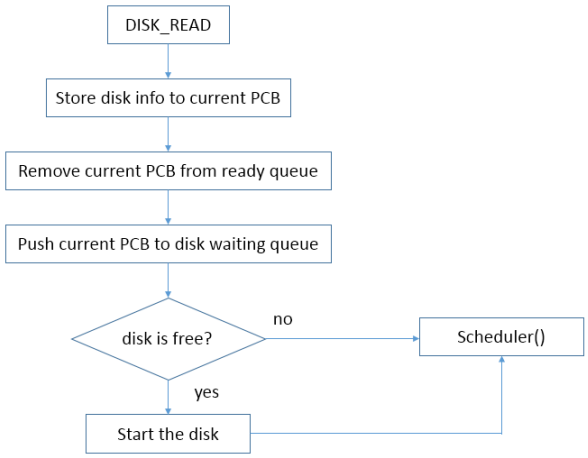

Figure 4. Disk Read

C. Disk Write

Disk write resembles disk read operation, the only difference is disk operation needs to be write instead of read. The flow of disk write is shown in figure 5.

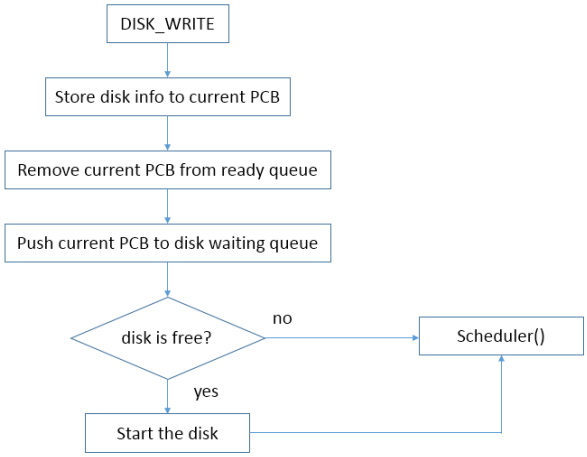

Figure 5. Disk Write

D. Disk Interrupt

Each time when a disk finishes its current operation either read or write, it goes to interrupt. In the disk interrupt, I need to move the process which finished disk operation from disk waiting queue back to ready queue. Then check for the current disk, whether there is at least one process waiting for disk operation, if so, start the disk for the topmost process in disk waiting queue. The flow of disk interrupt is shown in figure 6.

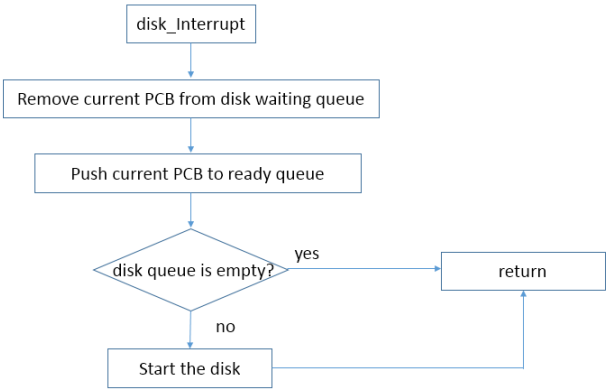

Figure 6. Disk Interrupt

E. Page Replacement

In this system, there are only 64 physical frames, when all these frames are occupied, then page replacement is needed. The procedure of page replacement is shown in figure 7. When there is no physical frames available in the system, the memory_mapping() function will search for an appropriate page to be replaced based on first in first out algorithm. Then I need to write the page been replaced to disk, and update shadow page table to record when the page information has been stored in the disk. Then I check whether it is the first time to use the logical address, if so, simply return the valid page. Otherwise, which means the page requested is in the disk not in memory, I need to read the page from disk, and then modify the page content.

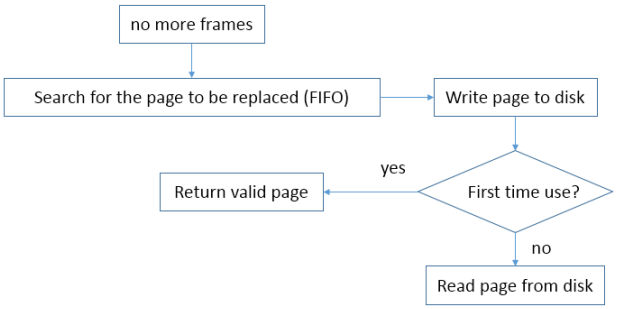

Figure 7. Page Replacement

III. Justification of High Level Design

Here, all the tests from test2a to test2g have been successfully tested. Detailed flows for these tests are to be explained in details in following sections.

A. test2a

Test2a simply checks the basic of MEM_READ and MEM_WRITE function. Its easy to pass.

B. test2b

Test2b also tests the MEM_READ and MEM_WRITE, while with more test data, and if my memory mapping system is not robust enough, test2b will find out by writing to the same physical address. The flow of test2b is shown in figure 8. Test2b first writes data to logical address 80, then enters a while loop, it write and read data, and compared whether data be written and data be read are the same. Also, it checks the data at address 80 matches data written at address 80.

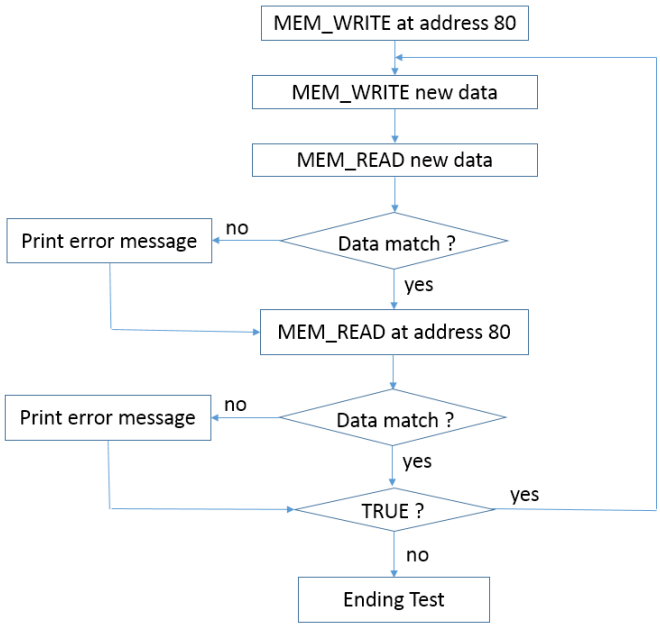

Figure 8. Test2b

C. Test2c

Test2c tests the basic operation of disk read and disk write. The flow for test2c is shown in figure 9. It first writes to disk and reads from disk, then compares data read and data written, repeats the procedure for 50 times, then read back data for 50 times to see whether data can be read successfully. After then, terminate the system. My OS system passes test2c successfully,

D. Test2d

Test2d basically tests multiply test2c, it first creates five processes, each process runs test2c. Then current process sleeps for a long time, then processes 1 to 5 run. For test2d, the main difference between test2d and test2c is that in test2c, only process 0 is using disk 1, so each time when disk interrupt is triggered, there is only one disk, while in test2d, 5 processes are using 3 disks, so when a disk interrupt has been triggered, there may be more than one disks finish its operation. So in the interrupt handler, I put the disk interrupt handler function into a while loop, that is each time one disk interrupt has been handled, it will check whether there is still another interrupt, in this way no interrupts will be missed.

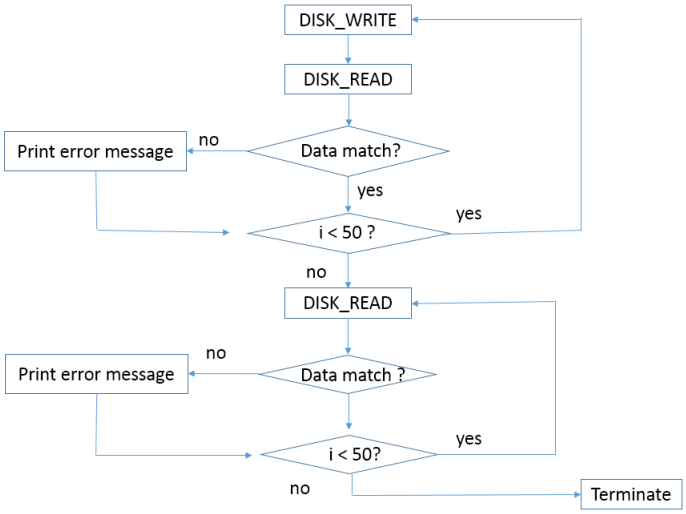

Figure 9. Test2c

E. Test2e

Test2e simply has more test cases on MEM_READ and MEM_WRITE.

F. Test2f

Test2f uses up all the 64 physical frames in the system, so to test page replacement feature in the system. The flow of test2f is shown in figure 10. In test2f, it simply has much more MEM_WRITE and MEM_READ requests than test2b, forcing system to do page replacement. The page replacement procedure is shown previously in section 2.5.

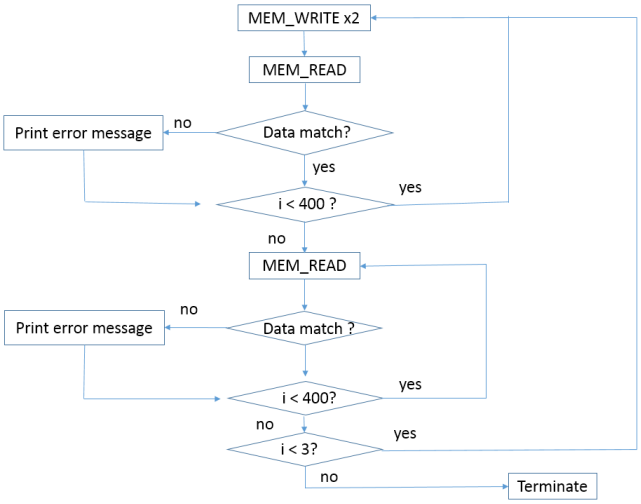

Figure 10. Test2f

G. Test2g

Test2g tests multiple copies of test2f. It first creates five processes which run test2f, then process 0 sleeps for 10000 to let each process finish its operation.

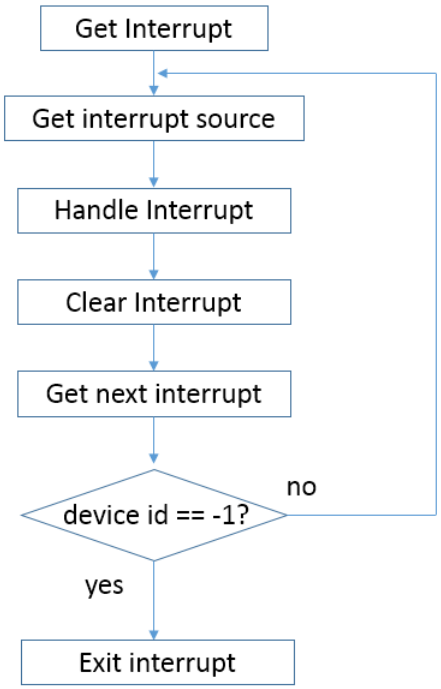

Figure 11. Test2f

IV. Unique Parts of My Project

In my project, including phase 1 and phase 2, I think I have several parts in my code which is unique.

A. PCB struct

In my system, my PCB stores a lot of information. Besides process ID, process name, I put message related information in PCB. Also, I put page table information in PCB, that is, each PCB has a 1024 length table, in my system, each logical address is unique, that is to say, if process 1 takes the logical address 100, then this logical address 100 cannot be used by other processes. In this way, I am customizing page table as if there is only one 1024 length page table. PCB struct also has disk related information. In this way, in my system, there is no need to set up other structs, no matter one process is in suspend queue, ready queue, disk waiting queue, I only need to check the corresponding item in the process.

B. File Organization

In my system, I have added 8 files, 4 header files and 4 source files, for these 4 header files, the first one inherits almost all the header files provided by professor, then other header files only need to inherit one other header file, the last header file added is inherited in base.c. In this way, it is pretty clear to put a function declaration to a corresponding header file.

C. Interrupt Handler

In my interrupt handler, I put the whole interrupt handling function into a loop. Once program runs into interrupt, it will

handle one interrupt, then I will get next interrupt, return value is the device id, if device id is not -1, meaning there is still one interrupt been unhandled, I need to handle the interrupt again. The loop exits until no more interrupts are to be triggered this time. Such a mechanism avoids the situation that the OS may has more than one interrupts at one time caused by different interrupt sources, and the OS only handles one of them. The flow is indicated in figure 11.

D. Current PCB & Ready Queue

In my operating system, current PCB also exists in the ready queue, in this way, it eases coding of some functions like reordering the ready queue once one process's priority has been changed.

V. Additional features

In phase 2, I didn't do test2h which tests the shared memory feature. My additional feature will be, when one process has been push into disk waiting list, I can still use suspend call to suspend to process.