Architectural Document 2

1. A list of what is include in my design.

1.1 Data Structures:

(1) Process Control Board (PCB) node structure defined in “Data.h” head file as “Data”. Except the basic processes’ information such as Process_ID, Process_Name, Process_Status, Process_Pointer(context), and Argument (WakeupTime/DiskID), I added PageTable and 6 other structures into the PCB node structure in Project 2. The six new added structures are Shadow_PageTable, Shared_Manager, current_Dir, parent_Dir, send_message, and receive_message. I will list these new created structures in detail next.

(2) Struct Shadow_PageTable is defined in “ShadowPageTable.h” head file, which is used for memory management. When the data in memory was swapped out to disk, the structure Shadow_PageTable will record the swap out location: “Diskid”, “Sector”, and “inDisk” status.

(3) Struct Shared_Manager is defined in “shared_manager.h” head file, which is used for shared memory management. It records the information of “StartingAddressofSharedArea” and “PagesInSharedArea” of the PageTable. It also records the “AreaTag”.

(4) Struct Directory is defined in “Data.h” head file, which is used for file system management. Both current_Dir, parent_Dir inside PCB node structure are belong to this Directory structure.

(5) Struct Message_Send and Message_Receive are defined in “Message.h” head file, which is used for send/receive message between processes.

Besides, I have three other structures for memory management purpose:

(6) Struct DiskManager is defined in “DishManager.h” head file, which is used to swap area management during memory management.

(7) Struct frame and Struct frameManager are defined in “FrameManager.h” head file, which are used to create FrameTable and manage frames during memory management.

Totally I have four queues in my design, except TimerQueue, ReadyQueue, DiskQueue mentioned in Project 1, I added another MessageQueue in Project 2:

(8) “MessageQueue.h” defines the message queue structure and its head and tail. Single linked list structure is applied on queue structure. And “MessageQueue.c” file includes all related functions to do message queue operations.

1.2 System Call Routines

1.2.1 *Existing in Project 1*

The one I made changes in Project 2 will be explained below.

GET_TIME_OF_DAY: Handled in svc

TERMINATE_PROCESS: **void TerminateProsess**(SYSTEM_CALL_DATA *SystemCallData);

long* **Terminate_with_ProcessID**(**long** ProcessID);

void Terminate_Whole_Simulation();

In the function of **Terminate_with_ProcessID,** I added some codes to update the FrameTable in Project 2. When a process is terminated, the frames it occupied will be released here for other non-terminated processes.

SLEEP: **void StartTimer**(SYSTEM_CALL_DATA *SystemCallData);

GET_PROCESS_ID: **void GetProcessID**(SYSTEM_CALL_DATA * SystemCallData);

CREATE_PROCESS: **void CreatProcess**(SYSTEM_CALL_DATA *SystemCallData);

PHYSICAL_DISK_WRITE: **void DiskWrite**(SYSTEM_CALL_DATA *SystemCallData);

PHYSICAL_DISK_READ: **void DiskRead**(SYSTEM_CALL_DATA *SystemCallData);

FORMAT: **void DiskFormat**(SYSTEM_CALL_DATA *SystemCallData);

char* **DiskSector**(**long** DiskID, **long** Sector);

I made some changes on the size of swap area. And I put all location parameters of disk format into “DiskFormat.h” head file.

CHECK_DISK: **void DiskCheck**(SYSTEM_CALL_DATA *SystemCallData);

OPEN_DIR: **void DiskOpenDIR**(SYSTEM_CALL_DATA *SystemCallData);

CREATE_DIR: **void DiskCreateDIR**(SYSTEM_CALL_DATA *SystemCallData);

CREATE_FILE: **void DiskCreateFile**(SYSTEM_CALL_DATA *SystemCallData);

1.2.2 New created in Project2:

DEFINE_SHARED_AREA:
void Define_Shared_Area(SYSTEM_CALL_DATA * SystemCallData);

SEND_MESSAGE: **void Send_MESSAGE**(SYSTEM_CALL_DATA * SystemCallData);

RECEIVE_MESSAGE: **void Receive_MESSAGE**(SYSTEM_CALL_DATA * SystemCallData);

1.3 Other functions

1.3.1 *Other functions in Project1*

The list of functions already in Project 1 will not be shown here again. I will only explain the ones I made changes:

-void Dispatcher():

Get PCB node from Ready Queue and then call “StartContext”. Before checking readyQueue, I added some codes to check the Mailbox and MessageQueue first. The Mailbox recorded all process_Id for the processes with receive_message action but did not receive the message yet. And MessageQueue had all messages sent out by any process. In this function, I will let it check MessageQueue with process_id provided by Mailbox. If there has messages for provided process in MessageQueue, it will move this process from Mailbox to readyQueue. That means, make this process ready to run.

1.3.2 *Other functions in Project2*

-void FaultHandler(**void**):

This function is used to receive hardware faults. In project 2, this function is also used to memory management.

-int FindFreeFrame():

This function is to find free frame by checking FrameTable.

-int CheckSwapBitmap(**long** DiskID):

This function is to find out free sector in swap area of given disk.

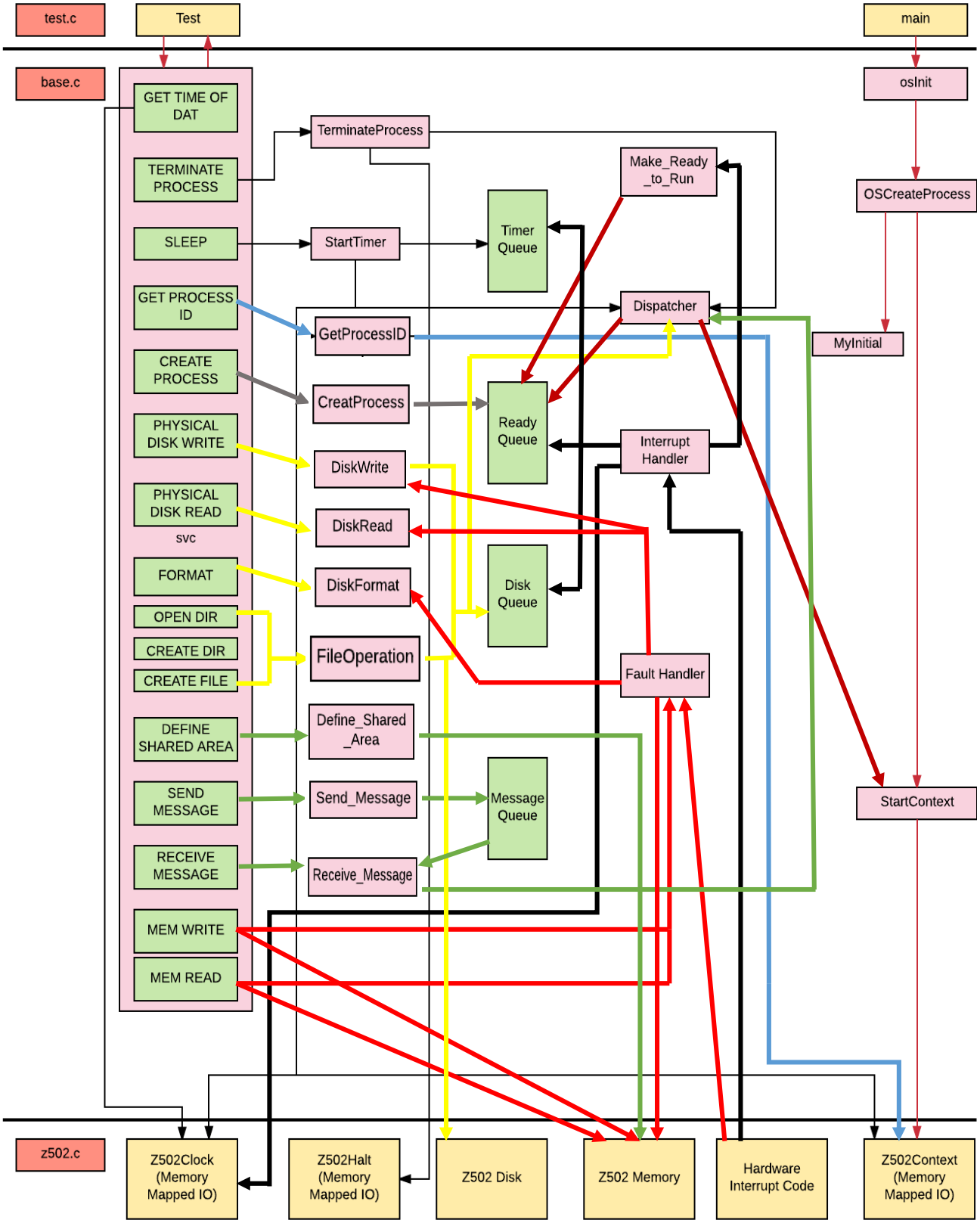
-int Find_Victim():

This function is to find victim to swap out in frames when all frames are occupied. The function follows LRU algorithm.

-void Find_Current_Free_Swap_Disk():

This function is to find the disk with available swap area. The rule here is to check if there has existing formatted disk first. If there have formatted disk, it will choose this formatted disk. If there no formatted disk at all, it will choose the first available disk.

2. High Level Design



3. Justification of the High Level Design.

The processes scheduling progress mentioned in Project 1 will not be repeated here. In this report, I will explain my design on Memory Management and message mechanism between processes.

3.1 *Memory Management*

Although “MEM_WRITE” and “MEM_READ” are directly called in test.c without go through “svc” function, I still put them inside “svc” routines in the diagram of high level design for easier explanation.

First time a process calling “MEM_WRITE” will introduce Hardware Interrupt to Fault Handler. Fault Handler need to transfer the virtual address to physical address, that is, fault handler has to find a free memory frame and assigned it to the virtual address. And the transfer of virtual address to memory address will be stored inside PageTable of this process. On the other hand, the FrameTable is used to record the ProcessID and virtual Page number for each frame. Finally the valid bit of PageTable will be set to 1. After the valid bit is set to “1”, both “MEM_WRITE” and “MEM_READ” will access the memory directly without going into fault handler again as long as they visit the same virtual address for same process.

When the frames is used up, there is no more free frame can be assigned to new virtual address. Now calling “MEM_WRITE” will introduce hardware interrupt to Fault Handler. Fault Handler need to find a victim frame X for example, swap out the data inside X into disk (access “Z502Memory” and “DiskWrite” function), record the new location of the data in Shadow PageTable of the process and update its PageTable as well(the valid bit and modified bit will be set to 0, the reserved bit will be set to 1). Then the Fault Handler will treat the frame X as a free frame and repeat the progresses explained in last paragraph.

Similarly, When the frames is used up and valid bit of accessing virtual page is 0, calling “MEM_Read” will introduce hardware interrupt to Fault Handler. That is because the data the process want to read is not in memory any more. If the reserved bit is 1, it means the data is stored in disk. Now the Fault Handler need to find a victim frame Y for example, swap out the data inside Y into disk (access “Z502Memory” and “DiskWrite” function), record the new location of the data in Shadow PageTable of the process and update its PageTable as well (the valid bit and modified bit will be set to 0, the reserved bit will be set to 1). Then the Fault Handler need to swap in the required data (access “DiskRead” function and “Z502Memory”) and update its PageTable (set valid bit to 1 and record frame address). And update FrameTable.

If need swap in/out data, Fault Handle will check if there exist formatted disk first. If there no formatted disk, Fault Handle need to access “DiskFormat” function to do disk format so that there has available swap area.

Some details of my design:
(1) When the data is swapped out, the shadow PageTable will be updated and records the data's “inDisk” status. If the same data need to be swapped out second time, Fault handler will check the data is modified or not after last time swapped in. If it was modified, the modified data will be updated in the same location on the disk swap area. If it was not modified, then the data is not going to be updated once more.

(2) “Find_Victim” function will follow LRU algorithm: For each frame, the algorithm will check the reference bit from its corresponding PageTable. If the reference bit is 1, the algorithm will set it to 0. If the reference bit is 0, it will check the modified bit then. If the modified bit is 0 as well, this frame will be pick up as a victim. If the modified bit is 1, the algorithm will go on to check next frame. The rule here is to choose the victim frame which has both reference bit and modified bit equal to 0 firstly. If there do not have 0 modified bit after checking all frames, the algorithm will pick the first 0 reference bit one.

3.*2 Shared Memory and Message mechanism between processes.*

“Define_Shared_Area” function will define the shared virtual address and link them with physical memory address. Update PageTable and FrameTable. And return unique “OurSharedID”. Several difference processes will be defined with a same physical area as their shared memory area.

“Send_MESSAGE**”** function will put the message into MessageQueue. “Receive_MESSAGE” will receive the message from MessageQueue. If there is no requiring message in MessageQueue yet, “Receive_MESSAGE” function will put the processID into Mailbox, then call Dispatcher. In “Dispatcher”, it will check the MessageQueue. If there is requiring message available in MessageQueue, “Dispatcher” will move the process (waiting for receive some message) from Mailbox to ReadyQueue.