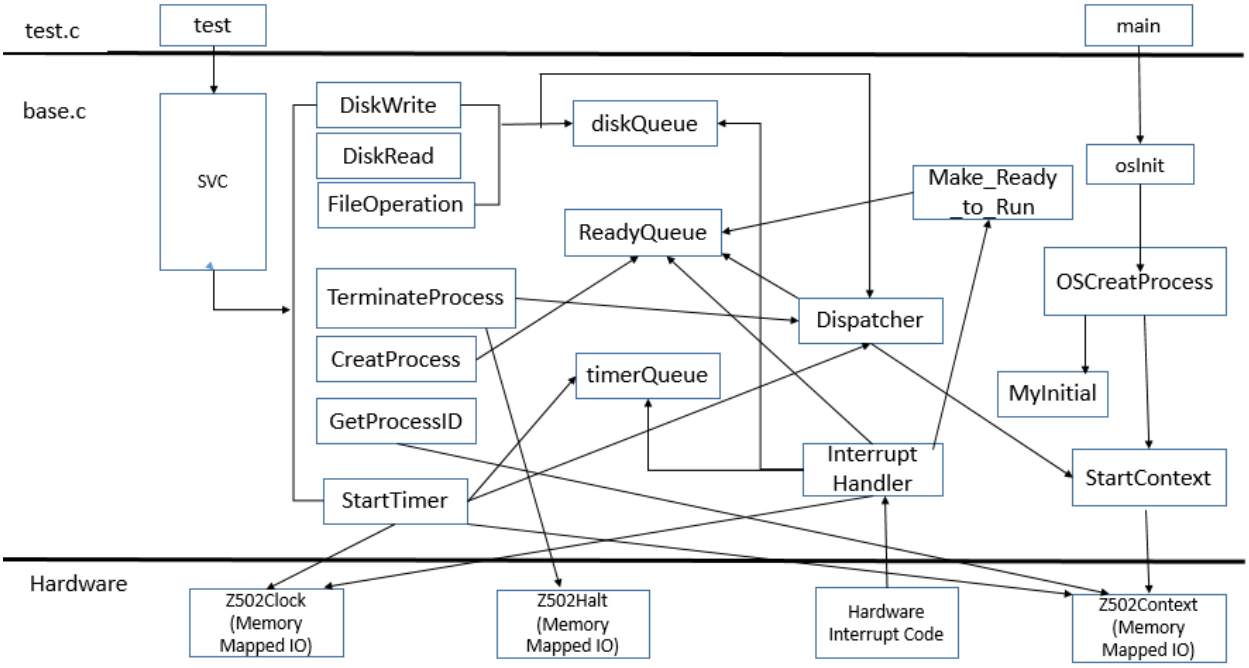
Architectural Document 1

1. A list of what is include in my design.
   (1) “Data.h” head file defines the Process Control Board (PCB) node structure.
   (2) “Data.c” source file includes the functions to create PCB node and release PCB node.
   (3) “TimerQueue.h” defines the timer queue structure and its head and tail. Single linked list structure is applied on queue structure.
   (4) “TimerQueue.c” includes all related functions to do timer queue operation.
   (5) “ReadyQueue.h” defines the ready queue structure and its head and tail. Same single linked list structure with timer queue is applied on queue structure.
   (6) “ReadyQueue.c” includes all related functions to do ready queue operation.
   (7) “DiskQueue.h” defines the disk queue structure and its head and tail. Same single linked list structure with timer queue is applied on queue structure.
   (8) “DiskQueue.c” includes all related functions to do disk queue operation.
   (9) All other Operating System operations are implemented in “base.c” source file:
      1) “osInit”: The first routine is called after the simulation begins. This function tells which test will be started and then it calls “OSCreatProcess” to create new process.
      2) “OSCreatProcess”: This function creates new process for the initial input test.
      3) “MyInitial”: This function creates PCB node and all queues to record the information of a new process.
      4) “StartContext”: Start a new context with Memory Mapped IO interface with hardware “Z502Context”.
      5) “Get_Current_ContextID”: Get context ID of the current executing PCB node with Memory Mapped IO interface with hardware “Z502Context”.
      6) “svc”: Handle all system calls.
      7) “GetProcessID”: Response to system call “GET_PROCESS_ID”.
      8) “CreatProcess”: Response to system call “CREATE_PROCESS”.
      9) “TerminateProcess”: Response to system call “TERNIMATE_PROCESS”.
      10) “Ternimate_with_ProcessID”: Called by “TerminateProcess”.
      11) “Terminate_Whole_Simulation”: Called by “TerminateProcess”.
      12) “InterruptHandler”: Handle Z502 hardware interrupt.
      13) “Make_Ready_to_Run”: Put PCB node into Ready Queue.
      14) “Dispatcher”: Get PCB node from Ready Queue and then call “StartContext”.
      15) “wasteTime” : Called by “Dispatcher”.
      16) “StartTimer”: Response to system call “SLEEP”.
      17) “PutNodeinTimerQueue”: Called by “StartTimer”.
      18) “CheckSamePCBName” : Check if there is existing PCB name in PCB nodes’ array.
      19) “FindExecutingPCBNode” : Get the PCB node with current executing context ID.

20) “DiskWrite”: Response to system call “PHYSICAL_DISK_WRITE”.
21) “DiskRead”: Response to system call “PHYSICAL_DISK_READ”.
22) “DiskFormat”: Response to system call “FORMAT”.
23) “DiskSector”: Called by “DiskFormat”.
24) “DiskCheck”: Response to system call “CHECK_DISK”.
25) “DiskOpenDIR”: Response to system call “OPEN_DIR”.
26) “DiskCreateDIR”: Response to system call “CREATE_DIR”.
27) “Dolock”: Lock the queue before queue operation.
28) “DoUnlock”: Unlock the queue after queue operation.

2. High Level Design Diagram.



Where the “FileOperation” includes all file system operations such as “DiskFormat”, “DiskOpenDIR”, “DiskCreateDIR”, etc.

3. Justification of the High Level Design.

The simulation will start in “main(int argc, char* argv[])”. The parameters received by “main” function is the test name, which will be passed to “osInit” to tell which test will be started. The “osInit” will call “OSCreatProcess” to create the new process for the initial input test. “MyInitial” is designed to create new PCB node and all empty queues (ready queue, timer queue and disk queue). The created PCB node will be put into a global array named “pcbnode[]”. All information about the new process will be recorded in PCB node. Finally, “OSCreatProcess” will call “StartContext” to create a new context that allow the test to run.

Now we go to test to run the test in user code “test.c”. Test will do multiple system calls which produce the software interrupts. And the software interrupts cause execution to go to “svc” (service routine) in “base.c”. “svc” deals with multiple system calls include “GetProcessID”, “CreatProcess”, “TerminateProcess”, “StartTimer”, “DiskRead”, “DiskWrite”, and so on.

If the system call is “SLEEP”, it will call “StartTimer” function. “StartTimer” will contact with “Z502Clock” to find out current time and contact with “Z502Context” to find out current executing PCB node. After that, “StartTimer” writes “wakeup time” into PCB node and put the PCB node into timer queue. Inside timer queue, the nearest “waking up” PCB node will be put in the front. That means the smallest “wakeup time” PCB node will be put in front of the timer queue. Then “StartTimer” check if this input PCB node is the first node in timer queue. If it is the first, “StartTimer” contact with “Z502Clock” to start this timer/sleep. Finally, “StartTimer” will call “Dispatcher”.

“Dispatcher” will check the ready queue continuously. If there no PCB node inside ready queue, it will wait until there has new PCB node goes in. As long as there is PCB node in ready queue, “Dispatcher” will get the first PCB node from ready queue and call “StartContext” to create new context to allow this PCB node to start. The ready queue in always First Come First Serve.

When it comes to the “wakeup time”, the “InterruptHandler” will receive a hardware interrupt from Z502. Then “InterruptHandler” will get the PCB node out of the timer queue, clean the “wakeup time” and call “Make_Ready_to_Run”. “Make_Ready_to_Run” will put the PCB node into ready queue. After that, “InterruptHandler” will check the timer queue to see if there still have other PCB nodes waiting inside the timer queue. If the timer queue is not empty, “InterruptHandler” will read the information of the first PCB node inside timer queue, contact with “Z502Clock” to find out current time, and start the new timer/sleep.

If the system call is “PHYSICAL_DISK_READ”, or “PHYSICAL_DISK_WRITE”, “svc” will call “DiskRead” or “DiskWrite”. Since there are total 8 disk in hardware. “DiskRead” or “DiskWrite” will check DiskID in disk queue to see if there is already have the same DiskID inside disk queue. If there is already exist same DiskID in disk queue, the coming PCB node should wait here. If there is no same DiskID in disk queue, then we put the coming PCB node in disk queue. Then contact hardware to do operation that read or write data. Finally, “DiskRead” or “DiskWrite” will call “Dispathcer”, and “Dispathcer” will do same action mentioned before.

When disk operation completes, the “InterruptHandler” will receive a hardware interrupt from Z502. Then “InterruptHandler” will get the PCB node out of the disk

queue, and call “Make_Ready_to_Run”. “Make_Ready_to_Run” will put the PCB node into ready queue.

If system call is “CREATE_PROCESS”, “svc” will call “CreatProcess”. All new created PCB node will be stored in global array “pcbnode[]” and put into ready queue.
If system call is “TERNIMATE_PROCESS”, “svc” will call “TerminateProcess”. If terminate the process with given process ID, “TerminateProcess” will check ready queue, timer queue and disk queue to see if there have other processes. If any one queue is not empty, it will call “Dispathcer” to start new process.Otherwise, it contacts with “Z502Halt” to terminate the whole simulation.

If system call is “GET_PROCESS_ID”, “svc” will call “GetProcessID”. According the input parameters, “GetProcessID” will either contact with “Z502Context” to find out current executing PCB node and return its process ID, or check the global array “pcbnode[]” to find out the PCB node with same process name and return its process ID.