**Project one report of CS 502 Operating System**

Hongmei Zong

The project one is the first phase implementation of my os502 operating system. I have put my effort to make the os502 run successfully and provide as much functionalities as I can in this phase.

The os502 operating system contains several important modules, structures, queues and global variables.

*The major components and queues in the os502:*

1. **SVC** – it deals with all kinds of user commands. The project one version of my design fully implements the following functions:
2. GET\_TIME\_OF\_DAY
3. CREATE\_PROCESS
4. TERMINATE\_PROCESS
5. GET\_PROCESSID
6. SLEEP
7. PHYSICAL\_DISK\_WRITE
8. PHYSICAL\_DISL\_READ
9. FORMAT
10. **Ready queue**- it contains the processes in order and those processes have the status of ready-to-run. The structure of ready queue contains a pointer to process control block and a pointer point to the next queue element. The queue has the FCFS scheduling behavior. Whenever the CPU is available, the process in the queue head position will start to run. The process always enqueue at tail and de queue at head. In my implementation, there are three places will enqueuer the process to ready queue, one is current process start a SLEEP system call, the other two are DISK READ or WRITE. Since these three system call take time to finish, so the CPU will give other processes a chance to continue run.
11. **Dispatch**- the main task of dispatch is to check if there are processes in ready queue and waiting for their turn to run. If no process, then advance the time and do nothing.
12. **Timer queue**- there is one timer in hardware, so we maintained a timer queue for those processes that need to do SLEEP. The timer queue structure contains the pointer to process control block, the wake up time and the pointer to the next queue element. The timer queue is in order of wake up time. The sooner wake up time, the close to head of queue. When enqueue the process, it need to compare the wake up time and insert to a proper position in the timer queue by wake up time. If the process enqueue at head position, the timer need to be reset. When timer interrupt happened, the head of the queue is dequeued from timer queue.
13. **Disk queue**- There are 8 disks in hardware, we maintained 8 disk queues for different disks. The structure of disk queue has diskID, sector, read/write, address, and pointers to PCB and next queue element. DiskID and sector are targeted disk and sector, the process need to access. Read/write used to identify the type of the disk operation. If disk read, read/write= 0, otherwise write, read/write =1. Address is the pointer pointing to the buffer address of data. The pointers to PCB and next queue element also included in the disk queue structure. Disk queue is the FCFS scheduling behavior. When the process had a DISK read or write, the process enqueued at disk queue. When disk interrupt occurred from a specific disk, the head element is dequeued from that disk queue.

*The structures defined in os502:*

1. **Process control block(PCB).** It keeps information of process ID, process name, pointer to the page table and the running status, context and its parent process ID to identify each process. Below is the syntax for defining PCB.

typedef struct {

INT32 processID;

char\* processName;

void \*pageTable;

INT32 errorReturned;

INT32 status;

INT32 parent;

long context;

} PCB;

1. **Ready queue**. See the syntax for defining ready queue structure.

struct readyQueue {

PCB \*pcb;

struct readyQueue \*next;

};

1. **Timer queue**. Below is the syntax for defining the timer queue.

struct timerQueue{

PCB \*pcb;

long endTime;

struct timerQueue \*next;

};

1. **Disk queue**. Below is the syntax for defining the disk queue

struct diskQueue {

PCB \*pcb;

long diskID;

long sector;

int read\_write; //read: =0; write: =1

char\* address;

struct diskQueue \*next;

};

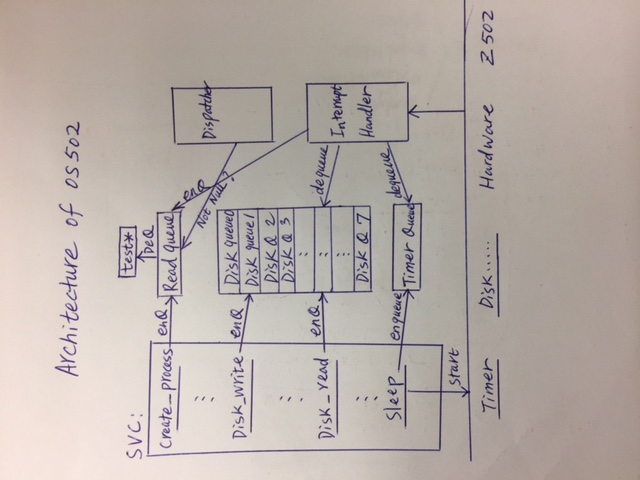
*Global variables defined and maintained in the os502.*

Those variables: **processCount**, \* **currentPCB**, **pcb\_list[\*PCB]** and **head**, **current**(tail) pointers for timer queue, ready queue and 8 disk queue defined and maintained in base.c. The processCount is used to keep tracking how many processes are alive in the operating system. Pointer of currentPCB is used to save the current running process PCB, since in this phase, operating system only has one running process, so one pointer for current running process. Pcb\_list[] is array of PCB pointers point to all alive processes separately. For each queue, a pointer to the head of queue and a pointer to the tail of the queue are used to track the queue.

*Source file and functionalities.*

1. Process.c
   1. CreateProcess():
   2. TerminateProcess():
   3. GetProcessID():
   4. StartProcess():
   5. terminateSelf():
   6. searchProcessName():
   7. addToPCBList():
   8. removePCB():
   9. dispatcher():
   10. readyEnQueue():
   11. readyDeQueue():
   12. readyQueueRemove():
   13. getNumofReadyProcess():
2. Disk.c
   1. diskEnQueue():
   2. diskDeQueue():
   3. getNumofDiskQueueProcess():
3. Timer.c
   1. timerEnQueue():
   2. timerDeQueue():
   3. startTimer():
   4. getNumofTimerQueueProcess():

*Architecture of os502*



*review and learned*

In the os502, Lock mechanism is adopted to the queue operations. The scheduler printers are added to the point of create process, sleep, disk write and disk read. The scheduler printer at the point of disk read/write, the print out result only shows the target disk which is read or written by the current process. For the tests from test1 to test 8, the results from the test have been analyzed and verified.

There are several road blockers during the implementation of the project one. When I implement the timer queue, I implemented four functions to do the enqueue operation, as createTimerQueue(), enQueueAtHead(), enQueueAtTail(), enQueueAtMed(). I exam these functions, the logic is right, but the timer queue performance is not steady. Sometimes works, but sometimes don’t. After I rewrite the enqueue function to one routine call. The timer queue works correctly all the time. The other problem is about terminate process. At beginning, I implemented as halt the process when the program run to terminate process point. I found out that there are still some tasks in the queues when I halt the process. Then I changed to call dispatcher finish the remaining task and halt the process. There are many more bugs found in the design and implement. By debugging, I have found most of them and fixed.