**Project1 report**

Jian Qiao

Important: I compile and test my program on Mac OS X v10.9, I hope you can compile and run my code on Mac OS since the lib I imported in my program probably will not work on your system.

**1. Architectural Document.**

**a) A list of WHAT is included in my design.**

**1.Function in base.c and Queue.c**

**In Queue.c**

1.Queue \*InitQueue();

This function used to initialize Queues and return this queue.

2.INT32 \*InitPCB2(SYSTEM\_CALL\_DATA \*SystemCallData,PCB pnode).

This function just used to initialize a PCB with SYSTEM\_CALL\_DATA, it returns PCB pid.

3.INT32 IsEmpty(Queue \*queue);

Use to check if the Queue empty or not.

4.PCB EnQueue(Queue \*queue,PCB pnode);

Push a PCB into the end of the target Queue and return this PCB.

5.PCB DeQueue(Queue \*queue);

Pop a PCB from the front of the target Queue and free the PCB. It returns the front of target queue.

6.PCB DeQueueWithoutFree(Queue \*queue);

Pop a PCB block from the front of the target Queue without free the PCB. It return the popped PCB.

7.void EnQueueWithPrior(Queue \*queue,PCB pnode);

Push a PCB block in to the target Queue and put it into a right position base on its priority.

8.void EnQueueWithwakeUpTime(Queue \*queue,PCB pnode);

Push a PCB block in to the target Queue and put it into a right position base on its wakeUptime.

9.void TerminateSelf(Queue \*queue,PCB pnode);

Pop the target PCB from target Queue and free the PCB. Normally, this function used to terminate the PCB that is running.

10.PCB DeleWithoutFree(Queue \*queue,PCB pnode);

Pop the target PCB from target Queue without free the PCB. It returns the popped PCB.

11.MSG EnQueueMsg(MsgQueue \*queue,MSG pnode);

Push a MSG node into MsgQueue and return the Msg front of target queue.

12.MsgQueue \*InitMsgQueue();

Used to initialize MsgQueue. It return the initialized queue.

13.MSG DeQueueMsg(MsgQueue \*queue);

Pop a MSG node from MsgQueue and return the Msg node.

**In base.c**

1.INT32 start\_timer(long \*SleepTime);

This function used for SLEEP() call, this function do two things:

a). Sleep the PCB that is currently running and push it into timerQueue.

b). Pop a PCB from the front of readyQueue and run this PCB by calling Z502SwitchContext().

2.long CreateProcess(PCB pnode);

This function used for CREATE\_PROCESS call , it push a initialized PCB to readyQueue. It returns an integer to indicate success or error.

3.INT32 GetIDByName(char\* name);

This function used in GET\_PROCESS\_ID call, it will get PCD block’s ID from Queues by its name and return it. If the name is null, just return the current running PCB’s ID.

4.int sendMessage(long sid,long tid,char\* msg,int msglength);

This function is used to push an Msg node into MsgQueue, and resume destination PCB if it is in suspendQueue. It returns an integer to indicate success or error.

5.int receiveMessage(long sid, char \*msg,int msglength,long \*actualLength, long \*actualSid);

This function is used for receive message and return a value to tell system the result. It return an integer to indicate success or error

6.void schedule\_printer();

It used for printing status of all Queues

7.int ChangePriorByID(long pid, int priority);

Used to change the priority of target PCB in CHANGE\_PRIORITY call. It return an integer to indicate success or error.

8.INT32 SuspendByID(long ID);

This function used in SUSPEND\_PROCESS It suspend target PCB base on its ID. It return an integer to indicate success or error.

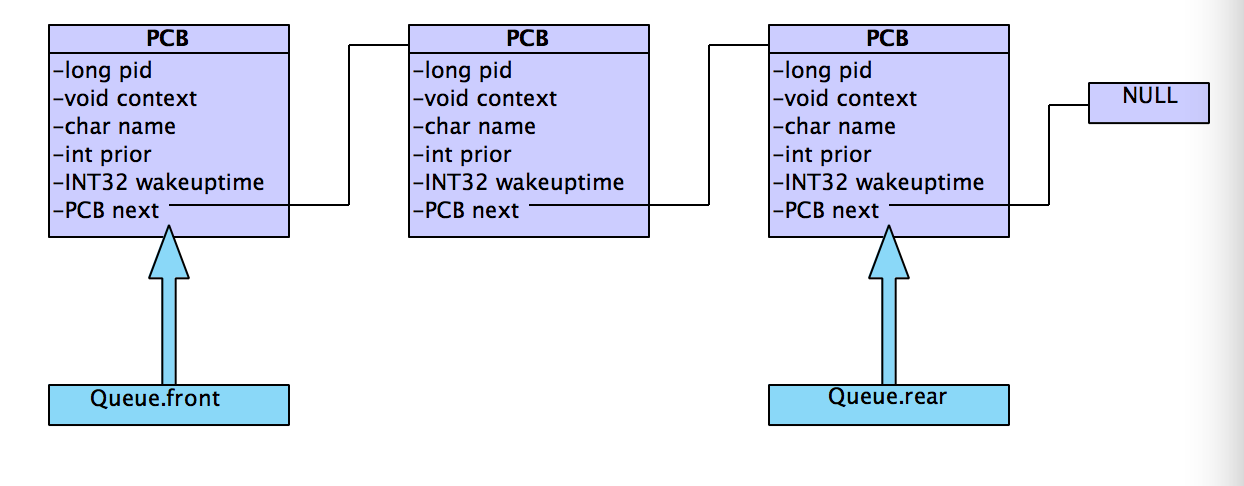
9.INT32 ResumeByID(long ID);

This function used in RESUME\_PROCESS call. It will resume a PCB base on its ID. It return an integer to indicate success or error.

**2.Structure**

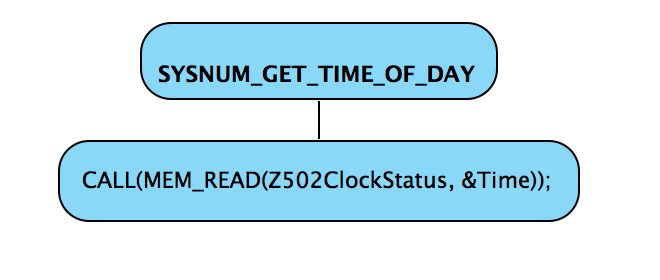
**Structure idea**

I used linkedlist to implement Queues that used to store PCBs or messages. There are four Queues in my system timerQueue, readQueue suspendQueue and MsgQueue(same idea but different Node structure).

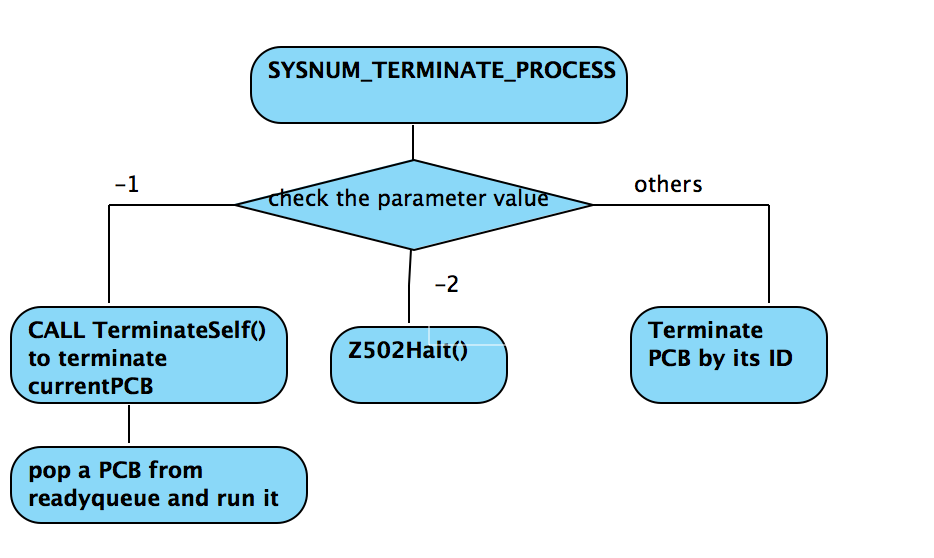


**b) High Level Design**

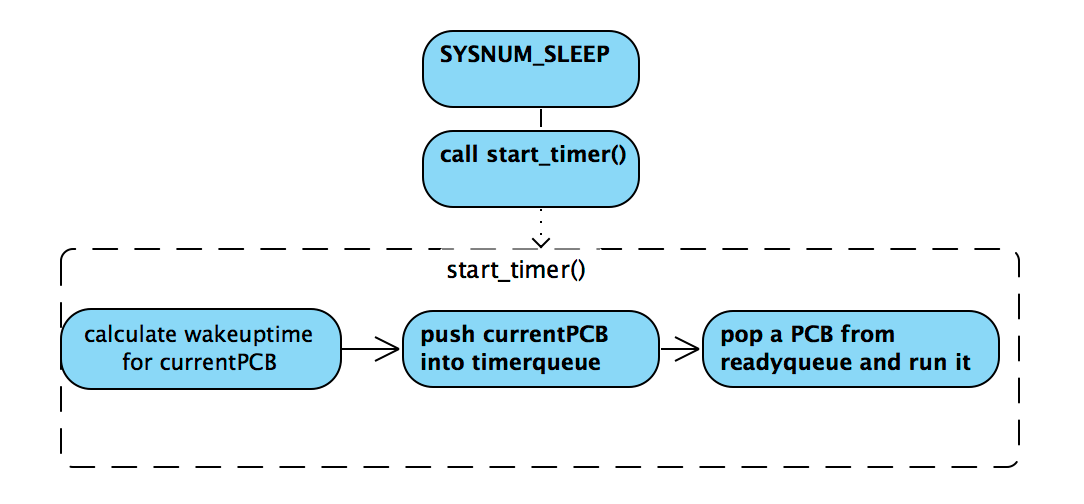
SYSNUM\_GET\_TIME\_OF\_DAY



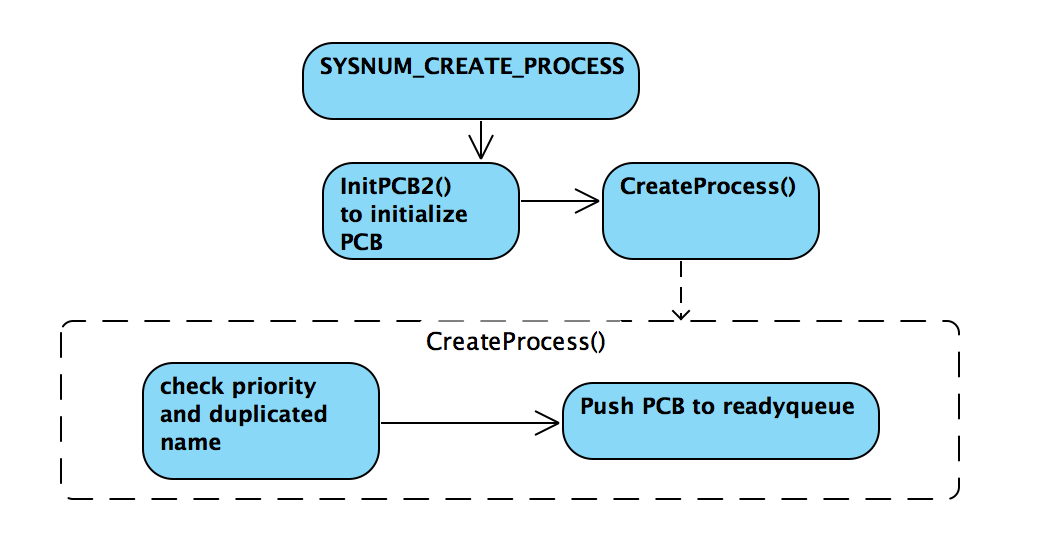
SYSNUM\_TERMINATE\_PROCESS



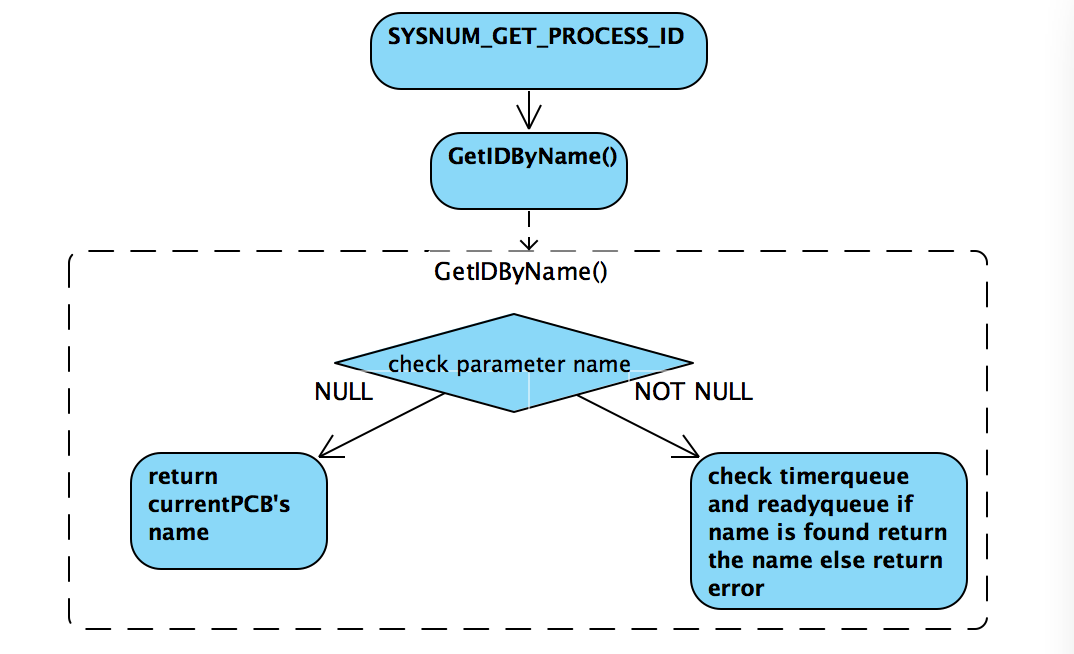
SYSNUM\_SLEEP



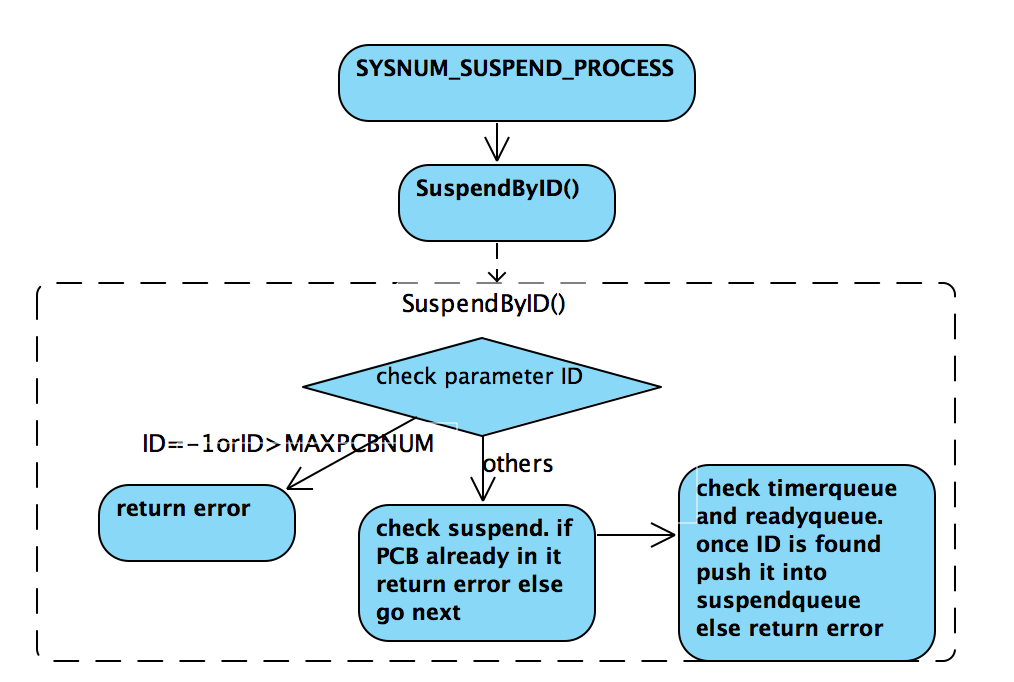
SYSNUM\_CREATE\_PROCESS



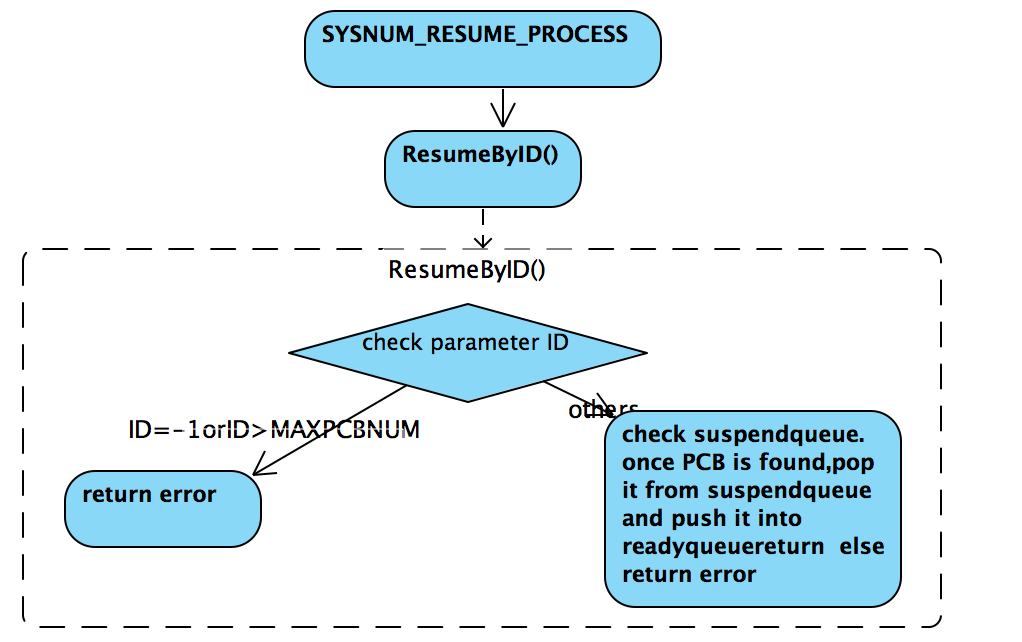
SYSNUM\_GET\_PROCESS\_ID



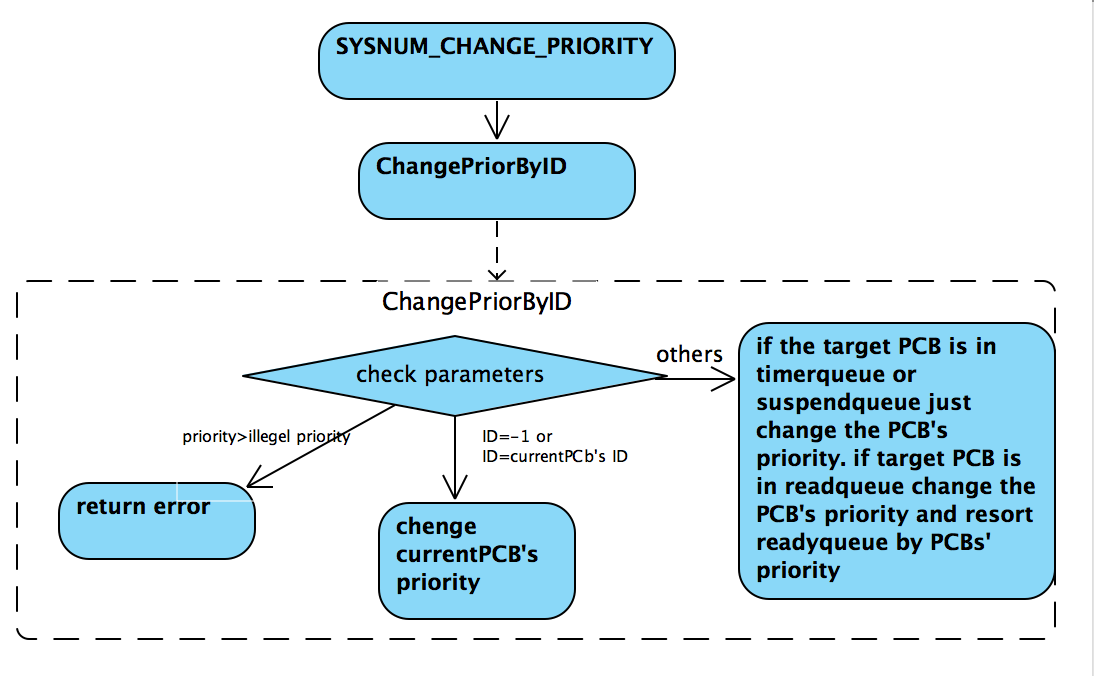
SYSNUM\_SUSPEND\_PROCESS



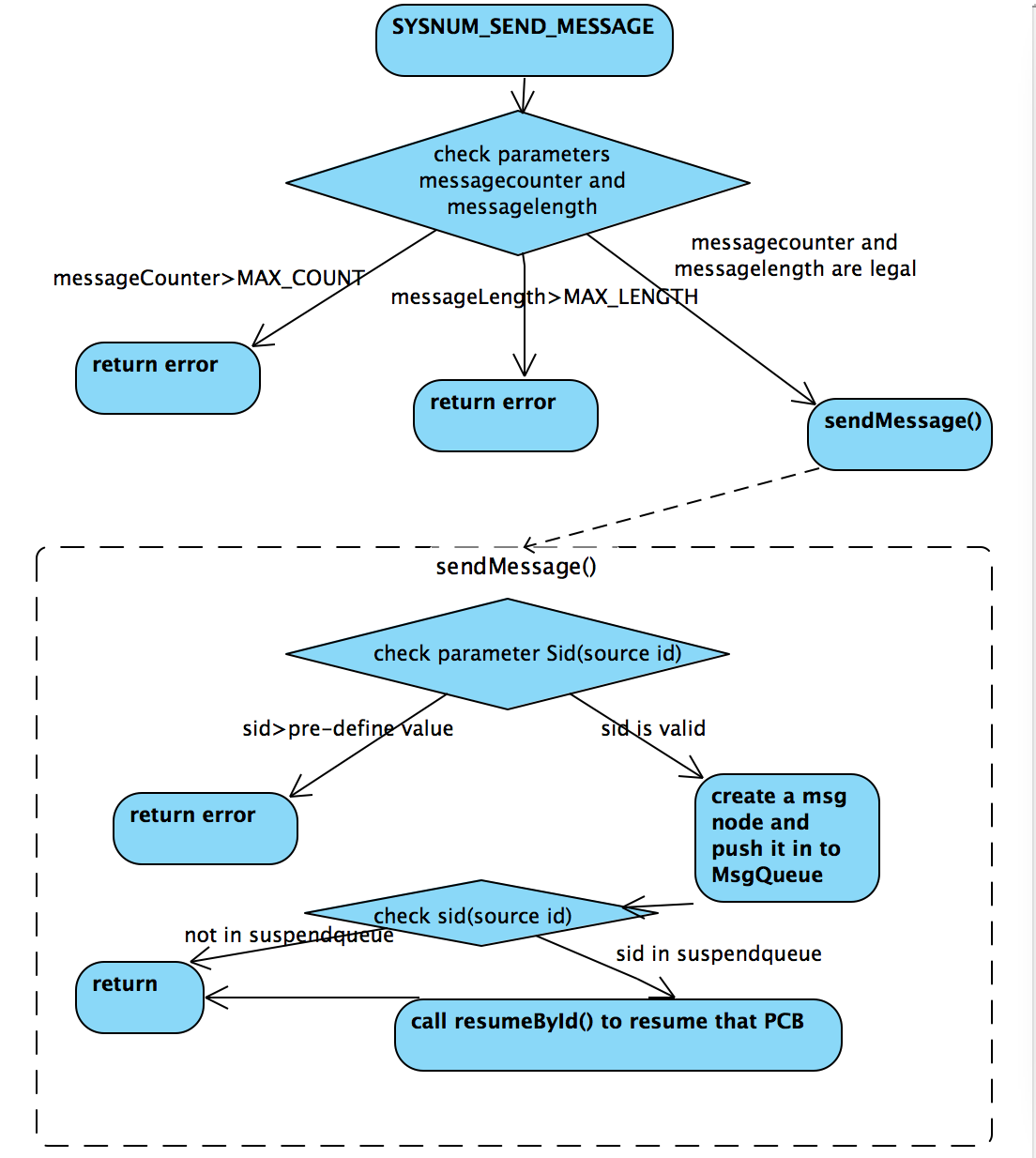
SYSNUM\_RESUME\_PROCESS



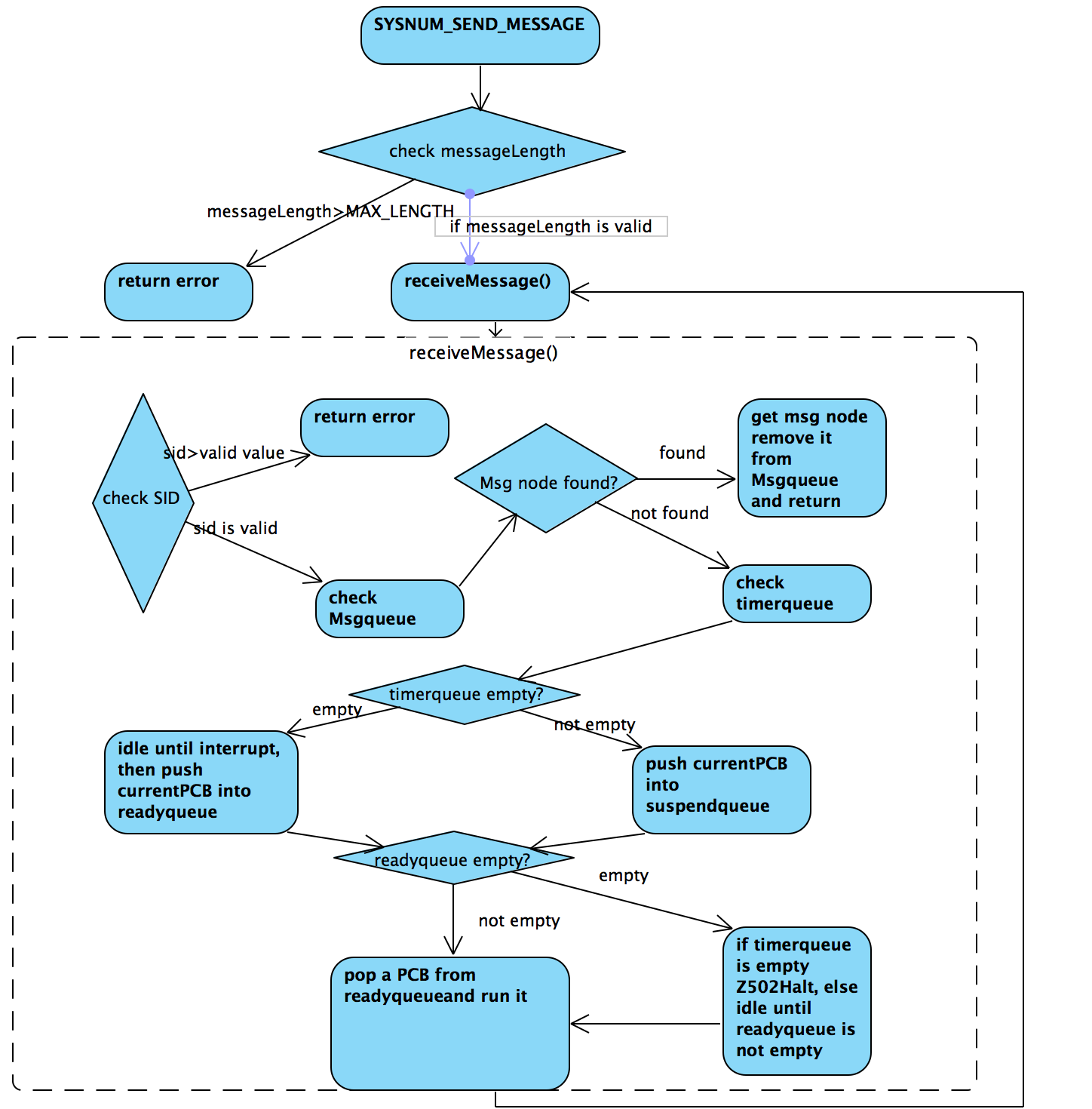
SYSNUM\_CHANGE\_PRIORITY



SYSNUM\_SEND\_MESSAGE



SYSNUM\_RECEIVE\_MESSAGE



**c) Justification of High Level Design.**

Here, I am going to introduce designs that are not clear and cannot demonstrate how it works in high-level design but it is crucial to run test successfully on my porgram.

1). I used three Queues timerQueue, readyQueue and suspendQueue to implement the simulation. readyQueue is used to store PCBs that the system is going to run. timerQueue is used to store PCBs that are sleep by system. timerQueue arranges in increasing order by WakeUptime. SuspendQueue is used to store PCBs that are suspended by system. Also there is a variable, called currentPCB, which represents the PCB that is running in the system. When Sleep() is called by system the currentPCB will be stored in timerQueue. System will pop a PCB from readyQueue and set the PCB as currentPCB.

2). Each test involves multiple system calls. To run each test without errors, the three queues are needed. Why I use three queues: 1 System only extracts PCB from readyQueue to run. 2 The sleep() call will push the running PCB into timerQueue, the PCB will stay in timerQueue until it wake up and interrupt by interrupt\_handler(). During that time, the PCB cannot be run by system. The same idea with above, The suspend() call will push the target PCB into suspendQueue, it will stay in suspendQueue until the PCB is resume by system, it also cannot be run at that time.

3). The way I implement interrupt\_handler() to make it works. First this function will be called when system reach the timer’s time we set up before (phase1). Once the function is called it will check timerQueue. If the timerQueue is not empty it will get current system time and compare this with the Wakeuptime of the front PCB in timerQueue. If the system time greater than or equal to the Wakeuptime, it indicates this PCB is no longer sleep, then system pop it from timerQueue and push it into readyQueue. System repeats the comparison until timerQueue is empty or Wakeuptime of the front PCB in timerQueue is greater than system time. interrupt\_handler() will end if timerQueue is empty. Otherwise, there is one more step, we need to set up a new timer for the further interrupting.

4). The way I implement fault\_handler() to make it works. We just need to halt the system when it is called with **Fault\_handler: Found vector type 4 with value 0** for phase1.

**d) Additional Features**

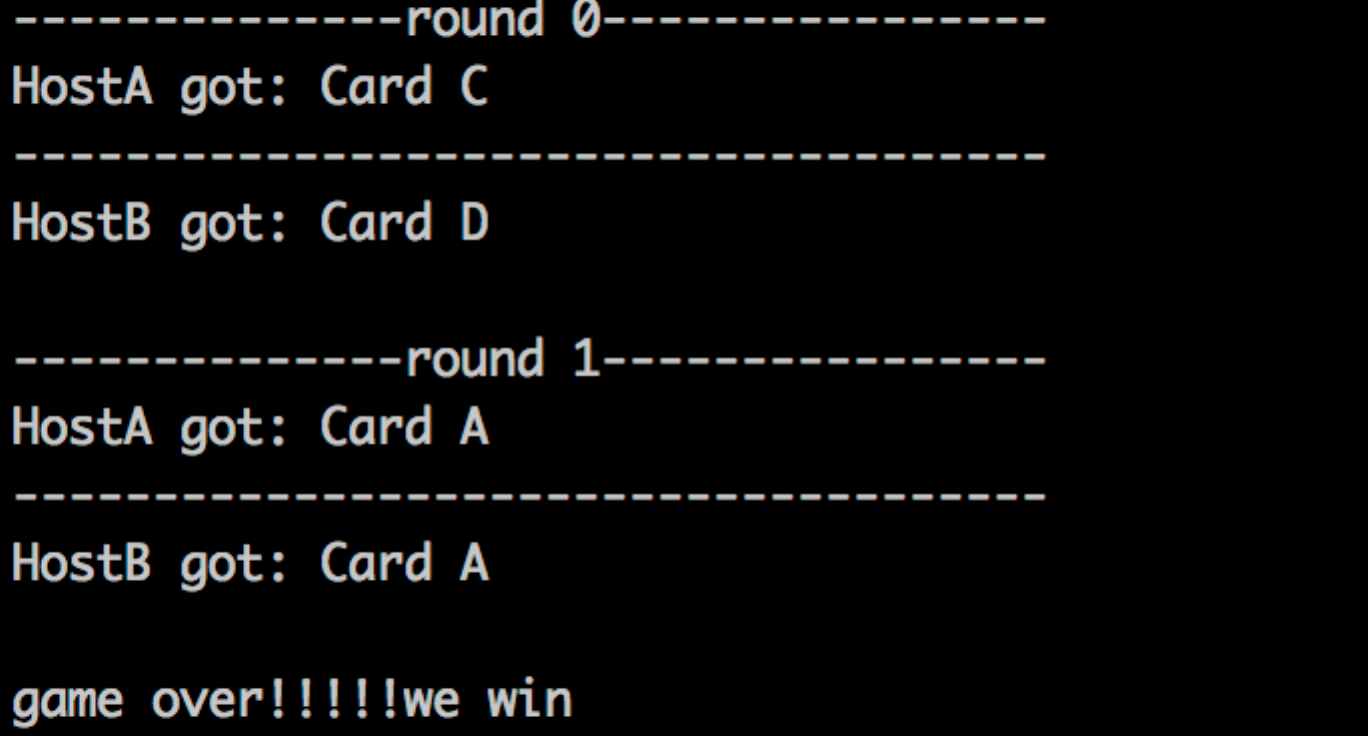
1). There a bug in my system, the interrupt\_handler() is always called before the schedule\_printer() finished. Thus the output of schedule\_printer() was chaos and disorder. I tried to fix it with **usleep()** function that is in unistd.h lib. I used usleep() function to sleep interrupt\_handler() for 20usec. The consequence is that the schedule\_printer() function can print out the schedule as I expected without interrupt most of the times.

2). In test1m, I used send and receive message to simulate a game. There are two processors in my in this test. HostA and HostB. And both HostA and HostB have a card box.

How it works:

First HostA draws a card from his box and sends a message to HostB to tell HostB what he got, then HostB receive this message and draws a card from his box next HostB response a message to HostA to tell HostA what he got. After HostA having this message he compares these two cards. If they got the same card they win this game otherwise repeat this process until 3 times. They will lose the game if they get no match 3 times in a row.

**Test1m result examples:**

**e) What anomalies and bugs did you find?**

There was always a error occur when I called CALL(Z502Idle())in a loop (but it works sometimes). It says **ERROR in Z502Idle. IDLE will wait forever since there is no event that will cause an interrupt.** I tried the way the system mentioned to fix it, but it does not work until I add schedule\_printer() function after CALL(Z502Idle()). I have no idea why it happened.

1. **Source Code**

Test.c(test1m)

base.c

Queue.h

Queue.c

1. **Test Format**

Use the SP\_print\_line() as the output format.

Example:

Time Target Action Run New Done State Populations

1667 0 Finish 0 READY : 2 3 1

1. **Test Results**

a) Test program 1a runs and gives expected output.

b) Test program 1b runs and gives expected output.

c) Test program 1c runs and gives expected output.

d) Test program 1d runs and gives expected output.

e) Test program 1e runs and gives expected output.

f) Test program 1f runs and gives expected output.

g) Test program 1g runs and gives expected output.

h) Test program 1h runs and gives expected output.

i) Test program 1i runs and gives expected output.

j) Test program 1j runs and gives expected output.

k) Test program 1k runs and gives expected output.

m) Test program 1m runs and gives expected output.(my design)

l) You said we should implement my test in test1l but there are some code in it and I found test1m is empty so I implement my test in test1m. I did not touch test1l.