**Operating System ( CS502 )**

**Project Report**

**Shuoqing Ding**

**2013/10/15**

1. **Project Contents**
   1. **System Calls**

* **SYSNUM\_GET\_TIME\_OF\_DAY**

Get Current time(number of time-units).

* **SYSNUM\_SLEEP**

Make process sleep a given time.

* **SYSNUM\_REVOKE(New)**

Revoke a given process that is sleeping (In timer queue).

* **SYSNUM\_CREATE\_PROCESS**

Create and initialize a process. Make it ready to run.

* **SYSNUM\_TERMINATE\_PROCESS**

Terminate a given process, remove it from system and free the memory. If no process left after terminating call Z502Halt.

* **SYSNUM\_GET\_PROCESS\_ID**

Get the process ID of the process whose name is given.

* **SYSNUM\_SUSPEND\_PROCESS**

Suspend the process by given Process ID.

* **SYSNUM\_RESUME\_PROCESS**

Resume the suspended process by given Process ID.

* **SYSNUM\_CHANGE\_PRIORITY**

Change the priority of certain process to a given value. Reorder ready queue if this process in ready queue.

* **SYSNUM\_SEND\_MESSAGE**

Send a message to a certain process or any process.

* **SYSNUM\_RECEIVE\_MESSAGE**

Receive a message from certain process or any process.

* 1. **Queue**
* **Timer Queue**

Contain sleeping processes which are waiting for interrupt. Processes are ordered by time\_of\_delay.

* **Ready Queue**

Contain processes which are ready to run. Processes are ordered by priority.

* **Suspend Queue**

Contain suspended processes.

* 1. **LinkList( No Order )**
* **PCBList**

Contain all PCBs in the system no matter what state it is, used for searching certain process.

* **MsgList**

Contain all messages in the system.

1. **Design and Justification**
   1. **Structure**

struct PCB{

int pid;

int time\_of\_delay; //Time to wake up

char name[64];

int state; //Running, Ready, Waiting

int pmode; //UserMode or KernelMode

int priority;

void \*context;

struct PCB \*next\_pcb;

};

PCB structure (as show above) designed to store all attributes of process, such as process name, process ID, priority and state (Including Running, Ready and Waiting). And that means each PCB is a process.To fit PQueue structure (as show below), we have a pointer which point to next PCB named next\_pcb.

typedef struct{

long length;

char name[32];

PCB \*head; //Point to the first PCB

} PQueue;

PQueue Structure (as show above) designed to store PCB. That is why I name it PQueue. It is a Linked List-Like structure, and the only difference is that each node of it is a PCB structure instead of an “LNode”.

There are three queues in the system: Timer Queue, Ready Queue and Suspend Queue, and they are quite similar. All of them used to store PCB, but ordered by different attribute. In order to avoid duplicate codes I design this Structure in a compatible way so that I can use it for all queues. The compatible part is mainly about the routine **AddtoQueue**(Talk it later), because order action happen only when adding a process into the queue.

typedef struct LNode{

void \*data;

struct LNode \*next;

}LNode, \*LinkList;

Sometime we need to find a certain process by pid or process name. But all processes store in different queues. That means we need to search all queues to find a certain process. Due to the pointer of PCB can only be used in one Queue we cannot use a PQueue to store all process. Consequently, I design this structure to store all the processes in the system. LinkList is a standard Linked List data structure, which have its own next pointer, so we can put all PCBs on it. Since it not only use for PCB, we use a void type for data. Actually I believe it can replace PQueue, maybe I will do it during B term.

typedef struct Message{

int id;

char data[256];

int length;

int sender\_pid;

int receiver\_pid;

}Message;

Another place to use LinkList is Message List. Message Structure contains all attributes of a message. All Messages store in the MsgList. Each Time a process send a message to someone. The message would be inserted into MsgList. After the target process receives that message, it will be removed.

* 1. **Routine**

All the routines wrote by me are showed below. I don’t introduce all of them but only some pivotal routines because the functions of most of them are just like their names.

void ResetTimer(); //Use the smallest delay time in timer queue to reset the timer (event queue).

void ReadytoGo(); //Move a process from timer queue into ready queue, and reset timer

void StartTimer( int delay ); //Put the sleep process into timer queue, reset timer and call Dispatcher.

void DispatchProcess(); //Let the first process of ready queue to run. If both ready queue and timer queue are empty then call Z502Idle(). If only ready queue is empty then wait until some process added into ready queue.

void DestoryRunningandDispath();

void SchedulerPrinter( char\* action, int target );

void SuspendProcess( PCB \*p, long \*Err );

void ResumeProcess( PCB \*p );

void ListTest();

void \*GetTestAddr( char \*test\_name );

int GetTestPrintLevel( char \*test\_name );

**PCB Part**

void OSCreateProcess( PCB \*\*p, void \*starting\_address, char \*name,

int priority, BOOL user\_or\_kernel){

//Create and initiate a process, and add it to ready queue

void DestoryProcess( PCB \*p );

**PQueue Part**

void InitQueue( PQueue \*\*q, char \*queue\_name );

void AddtoQueue( PQueue \*q, PCB \*p, int order\_type ); //Add PCB into the certain queue by given order and change the state of process. If queue is timer queue then order by delay\_of\_time. If queue is ready ready queue then order by priority( smaller is greater ).

BOOL IsEmptyQueue( PQueue \*q );

BOOL HasPCB( PQueue \*q, PCB \*p );

void GetFirstPCB( PCB \*\*p, PQueue \*q ); //Get the first PCB in given queue. First PCB may means the smallest time\_of\_delay or the smallest(greatest) priority.

void PrintQueue( PQueue \*q );

int RemoveFromQueue( PQueue \*q, PCB \*p );

**LinkList Part**

**L**inkList CreateNullList();

int InsertIntoList( LinkList L, void \*data );

LinkList SearchByPid( LinkList L, int pid );

LinkList SearchByPname( LinkList L, char \*pname );

BOOL DeleteByPid( LinkList L, int pid );

BOOL DeleteByPname( LinkList L, char \*pname );

int GetLength( LinkList L );

**Message Part**

Message \*GetMsg( int sender\_id, int receiver\_id ); //Get Message by given sender’s pid and given receiver’s pid. If sender pid is -1, then return the first message sent to receiver. If receiver’s pid = -1, then return the first message sent by sender.

BOOL DeleteMsg( int msg\_id );

Message \*CreateMsg( char \*msg, int len, int sender, int receiver );

* 1. **Procedure**











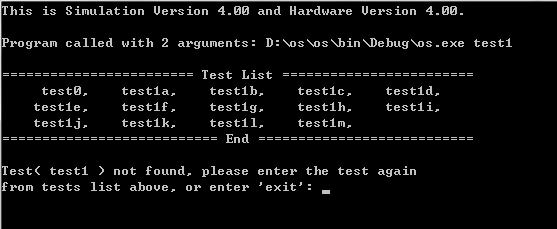






1. **Additional Features**
   1. SYSNUM\_REVOKE

The new system call can revoke a sleeping process. I think it is a very useful system call. When CPU is idle and no process in ready queue, we can revoke a sleeping process instead of wait it wakes up to make our system more efficient.

3.2 A very friendly user interface to help user choose the test after they run the program with wrong argument. The screenshot shows below.

3.3 I design a routine to load the printer level according to the test name. The system will print everything by this configuration. The printer level defined on the top of *base.c* file. Its’ rule is:

**Level Interrupt/svc/fault Scheduler Memory**

0 0(Limited) 0(No) 0(No)

1 0 0 1(Yes)

2 0 1(Yes) 0

3 0 1 1

4 1(Full) 0 0

5 1 0 1

6 1 1 0

7 1 1 1

1. **Bugs**

There is a thread safe problem in state printer. Because the buffer of printer shared by all thread with no Lock. Sometime when two thread call printer in the same time, the output will be overlap strangely. There are two solutions for this problem. The simple one which I used is using Lock when call printer. Another one is creating a new buffer for printer each time when printer is called.

1. **Code List**

* **linklist.h/linklist.c**

Define LNode structure and some related global variables.

LNode structure is a standard LinkList, I use it to store all PCBs (PList) and Messages.

* **process.h/process.c**

Define PCB structure and some related global variables.

A PCB is a process. It contains the necessary attributes of process.

* **queues.h/queues.c**

Define PQueue structure and some related global variables.

PQueue structure is only used for PCB, all of Queues (timerQ, readyQ, suspendQ) use the same structure.

* **message.h/message.c**

Define Message structure and some related global variables.

Message structure is used for communication between processes, and store in LinkList Structure.