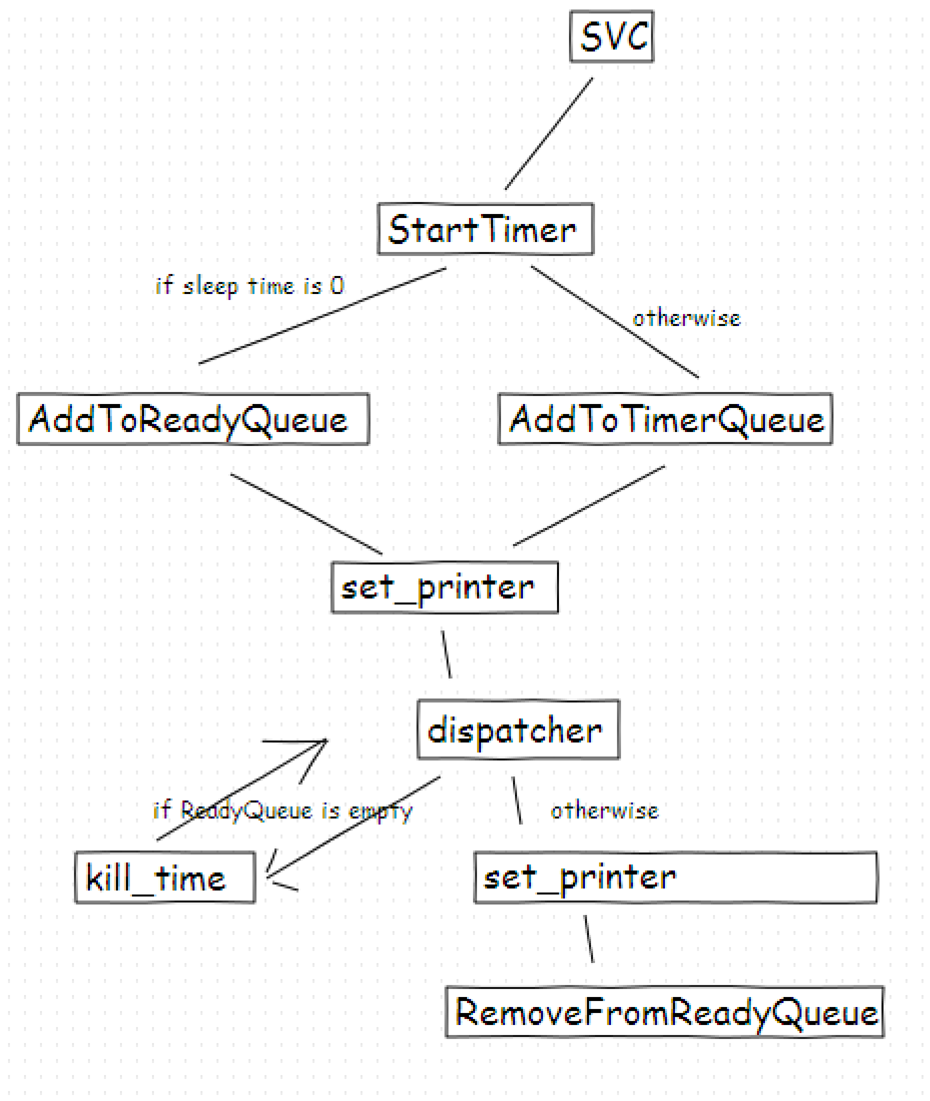
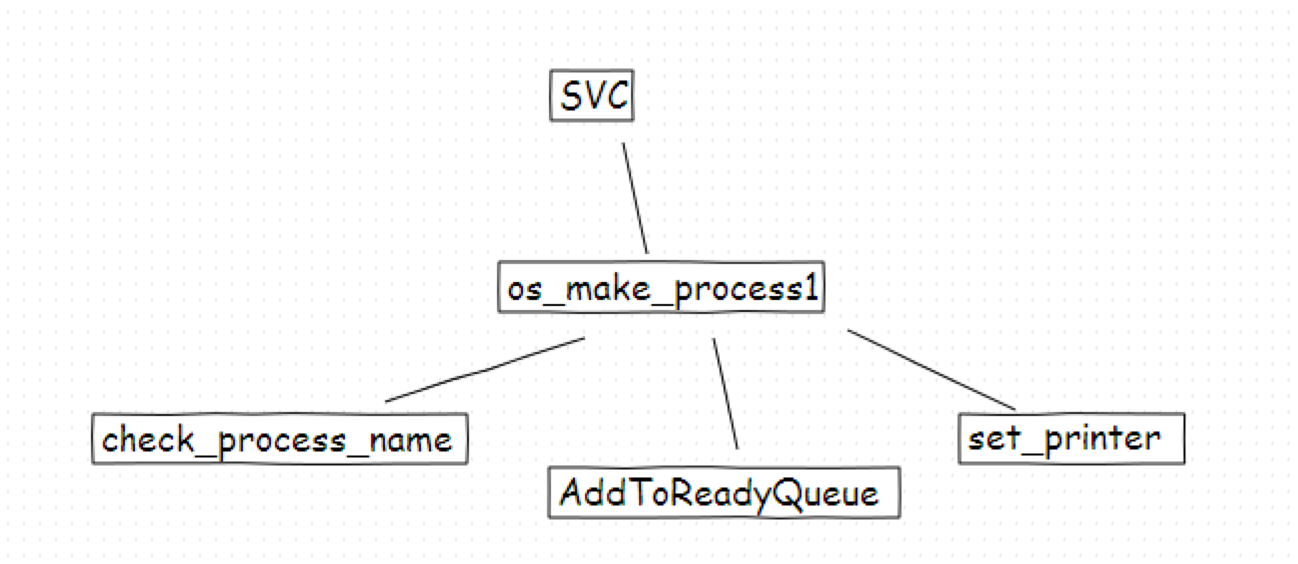
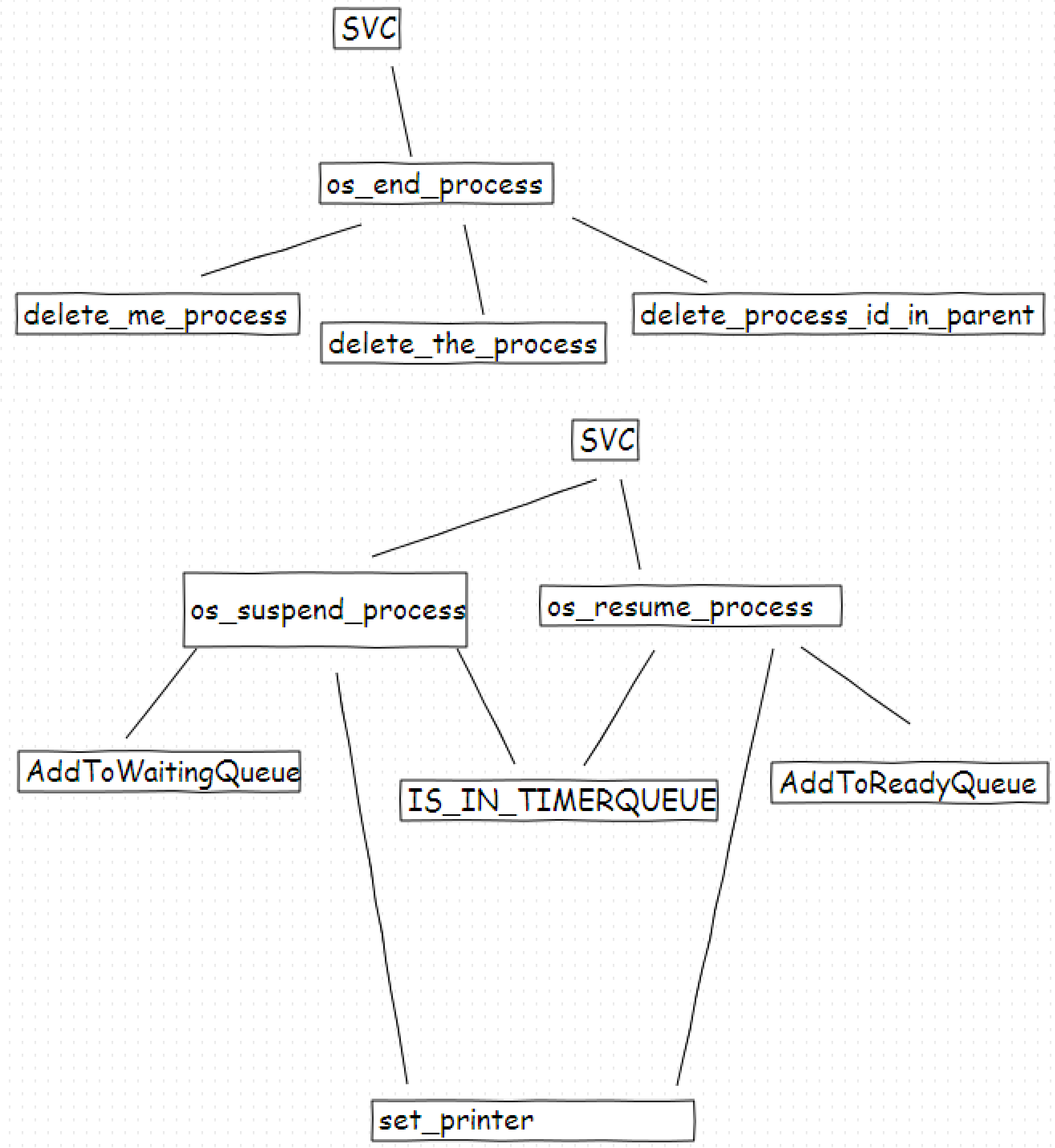
a) architectural and policy overview
For different system calls from user model, I design different routine in kernel model. And other routines are designed for dealing with frequently used functions, such as adding a process to the ReadyQueue, extracting a process from TimerQueue and so on. For the high level design please refer to session c), the detailed design please directly refer to the code. Since very detailed explanation exists before and in each routine.

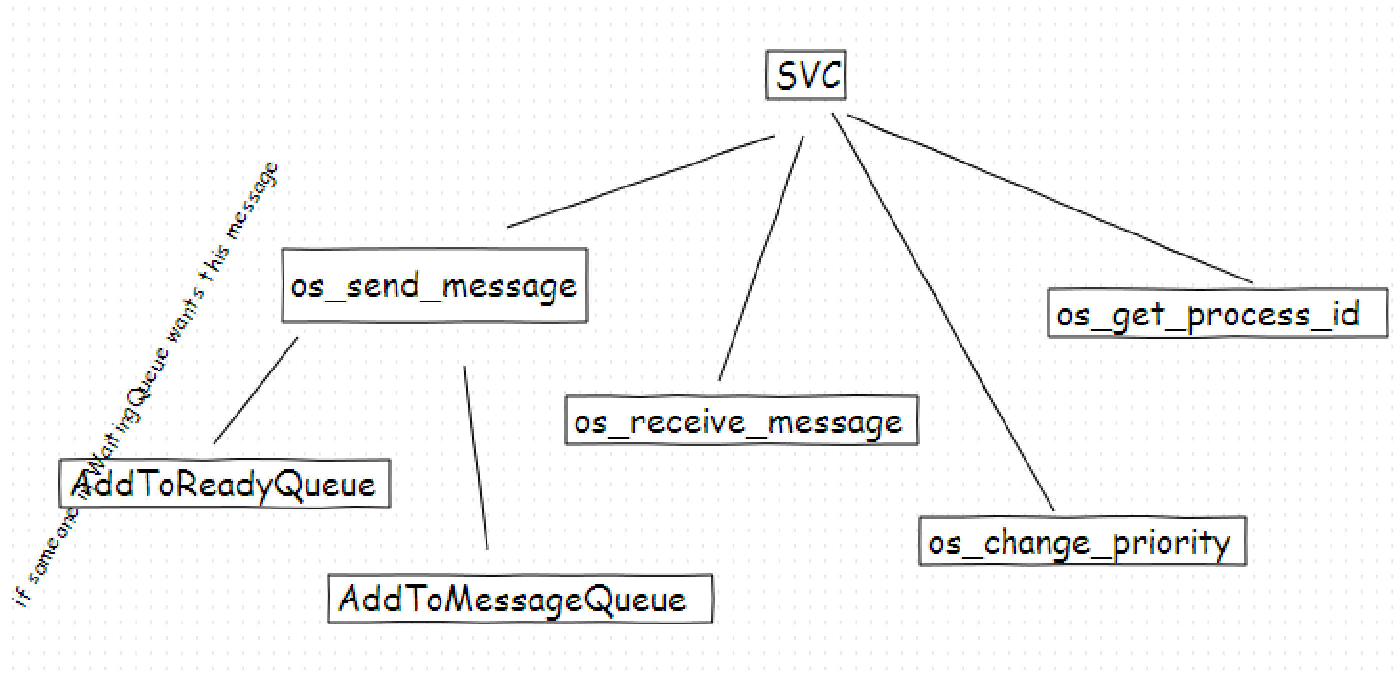
b) Elements of my design
To implement all the system calls from user in the kernel model, I design my program to including these funtion:
1) Interacting with hardware:
Getting the time from the system, Setting the timer, Fault handling
2)Operating on processes:
Process creation, Process termination, Getting process id, Changing priority of a process, Sending and Receiving messages between processes
3) Process scheduling:
Priority scheduling in ReadyQueue, Sleeping a process for certain time, Suspending a process, Resuming a process, Interrupt handling
4) Process Synchronization
Locking shared Queues, deadlock prevention,
Typical errors and illegal system call can be handled.

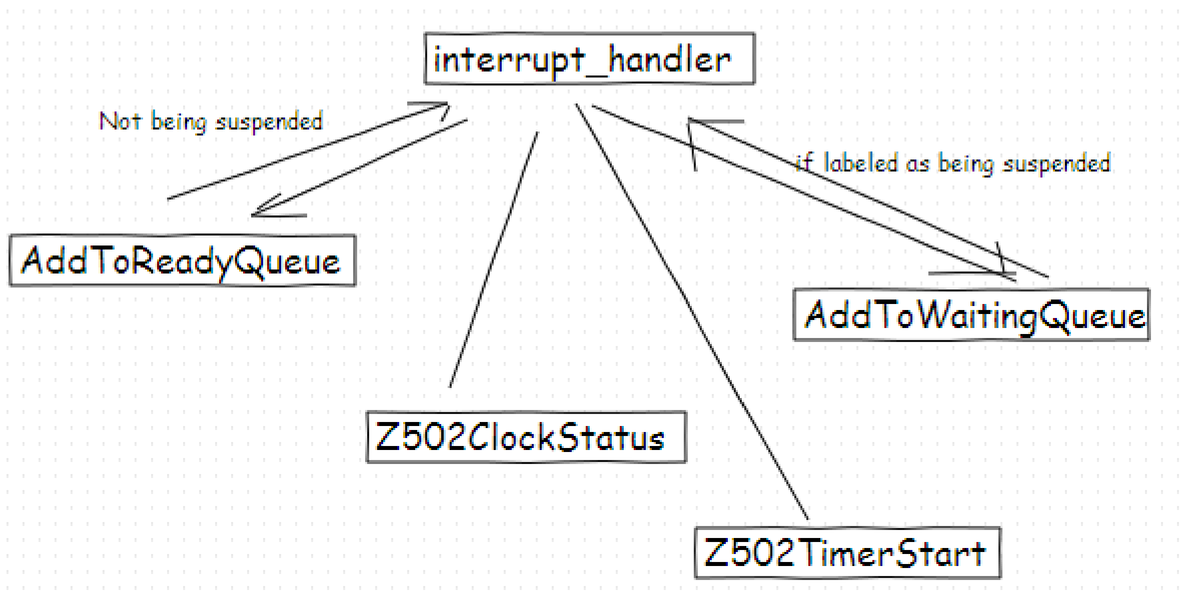
c) High Level Design:
(As I cannot express my high level design in a single small picture, I spit the picture into small pieces and make them as clear as I can. The other part of the base.c is identical to the diagram of in StartHere.ppt hand out by JB)


SVC
StartTimer
if sleep time is 0
otherwise
AddToReadyQueue
AddToTimerQueue
set_printer
dispatcher
if ReadyQueue is empty
otherwise
kill_time
set_printer
RemoveFromReadyQueue


SVC
os_make_process1
check_process_name
AddToReadyQueue
set_printer


SVC
os_end_process
delete_me_process
delete_the_process
delete_process_id_in_parent
SVC
os_suspend_process
os_resume_process
AddToWaitingQueue
IS_IN_TIMERQUEUE
AddToReadyQueue
set_printer





d) Justification of High Level Design.
In the user model, multiple system calls are used. Hence in SVC of kernel model, I define multiple routines to deal with different system calls.

So, when system calls for sleeping certain time, StartTimer routine is invoked. It check whether the time is 0, if so directly add the current process to ReadyQueue, that's the function of AddToReadyQueue. Otherwise, it will add the current process to TimerQueue, that's the function of AddToTimerQueue. And then it deals with print format, using the routine set_printer, then print to let user know what's happening in the system. Then dispatcher is invoked. It checks whether ReadyQueue is empty, if not find the first in ReadyQueue to run. Otherwise, it calls a kill_time routine to spend time and waits for some process to be ready and then extract the first process from the ReadyQueue, which is the function of another routine----RemoveFromReadyQueue, and then runs it.

The others are like we discussed above. And detailed discussion is in the code please refer to that part.

The way I design my project this way lies in that lot of these routines such as AddToReadyQueue, set_printer are invoked by different system calls from user model. It is efficient and concise to write such a function unit into a certain routine and deals with certain problems.

e) Additional Features.

Tries to implement this: When each process is created, add this pid of this new process to its parent. When considering terminating a certain process, we check whether that process has children. If so, we need to wait until all the children are terminated. When using parameter -2, it means terminating the process and all the children of the certain process. This rule also applies to the children of the children of the process and so on.

Accomplishment: add_child, check_children_process, delete_process_id_in_parent routines are completed. But delete_family_process has not been finished since time is limited. Details please refer to base.c

Also PrintReadyQueue, PrintTimerQueue and PrintWaitingQueue are used for testing. And could help decide what's going on in the system.

Jiefeng He
10/18/2012