A list of WHAT is included

`Queue *InitQueue()`

initiate a new queue, return the queue

`int IsEmpty(Queue *pqueue)`

check if the queue is empty. If the queue is empty, return 1, if it's not empty, return 0.

`PNode EnQueue(Queue *pqueue,PCB pitem)`

insert a new item to the rear of the queue

`PNode DeQueue(Queue *pqueue,PCB *pitem)`

`delete the first item of the queue`

`PNode RemoveAdvanced(Queue *pqueue,INT32 pid){`

remove the PCB with pid from queue, return the removed PCB

`INT32 Remove(Queue *pqueue,INT32 pid){`

remove the PCB with process_id pid from queue.
If the PCB with process_id pid is removed successfully, return 0.
If couldn’t find PCB with process_id pid, then return -1.

`PNode EnQueuePriority(Queue *pqueue,PCB pitem){`

Enqueue PCB to the queue in priority order.

`PNode EnQueueTimer(Queue *pqueue,PCB pitem){`

Enqueue PCB to timer_queue in duetime order, i.e. if the new PCB’s execution timer is earlier than another PCB, it appears earlier than the other PCB.

void OSCreateProcess(void *starting_address)

Create main process. i.e. test1a, test1b, test1c….

INT32 create_process(char process_name[], void *starting_address, INT32 initial_priority)

Create children process. This is called by svc.
if the new process is legal, enqueue it to ready_queue, enqueue it to PCB_queue.
Set currentPCB to the new PCB.
Return it's pid.

void interrupt_handler(void)

Removes the first PCB of timer_queue. Enqueue it to ready_queue in priority order.
If timer_queue is not NULL, start another timer.

void dispatch ()

Run the first process on ready_queue.

SVC_HANDLER

SYSNUM_GET_TIME_OF_DAY

Get the system time(absolute time)

SYSNUM_SLEEP

A process sleep a certain amount of time. (see below)

SYSNUM_GET_PROCESS_ID

Get process ID.
If process name is empty, get the process ID of currently running process;
If process name is not in PCB_queue (a queue stores all processes), return ERR_BAD_PARAM;
else, return the ID of this process and ERR_SUCCESS.

SYSNUM_CREATE_PROCESS

Create a process and check if its legal:
If the process trying to create has illegal priority, return ERR_BAD_PARAM;
If the process trying to create has exceed the MAX_PROCESS_NUMBER, return ERR_BAD_PARAM;
If the process trying to create has the same name as an already existing process, return ERR_BAD_PARAM;
Else, put new PCB to PCB_queue and ready_queue.

SYSNUM_TERMINATE_PROCESS

If SystemCallData->Argument[0] is -2, exit the whole program;
If SystemCallData->Argument[0] is -1, terminate the process itself;
Else, terminate the process with process ID SystemCallData->Argument[0] if the process exist, remove it from PCB_queue, remove it from ready_queue, if not exist , return ERR_BAD_PARAM.

SYSNUM_SUSPEND_PROCESS

If SystemCallData->Argument[0] is -1, return ERR_BAD_PARAM;
If there is PCB with process ID SystemCallData->Argument[0] in timer_queue, return ERR_BAD_PARAM;

If there is PCB with process ID SystemCallData->Argument[0] in suspend_queue, return ERR_BAD_PARAM
If there is PCB with process ID SystemCallData->Argument[0] in ready_queue, it's a legal suspend, return ERR_BAD_PARAM.

SYSNUM_RESUME_PROCESS

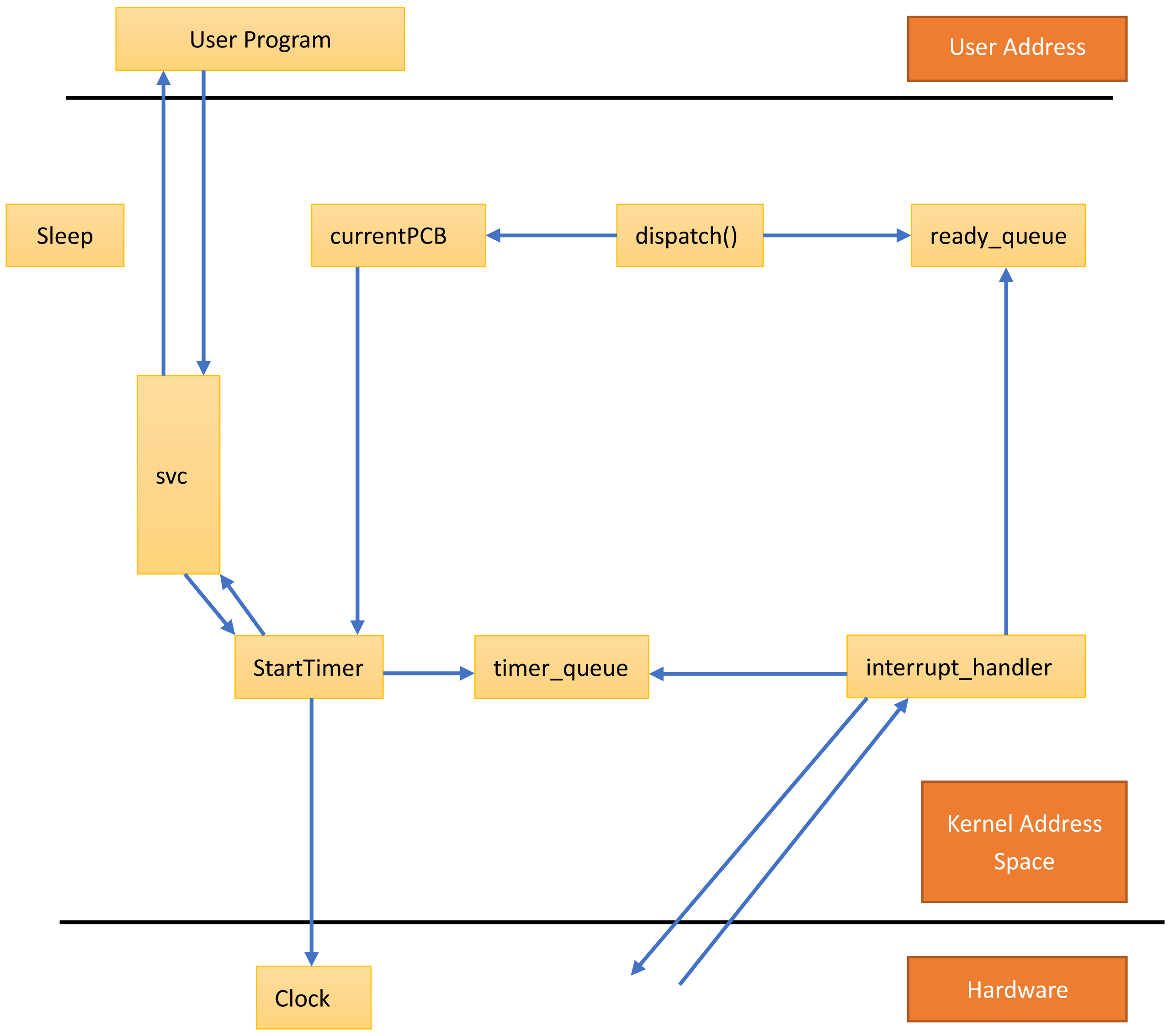
If the process with process ID SystemCallData->Argument[0] is in suspend_queue, then remove it from suspend_queue, add it to ready_queue. Then return success.
Else, return ERR_BAD_PARAM.

SYSNUM_CHANGE_PRIORITY

If the pid is not in PCB_queue, return ERR_BAD_PARAM;
If initial priority of PCB is illegal, return ERR_BAD_PARAM;
Else, remove this PCB from PCB_queue and ready_queue, change its priority, and enqueue it into PCB_queue and ready_queue. Then, return success.

High Level Design

Here is an overview of how test1c flows:



Justification of High Level Design

Here is justification of test1c:

1. Create 5 processes.
2. Now sleep see if one of the five processes has terminated.
3. While error code is ERR_SUCCESS. Get into the loop, sleep 1000. Run all the processes that've been created.
4. In test1x, we assign the process a random sleep time, SYSNUM_SLEEP begins.
5. Add this process to timer_queue in duetimer order(when the process should start running, absolute time), remove this process from ready_queue.
6. Start a timer to counted down(the time this process should execute – system time).
7. When the timer reaches an end, it automatically call interrupt_handler which remove the first process off timer_queue, add it to ready_queue in priority order.
8. call dispatch(), it calls SWITCH_CONTEXT to run the first process on ready_queue.
9. Do a GET_PROCESS_ID to get the error code. When error code is not ERR_SUCCESS, end the loop.
10. Terminate the whole simulation.

test running results:

test1a runs successfully.
test1b runs successfully.
test1c fails. (I think it’s close, but it still doesn’t work.)
test1d fails.
test1e runs successfully.
test1f fails.
test1g runs successfully.
test1h fails.
test1i is unfinished.
test1j is unfinished.
test1k runs successfully.
test1l is unfinished.