Lab 2 report - Group 53 - Venkata Sai Dinesh Uddagiri, Souptik Paul

Data Structures
We have added new variable “int64_t Time_To_Wait;” to structure thread defined in thread.h . The purpose of Time_To_Wait is to make the currently running thread, wait for a specific amount of time.

Algorithms
Previous implementation of sleep uses busy wait. Therefore, our objective is to get rid of busy waiting as it waste CPU cycles, which is inefficient.

Implementation in timer_sleep() function defined in ‘devices/timer.c’:
As a preliminary step, we had used the "ASSERT" function to confirm whether interrupts were ON. The program executes the subsequent instructions when the conditional expression in an assert statement is true.

Then we are disabling the interrupt and storing the previous state in the variable "enum intr level precedingState = intr disable();" for later reactivating.

We have retrieved the active thread and set its time waiting to the specific number of ticks and then we are blocking the active thread using "thread block()".

Using "intr set level," we reset the interrupt status back to the one that existed before we deactivated the interrupt.

Implementation in timer_interrupt() function defined in ‘devices/timer.c’:
The timer_interrupt() is triggered and the interrupt handler is run for each tick of the clock.

We have made full use of the timer_interrupt() function and iterated over all the threads using thread_foreach().

Internally, thread_foreach() invokes the check threads() function, which determines if a certain thread is currently blocked.

If the state is blocked, we check its remaining waiting time and accordingly decrement the waiting time, with every timer tick.

If the waiting the time reaches zero then, we use "thread_unblock(currentThread)" to unblock the current thread.

Synchronization

In the previous implementation, we were constantly checking if the amount of time, waited by there thread is lesser that the time it is supposed to wait, then we were using thread_yield(), to the pause the execution of the current thread. When the time waited by the current thread is greater than the time, it was supposed to wait, it starts executing. As the condition was being constantly checked, the processed was using memory and hence it was busy waiting.

The int64 t Time_To_Wait variable, which tracks thread blocked time and is added to the struct thread data structure to guarantee that all threads have this property, and has been used to ensure thread synchronisation. In order to guarantee that the determination of the thread status and sleep time is made for each thread, we introduced the new function thread_blocked_checker to the thread for each function. If the thread, satisfies the condition then it will be woken up and place in the ready queue. As we not constantly checking the condition, the problem of busy waiting has been replaced with a better form of synchronisation.

Rationale

To overcome busy wait, when the interrupt occurred we are disabling the interrupts, such that , to temporarily prevent the CPU from responding to the interrupts. Then we are defining the newly defined variable Time_To_Wait for the current thread to the specified number of ticks. Then we are blocking the current thread, until the specified number of ticks, using the variable Time_To_Wait. Once the current thread state was changed from running to blocked, we are setting interrupt level to the previous state. So that CPU can start responding to it. With our new implementation there is only one thread running at a time, and the check will only be made once the thread has completed the specified number of ticks, through the Time_To_Wait variable. Hence we can conclude that busy waiting problem has solved and there is no wastage of CPU resources.