OS:Assigment3: Pintos

Submission Report by: Suraj Garg(2018202003) and Prakash Nath Jha(2018201013)

-----------------------------------------------------------------------------------------------------------------------------------

**Part 1: Getting Started**

* Installed and setup the pintos using QEMU
* Successfully executed hello.c

-----------------------------------------------------------------------------------------------------------------------------------**Part 2: Sleeping of threads and Removal of Busy-waiting**

-----------------------------------------------------------------------------------------------------------------------------------

Changed Files:

2.1: thread.h

* Added a member variable named "**wakeup\_tick**" in struct thread to keep track of wake-up time for each thread.
* Added the declaration of comparator function to sort the threads based on wakeup time

in non-decreasing order, whose declaration is as below:

**bool thread\_wakeup\_ticks\_less(const struct list\_elem \*, const struct list\_elem \*, void \*);**

2.2: thread.c

* Definition provided for

**bool thread\_wakeup\_ticks\_less(const struct list\_elem \*, const struct list\_elem \*, void \*);**

2.3: timer.c

* Declared "**static struct list sleep\_list**" to keep track of sleeping threads in the

non-decreasing order of their wakeup time.

* function "*void timer\_init()*" is modified to initialize sleep\_list
* function "*void timer\_sleep ()"* is modified in following way:

This function is called for a thread whenever that needs to be blocked,

so this function does the following:

1. Interrupt disabled

2. Adds that thread into the sleep\_list in non-decreasing order of wakeup time

3. Sets the time to wake-up for that thread

4. Blocks the thread

5. Interrupt enabled

* function "*timer\_interrupt ()*" is modified in following way:

It compares the timer\_ticks() (which gives the number of elapsed ticks since the OS has booted)

with the wakeup time of thread present at the front of the sleep\_list(if list is not empty)

and if found greater then:

1. removes that thread from the sleep\_list

2. Unblocks it

Note:

1. Busy Waiting eliminated from timer\_sleep() method:

Previously this function suspends execution of the calling thread until time has advanced by

at least x timer ticks. Unless the system is otherwise idle, the thread need not wake up after

exactly x ticks. Just put it on the ready queue after they have waited for the right amount of time.

**Our Impl**: In timer\_sleep() method first interrupts are disabled, then wakeup time is set in thread's wakeup\_tick variable and it is added to the sleep\_list in non-decreasing order of their wakeup time and after that thread is blocked by calling thread\_block() method and whenever any timer interrupt occurred, then in timer\_interrupt() method the threads in the sleep\_list are checked to wake-up.

If they have waited for the specified sleep duration then our impl removes them from the sleep\_list, unblock them and put them in the ready\_list. After that they will be scheduled later according to the scheduling policy.

2. Synchronization among threads when when multiple threads call timer\_sleep() simultaneously:

Interrupts are disabled before adding a thread to sleep\_list and blocking it to acheive synchronization among mutiple threads calling this function.

----------------------------------------------------------------------------------------------------------------------------------- **Part 3: Implementation of priority scheduling**

-----------------------------------------------------------------------------------------------------------------------------------

Changed Files:

2.1: synch.h

* Added the declaration of comparator function to sort the threads based on priority

in non-increasing order, whose declaration is as below:

**bool sema\_priority\_high (const struct list\_elem \*, const struct list\_elem \*, void \*);**

2.2: synch.c

* Definition provided for:

**bool sema\_priority\_high (const struct list\_elem \*, const struct list\_elem \*, void \*);**

* function "*void sema\_down ()*" is modified in following way:

if down operation on the semaphore is unsuccessful then thread are added in the sema's waiter list in non-increasing order of their priorities instead of adding them in FIFO order.

* function "*void sema\_up ()*" is modified in following way:

whenever a thread needs to be removed from sema's waiter\_list, always highest priority thread will be popped because we have maintained the waiter\_list in order of the threads' priority. After removing from waiter\_list its priority is compared with the currently running thread and if found greater then the currently running thread will be preempted.

* function "*void cond\_signal ()*" is modified in following way:

cond->waiter is sorted in non-increasing order of the thread's priorities before calling

sema\_up() so that in sema\_up always highest priority thread will be removed from the

cond->waiter list.

2.3: thread.h

* Added the declaration of comparator function to sort the threads based on priority

in non-increasing order, whose declaration is as below:

**bool thread\_priority\_higher(const struct list\_elem \*, const struct list\_elem \*, void \*);**

* Added the declaration of a function which compares and the priorities of the currently running

thread and thread at the head of the ready\_list and if found greater then preempt the

currently running thread.

**void check\_thread\_preemption(void);**

2.4: thread.c

* Definition provided for functions:

**bool thread\_priority\_higher(const struct list\_elem \*, const struct list\_elem \*, void \*);**

**void check\_thread\_preemption(void);**

* function "*thread\_create()*" is modified in following way:

when a new thread is successfully created and placed in the ready\_lista according to its

priority then a call to the function check\_thread\_preemption()

is made which compare the priority of newly created thread with currently running

thread and if found greater then currently running thread will be preempted.

* function "*thread\_unblock()*" is modified in following way:

Adding unblocking thread in the ready\_list based on the priority

* function "*thread\_yield()*" is modified in following way:

Adding unblocking thread in the ready\_list based on the priority

* function "*thread\_set\_priority()*" is modified in following way:

Sets the current thread's priority to NEW\_PRIORITY.

check to yield itself if new priority is less than old priority

Note:

1. Ensuring that the highest priority thread is waking first:

Each thread is added into ready\_list in the non-increasing order of its priority so front of the ready\_list contains the highest priority thread. Whenever any thread needs to be scheduled then scheduler picks the thread from the front of the ready\_list.