Pintos scheduler

Group 16
14CS30011 - Kaustubh Hiware
14CS30017 - Surya Midatala

(i) how the existing Pintos Scheduler works (the scheduling policy is simple, explain how a context switch happens stating the functions called in sequence and what does each function do)

A context switch occurs in pintos in three cases:

- A thread is blocked (thread_block()),
- A thread is exited (thread_exit()),
- CPU is yielded and current thread is put to sleep(thread_yield()).

In each of these cases, the current thread is stopped and a new thread is scheduled by calling the schedule() function.
In the schedule function, the next thread to run is determined by the next_thread_to_run() function. The function does it by searching for the first non-empty queue and returns it to schedule(). To actually switch the threads, a switch_threads() function is called. This function stores the current thread's registers and cpu variables on the stack and restores the new thread's registers and variables from the stack and then returns.
Then schedule() calls thread_schedule_tail() function which marks the new thread as running, and if the older thread is dying, frees its memory.

(ii) data structures you have added/modified

In **threads.h**, struct thread was modified adding, 'level' to keep a track of the current level (1 or 2), 'num_run' to store the number of times the process has run in Level 1 (A process in Level 2 is not concerned with number of times it has run, only with its waiting time) and 'waiting_time' to store the time a process has been waiting in Level 2 while Level 1 was empty.

(iii) which functions you have added/modified (for added function (if any), explain what it does; for modified functions explain what you modified)

Thread_init function was modified as follows:

Ready_list_1 and Ready_list_2 are initialised along with the new attributes added to struct thread.

Thread_tick function was modified as follows:

After each tick, we update waiting_time for the waiting threads, num_run for the running thread and count. Then we check whether any waiting thread has been waiting for more than 6T. If so, it is sent to Level 1 queue.

Thread_block function was modified as follows:

We check the level the thread was in before it was run and add it to the end of that queue.

Thread_yield function was modified as follows:

If a thread belongs to Level 1 queue, we check if it has been run twice and send it to Level 2 queue if it has. Otherwise we send it to Level 1 Queue. If it belongs to Level 2 it is simply sent back to level 2.

Next_thread_to_run function is modified as follows:

If both queues are empty, idle thread is returned. If Level 1 Queue is empty, A thread is popped from Level 2 Queue. Otherwise, a thread is popped from Level 1 Queue.

Schedule function is modified as follows:

If a thread from level 2 is scheduled, Waiting time of the the scheduled thread is set to 0.

Print statements are added to the following functions:

Thread_tick, Thread_create, Thread_block, Thread_unblock, Thread_exit, Thread_yield, kernel_thread, schedule

(iv) which files you have modified.

The files changed were **threads.c** and **threads.h**.

Pintos memory management

(i) how the new Pintos memory management works (explain the within-page memory allocator)

The minimum block size allocated is 16 bytes.We will be finding the largest block of power 2 that satisfies the request. If the size exceeds 1KB, a new page will be requested using palloc. For blocks smaller than 1KB, we will recursively continue finding the appropriate size block until the right size is found.
An address some bytes above the available address (A) is returned and information such as current level and occupied blocks is kept in that space. A global lock is used instead of a lock per arena, and it is used whenever a block is to be allocated or freed.
When a block is to be freed, we will start at the hidden address(A) in order to get information about the buddies of the block to be freed.In case the size of block to be freed in more than 1KB, the page will be freed correspondingly.

(ii) data structures you have added/modified

arena and **block** have been modified. A list element has directly been added in arena, and information regarding current level (a measure of size) and occupied blocks count is additionally stored in block.

(iii) which functions you have added/modified (for added function (if any), explain what it does; for modified functions explain what you modified)

malloc has been modified to change to buddy memory allocation.
In **malloc** we allocate the smallest size block that can satisfy the request and allocates new pages as required.
PrintMemory function prints out the starting addresses for the free memory blocks for every page allocated, in the same format as given in the assignment.
Smaller_block is a utility function that just returns if first block is smaller than the second or not.

(iv) which files you have modified.

The file changed was **malloc.c**.