Operating System Design MP3 Report

Pin-Xuan Lee (plee18)

Yu-Lin Chien (ychien8)

Shuo-Yang Wang (swang234)

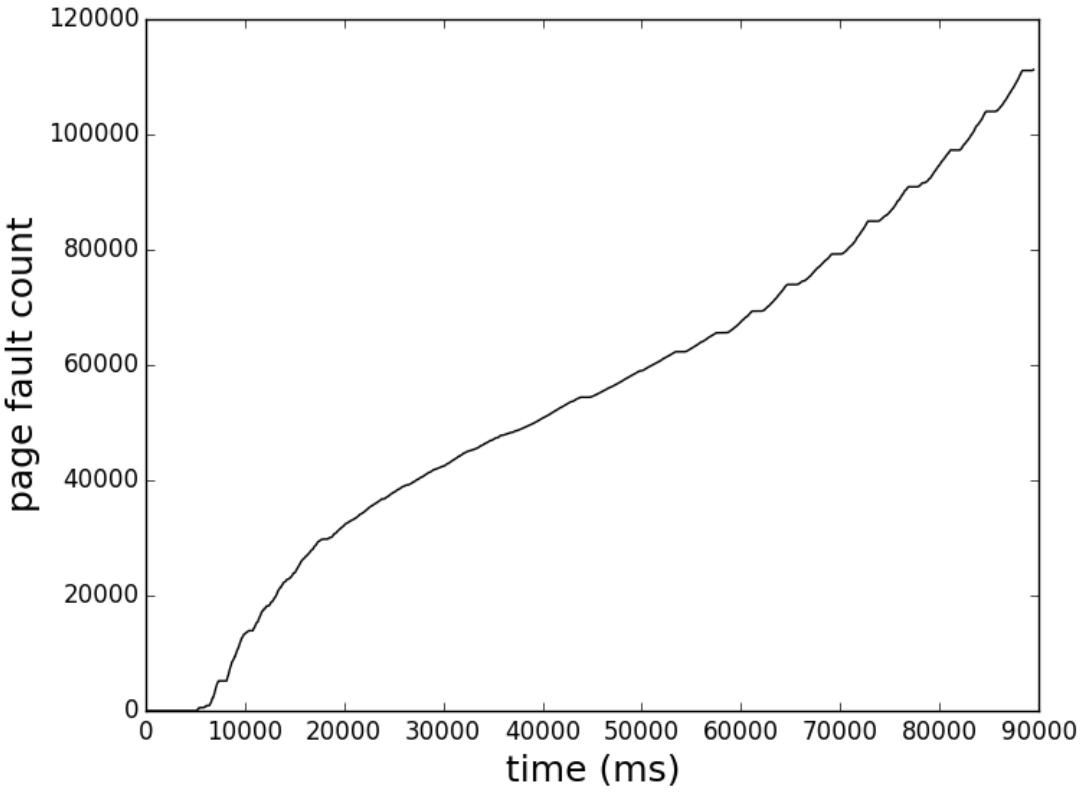
Wei-Tze Tsai (wtsai10)

Contain files:

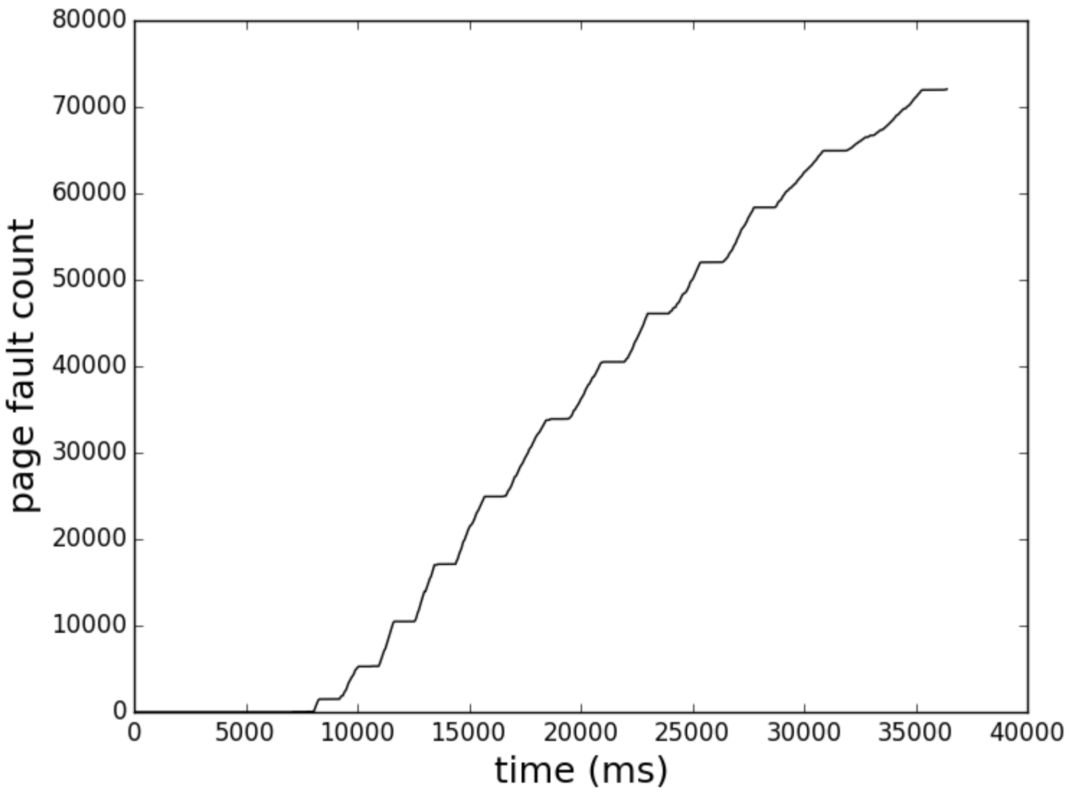
Makefile mp3.c mp3_given.h work.c monitor.c reinstall_and_test_module.sh

Graphs and logical analysis for the case study 1

Graph 1.1 (work processes 1 and 2) (x-axis is time (ms), y-axis is page fault count)



Graph 1.2 (work processes 3 and 4) (x-axis is time (ms), y-axis is page fault count)



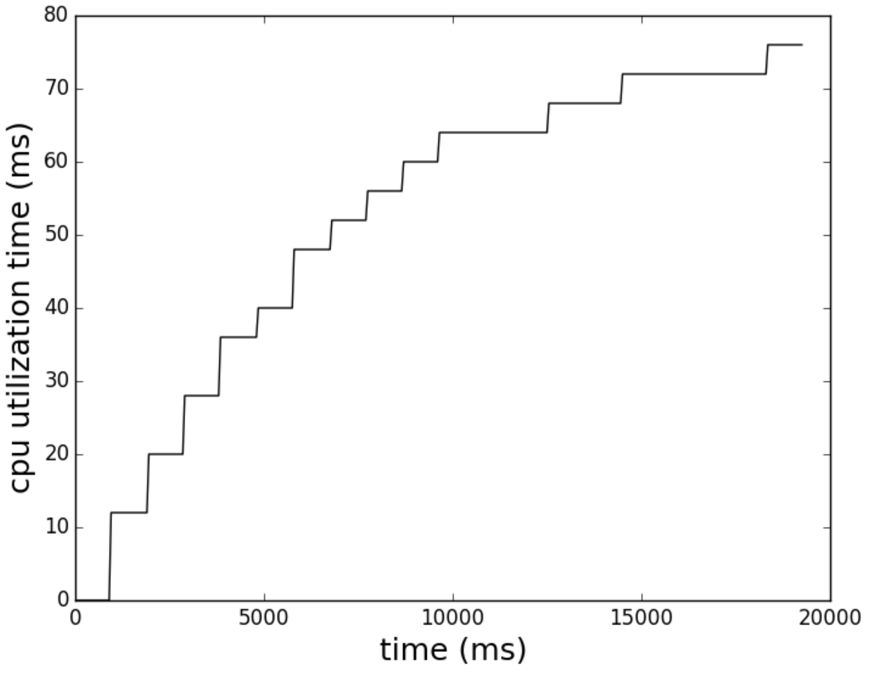
Work process 1 and work process 3 are the same. Work process 2 and work process 4 have the same #memory access, but different method to access memory. Work process 2 randomly access memory, while work process 4 has locality-based memory access method.

From the above two graphs, we can tell that the page fault count of two random memory access processes is much higher than one process having random memory access and the other having localized memory access.

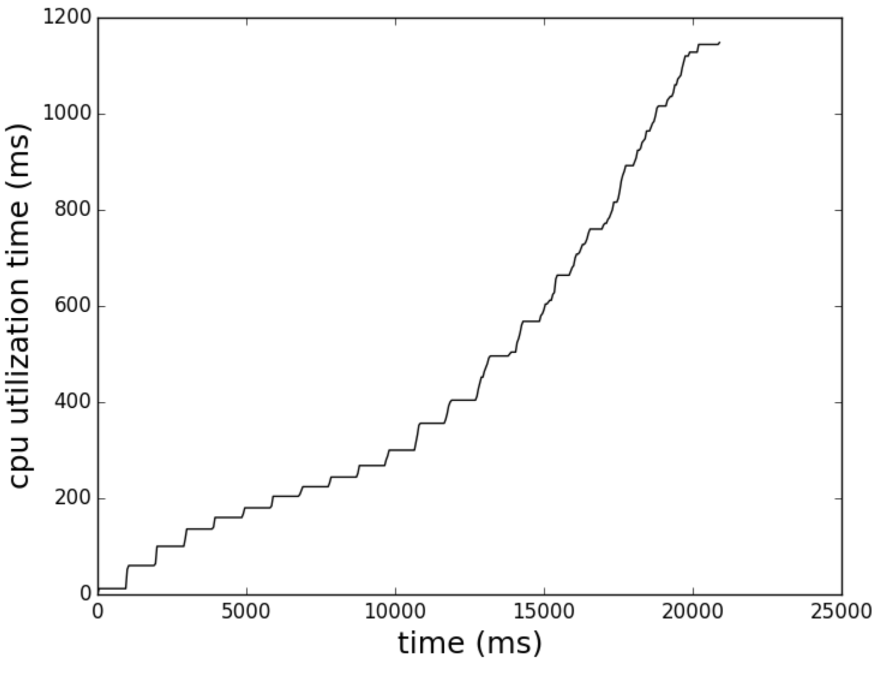
In addition, the time required for two random memory access processes to complete is also much higher than one process having random memory access and the other having localized memory access.

Graphs and logical analysis for the case study 2

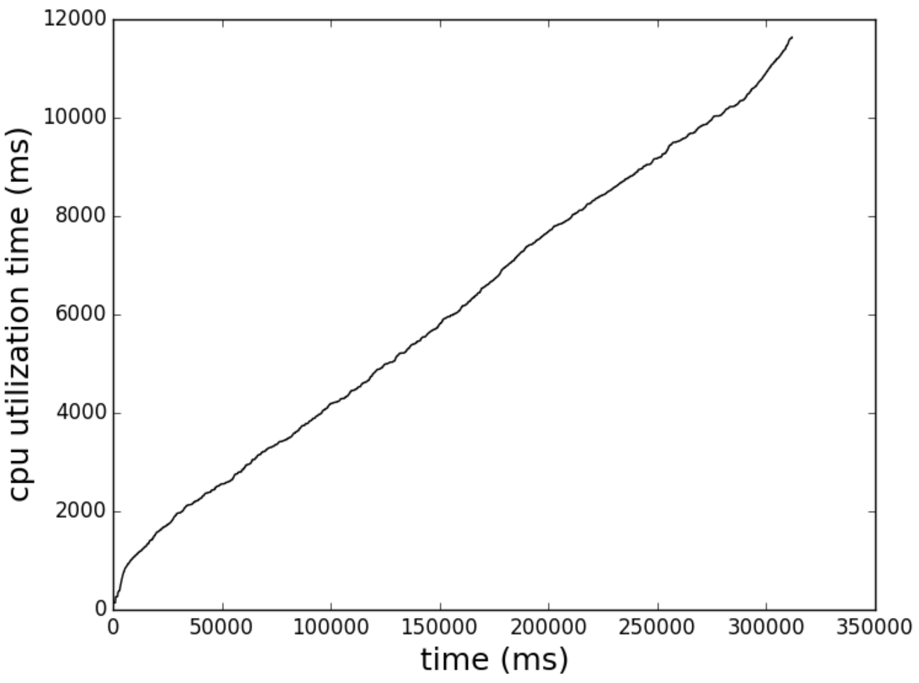
Graph 2.1 (N = 1) (x-axis is time (ms), y-axis is cpu utilization time (ms))



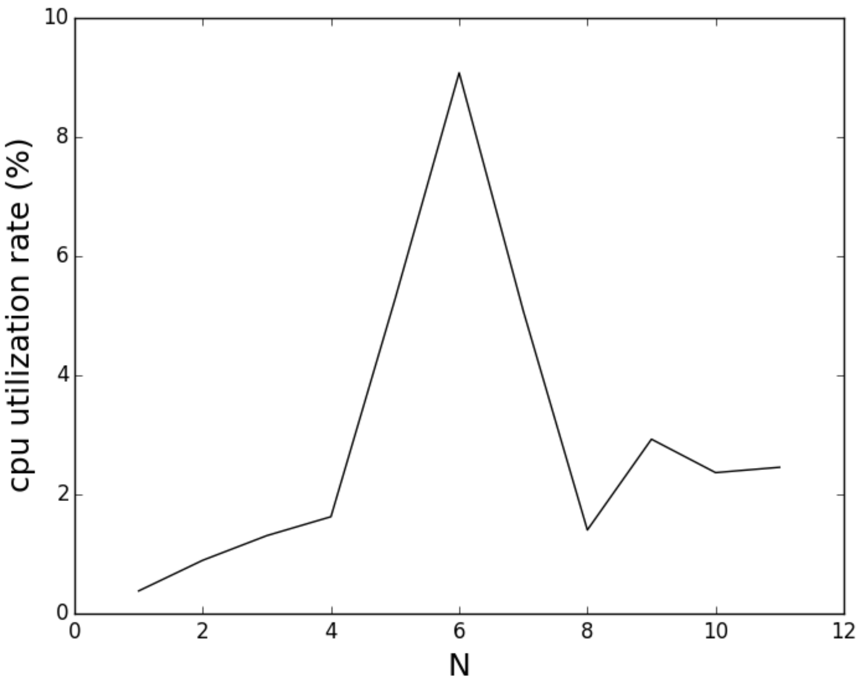
Graph 2.2 (N = 5) (x-axis is time (ms), y-axis is cpu utilization time (ms))



Graph 2.3 (N = 11) (x-axis is time (ms), y-axis is cpu utilization time (ms))



The difference between Graph 2.1, Graph 2.2, and Graph 2.3 are that the CPU utilization time: Graph 2.2 > Graph 2.3 > Graph 2.1.



The reason of above phenomenon might due to thrashing. The CPU utilization will increase while degree of multiprogramming increases. However, when the degree of multiprogramming reaches a threshold, the CPU utilization will decrease dramatically because the working memory is too large, and this will cause lots of page faults happen.

Implementation and design decisions:

mp3_init():

Create /proc/mp3 directory and create /proc/mp3/status file.

Initialize proc file, spinlock, workqueue, character device, and use vmalloc() to set profile buffer.

mp3_register_process():

Get pid for the process, create and insert mp3_task_struct into linked list, and queue delayed work into work queue.

mp3_unregister_process():

Remove the process form the linked list, and release memory of the process. Make sure all works are halt and flushed if all processes have unregistered.

mp3_write():

Get either register or unregister messages from the process.

mp3_work_function():

Calculate minor page fault, major page fault, and cpu time, and write them into profiler buffer.

mp3_cdev_mmap():

Use vmalloc_to_page(virtual_address) and vm_insert_page() to map the physical pages from the buffer to the virtual address space of a requested user process

mp3_exit():

Remove proc file and free everything setting up at mp3_init()