Report for CS423 MP3

Name: Xiaocong Yu
netID: xy21
Email: xy21@illinois.edu

Design and implementation

The work flow of the machine problem is implemented as described in the instruction. To make it simple and clear, I conclude it as three process: module interface for register and unregister, mmap character device callback, delayed work for regular update.

Module Interface:
After the input is parsed by the write callback of proc file, the request will be dispatched to corresponding entry function. The register entry will allocate one memory for new struct by using kmalloc(supposed to use kmemcache for better performance in handling more concurrent job) and add it to the linked list of registered process.

Workqueue:
The workqueue is initialized during the init process of module and destroyed during the exit process of module. The delayed work is initialied when the first process is registered and is destroyed when the only process is unregistered. The work is queued repeatedly in the work_callback function by passing delay parameter as 0.05 second(5 jiffiies on x86 linux).

Mmap character device callback:
The character device is created during the init process of module and removed during the exit process of module. The device is initialized with file_operations struct which specifies the call back function pointer for mmap. In the mmap_callback function, I translate the virtual memory address in unit of PAGE_SIZE to physical memory, then remap the user's memory pointer in vm_area_t to the physical address. By iterating till the limit of the memory address, I map the physical address where kernel address points to to user memory address. In the experiment, I have to create the file representation of the device by mknod so that monitor program can use glibc to mmap with character device.

Case Study 1: Thrasing and Locality

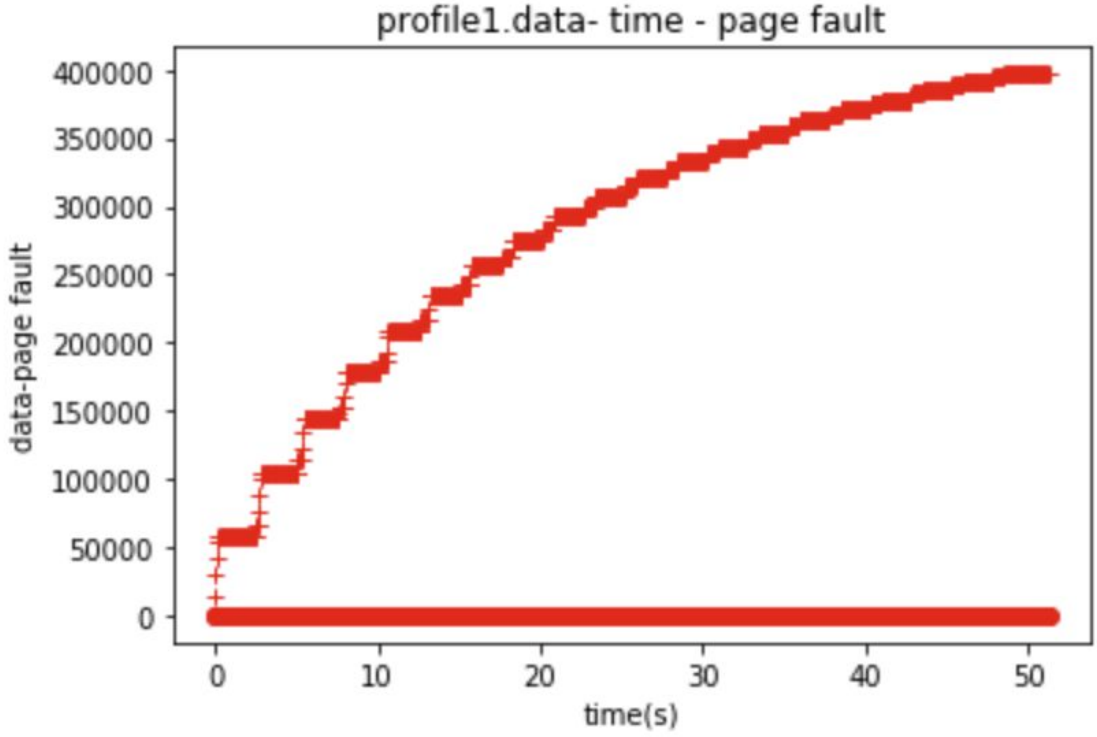

Fig 1.1 Case 1 Work process 1 & work process 2 &

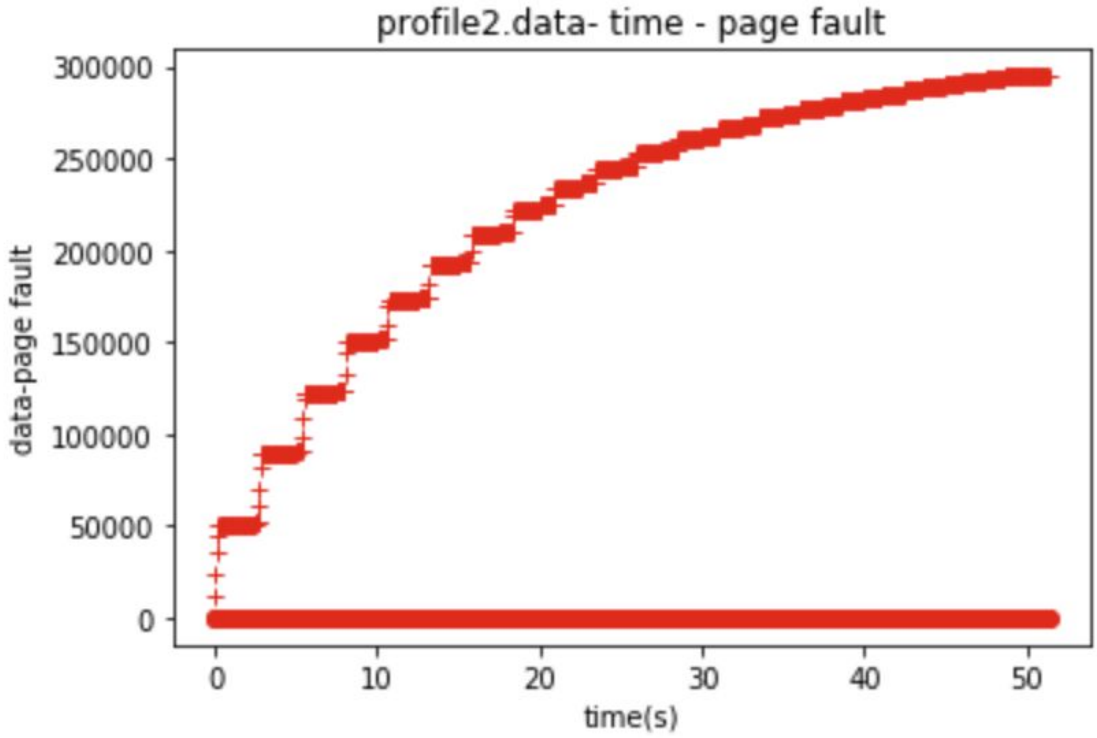

Fig 1.2 Case 2 Work process 3 & work process 4 &

Analysis: (the difference between two profiles in perspective of quatity)

In the first case, I launch two random access work at the same time, the data collected shows that the minor fault is increasing as time went by. And in the second case I launch one random access work and one local access work at the same time, the minor page fault is also increasing.

Memory allocated for two case are same, and the only different argument for two case is the access pattern. Since the random access requires more jumps in the memory space than local access, it will cause more page faults during its execution by probability. The figure justifies the suppose that the max accumulated value of page fault for case 1 is larger than(about 33%) case 2 since it executes more random access on memory.

Two work in a case will still only require for 2000 MB whereas our virtual machine is granted with 5 GB memory, so we will seldomly have major page fault in both cases.

The data collected for two cases are same in length, so the execution time for two cases are almost the same. For each work, the iteration is fixed, so the execution time is positive related to the number of work when there is low concurrency.

Case Study 2: N random access work, concurrent

In this case we run N work simultaneously, each work's will allocate for 200MB memory, random access pattern, 10000 times data access.

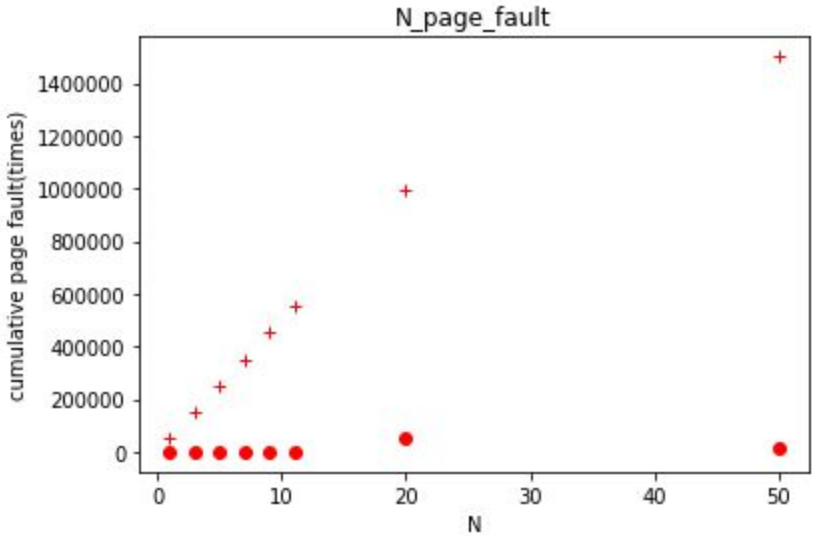

Fig 2.1 The page fault times for different N(dot for major page fault, cross for minor page fault)*

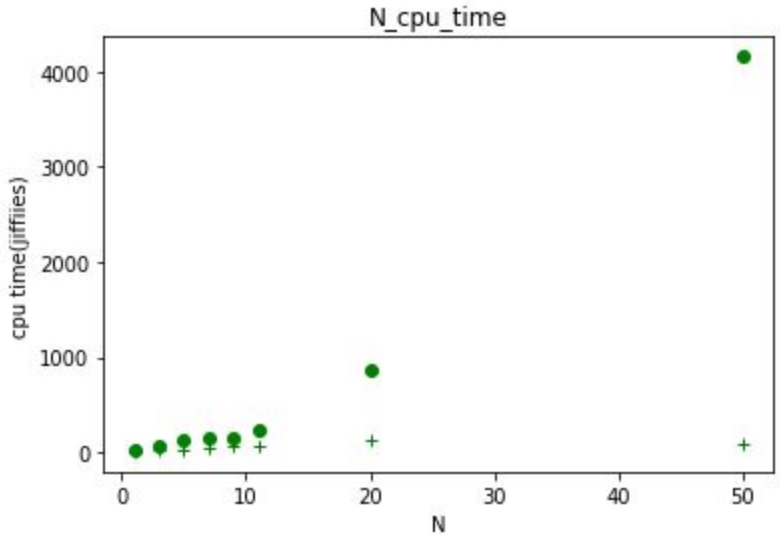

Fig 2.2 The CPU time for different N(dot for stime, cross for utime)*

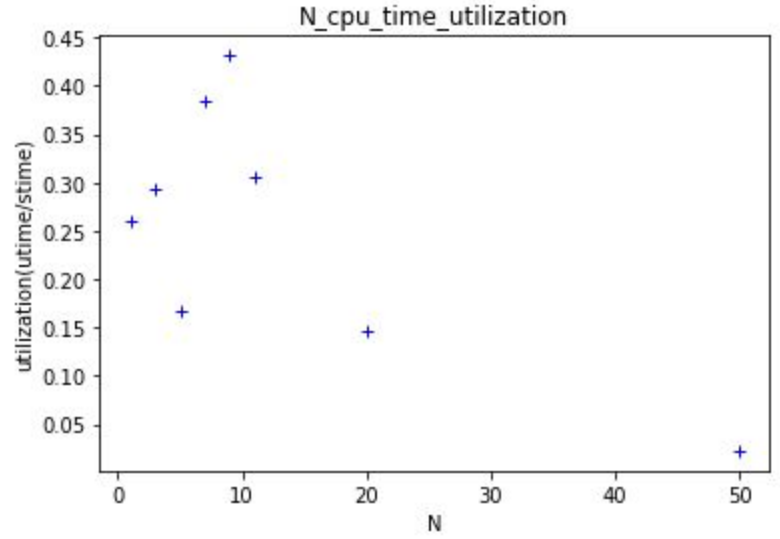

Fig 2.2 The CPU utilization for different N

Analysis:

The cpu utilization time is linearly increasing for both stime(kernel space) and utime(user space). This is because we are using multi-process to carry out the work. And there is not much concurrency, so the utilization rate is kept in a relatively normal range.

Extreme case:
As I increase N for test, the utilization rate will have a obvious drop tendency. This is mostly because the increased concurrency will make the page fault rate increasing, thus the program will cost more time in dealing with the waiting for page fault handler to register the swapped page to its MMU and update its page table more frequently.