Lab 1 Report for CS380L

Omeed Tehrani

February 16, 2024

Introduction

Let's start with reproducibility. You will write a report for this lab, and in your report you will include details about your system. Think about what it would take to recreate your results. I won't spell out exactly what information you should include, but please include everything relevant while not bogging down your reader. You should report things like the kernel version for your host and guest system.

Environment

For this second lab, I decided to go with CloudLab for my host machine. I set up a new experiment using the default profile, which is an Ubuntu 22.04 LTS, which is under the code name "jammy". I ran a sudo apt install –install-recommends linux-generic-hwe-22.04, which allowed me to gain access to a newer kernel version (6.5) from the (5.15) version on the original machine launch. I am on the Emulab cluster. The system is running an x86-64 architecture, which I confirmed again by running the `getconf LONG_BIT` command. Using the `lscpu` command, I found that my system has 8 CPUs, specifically the Intel Xeon E5530 processors, each with 4 cores and 2 threads per core. Next, I called `getconf -a | grep CACHE`, which gave me insight into configuration values related to the system cache. The output was the following:

```
omeed26@node0:~$ getconf -a | grep CACHE
LEVEL1_ICACHE_SIZE                 32768
LEVEL1_ICACHE_ASSOC
LEVEL1_ICACHE_LINESIZE             32
LEVEL1_DCACHE_SIZE                 32768
LEVEL1_DCACHE_ASSOC                8
LEVEL1_DCACHE_LINESIZE             64
LEVEL2_CACHE_SIZE                  262144
LEVEL2_CACHE_ASSOC                 8
LEVEL2_CACHE_LINESIZE              64
LEVEL3_CACHE_SIZE                  8388608
LEVEL3_CACHE_ASSOC                 16
LEVEL3_CACHE_LINESIZE              64
LEVEL4_CACHE_SIZE                  0
LEVEL4_CACHE_ASSOC
LEVEL4_CACHE_LINESIZE
```

This tells me that level 1 data cache capacity is 32,768 and is 8-way set associative. Next, I ran `sudo apt-get update`, `sudo apt-get install cpuid` and then `cpuid | grep -i tlb`. Here is a snippet

of the output below:

```
   cache and TLB information (2):
      0x5a: data TLB: 2M/4M pages, 4-way, 32 entries
      0x03: data TLB: 4K pages, 4-way, 64 entries
      0x55: instruction TLB: 2M/4M pages, fully, 7 entries
      0xb2: instruction TLB: 4K, 4-way, 64 entries
      0xca: L2 TLB: 4K pages, 4-way, 512 entries
   L1 TLB/cache information: 2M/4M pages & L1 TLB (0x80000005/eax):
   L1 TLB/cache information: 4K pages & L1 TLB (0x80000005/ebx):
   L2 TLB/cache information: 2M/4M pages & L2 TLB (0x80000006/eax):
   L2 TLB/cache information: 4K pages & L2 TLB (0x80000006/ebx):
```

Observing the snippet from the output above, we can see that the data TLB capacity is 32 entries for 2M/4M pages, and 64 entries for 4K pages. These are both 4-way set associative. We can also observe the existence of a level two TLB, which has a capacity of 512 entries for 4K pages, and is 4-way set associative. This layer can be useful in performance improvement.

Memory Mapping

0.1 Part 1 and Part 2

In this section, we were instructed to write a program that opens, reads and prints the /proc/self/maps file. I chose C++ for the language to write this program. In the main root user directory, I created a new file with vim called `proc_program.cpp`. The program is quite simple, I simply set up my pointers and print the memory map to the standard output. I also ensure to check for the system call error (as per the instructions). I compile it using `g++ -o proc_program proc_program.cpp`. The code is below:

```
#include <cstdio>
#include <cstdlib>

int main() {
        FILE *file;
        char buffer[1024];

        file = fopen("/proc/self/maps", "r");

        // for every system call, we need to check if the return code is
        // less than zero, if it is, we need to call perror (lab1 note)
        if (file == nullptr) {
                // nullptr is supposedly safer?
                perror("Error in system call fopen");
                return EXIT_FAILURE;
        }

        while (fgets(buffer, sizeof(buffer), file) != nullptr) {
                printf("%s", buffer);
        }

        fclose(file);
```

```
        return EXIT_SUCCESS;
}
```

Next, I run the program. The results are reported below:

```
omeed26@node0:~$ ./proc_program
55bd4c273000-55bd4c274000 r--p 00000000 08:03 3997722                    /users/omeed26/proc_program
55bd4c274000-55bd4c275000 r-xp 00001000 08:03 3997722                    /users/omeed26/proc_program
55bd4c275000-55bd4c276000 r--p 00002000 08:03 3997722                    /users/omeed26/proc_program
55bd4c276000-55bd4c277000 r--p 00002000 08:03 3997722                    /users/omeed26/proc_program
55bd4c277000-55bd4c278000 rw-p 00003000 08:03 3997722                    /users/omeed26/proc_program
55bd4dec6000-55bd4dee7000 rw-p 00000000 00:00 0                          [heap]
7fb400655000-7fb400658000 rw-p 00000000 00:00 0
7fb400658000-7fb400680000 r--p 00000000 08:03 248540                     /usr/lib/x86_64-linux-gnu/l
7fb400680000-7fb400815000 r-xp 00028000 08:03 248540                     /usr/lib/x86_64-linux-gnu/l
7fb400815000-7fb40086d000 r--p 001bd000 08:03 248540                     /usr/lib/x86_64-linux-gnu/l
7fb40086d000-7fb400871000 r--p 00214000 08:03 248540                     /usr/lib/x86_64-linux-gnu/l
7fb400871000-7fb400873000 rw-p 00218000 08:03 248540                     /usr/lib/x86_64-linux-gnu/l
7fb400873000-7fb400880000 rw-p 00000000 00:00 0
7fb400886000-7fb400888000 rw-p 00000000 00:00 0
7fb400888000-7fb40088a000 r--p 00000000 08:03 246601                     /usr/lib/x86_64-linux-gnu/l
7fb40088a000-7fb4008b4000 r-xp 00002000 08:03 246601                     /usr/lib/x86_64-linux-gnu/l
7fb4008b4000-7fb4008bf000 r--p 0002c000 08:03 246601                     /usr/lib/x86_64-linux-gnu/l
7fb4008c0000-7fb4008c2000 r--p 00037000 08:03 246601                     /usr/lib/x86_64-linux-gnu/l
7fb4008c2000-7fb4008c4000 rw-p 00039000 08:03 246601                     /usr/lib/x86_64-linux-gnu/l
7ffe89143000-7ffe89164000 rw-p 00000000 00:00 0                          [stack]
7ffe891bd000-7ffe891c1000 r--p 00000000 00:00 0                          [vvar]
7ffe891c1000-7ffe891c3000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

To understand the output, it was integral for me to first understand what `/proc/self/maps` is. `/proc` is supposedly referred to as a process information pseudo-file system. More specifically, it enables us to obtain information about a processes region of memory. It can be called on a process-id when that is specified in the path, but when it is not (and you use self), it will output a memory map of the process that is accessing the file, which in this case, is my `proc_program.cpp`. Each row here is describing a region of contiguous virtual memory, divided up into: the starting and ending address within its region, permissions for page access in that region of memory, offset in the file where a mapping began (if it is zero, the memory was not mapped from a file), major and minor device numbers, the inode number and the path name of the file that has been mapped into this particular address space. Generally, the reason we are seeing so many rows, is because each row represents some different memory region that does a specific task to execute the program. What we are seeing is a lot of different parts of this program mapped to different permissions. Next, we can see a heap and stack entry, which is likely related to the memory allocation of our program and the storage of addresses and other components of the program occur. The zeros we are seeing are likely intentional anonymity. Finally, there are rows for libraries such as libc, which is important to help us perform broad operations like linking and input and output. There are also some interesting things such as vvar, vdso, and vsyscall which I will explain below.

In this case, the base address of my executable is 0x55bd4c274000. The reasoning for this is that the permissions to execute only exist within the second line of output. The reference to this base address is what allows us to execute `./proc_program` Additionally, the start address of libc is 0x7fb400658000.

I inferred this from the path. The assignment handout asks us to explain why these numbers are so different. To understand this, I dug a bit into some Ubuntu wikis:
https://wiki.ubuntu.com/Security/FeaturesAddress_Space_Layout_Randomisation_.28ASLR.29.
Basically, I found that each execution of a program will result in a different mmap memory space layout (leading to the dynamically loaded libraries loading into different locations). This is apparently intentional, as it makes it more difficult to locate in memory where to jump to in something supposedly known as "return to libc" exploit. It has actually been like this since early Ubuntu versions!

One of the most interesting things that I found in my output was `vvar`, `vdso` and `vsyscall`. vvar has values (such as kernel variables) that were not available in user-space, but we want them to be. Supposedly this can be useful in variables necessary for virtual dynamic shared object (vdso) (https://lwn.net/Articles/615809/), which we notice is actually the next thing on the row list. The reason we see `vsyscall` after `vdso`, is because `vdso` was introduced to solve the security holes of `vsyscall`. Generally, the purpose of these last two system calls is to allow us to make a system call in user-space to reduce the total amount of context switching with the kernel. This can be useful with things like time, which they highlight in the linux `vdso` article.

getrusage

For this portion, I added an additional snippet to the bottom of my C++ code:

```
struct rusage ru;
if (getrusage(RUSAGE_SELF, &ru) != 0) {
     perror("Error in system call getrusage");
     return EXIT_FAILURE;
} else {
     printf("Utime (tv_sec): %ld.%09ld \n", ru.ru_utime.tv_sec);
     printf("Utime (tv_usec): %ld.%09ld \n", ru.ru_utime.tv_usec);
     printf("Stime (tv_sec): %ld.%09ld \n", ru.ru_stime.tv_sec);
     printf("Stime (tv_usec): %ld.%09ld \n", ru.ru_stime.tv_usec);
     printf("Maxrss: %ld \n", ru.ru_maxrss);
     printf("Minflt: %ld \n", ru.ru_minflt);
     printf("Majflt: %ld \n", ru.ru_majflt);
     printf("Inblock: %ld \n", ru.ru_inblock);
     printf("Outblock: %ld \n", ru.ru_oublock);
     printf("Voluntary C.S: %ld \n", ru.ru_nvcsw);
     printf("Involuntary C.S: %ld \n", ru.ru_nivcsw);
}
```

The output was the following:

```
Utime (tv_sec): 0.231
Utime (tv_usec): 1035
Stime (tv_sec): 0.0
Stime (tv_usec): 0.0
Maxrss: 2524
Minflt: 81
Majflt: 0
Inblock: 0
Oublock: 0
```

```
Voluntary C.S: 0
Involuntary C.S: 0
```

These fields are the ones we were instructed to obtain. I learned about them and their types (long and timeval struct) using the `getrusage` manual page. The Utime is the total amount of time that is spent executing in user mode. The stime is the total amount of time spent executing in kernel mode. These are both represented in the timeval struct which contains seconds and microseconds. Maxrss is the maximum resident set size in KB. Minflt is the number of page faults that were serviced without I/O. Majflt is the number of page faults serviced that do require I/O. Inblock and Oublock represents the number of times the file system performed inputs and outputs. Voluntary and Involuntary Context Switching, which represent the number of times a context switch occurred due a process voluntarily giving up before the time slice was completed vs. the context switch occurring because a process with more priority became runnable or the time slice ran out.
Overall, this code I wrote will be very useful when I am running metrics on the provided functions such as `do_mem_access()`.

Perf Event Open

0.2 Part 1

Perf_Event_Open is a system call in Linux. It is used to set up performance monitoring for things like cache misses, cache hits, instruction execution, CPU data, etc. When you make a call to `perf_event_open()`, it returns a file descriptor to be used in subsequent system calls. The quirk of this system call is that `glibc` does not provide a wrapper for the `perf_event_open()` function, meaning that we have to use syscall(2) to open perf events. Here is an example of what it would look like:

```
#include <linux/perf_event.h>
#include <sys/syscall.h>
// to directly invoke system calls, we need to include the header file.
#include <unistd.h>

static long perf_event_open(struct perf_event_attr *hw_event, pid_t pid,
                            int cpu, int group_fd, unsigned long flags) {
          int ret;
          ret = syscall(SYS_perf_event_open, hw_event, pid, cpu,
                        group_fd, flags);
          return ret;
      }
```

Next, now that we can use the `perf_event_open()` system call, we have to set it up with whatever we are trying to performance measure in our code. For example, in the linux manual, they discuss an example on how to measure the instruction count of a call to printf:

```
int
 main(void)
 {
     int fd;
     long long count;
     struct perf_event_attr pe;
```

```
    memset(&pe, 0, sizeof(pe));
    pe.type = PERF_TYPE_HARDWARE;
    pe.size = sizeof(pe);
    pe.config = PERF_COUNT_HW_INSTRUCTIONS;
    pe.disabled = 1;
    pe.exclude_kernel = 1;
    pe.exclude_hv = 1;

    fd = perf_event_open(&pe, 0, -1, -1, 0);
    if (fd == -1) {
       fprintf(stderr, "Error opening leader %llx\n", pe.config);
       exit(EXIT_FAILURE);
    }

    ioctl(fd, PERF_EVENT_IOC_RESET, 0);
    ioctl(fd, PERF_EVENT_IOC_ENABLE, 0);

    printf("Measuring instruction count for this printf\n");

    ioctl(fd, PERF_EVENT_IOC_DISABLE, 0);
    read(fd, &count, sizeof(count));

    printf("Used %lld instructions\n", count);

    close(fd);
}
```

In this code, we initialize a `perf_event_attr()` struct to configure the system call `perf_event_open()`. Next, we have to specify the event in conjunction with our type field. In this case, they selected `PERF_COUNT_HW_INSTRUCTIONS`, but the documentation provides a diverse range of options such as `PERF_COUNT_HW_BRANCH_MISSES`, `PERF_COUNT_SW_PAGE_FAULTS_MAJ`, etc. All the boolean values that are set to 1 are indicating that we don't want to include events in the kernel, hypervisor, etc. We initialize the performance counters with their starting values and config, then use the input output control (ioctl) to set up the counters, enable them, disable them, and then read the respective values after the event has occurred. Answering the question in our handout, I agree, it is quite wild how we call it!

0.3 Part 2

The question that we are trying to answer in this section is "Does using the syscall routine mean there is a syscall opcode in your program? Check using objdump -d and explain what you find in your report." To accomplish this, I decided to write two very barebone programs, one with the `perf_event_open()` wrapper on the system call, and one without. I avoided doing any `ioctl`, and just focused on making the system call and ensuring there was no error. I made two files, `obj_dump_exp1.cpp` and `obj_dump_exp2.cpp`, and compiled them respectively. I show snippets below:

```
omeed26@node0:~$ sudo objdump -d ./obj_dump_exp1

./obj_dump_exp1:     file format elf64-x86-64

Disassembly of section .init:
```

```
0000000000001000 <_init>:
    1000: f3 0f 1e fa           endbr64
    1004: 48 83 ec 08           sub    $0x8,%rsp
    1008: 48 8b 05 d9 2f 00 00  mov    0x2fd9(%rip),%rax        # 3fe8 <__gmon_start__@Base>
    100f: 48 85 c0              test   %rax,%rax
    1012: 74 02                 je     1016 <_init+0x16>
    1014: ff d0                 call   *%rax
    1016: 48 83 c4 08           add    $0x8,%rsp
    101a: c3                    ret
```

In experiment one, it I observed no `syscall` instruction or syscall links/jumps in the disassembly across all sections. But after observing the output of experiment 2:

```
00000000000011c9 <_ZL15perf_event_openP15perf_event_attriiim>:
...
10d4: f2 ff 25 f5 2e 00 00  bnd jmp *0x2ef5(%rip)        # 3fd0 <syscall@GLIBC_2.2.5>
...
1205: b8 00 00 00 00        mov    $0x0,%eax
120a: e8 c1 fe ff ff        call   10d0 <syscall@plt>
120f: 89 45 fc              mov    %eax,-0x4(%rbp)
1212: 8b 45 fc              mov    -0x4(%rbp),%eax
1215: 48 98                 cltq
1217: c9                    leave
1218: c3                    ret
...
```

The answer to the question is yes. Why? Although we don't have an instruction that is a direct syscall, what we are seeing in this disassembly is two things: (1) `call` instruction for `perf_event_open()` with a `@plt` suffix on the syscall. This indicates that the call is made through the Procedure Linkage Table (PLT), which is a mechanism for dynamic linking. The actual address of the syscall is typically resolved during runtime by the dynamic linker. (2) a BND JMP instruction that targets the syscall function at a particular address. These things allow me to infer that there is likely the existence of a syscall opcode in our program.

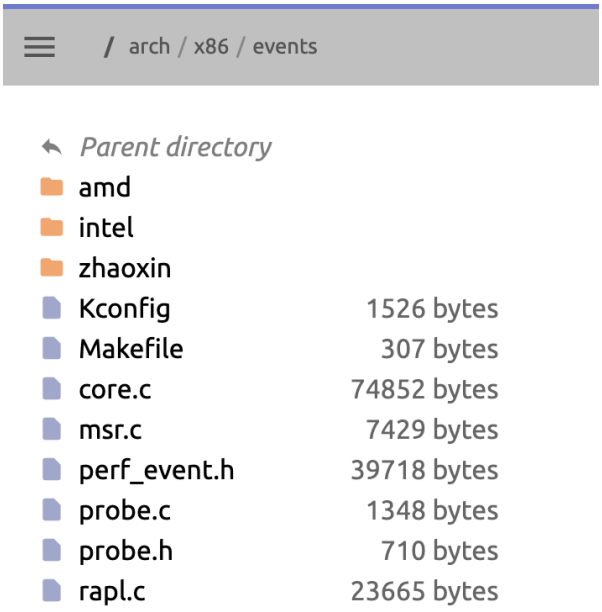
0.4 Part 3 to Part 7

For this section, first I had to install the proper tools for my current linux kernel on the Ubuntu 22.04 system using `sudo apt install linux-tools-$(uname -r)`. Next, I use the `perf list` command with specification and sudo: `sudo perf list cache`. My event output for L1 Data Cache and Data TLB is below:

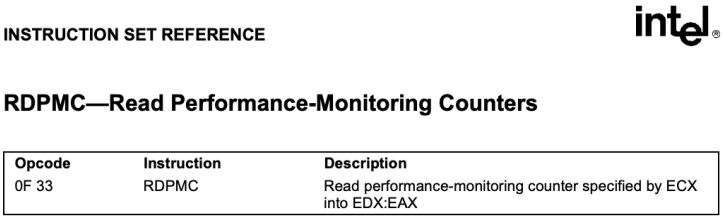
```
L1-dcache-load-misses                              [Hardware cache event]
L1-dcache-loads                                    [Hardware cache event]
L1-dcache-prefetch-misses                          [Hardware cache event]
L1-dcache-prefetches                               [Hardware cache event]
L1-dcache-store-misses                             [Hardware cache event]
L1-dcache-stores                                   [Hardware cache event]
...
dTLB-load-misses                                   [Hardware cache event]
dTLB-loads                                         [Hardware cache event]
dTLB-store-misses                                  [Hardware cache event]
```

```
dTLB-stores                                              [Hardware cache event]
```

On the provided LXR website, I found my linux kernel version on my machine, and navigated to the directory that is shown below:
`perf_event.h`:



These files likely define and handle x86 CPU-specific performance events for the kernel. The important note here is that names like "L1-dcache-load-misses", "dTLB-loads", etc.. can not be found in the linux kernel source code. This is because they are hardware events that can be mapped directly to the CPU events for a particular CPU vendor such as Intel or AMD. One thing I observed in both the `perf_event.h` file and the Intel x86 ISA was the RDPMC instruction, which likely provides a way for the user to access the neccesary performance counters for the perf command.


INSTRUCTION SET REFERENCE

RDPMC—Read Performance-Monitoring Counters

Opcode	Instruction	Description
0F 33	RDPMC	Read performance-monitoring counter specified by ECX into EDX:EAX

0.5 Part 4

To summarize the behavior of the function, I had to first make the required modifications that the assignment handout suggested:

```
omeed26@node0:~$ cat do_mem_access.cpp
```

```
// used C libraries that are easier to use.
#include <stdio.h>
#include <stdlib.h>

int opt_random_access = 0; // subject to change

#define CACHE_LINE_SIZE 64
#define MEM_SIZE (1024 * 1024 * 1024)

// PROVIDED PROGRAM GOES HERE

int main() {
 char* p = (char*) malloc(MEM_SIZE);
 if (p == nullptr) {
 printf("Failure in malloc.");
 return 1;
 } else {
 do_mem_access(p, MEM_SIZE);
 printf("Success in malloc and mem_access.\n");
 }
 // deallocate
 free(p);
 return 0;
}
```

At its core, this code is simulating a memory access pattern within some 1GB region pointed at by `char* p`. It simulates multiple memory accesses, with the `opt_random_access` global variable allowing us to switch between random and subsequent memory accessing. I gained some general useful insights from the perf utility tool about the differences between these patterns, but as per the TA's instructions and assignment page, my task is to monitor the performance counter metrics programatically using the lower level `perf_event_open()` interface that I discussed earlier in my write up. With this in mind, I began to make the necessary modifications to my program to effectively run the experiments. I describe them sequentially below:

1. First, I had to lock my process onto a single processor. This is important because it can help our system from scheduling some alternative process during execution and reduce the amount of variability that we have in our performance results. From my earlier analysis, I know that my system has 8 CPUs. Using this command: `awk '/ processor/print $3' /proc/cpuinfo`, I found all CPU ids on my system, which ranged from 0 to 7. Next, I dug into the Linux manual, and found a function called `sched_setaffinity()`, which can set a CPU affinity mask on a multiprocessing system to obtain performance benefits. The function header is the following: `int sched_getaffinity(pid_t pid, size_t cpusetsize, cpu_set_t *mask)`. They note that we should specify 0 as PID to set the attribute for the calling thread, which is that current program that we are running. After setting up the parameters, it is as simple as adding our CPU to the set of allowed CPUs and enabling the thread to run on the specified CPU. I made a simple program to set the CPU affinity, execute a basic task and confirm the main thread is running on the CPU specified. The code I wrote is below:

```
omeed26@node0:~$ cat lock_test.cpp
#include <stdio.h>
#include <stdlib.h>
// allows us to use affinity and getcpu() to test.
#include <sched.h>
#include <unistd.h>

int main() {

    int cpu_id = 4;
    pid_t pid = 0;
    cpu_set_t mask;
    CPU_ZERO(&mask);
    CPU_SET(cpu_id, &mask);
    size_t cpusetsize = sizeof(mask);
    if(sched_setaffinity(pid, cpusetsize, &mask) == -1) {
        perror("Oh no. CPU Set Operation Failed.");
        return EXIT_FAILURE;
    }

    // some basic operations :)
        for (int i = 0; i < 10; i++) {
                printf("%d", i);
        }
        printf("\n");
        for (int i = 0; i < 10; i++) {
                printf("%d", i);
        }
        printf("\n");

    printf("Basic Operations Complete. \n");
    printf("CPU that was used: %d\n", sched_getcpu());

    return 0;
}
```

After compiling and running this program, I observed that my code was working on a baseline program, since the initial CPU set was 4 as well:

```
omeed26@node0:~$ ./lock_test
0123456789
0123456789
Basic Operations Complete.
CPU that was used: 4
```

2. Next, we need to flush the L1 data cache before enabling perf events. This is again to minimize external factors when we are doing performance measurement with perf, especially because of how frequently L1 cache is accessed by the CPU. The best technique here is to allocate a buffer that

is larger than the cache (to ensure full range of eviction), and then write random data to it. From the earlier part of the assignment, I know that my LEVEL1_ICACHE_SIZE is 32768 bytes, or 32 KB. I will allocate a 64 KB buffer, write the random data, and then perform the measurement. Here is my code below:

```
static int flush_the_cache() {
    size_t kb_to_flush = 64 * 1024;
    char* buffer = (char*) malloc(kb_to_flush);
    if (buffer == nullptr) {
        printf("Failed to allocate for cache flush.\n");
        return EXIT_FAILURE;
    }
    for (size_t i = 0; i < kb_to_flush; i++) {
        buffer[i] = (char) i;
    }
    free(buffer);
    return EXIT_SUCCESS;
}
```

3. Now that I've done the intermediary steps, I can modify the first experimental function to support these operations, and begin the low-level experimentation. This component of the lab was one of the most time consuming, and required a depth of steps to be able to obtain actual performance metrics. The code that I wrote is **very** long, so I will sequentially describe the steps that I took to accomplish this part of the lab. Every step here has `perror` and `RETURN_EXIT` setup, with success console outputs to ensure that every step is working prior to the `ioctl` and `read`.

   (a) First, I added the `perf_event_open()` function wrapper with the system call to my `do_mem_access` file with all the provided code and global constants/values. Additionally, I declare a new struct to serve as a reading format for my performance counter group leaders, this was inspired by a very useful stack overflow article. This is what it looks like:

```
struct read_format {
  uint64_t nr;
  struct {
    uint64_t value;
    uint64_t id;
  } values[];
};
```

   (b) Next, I started by locking my main thread to CPU ID 4.

   (c) I declared a loop that runs through 5 trials, and immediately called upon my `flush_the_cache()` function, since we want a clean slate for performance counting on every trial.

   (d) Next, I allocate a memory pointer for accessing, and cast a buffer to the read format struct to enable our buffer to be used.

   (e) Next, I begin to follow an "event leader" paradigm. The core bottleneck here was that when you are doing event creation and `ioctl`, there is a limit on the number of events that a file descriptor leader can handle concurrently, which is likely due to hardware limitations and other system configurations. I found that the max amount of events a leader can handle

including its own is 4. Additionally, the linux kernel has settings in the `perf_event_paranoid` file that controls security restrictions for non-root users accessing performance data. In my system, it was set to 4 (which I found by doing `cat /proc/sys/kernel/perf_event_paranoid`. This means that its at a **very** high level of paranoia. What I did to ensure I could access all the necessary performance data for my events, was to manually set the value to `-1` on my current machine instance (since it resets on reboot). I did this command: `sudo sh -c 'echo -1 > /proc/sys/kernel/perf_event_paranoid'`. This is actually what made my code work for prefetching metrics.

(f) Next, I set up the leader events and child events using a very similar structure to the examples in the Linux manual. As per my `perf list` usage, I knew there are a total of 10 events to handle. Here is a skeleton of all events below:

```
// <Event Type>
int fd;
uint64_t id;
long long count;
struct perf_event_attr pe;
memset(&pe, 0, sizeof(pe));
pe.type = PERF_TYPE_HW_CACHE; // measuring hardware CPU cache event.
pe.size = sizeof(pe);
pe.config = (PERF_COUNT_HW_CACHE_L1D |
(PERF_COUNT_HW_CACHE_OP_READ << 8) |
(PERF_COUNT_HW_CACHE_RESULT_MISS << 16));
pe.disabled = 0;
pe.exclude_kernel = 1;
pe.exclude_hv = 1;
pe.read_format = PERF_FORMAT_GROUP | PERF_FORMAT_ID;
fd = perf_event_open(&pe, 0, -1, fd_leader, 0);
ioctl(fd, PERF_EVENT_IOC_ID, &id);
if (fd == -1) {
    perror("Perf Event Open Failed. <Event Type> \n");
    return EXIT_FAILURE;
} else {
    printf("Perf Event Open Successful.\n");
    printf("<Event Type>\n");
    printf("-----------------------\n");
}
```

One of the common things that changed across each event type setup was the configuration. Once I set the type, the documentation was quite trivial to map to the events off of my `perf list`. For example, `L1-dcache-load-misses` is equivalent to `pe.config = (PERF_COUNT_HW_CACHE_L1D | (PERF_COUNT_HW_CACHE_OP_READ << 8) |(PERF_COUNT_HW_CACHE_RESULT_MISS << 16))`. Files like `builtin-stat.c` in the Linux source code also helped a lot in setting up these configurations. All leaders are initially disabled with `pe.disabled = 1`, but all child events controlled by that leader are `pe.disabled = 0`. This is because we want the leader to control the entire `RESET` and `ENABLE` flow when we leverage `iocttl`. Additionally, the `ioctl(fd, PERF_EVENT_IOC_ID, id)` line exists in every event setup, which is useful for identifying the event in later performance analysis, since that pointer will store the id of the perf event.

(g) Now, since all events are setup, we can leverage the `ioctl` feature to begin and stop the performance counting. The respective code is below:

```
// begin i/o control, leader controls all flow.
ioctl(fd_leader, PERF_EVENT_IOC_RESET, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_2, PERF_EVENT_IOC_RESET, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_3, PERF_EVENT_IOC_RESET, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_4, PERF_EVENT_IOC_RESET, PERF_IOC_FLAG_GROUP);

ioctl(fd_leader, PERF_EVENT_IOC_ENABLE, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_2, PERF_EVENT_IOC_ENABLE, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_3, PERF_EVENT_IOC_ENABLE, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_4, PERF_EVENT_IOC_ENABLE, PERF_IOC_FLAG_GROUP);

do_mem_access(p, MEM_SIZE);

ioctl(fd_leader, PERF_EVENT_IOC_DISABLE, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_2, PERF_EVENT_IOC_DISABLE, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_3, PERF_EVENT_IOC_DISABLE, PERF_IOC_FLAG_GROUP);
ioctl(fd_leader_4, PERF_EVENT_IOC_DISABLE, PERF_IOC_FLAG_GROUP);
```

Each leader here is controlling different things. `fd_leader` is controlling L1-dcache-load-misses, L1-dcache-loads and L1-dcache-prefetch-misses. `fd_leader_2` is controlling L1-dcache-prefetches, L1-dcache-store-misses and L1-dcache-stores. `fd_leader_3` is controlling dTLB-load-misses, dTLB-loads and dTLB-store-misses. And finally, `fd_leader_4` controls dTLB-stores.

(h) Next, I declare `uint64_t` variables up to 10. This type seemed common across many forums, likely because 64 bits ensures accuracy, better for performance, etc.

(i) Finally, I read from each leader file descriptor into our buffer declared earlier, iterate through an array and assign the counts based on matching ids:

```
read(fd_leader, buf, sizeof(buf));
for (int i = 0; i < rf->nr; i++) {
    if (rf->values[i].id == id) {
        val1 = rf->values[i].value;
    } else if (rf->values[i].id == id_2) {
        val2 = rf->values[i].value;
    } else if (rf->values[i].id == id_3) {
        val3 = rf->values[i].value;
    }
}

printf(FORMATTING + VALUE, using PRIu64)
```

(j) I finally print all the values and place them into an excel file to take the average, standard deviation and other analyses. This is a general idea of what the function looks like For `getrusage`, I simply reused the code that I explained in an earlier `getrusage` section. I placed one print sequence right before the enabling of `ioctl`, and one after disabling. I did not have time to generalize everything down to functions, so I have two files currently: (1) a memory access malloc file (2) a memory access mmap file, that has functions for all the different mmap data collections. For MMAP, an example signature in multiple functions was `char* p = (char*) mmap(NULL, MEM_SIZE, PROT_READ | PROT_WRITE,`

`MAP_PRIVATE, fd, 0)`. The idea was to play around with the different definitions from `mman.h` to achieve the desired functionality. For file descriptors, an important component to avoid segmentation faulting was truncation to our desired 1GB size. Error checking was also very important! Additionally, for mmap, there was a new way I had to deallocate:

```
// deallocate memory pointer.
if (munmap(p, MEM_SIZE) == -1) {
    perror("Oh no. Memory Deallocation Failed.");
    return EXIT_FAILURE;
} else {
    printf("Memory Deallocation Successful.\n");
}
```

Overall, the majority of the code for this entire section can be found in my zip file submission.

1 Perf Event and Getrusage Results for MALLOC and MMAP

In this section, I report all performance event and getrusage results for basic malloc and mmap for the `do_mem_access()` function. I do it for random and sequential memory patterns. The charts were created in Google Sheets by transferring experimental data over from my machine.

1.1 Sequential Access Patterns (opt_random_access = 0)

Task Name	do_mem_access_malloc_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000002125	36.00032182	72.00056505	108.0008246	145.0000661	72.20035595	57.23819712
Utime (tv_usec) - before	2125	321815	565053	824646	66115	355950.8	344151.7329
Stime (tv_sec) - before	0	0.000447973	0.000927911	1.000415852	1.000899818	0.4005383108	0.5478317927
Stime (tv_usec) - before	0	447973	927911	415852	899818	538310.8	385814.5165
Maxrss - before	1912	1049984	1049984	1049984	1049984	840369.6	468712.0475
Minflt - before	107	262247	524383	786519	1048655	524382.2	414474.6733
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	1	74	167	256	347	169	138.29859
Utime (tv_sec) - after	0.000002125	36.00032182	72.00056505	108.0008246	145.0000661	72.20035595	57.23819712
Utime (tv_usec) - after	2125	321815	565053	824646	66115	355950.8	344151.7329
Stime (tv_sec) - after	0	0.000447973	0.000927911	1.000415852	1.000899818	0.4005383108	0.5478317927
Stime (tv_usec) - after	0	447973	927911	415852	899818	538310.8	385814.5165
Maxrss - after	1912	1049984	1049984	1049984	1049984	840369.6	468712.0475
Minflt - after	107	262247	524383	786519	1048655	524382.2	414474.6733
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	1	74	167	256	347	169	138.29859
L1 HW Cache Read Misses	202141992	202280091	202348750	202447977	202301512	202304064.4	111306.2675
L1 HW Cache Read Accesses	16907834302	16908927904	16947975676	16948858850	16907479484	16924215243	22101974.65
L1 HW Cache Write Misses	2775116	2759675	2770053	2754508	2781731	2768216.6	11118.65254
L1 HW Cache Write Accesses	6456332505	6456528942	6456814868	6457137853	6457816733	6456926180	583208.4748
L1 HW Cache Prefetch Misses	235026360	234931666	234923207	235410630	234845466	235027465.8	223599.7838
L1 HW Cache Prefetch Accesses	1831304682	1831408038	1831432736	1831550875	1831742037	1831487674	166988.8099
Data TLB Load Misses	163575	164007	163280	163301	164030	163638.6	365.8979366
Data TLB Load Accesses	23089683466	23089142456	23041558816	23029959131	23091335180	23068335810	30030762.78
Data TLB Store Misses	2705544	2706358	2700831	2699172	2702351	2702851.2	3058.461165
Data TLB Store Accesses	6626395260	6625669456	6624204322	6626508668	6624855952	6625526732	992172.2824

Task Name	do_mem_access_mmap_private_anon_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000002241	36.00019506	72.00035754	108.0005275	144.0007245	72.00036136	56.92127884
Utime (tv_usec) - before	2241	195056	357544	527489	724490	361364	281113.9804
Stime (tv_sec) - before	0	0.000472039	0.00094402	1.000391993	1.000847955	0.4005312014	0.5478037208
Stime (tv_usec) - before	0	472039	944020	391993	847955	531201.4	379409.454
Maxrss - before	1908	1049984	1049984	1049984	1049984	840368.8	468713.8363
Minflt - before	107	262246	524382	786518	1048654	524381.4	414474.357
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	1	36	84	126	168	83	67.09694479
Utime (tv_sec) - after	0.000002241	36.00019506	72.00035754	108.0005275	144.0007245	72.00036136	56.92127884
Utime (tv_usec) - after	2241	195056	357544	527489	724490	361364	281113.9804
Stime (tv_sec) - after	0	0.000472039	0.00094402	1.000391993	1.000847955	0.4005312014	0.5478037208
Stime (tv_usec) - after	0	472039	944020	391993	847955	531201.4	379409.454
Maxrss - after	1908	1049984	1049984	1049984	1049984	840368.8	468713.8363
Minflt - after	107	262246	524382	786518	1048654	524381.4	414474.357
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	1	36	84	126	168	83	67.09694479
L1 HW Cache Read Misses	202976591	202336114	201859759	201732796	201361312	202053314.4	622791.7696
L1 HW Cache Read Accesses	16907804762	16945893543	16937022218	16900813717	16900962903	16918499429	21378851.99
L1 HW Cache Write Misses	2806295	2770061	2721273	2765927	2753058	2763322.8	30709.71739
L1 HW Cache Write Accesses	6454231307	6458898995	6460763442	6457930497	6457939786	6457952805	2379347.094
L1 HW Cache Prefetch Misses	220868223	220479018	219223167	220416585	220864046	220370207.8	674831.1656
L1 HW Cache Prefetch Accesses	1833994797	1835521511	1836123722	1835270783	1835326495	1835247462	777639.7187
Data TLB Load Misses	170105	170001	170364	169910	170726	170221.2	329.4126591
Data TLB Load Accesses	23083344560	23023187009	23065686754	23088867285	23093671351	23070951392	28722461.02
Data TLB Store Misses	2645790	2641587	2646157	2648106	2648792	2646086.4	2816.757586
Data TLB Store Accesses	6633675260	6631141666	6618001729	6631130760	6631250812	6629040045	6265082.381

Task Name	do_mem_access_mmap_private_file_backed_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0	36.00028662	72.00053587	108.0007358	145.0000043	72.20031252	57.23816983
Utime (tv_usec) - before	0	286618	535867	735823	4282	312518	324969.0912
Stime (tv_sec) - before	0.000002097	0.000675974	1.00036881	2.000163837	2.00084377	1.000410898	1.000082442
Stime (tv_usec) - before	2097	675974	368810	163837	843770	410897.6	349131.8695
Maxrss - before	1908	1050112	1050112	1050112	1050112	840471.2	468771.0797
Minflt - before	108	262247	524383	786519	1048655	524382.4	414474.357
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	0	67	145	225	296	146.6	118.635155
Utime (tv_sec) - after	0	36.00028662	72.00053587	108.0007358	145.0000043	72.20031252	57.23816983
Utime (tv_usec) - after	0	286618	535867	735823	4282	312518	324969.0912
Stime (tv_sec) - after	0.000002097	0.000675974	1.00036881	2.000163837	2.00084377	1.000410898	1.000082442
Stime (tv_usec) - after	2097	675974	368810	163837	843770	410897.6	349131.8695
Maxrss - after	1908	1050112	1050112	1050112	1050112	840471.2	468771.0797
Minflt - after	108	262247	524383	786519	1048655	524382.4	414474.357
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	0	67	145	225	296	146.6	118.635155
L1 HW Cache Read Misses	202559134	202448189	203358500	200995732	201721679	202216646.8	895874.4436
L1 HW Cache Read Accesses	16948223647	16942632678	16939953647	16901969181	16944811809	16935518192	18997534.15
L1 HW Cache Write Misses	2799023	2778908	2817028	2744273	2761692	2780184.8	28920.45303
L1 HW Cache Write Accesses	6457812180	6458853593	6445203202	6457240679	6457830546	6455388040	5723101.088
L1 HW Cache Prefetch Misses	221254240	221079925	221879873	220409065	220562846	221037189.8	587067.8979
L1 HW Cache Prefetch Accesses	1834757548	1835120006	1831021369	1834791899	1834912865	1834120737	1738395.38
Data TLB Load Misses	170290	170040	171199	170437	170244	170442	446.3703619
Data TLB Load Accesses	23067937190	23072766384	23113539834	23128217877	23070064660	23090505189	28260558.71
Data TLB Store Misses	2645855	2647401	2653690	2651440	2643553	2648387.8	4128.964483
Data TLB Store Accesses	6651434208	6650714424	6654961793	6652721329	6651060350	6652178421	1731266.282

Task Name	mmap_private_file_backed_populate_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0	36.00019661	72.00035915	108.0005928	144.0009338	72.00041647	56.92135582
Utime (tv_usec) - before	0	196607	359147	592804	933807	416473	361871.5853
Stime (tv_sec) - before	0.000508912	1.000076018	1.00064398	2.000210196	2.000779881	1.200443797	0.836671244
Stime (tv_usec) - before	508912	76018	643980	210196	779881	443797.4	294574.4693
Maxrss - before	1049728	1050064	1050064	1050064	1050064	1049996.8	150.2637681
Minflt - before	262252	524400	786544	1048688	1310832	786543.2	414487.3224
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	2	78	146	201	261	137.6	101.5593423
Utime (tv_sec) - after	0	36.00019661	72.00035915	108.0005928	144.0009338	72.00041647	56.92135582
Utime (tv_usec) - after	0	196607	359147	592804	933807	416473	361871.5853
Stime (tv_sec) - after	0.000508912	1.000076018	1.00064398	2.000210196	2.000779881	1.200443797	0.836671244
Stime (tv_usec) - after	508912	76018	643980	210196	779881	443797.4	294574.4693
Maxrss - after	1049728	1050064	1050064	1050064	1050064	1049996.8	150.2637681
Minflt - after	262252	524400	786544	1048688	1310832	786543.2	414487.3224
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	2	78	146	201	261	137.6	101.5593423
L1 HW Cache Read Misses	171735038	171548924	172071987	171345251	171871577	171714555.4	281359.2761
L1 HW Cache Read Accesses	16942441116	16942634532	16932571755	16945585287	16944593067	16941565151	5199082.39
L1 HW Cache Write Misses	2703874	2698375	2716212	2714033	2693813	2705261.4	9711.717835
L1 HW Cache Write Accesses	6444124488	6458642539	6460339483	6458379067	6458855822	6456068280	6720012.626
L1 HW Cache Prefetch Misses	217418196	214542948	218896956	215776103	218669968	217060834.2	1876076.19
L1 HW Cache Prefetch Accesses	1831481863	1835611392	1836037667	1835397524	1835693707	1834844431	1893811.4
Data TLB Load Misses	26297	26184	26662	26115	26668	26385.2	263.5634648
Data TLB Load Accesses	23053212536	22980257488	23059771331	23011653266	23006159683	23022210861	33545049.53
Data TLB Store Misses	2128558	2124982	2128735	2125278	2124452	2126401	2072.045849
Data TLB Store Accesses	6689734957	6683289506	6677730692	6682875005	6687019060	6684129844	4555073.985

Task Name	mmap_shared_file_backed_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.00000227	37.00072192	75.0009113	114.0001159	152.0004671	75.60044369	60.24371979
Utime (tv_usec) - before	2270	721922	911299	115879	467067	443687.4	386952.6376
Stime (tv_sec) - before	0	1.000979051	4.000560525	7.000011746	9.000750661	4.200460397	3.834083423
Stime (tv_usec) - before	0	979051	560525	11746	750661	460396.6	440605.8032
Maxrss - before	1908	1049856	1049860	1049860	1049860	840268.8	468657.9346
Minflt - before	107	1176678	2655326	4133974	5707044	2734625.8	2275111.438
Majflt - before	0	0	0	0	2	0.4	0.894427191
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	9412552	19144664	28876768	39846824	19456161.6	15684708.21
Voluntary - before	1	88	195	314	462	212	182.4075108
Involuntary - before	1	108	225	338	453	225	179.3167588
Utime (tv_sec) - after	0.00000227	37.00072192	75.0009113	114.0001159	152.0004671	75.60044369	60.24371979
Utime (tv_usec) - after	2270	721922	911299	115879	467067	443687.4	386952.6376
Stime (tv_sec) - after	0	1.000979051	4.000560525	7.000011746	9.000750661	4.200460397	3.834083423
Stime (tv_usec) - after	0	979051	560525	11746	750661	460396.6	440605.8032
Maxrss - after	1908	1049856	1049860	1049860	1049860	840268.8	468657.9346
Minflt - after	107	1176678	2655326	4133974	5707044	2734625.8	2275111.438
Majflt - after	0	0	0	0	2	0.4	0.894427191
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	9412552	19144664	28876768	39846824	19456161.6	15684708.21
Voluntary - after	1	88	195	314	462	212	182.4075108
Involuntary - after	1	108	225	338	453	225	179.3167588
L1 HW Cache Read Misses	234259992	250471915	246619767	242024295	239462158	242567625.4	6281989.216
L1 HW Cache Read Accesses	16914733659	16933350302	16913228325	16940113268	16957830783	16931851267	18609604.31
L1 HW Cache Write Misses	3716662	4289152	4137132	3928051	3841005	3982400.4	230000.9923
L1 HW Cache Write Accesses	6452252221	6461963970	6462595296	6451007035	6458222475	6457208199	5377803.469
L1 HW Cache Prefetch Misses	247253316	260956121	258185754	255146670	253192114	254946795	5218029.535
L1 HW Cache Prefetch Accesses	1831411919	1832054066	1832892551	1830536719	1832760692	1831931189	979997.295
Data TLB Load Misses	770891	944058	937554	1027931	990403	934167.4	98390.20685
Data TLB Load Accesses	23170769820	23152481811	23164228496	23148831864	23110010705	23149264539	23652688.27
Data TLB Store Misses	6245263	7416835	7348652	7805755	7562973	7275895.6	602188.7955
Data TLB Store Accesses	6704693363	6703810863	6706190589	6716671574	6708205167	6707914311	5171031.365

Task Name	mmap_shared_file_backed_populate_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000003952	37.00018011	75.00043939	113.0006186	151.0002195	75.20029231	59.76802137
Utime (tv_usec) - before	3952	180112	439390	618631	219486	292314.2	239353.2868
Stime (tv_sec) - before	0.000079059	1.000538094	3.000810549	6.000736393	9.000031013	3.800439022	3.701305585
Stime (tv_usec) - before	79059	538094	810549	736393	31013	439021.6	364801.9291
Maxrss - before	1049856	1050164	1050164	1050164	1050164	1050102.4	137.7417874
Minflt - before	16493	903273	2398305	3987762	5335340	2528234.6	2178089.112
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	6963144	16695256	27264288	36159464	17416430.4	14674850.47
Voluntary - before	1	66	185	325	418	199	173.8720794
Involuntary - before	0	120	265	391	508	256.8	203.6165514
Utime (tv_sec) - after	0.000003952	37.00018011	75.00043939	113.0006186	151.0002195	75.20029231	59.76802137
Utime (tv_usec) - after	3952	180112	439390	618631	219486	292314.2	239353.2868
Stime (tv_sec) - after	0.000079059	1.000538094	3.000810549	6.000736393	9.000031013	3.800439022	3.701305585
Stime (tv_usec) - after	79059	538094	810549	736393	31013	439021.6	364801.9291
Maxrss - after	1049856	1050164	1050164	1050164	1050164	1050102.4	137.7417874
Minflt - after	16493	903273	2398305	3987762	5335340	2528234.6	2178089.112
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	6963144	16695256	27264288	36159464	17416430.4	14674850.47
Voluntary - after	1	66	185	325	418	199	173.8720794
Involuntary - after	0	120	265	391	508	256.8	203.6165514
L1 HW Cache Read Misses	223493914	240697313	250905350	236796159	237981451	237974837.4	9819578.638
L1 HW Cache Read Accesses	16916605698	16946967912	16961233586	16903293547	16912831518	16928186452	24669824.11
L1 HW Cache Write Misses	3302748	3809581	4189605	3708344	3731452	3748346	315793.5936
L1 HW Cache Write Accesses	6455474494	6447378009	6454556029	6459605202	6454765448	6454355836	4407526.744
L1 HW Cache Prefetch Misses	237720695	252879955	258588556	249451395	249983419	249724804	7628259.048
L1 HW Cache Prefetch Accesses	1833036835	1829402488	1830392150	1833265845	1831834257	1831586315	1671932.635
Data TLB Load Misses	501327	897850	948252	821941	819808	797835.6	174362.5391
Data TLB Load Accesses	23100676079	23108540933	23096020493	23132249764	23152856091	23118068672	23930473.2
Data TLB Store Misses	5096439	7505050	7909452	7034908	7062804	6921730.6	1081694.89
Data TLB Store Accesses	6673433257	6691472816	6686655296	6685624305	6680613719	6683559879	6849933.667

Task Name	mmap_private_file_backed_memset_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000080021	36.00011072	72.00012801	108.0001646	144.0001999	72.00013665	56.92104431
Utime (tv_usec) - before	80021	110718	128010	164612	199879	136648	46727.24508
Stime (tv_sec) - before	0.000608161	1.000279954	1.000939892	2.000587823	3.000239727	1.400531111	1.140043213
Stime (tv_usec) - before	608161	279954	939892	587823	239727	531111.4	284710.0865
Maxrss - before	1049728	1050060	1050060	1050060	1050060	1049993.6	148.4749137
Minflt - before	262252	524399	786543	1048687	1310831	786542.4	414487.0062
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	3	77	168	263	358	173.8	141.8298276
Utime (tv_sec) - after	0.000080021	36.00011072	72.00012801	108.0001646	144.0001999	72.00013665	56.92104431
Utime (tv_usec) - after	80021	110718	128010	164612	199879	136648	46727.24508
Stime (tv_sec) - after	0.000608161	1.000279954	1.000939892	2.000587823	3.000239727	1.400531111	1.140043213
Stime (tv_usec) - after	608161	279954	939892	587823	239727	531111.4	284710.0865
Maxrss - after	1049728	1050060	1050060	1050060	1050060	1049993.6	148.4749137
Minflt - after	262252	524399	786543	1048687	1310831	786542.4	414487.0062
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	3	77	168	263	358	173.8	141.8298276
L1 HW Cache Read Misses	205024197	203560141	203440260	203324664	202942552	203658362.8	797881.4771
L1 HW Cache Read Accesses	16942005284	16942687199	16932786170	16935322584	16946471596	16939854567	5633284.267
L1 HW Cache Write Misses	2790088	2732381	2734602	2723812	2715295	2739235.6	29434.08798
L1 HW Cache Write Accesses	6442251279	6443947241	6460524791	6459278049	6457537342	6452707740	8855566.35
L1 HW Cache Prefetch Misses	249201803	249026735	249009420	249020136	248669341	248985487	193788.3152
L1 HW Cache Prefetch Accesses	1830510850	1831158141	1835920188	1835572113	1835087316	1833649722	2597006.586
Data TLB Load Misses	26862	26073	26763	26655	26870	26644.6	331.2556415
Data TLB Load Accesses	22956977991	22952160318	22958914709	22963434678	22905957455	22947489030	23566622.93
Data TLB Store Misses	2127399	2125495	2126545	2126771	2121478	2125537.6	2370.817749
Data TLB Store Accesses	6669780105	6668136096	6653379397	6650496583	6665917411	6661541918	8931993.365

1.2 Random Access Patterns (opt_random_access = 1)

Task Name	do_mem_access_malloc_1.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0	36.00040251	72.00091302	109.0003812	145.0008827	72.40051587	57.39626737
Utime (tv_usec) - before	0	402512	913015	381158	882665	515870	383856.073
Stime (tv_sec) - before	0.000002294	0.000799879	1.000450718	2.000142991	2.000803125	1.000439801	1.000036053
Stime (tv_usec) - before	2294	799879	450718	142991	803125	439801.4	367861.4641
Maxrss - before	1908	1049984	1050064	1050064	1050064	840416.8	468740.6704
Minflt - before	108	466515	932809	1399507	1865791	932946	737499.7613
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	1	100	189	285	381	191.2	149.4329281
Utime (tv_sec) - after	0	36.00040251	72.00091302	109.0003812	145.0008827	72.40051587	57.39626737
Utime (tv_usec) - after	0	402512	913015	381158	882665	515870	383856.073
Stime (tv_sec) - after	0.000002294	0.000799879	1.000450718	2.000142991	2.000803125	1.000439801	1.000036053
Stime (tv_usec) - after	2294	799879	450718	142991	803125	439801.4	367861.4641
Maxrss - after	1908	1049984	1050064	1050064	1050064	840416.8	468740.6704
Minflt - after	108	466515	932809	1399507	1865791	932946	737499.7613
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	1	100	189	285	381	191.2	149.4329281
L1 HW Cache Read Misses	233108360	232864708	232650378	232815029	232352393	232758173.6	279951.0745
L1 HW Cache Read Accesses	16947287773	16937929981	16947705451	16934769489	16911588870	16935856313	14709439.35
L1 HW Cache Write Misses	6706286	6704247	6675343	6683682	6682387	6690389	13965.64304
L1 HW Cache Write Accesses	6455775677	6445484493	6456154330	6459937276	6456462515	6454762858	5447786.002
L1 HW Cache Prefetch Misses	236888358	236641069	236902327	237020498	236948268	236880104	143227.5529
L1 HW Cache Prefetch Accesses	1818005581	1815069877	1818183949	1819335008	1818302586	1817779400	1600778.909
Data TLB Load Misses	2204384	2215241	2204446	2214522	2208872	2209493	5250.954104
Data TLB Load Accesses	22972265467	23011240051	22974877357	23019464898	23023974814	23000364517	24898292.38
Data TLB Store Misses	1806976	1809871	1807154	1806195	1805592	1807157.6	1641.13019
Data TLB Store Accesses	6858919302	6862112935	6857896696	6845800337	6857299956	6856405845	6212632.564

Task Name	do_mem_access_mmap_private_anon_1.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.00000217	36.00038816	72.00081156	109.0001878	145.0005958	72.40039711	57.3961476
Utime (tv_usec) - before	2170	388163	811562	187827	595805	397105.4	320588.2054
Stime (tv_sec) - before	0	0.000750708	1.000447911	2.000183836	2.000935319	1.000463555	1.000092182
Stime (tv_usec) - before	0	750708	447911	183836	935319	463554.8	386777.9753
Maxrss - before	1908	1049984	1050060	1050060	1050060	840414.4	468739.3286
Minflt - before	106	466512	932806	1399504	1865788	932943.2	737499.4451
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	1	83	172	277	379	182.4	150.4121006
Utime (tv_sec) - after	0.00000217	36.00038816	72.00081156	109.0001878	145.0005958	72.40039711	57.3961476
Utime (tv_usec) - after	2170	388163	811562	187827	595805	397105.4	320588.2054
Stime (tv_sec) - after	0	0.000750708	1.000447911	2.000183836	2.000935319	1.000463555	1.000092182
Stime (tv_usec) - after	0	750708	447911	183836	935319	463554.8	386777.9753
Maxrss - after	1908	1049984	1050060	1050060	1050060	840414.4	468739.3286
Minflt - after	106	466512	932806	1399504	1865788	932943.2	737499.4451
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	1	83	172	277	379	182.4	150.4121006
L1 HW Cache Read Misses	234053420	233354594	232926279	233105520	232567627	233201488	555868.5092
L1 HW Cache Read Accesses	16937320923	16944046085	16943233784	16934855470	16908843671	16933659987	14406680.68
L1 HW Cache Write Misses	6516312	6492817	6493688	6501788	6478895	6496700	13708.44763
L1 HW Cache Write Accesses	6458575055	6456820413	6459099704	6460091943	6457310348	6458379493	1328837.355
L1 HW Cache Prefetch Misses	238078367	237088172	236875406	237141742	236802625	237197262.4	512508.2356
L1 HW Cache Prefetch Accesses	1818440201	1817943526	1818886070	1818927023	1818225325	1818484429	423931.5971
Data TLB Load Misses	2221414	2211219	2216066	2219503	2215019	2216644.2	3979.07385
Data TLB Load Accesses	23026310074	22977488307	22976131963	23031743479	23029873622	23008309489	28825003.12
Data TLB Store Misses	1761939	1751010	1768992	1762967	1767952	1762572	7148.380201
Data TLB Store Accesses	6855493573	6867955093	6866672310	6852835248	6864785697	6861548384	6898554.351

Task Name	do_mem_access_mmap_private_file_backed_1.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000002173	36.00030563	72.00064726	108.0009747	145.0003042	72.2004468	57.23829877
Utime (tv_usec) - before	2173	305632	647263	974704	304218	446798	373101.6107
Stime (tv_sec) - before	0	0.00084762	1.000615635	2.000431971	3.000303606	1.200439766	1.303792929
Stime (tv_usec) - before	0	847620	615635	431971	303606	439766.4	319870.768
Maxrss - before	1908	1050112	1050188	1050188	1050188	840516.8	468796.572
Minflt - before	107	278617	557114	835615	1114120	557114.6	440350.9589
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	1	49	102	169	254	115	99.69704108
Utime (tv_sec) - after	0.000002173	36.00030563	72.00064726	108.0009747	145.0003042	72.2004468	57.23829877
Utime (tv_usec) - after	2173	305632	647263	974704	304218	446798	373101.6107
Stime (tv_sec) - after	0	0.00084762	1.000615635	2.000431971	3.000303606	1.200439766	1.303792929
Stime (tv_usec) - after	0	847620	615635	431971	303606	439766.4	319870.768
Maxrss - after	1908	1050112	1050188	1050188	1050188	840516.8	468796.572
Minflt - after	107	278617	557114	835615	1114120	557114.6	440350.9589
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	1	49	102	169	254	115	99.69704108
L1 HW Cache Read Misses	204284403	203627042	203378974	203580561	203195288	203613253.6	412699.2298
L1 HW Cache Read Accesses	16935588955	16939795887	16937229746	16934805744	16941454763	16937775019	2807009.791
L1 HW Cache Write Misses	7601178	7597638	7579224	7583119	7572749	7586781.6	12171.31436
L1 HW Cache Write Accesses	6444879602	6444331473	6462104908	6445536478	6460236000	6451417692	8937686.229
L1 HW Cache Prefetch Misses	234185997	233850593	234255628	233501954	234260297	234010893.8	330709.0798
L1 HW Cache Prefetch Accesses	1815972810	1815975667	1821033148	1816297484	1820419655	1817939753	2556489.419
Data TLB Load Misses	2028612	2033160	2033093	2034156	2031929	2032190	2150.097091
Data TLB Load Accesses	22973781218	22970535940	22978111826	22982047726	22971617440	22975218830	4795035.02
Data TLB Store Misses	1457917	1454832	1451431	1459012	1454764	1455591.2	2986.600358
Data TLB Store Accesses	6872982197	6873441977	6856771945	6874880320	6854195430	6866454374	10080510.58

Task Name	mmap_private_file_backed_populate_1.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0	36.00019661	72.00035915	108.0005928	144.0009338	72.00041647	56.92135582
Utime (tv_usec) - before	0	196607	359147	592804	933807	416473	361871.5853
Stime (tv_sec) - before	0.000508912	1.000076018	1.00064398	2.000210196	2.000779881	1.200443797	0.836671244
Stime (tv_usec) - before	508912	76018	643980	210196	779881	443797.4	294574.4693
Maxrss - before	1049728	1050064	1050064	1050064	1050064	1049996.8	150.2637681
Minflt - before	262252	524400	786544	1048688	1310832	786543.2	414487.3224
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	2	78	146	201	261	137.6	101.5593423
Utime (tv_sec) - after	0	36.00019661	72.00035915	108.0005928	144.0009338	72.00041647	56.92135582
Utime (tv_usec) - after	0	196607	359147	592804	933807	416473	361871.5853
Stime (tv_sec) - after	0.000508912	1.000076018	1.00064398	2.000210196	2.000779881	1.200443797	0.836671244
Stime (tv_usec) - after	508912	76018	643980	210196	779881	443797.4	294574.4693
Maxrss - after	1049728	1050064	1050064	1050064	1050064	1049996.8	150.2637681
Minflt - after	262252	524400	786544	1048688	1310832	786543.2	414487.3224
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	2	78	146	201	261	137.6	101.5593423
L1 HW Cache Read Misses	171735038	171548924	172071987	171345251	171871577	171714555.4	281359.2761
L1 HW Cache Read Accesses	16942441116	16942634532	16932571755	16945585287	16944593067	16941565151	5199082.39
L1 HW Cache Write Misses	2703874	2698375	2716212	2714033	2693813	2705261.4	9711.717835
L1 HW Cache Write Accesses	6444124488	6458642539	6460339483	6458379067	6458855822	6456068280	6720012.626
L1 HW Cache Prefetch Misses	217418196	214542948	218896956	215776103	218669968	217060834.2	1876076.19
L1 HW Cache Prefetch Accesses	1831481863	1835611392	1836037667	1835397524	1835693707	1834844431	1893811.4
Data TLB Load Misses	26297	26184	26662	26115	26668	26385.2	263.5634648
Data TLB Load Accesses	23053212536	22980257488	23059771331	23011653266	23006159683	23022210861	33545049.53
Data TLB Store Misses	2128558	2124982	2128735	2125278	2124452	2126401	2072.045849
Data TLB Store Accesses	6689734957	6683289506	6677730692	6682875005	6687019060	6684129844	4555073.985

Task Name	mmap_shared_file_backed_1.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000002241	37.00021625	74.00065599	112.0001838	149.0006879	74.40034924	58.97732493
Utime (tv_usec) - before	2241	216252	655989	183795	687900	349235.4	305880.5758
Stime (tv_sec) - before	0	1.000615663	3.000359099	5.000194323	7.000130089	3.200259835	2.863513397
Stime (tv_usec) - before	0	615663	359099	194323	130089	259834.8	237227.2018
Maxrss - before	1912	1049984	1049988	1049988	1049988	840372	468713.3891
Minflt - before	106	923836	2012860	3161991	4334328	2086624.2	1726243.167
Majflt - before	0	0	1	1	1	0.6	0.5477225575
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	7258920	14390312	21681816	28962560	14458721.6	11439305.86
Voluntary - before	1	22	43	62	84	42.4	32.57759967
Involuntary - before	1	112	250	394	537	258.8	214.3378175
Utime (tv_sec) - after	0.000002241	37.00021625	74.00065599	112.0001838	149.0006879	74.40034924	58.97732493
Utime (tv_usec) - after	2241	216252	655989	183795	687900	349235.4	305880.5758
Stime (tv_sec) - after	0	1.000615663	3.000359099	5.000194323	7.000130089	3.200259835	2.863513397
Stime (tv_usec) - after	0	615663	359099	194323	130089	259834.8	237227.2018
Maxrss - after	1912	1049984	1049988	1049988	1049988	840372	468713.3891
Minflt - after	106	923836	2012860	3161991	4334328	2086624.2	1726243.167
Majflt - after	0	0	1	1	1	0.6	0.5477225575
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	7258920	14390312	21681816	28962560	14458721.6	11439305.86
Voluntary - after	1	22	43	62	84	42.4	32.57759967
Involuntary - after	1	112	250	394	537	258.8	214.3378175
L1 HW Cache Read Misses	259297137	262306613	265888164	266544963	265488846	263905144.6	3049514.493
L1 HW Cache Read Accesses	16920118927	16935483113	16908359787	16908421054	16942461843	16922968945	15570554.79
L1 HW Cache Write Misses	7743746	7763128	7756221	7758463	7842443	7772800.2	39586.11739
L1 HW Cache Write Accesses	6464666470	6454519960	6459376392	6454325592	6452505084	6457078700	4947179.991
L1 HW Cache Prefetch Misses	188409512	192234257	193721451	194542462	195837922	192949120.8	2854797.751
L1 HW Cache Prefetch Accesses	1819507894	1815946841	1817124961	1815704207	1814876763	1816632133	1797559.206
Data TLB Load Misses	3443629	3766933	3852714	3856476	3920708	3768092	189434.5794
Data TLB Load Accesses	23011911825	23026562548	23063207106	23077404989	23033724702	23042562234	26989663.82
Data TLB Store Misses	4076597	4719376	4907782	4938908	5037569	4736046.4	386236.0796
Data TLB Store Accesses	6910679119	6917030182	6914252267	6915982538	6919723326	6915533486	3361655.356

Task Name	mmap_shared_file_backed_populate_1.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0	37.00047072	74.00071237	112.0001712	149.0005655	74.40038397	58.9772445
Utime (tv_usec) - before	0	470720	712373	171206	565549	383969.6	291995.9361
Stime (tv_sec) - before	0.0000816	1.000885837	3.000746815	5.000297715	6.000976446	3.000597683	2.549657683
Stime (tv_usec) - before	81600	885837	746815	297715	976446	597682.6	388927.0353
Maxrss - before	1049728	1049988	1049988	1049988	1049988	1049936	116.2755348
Minflt - before	16492	1152969	2238065	3284492	4362451	2210893.8	1711516.425
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8058848	14816728	21564784	28653656	14618803.2	11202737.79
Voluntary - before	1	20	40	57	80	39.6	30.85935839
Involuntary - before	1	140	270	415	563	277.8	221.2638696
Utime (tv_sec) - after	0	37.00047072	74.00071237	112.0001712	149.0005655	74.40038397	58.9772445
Utime (tv_usec) - after	0	470720	712373	171206	565549	383969.6	291995.9361
Stime (tv_sec) - after	0.0000816	1.000885837	3.000746815	5.000297715	6.000976446	3.000597683	2.549657683
Stime (tv_usec) - after	81600	885837	746815	297715	976446	597682.6	388927.0353
Maxrss - after	1049728	1049988	1049988	1049988	1049988	1049936	116.2755348
Minflt - after	16492	1152969	2238065	3284492	4362451	2210893.8	1711516.425
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8058848	14816728	21564784	28653656	14618803.2	11202737.79
Voluntary - after	1	20	40	57	80	39.6	30.85935839
Involuntary - after	1	140	270	415	563	277.8	221.2638696
L1 HW Cache Read Misses	267956364	264526738	263636412	264204775	263091187	264683095.2	1910316.98
L1 HW Cache Read Accesses	16902671751	16938764410	16926445756	16897271887	16941753563	16921381473	20458258.4
L1 HW Cache Write Misses	6777309	6739613	6720608	6719700	6726334	6736712.8	24046.73601
L1 HW Cache Write Accesses	6459760630	6451139643	6458462643	6459890580	6449162341	6455683167	5128798.533
L1 HW Cache Prefetch Misses	267231496	265277683	264452552	264827134	264316555	265221084	1184163.271
L1 HW Cache Prefetch Accesses	1816833241	1814594443	1816810901	1817195152	1814201154	1815926978	1411117.978
Data TLB Load Misses	3357025	3331979	3274797	3260546	3226794	3290228.2	53263.48292
Data TLB Load Accesses	23089517019	23041440984	23048216975	23048228249	23063400477	23058160741	19285708.16
Data TLB Store Misses	4890383	4797050	4633110	4668255	4558076	4709374.8	133087.7471
Data TLB Store Accesses	6920173347	6928267733	6917737442	6923281498	6920655360	6922023076	4007435.907

Task Name	mmap_private_file_backed_memset_1.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000088346	36.00047262	72.00086244	109.0001634	145.0005254	72.40042245	57.3960808
Utime (tv_usec) - before	88346	472623	862443	163438	525407	422451.4	310412.6338
Stime (tv_sec) - before	0.000602362	1.000264021	1.000895959	2.000575892	3.000239853	1.400515617	1.140048716
Stime (tv_usec) - before	602362	264021	895959	575892	239853	515617.4	271645.0527
Maxrss - before	1049728	1050064	1050064	1050064	1050064	1049996.8	150.2637681
Minflt - before	262250	524398	786542	1048686	1310830	786541.2	414487.3224
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	3	83	167	248	325	165.2	127.9265414
Utime (tv_sec) - after	0.000088346	36.00047262	72.00086244	109.0001634	145.0005254	72.40042245	57.3960808
Utime (tv_usec) - after	88346	472623	862443	163438	525407	422451.4	310412.6338
Stime (tv_sec) - after	0.000602362	1.000264021	1.000895959	2.000575892	3.000239853	1.400515617	1.140048716
Stime (tv_usec) - after	602362	264021	895959	575892	239853	515617.4	271645.0527
Maxrss - after	1049728	1050064	1050064	1050064	1050064	1049996.8	150.2637681
Minflt - after	262250	524398	786542	1048686	1310830	786541.2	414487.3224
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	3	83	167	248	325	165.2	127.9265414
L1 HW Cache Read Misses	232556181	232054533	233041853	233880685	232677243	232842099	679659.7214
L1 HW Cache Read Accesses	16942716026	16940347030	16938554748	16938069146	16941798934	16940297177	2006786.663
L1 HW Cache Write Misses	6511695	6521044	6539109	6557146	6534971	6532793	17470.46618
L1 HW Cache Write Accesses	6457582730	6445478378	6444493778	6460136467	6457359893	6453010249	7413995.675
L1 HW Cache Prefetch Misses	237295486	236314220	237108758	238400279	237232494	237270247.4	744832.3535
L1 HW Cache Prefetch Accesses	1818690020	1815216708	1815086956	1819220397	1818684176	1817379651	2045847.686
Data TLB Load Misses	1871503	1872626	1874446	1873028	1868287	1871978	2316.383712
Data TLB Load Accesses	23000587947	23004951369	23007403309	23006860877	22967606978	22997482096	16914128.9
Data TLB Store Misses	865491	865637	862266	866596	864667	864931.4	1639.830875
Data TLB Store Accesses	6852583693	6865988082	6869610059	6853133135	6865407712	6861344536	7914705.705

1.3 Why can your data TLB miss count be lower than the total number of data pages your application accesses?

The answer is likely due to TLB prefetching. Based on the hardware, the computer has the capability to predict the virtual addresses that the application will access and prefetch those entries. In a lot of the data from the tables, we can clearly see that the ratio of prefetch misses to prefetch accesses displays a significant majority of prefetches are successful. For example, just grabbing a basic entry out of one of the tables:

```
prefetch_misses = [236888358, 236641069, 236902327,
                   237020498, 236948268, 236880104]
prefetch_accesses = [1818005581, 1815069877, 1818183949,
                   1819335008, 1818302586, 1817779400]
```

If we calculate a fraction of prefetch misses to prefetch accesses, the average percent of success is approximately 87 percent, which is quite high and indicates the system is optimizing with prefetching.

1.4 General Table Analysis

Firstly, if one thing is clear, the results are quite stable. The numbers seem quite consistent, and generally in the ballpark of the `perf -e` command.

Firstly, we were supposed to calculate the miss rate for cache and TLB. We can do this simply by using the general formula (CACHE MISSES / CACHE ACCESSES) * 100:

do_mem_access_malloc_0	
L1 HW CACHE MISS RATE:	1.194274441
L1 HW CACHE WRITE MISS RATE:	0.04278945485
L1 HW CACHE PREFETCH MISS RATE:	12.00765632
DATA TLB LOAD MISS RATE:	0.0007378161269
DATA TLB STORE MISS RATE:	0.03991658493
do_mem_access_malloc_1	
L1 HW CACHE MISS RATE:	1.374351372
L1 HW CACHE WRITE MISS RATE:	0.1036504229
L1 HW CACHE PREFETCH MISS RATE:	13.03128993
DATA TLB LOAD MISS RATE:	0.009606339057
DATA TLB STORE MISS RATE:	0.02635721456
do_mem_access_mmap_private_anon_0	
L1 HW CACHE MISS RATE:	1.194038733
L1 HW CACHE WRITE MISS RATE:	0.04306766352
L1 HW CACHE PREFETCH MISS RATE:	12.051398
DATA TLB LOAD MISS RATE:	0.0007381475572
DATA TLB STORE MISS RATE:	0.03981233864

These rates make a lot of sense, and are quite in the ballpark of each other, especially the L1 HW Cache Miss Rate.

2 Running STRACE

For this section, I ran one of my scripts with strace and outputted everything to a file: `sudo strace -o do_mem_access_malloc_strace.txt ./do_mem_access_malloc`. Next, I ran `cat do_mem_access_malloc_strace.txt`, and got an extremely big output. Here are the outputs requested from us:

```
omeed26@node0:~$ cat do_mem_access_malloc_strace.txt
    ...
    arch_prctl(0x3001 /* ARCH_??? */, 0x7ffddf933590) = -1 EINVAL (Invalid argument)
    ...
    arch_prctl(ARCH_SET_FS, 0x7fd0b30d9740) = 0
    ...
    access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
```

`arch_prctl()` is a system call that sets the architecture specific process or thread state.
The system call related to `/etc/ld.so.preload` is the `access()` system call, which checks to see if the calling process can access a specific file path with some mode, so in this case, R_OK represents a read.
The `/etc/ld.so.preload` is a list of shared libraries to be loaded before all others.

These are in the stack trace because the process is attempting to configure the thread storage and work with the dynamic linker to execute the program successfully.

3 Effect of Background Activity

For this section, I decided to run a memory intensive background activity program on CPU ID 5. This was my code:

```
omeed26@node0:~$ cat background_activity.cpp
    #include <stdio.h>
    #include <stdlib.h>
    // allows us to use affinity and getcpu() to test.
    #include <sched.h>
    #include <unistd.h>
    #include <iostream>
    #include <numeric> // For std::accumulate

    int main() {
        int cpu_id = 5;
        pid_t pid = 0;
        cpu_set_t mask;
        CPU_ZERO(&mask);
        CPU_SET(cpu_id, &mask);
        size_t cpusetsize = sizeof(mask);
        if(sched_setaffinity(pid, cpusetsize, &mask) == -1) {
            perror("Oh no. CPU Set Operation Failed.");
            return EXIT_FAILURE;
        }

        while (true) {
            // allocate some memory
            size_t arraySize = 1000000;
            int* array = (int*) malloc(arraySize * sizeof(int));
            if (array == nullptr) {
                printf("Failed to allocate for cache flush.\n");
                return EXIT_FAILURE;
            }
            // fill the array with some values
            for (size_t i = 0; i < arraySize; i++) {
                array[i] = i;
            }
            // iterate over the array and sum the values
            for (size_t i = 0; i < arraySize; i++) {
                array[i] = array[i] + 1;
            }
            // free the memory
            free(array);

        }
    }
```

After this began running, I SSHed with another terminal and ran `sudo ./do_mem_access_malloc >> do_mem_access_malloc_background.txt`. After formatting, these were the results:

Task Name	WITH BACKGROUND ACTIVITY --> MALLOC!						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0	37.00004578	74.00015446	111.0002092	148.0003319	74.00014829	58.50226753
Utime (tv_usec) - before	0	45778	154461	209245	331944	148285.6	132269.6291
Stime (tv_sec) - before	0.000001643	0.000778293	1.000487808	2.000275779	2.000999675	1.00050864	1.00012395
Stime (tv_usec) - before	1643	778293	487808	275779	999675	508639.6	395452.9957
Maxrss - before	1912	1049984	1050060	1050060	1050060	840415.2	468737.5398
Minflt - before	108	466514	932808	1399506	1865790	932945.2	737499.4451
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	1	85	187	307	442	204.4	175.2934682
Utime (tv_sec) - after	0	37.00004578	74.00015446	111.0002092	148.0003319	74.00014829	58.50226753
Utime (tv_usec) - after	0	45778	154461	209245	331944	148285.6	132269.6291
Stime (tv_sec) - after	0.000001643	0.000778293	1.000487808	2.000275779	2.000999675	1.00050864	1.00012395
Stime (tv_usec) - after	1643	778293	487808	275779	999675	508639.6	395452.9957
Maxrss - after	1912	1049984	1050060	1050060	1050060	840415.2	468737.5398
Minflt - after	108	466514	932808	1399506	1865790	932945.2	737499.4451
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	1	85	187	307	442	204.4	175.2934682
L1 HW Cache Read Misses	231879664	231181825	231539469	231762894	231217692	231516308.8	314013.9578
L1 HW Cache Read Accesses	16937883285	16939500881	16947895759	16902301520	16936780827	16932872454	17639455.42
L1 HW Cache Write Misses	6365880	6352265	6359748	6360653	6356419	6358993	5065.763861
L1 HW Cache Write Accesses	6442467336	6443430374	6440940983	6455210743	6448121417	6446034171	5788106.432
L1 HW Cache Prefetch Misses	240330347	240510388	239997727	241053350	240257494	240429861.2	394200.5656
L1 HW Cache Prefetch Accesses	1814932321	1815191651	1814555712	1818482884	1816558302	1815944174	1607806.405
Data TLB Load Misses	2213490	2210595	2215883	2217495	2209516	2213395.8	3387.322202
Data TLB Load Accesses	22983676714	22990758326	22980091785	23011719803	22979937234	22989236772	13311914.29
Data TLB Store Misses	1820753	1811432	1827161	1821301	1820743	1820278	5637.281792
Data TLB Store Accesses	6915341781	6911684493	6913994654	6907664805	6910594844	6911856115	2996270.794

I noticed immediately that the amount of average context switching increased. Additionally, the amount of cache misses that were occuring also increased. For example, on read misses, there was an increase of 29,737,672. On DTLB load misses, there was an increase of 2,049,915. Across the board there were increases in cache misses. Interestingly enough, based off a wall clock... a run with and without memory intensive background activity yielded a runtime of around 3 minutes and 10 seconds.

4 Get Memory Size

This is a helper function to the compete for memory function provided to us. This function calculates and returns the total size of the physical memory in the system or RAM in bytes by using the sysconf().

```
long get_mem_size() {
    long page_sz = sysconf(_SC_PAGE_SIZE);
    long physical_pages = sysconf(_SC_PHYS_PAGES);
    return physical_pages * page_sz;
}
```

4.1 Why do we call the fflush? Would this fflush be necessary if we fprintf'ed to stderr?

The code that was provided to us alongside the function we built, basically intentionally competes for memory by continuously allocating memory with mmap. fflush() is designed to flush a stream. This

means that it is used to manually flush out an output buffer that comes from stdout, which is the default output stream in the program. Generally, flushes do occur with stdout, but usually when the buffer is full. The manual call makes sure that our stdout is visible no matter what. The fflush() would not be necessary if we fprintf'ed to stderr, that is becuase it does not have a buffer.

4.2 Compete For Memory: Malloc()

In this section I compare getrusage and perf event outputs of with/without another process running compete_for_memory. You may use table or chart to show your experimental numbers, and explain them as much as you can. I launch the program in the background on CPU 6, open another terminal, and follow a similar procedure to the background activity portion I did earlier. The command was: `sudo ./do_mem_access_malloc >> do_mem_access_malloc_compete.txt`. First, let's look at a WITHOUT case for memory access malloc():

Task Name	do_mem_access_malloc_0.txt						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000002125	36.00032182	72.00056505	108.0008246	145.0000661	72.20035595	57.23819712
Utime (tv_usec) - before	2125	321815	565053	824646	66115	355950.8	344151.7329
Stime (tv_sec) - before	0	0.000447973	0.000927911	1.000415852	1.000899818	0.400538310	0.5478317927
Stime (tv_usec) - before	0	447973	927911	415852	899818	538310.8	385814.5165
Maxrss - before	1912	1049984	1049984	1049984	1049984	840369.6	468712.0475
Minflt - before	107	262247	524383	786519	1048655	524382.2	414474.6733
Majflt - before	0	0	0	0	0	0	0
Inblock - before	0	0	0	0	0	0	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	1	1	1	1	1	1	0
Involuntary - before	1	74	167	256	347	169	138.29859
Utime (tv_sec) - after	0.000002125	36.00032182	72.00056505	108.0008246	145.0000661	72.20035595	57.23819712
Utime (tv_usec) - after	2125	321815	565053	824646	66115	355950.8	344151.7329
Stime (tv_sec) - after	0	0.000447973	0.000927911	1.000415852	1.000899818	0.400538310	0.5478317927
Stime (tv_usec) - after	0	447973	927911	415852	899818	538310.8	385814.5165
Maxrss - after	1912	1049984	1049984	1049984	1049984	840369.6	468712.0475
Minflt - after	107	262247	524383	786519	1048655	524382.2	414474.6733
Majflt - after	0	0	0	0	0	0	0
Inblock - after	0	0	0	0	0	0	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	1	1	1	1	1	1	0
Involuntary - after	1	74	167	256	347	169	138.29859
L1 HW Cache Read Misses	202141992	202280091	202348750	202447977	202301512	202304064.4	111306.2675
L1 HW Cache Read Accesses	16907834302	16908927904	16947975676	16948858850	16907479484	16924215243	22101974.65
L1 HW Cache Write Misses	2775116	2759675	2770053	2754508	2781731	2768216.6	11118.65254
L1 HW Cache Write Accesses	6456332505	6456528942	6456814868	6457137853	6457816733	6456926180	583208.4748
L1 HW Cache Prefetch Misses	235026360	234931666	234923207	235410630	234845466	235027465.8	223599.7838
L1 HW Cache Prefetch Accesses	1831304682	1831408038	1831432736	1831550875	1831742037	1831487674	166988.8099
Data TLB Load Misses	163575	164007	163280	163301	164030	163638.6	365.8979366
Data TLB Load Accesses	23089683466	23089142456	23041558816	23029959131	23091335180	23068335810	30030762.78
Data TLB Store Misses	2705544	2706358	2700831	2699172	2702351	2702851.2	3058.461165
Data TLB Store Accesses	6626395260	6625669456	6624204322	6626508668	6624855952	6625526732	992172.2824

And this was WITH:

Task Name	COMPETE FOR MEMORY: MALLOC()						
Trial	0	1	2	3	4	AVERAGE	STD
Utime (tv_sec) - before	0.000002272	37.00040404	73.00073835	110.0002153	146.0005888	73.20038976	57.71237237
Utime (tv_usec) - before	2272	404042	738354	215329	588794	389758.2	292365.1011
Stime (tv_sec) - before	0	1.000293523	2.000150788	2.000863653	3.000688925	1.600399378	1.14043729
Stime (tv_usec) - before	0	293523	150788	863653	688925	399377.8	364655.2471
Maxrss - before	1888	1050112	1050152	1050152	1050152	840491.2	468793.4407
Minflt - before	106	466513	932807	1399505	1865789	932944	737499.7613
Majflt - before	2	2	2	2	2	2	0
Inblock - before	56	56	56	56	56	56	0
Outblock - before	0	8	24	32	40	20.8	16.58915308
Voluntary - before	3	272	272	272	272	218.2	120.3004572
Involuntary - before	0	96	214	311	409	206	163.4273539
Utime (tv_sec) - after	0.000002272	37.00040404	73.00073835	110.0002153	146.0005888	73.20038976	57.71237237
Utime (tv_usec) - after	2272	404042	738354	215329	588794	389758.2	292365.1011
Stime (tv_sec) - after	0	1.000293523	2.000150788	2.000863653	3.000688925	1.600399378	1.14043729
Stime (tv_usec) - after	0	293523	150788	863653	688925	399377.8	364655.2471
Maxrss - after	1888	1050112	1050152	1050152	1050152	840491.2	468793.4407
Minflt - after	106	466513	932807	1399505	1865789	932944	737499.7613
Majflt - after	2	2	2	2	2	2	0
Inblock - after	56	56	56	56	56	56	0
Outblock - after	0	8	24	32	40	20.8	16.58915308
Voluntary - after	3	272	272	272	272	218.2	120.3004572
Involuntary - after	0	96	214	311	409	206	163.4273539
L1 HW Cache Read Misses	232473399	231028572	231493766	231351286	231351386	231539681.8	549116.9237
L1 HW Cache Read Accesses	16937595126	16905870423	16942133978	16939651533	16935593417	16932168895	14900282.83
L1 HW Cache Write Misses	6631056	6597410	6594570	6594966	6614091	6606418.6	15957.54871
L1 HW Cache Write Accesses	6455721925	6455334902	6444668186	6458291599	6458471446	6454497612	5678981.123
L1 HW Cache Prefetch Misses	236562638	235718876	235432092	235979564	235734480	235885530	425297.9294
L1 HW Cache Prefetch Accesses	1818017990	1818022783	1815027389	1818889816	1819003308	1817792257	1613991.401
Data TLB Load Misses	2248931	2213100	2209728	2212365	2213747	2219574.2	16481.91017
Data TLB Load Accesses	23020650624	23026719989	23017447245	23020777558	23022618095	23021642702	3393552.236
Data TLB Store Misses	1792730	1810962	1807404	1802849	1816814	1806151.8	9075.265021
Data TLB Store Accesses	6848798840	6860571294	6860324709	6846132868	6846849271	6852535396	7289295.006

Similar to the background activity portion, we can see from the tables there was a very clear increase in context switching and general read and write misses.

5 page replacement change

Due to time constraints, I was unable to finish this portion of the lab.

6 Total Hours Spent:

I would say I spent approximately 80 hours on this lab.