Fast System Calls: Performance and Security Implications

UG Project Final Report

Mentor:
Prof. Debadatta Mishra

Group Members:
Sarthak(190772) and Manish(190477)

April 25, 2023

Contents

1 Introduction

System calls play a crucial role in allowing users to access the services provided by the operating system. These services are essential for performing privileged operations. In earlier implementations, accessing these services involved using the INT 0x80 instruction to enter the kernel, which was a relatively slow process as it required reading from memory. However, a faster system call was later introduced that only manipulated certain general purpose and special registers to switch to kernel mode. This made the process much lighter and faster. To achieve this, the system call involved finding the kernel address to jump to from LSTAR MSR registers and modifying the CS and SS selector registers from STAR MSR.
In this project, we establish the behavior of syscall instruction in speculative mode and modify it to show that forbidding speculative execution with syscall entails a significant performance penalty.

2 Code

Please check out the codebase at SpecPriv.

3 Reverse engineering the syscall instruction

The aim of our study is to examine the behavior and interaction of syscalls with neighboring instructions in both speculative and non-speculative modes. Specifically, we are interested in determining whether in speculative mode, user-space (PC + 4) instructions are fetched and executed, or if kernel-space instructions are fetched and executed, or if the process stalls.
To achieve this, we have implemented a micro-benchmark to train the branch predictor on a syscall instruction, which is present in an if block. We then investigate the scenario where the predictor fails the first time. Through timing measurements on data blocks, we investigate whether any specific load instruction, whether in user-space or kernel-space, was speculatively executed. This study will help us gain a better understanding of how syscalls interact with other instructions and the potential implications of speculative execution in such scenarios.

3.1 Real machine experiments

3.1.1 System Details

We experimented on a Intel x86 machine based on Skylake architecture with 32 cores and 3 layered caches with L1 cache being composed of instruction and data cache.

3.1.2 Micro benchmark code

```
long makesyscall(int condition)
-
   long retval;
   // Flushing buf from cache
   asm volatile(
                "clflush (%0);"
                "mfence;"
                :
                :"r" (buf)
                : "memory"
                );


   if(condition)-
          // The assembly below executes write(fd, buf, 64)
          asm volatile("mov $1, %%rax;"    //SYSCALL write from /usr/include/
    x8664-linux-gnu/asm/unistd64.h
               "mov %1, %%rdi;"
               "mov %2, %%rsi;"
               "mov $64, %%rdx;"
               "syscall;"
               // Accessing buffer in entry64.S for kernel mode speculation
               "mov %%rax, %0;" // For usermode speculation
               :"=m" (retval)
               :"r" (fd), "r"(buf)
               :"rax", "rdi", "rsi", "rdx", "rcx", "memory"
              );

   return retval;


int main()
-
   //Section 1
   register long sum = 0;
   long timearray[101];
   fd = open("some.txt", ORDWR—OCREAT, 0644);
   assert(fd ¿ 0);

   //Section 2
   buf=malloc(1 ¡¡ 30);
```

```
41    memset(buf, '1', 1 ¡¡ 30);
42    free(buf);
43    buf=malloc(64);
44    memset(buf, 0, 64);
45
46    //Section 3
47    for(int i=1; i¡=100; i++)-
48         long start, end;
49         makesyscall(i¡=20);
50         start = rdt();
51         sum += buf[5];
52         end = rdt();
53         timearray[i] = end - start;
54         lseek(fd, 0, SEEKSET);
55
56    close(fd);
57
58    //Section 4
59    for(int i=1; i¡=100; i++)
60          printf("%d:%ld ", i, timearray[i]);
61    printf(""n sum %ld"n", sum);
62
```

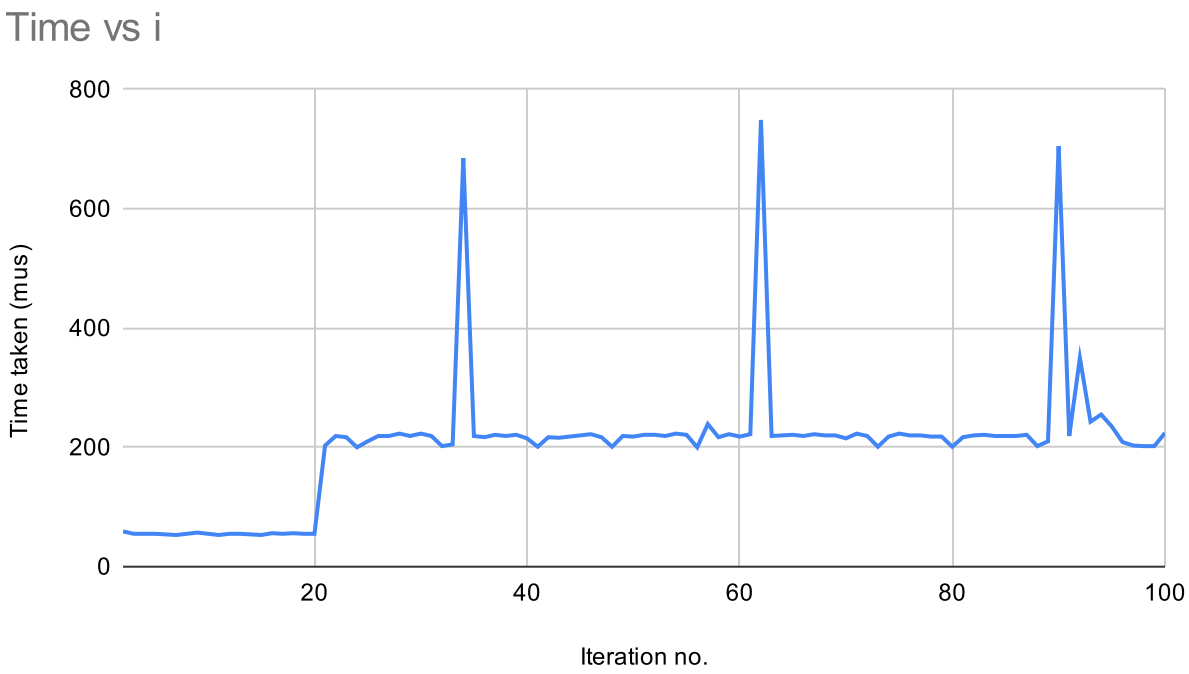


Figure 1: Time vs Iteration number (User mode simulation)

We can observe that there is a sharp increase at i = 20, and there is no speculative execution for i = 21, 22 etc.

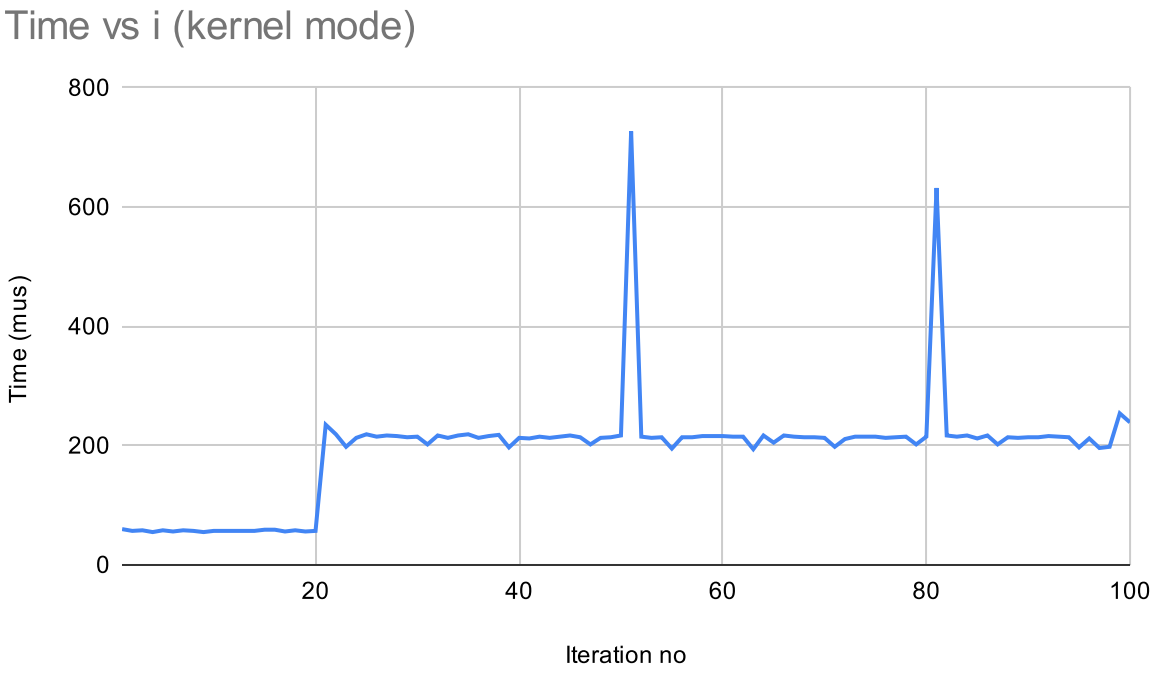


Figure 2: Time vs Iteration number (Kernel mode simulation)

Here too, in kernel mode speculation, we can observe that there is a sharp increase at i = 20, and there is no speculative execution for i = 21, 22 etc.

3.2 Gem5 experiments

We chose to confirm our experiments on the Gem5 simulator, which emulates both hardware and software. We utilized our modified kernel and code snippet, and established a workload that boots with a KVM CPU for quicker experiments before switching to an O3 CPU for more thorough out-of-order simulations. ClassicCache was employed instead of the default RubyCache to allow for clflush instructions.
Our Gem5 experiments yielded consistent results with our experiments on a physical machine.
We modified entry_64.S to modify the behavior of syscall after jumping to jump target (value stored at IA32_LSTAR_MSR). It was modified to access the buffer for only a specific call to syscall.

3.3 Gem5 Results

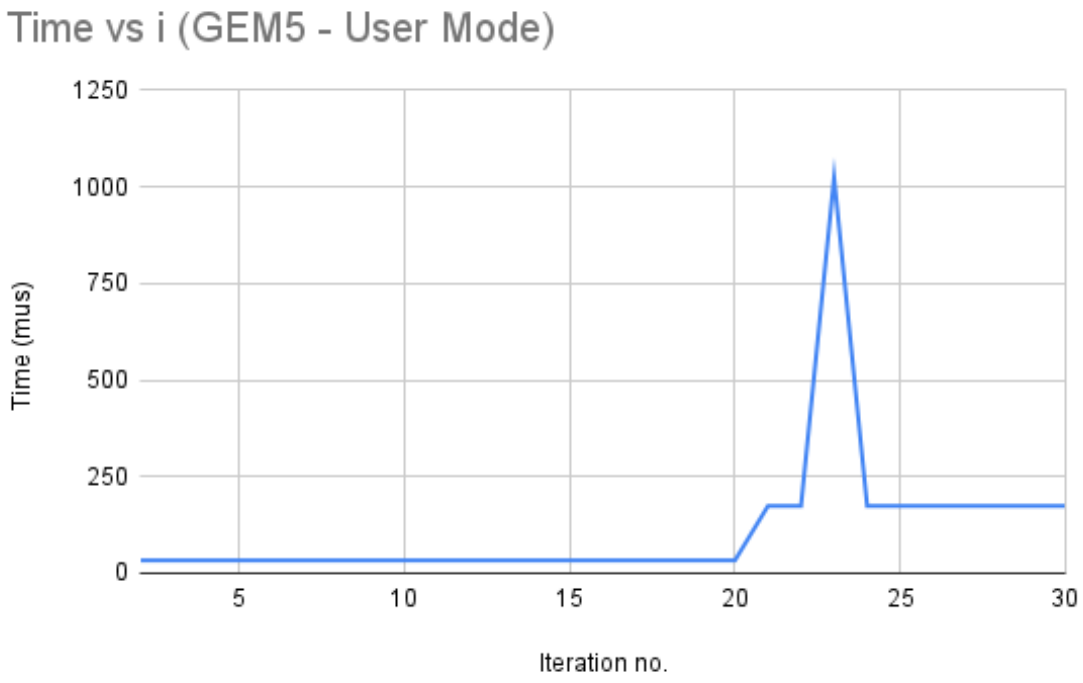


Figure 3: Gem5 - User Mode

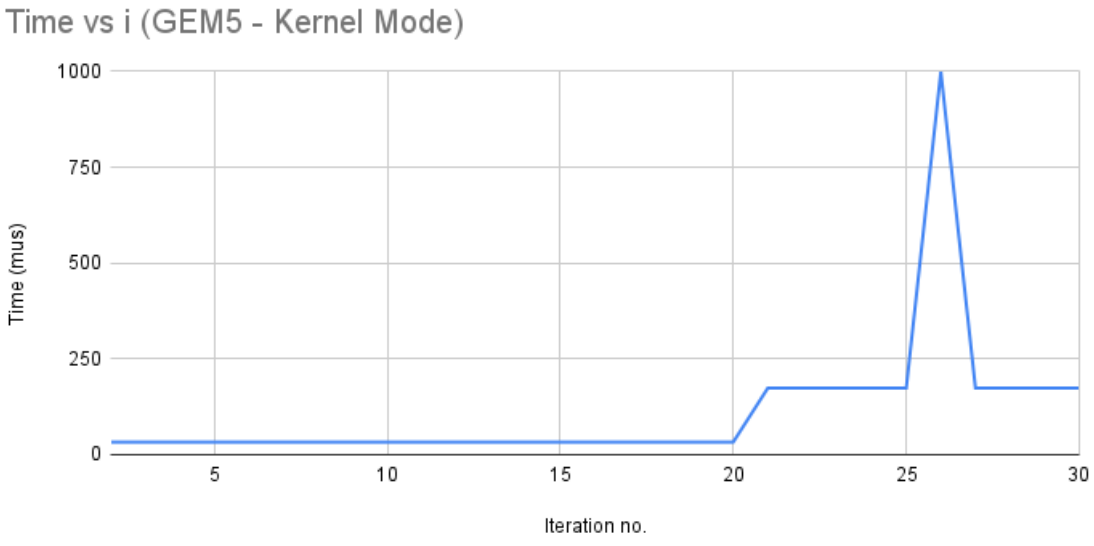


Figure 4: Gem5 - Kernel Mode

3.4 Conclusion from all experiments

We observed that the buffer was not obtained from user-space or kernel-space neither in real machines nor in Gem5 simulator. This indicates that the syscall completely prohibits executing any instruction speculatively in either user mode or kernel mode.

4 Analysis of syscall implementation in Gem5

In Gem5 syscall instruction is implemented using 25 MicrrOps. Every MicrrOps in Gem5 has 3 flags to focus on:
IsNonSpeculative flag: If the instruction has non speculative flag true means that when it is issued into the rob its execution only start when it comes to the head of the rob. Instruction after it can exeute freely.
IsSerializeAfter flag: SerializeAfter marks the next instruction as serializeBefore. SerializeBefore makes the instruction wait in rename until the ROB is empty.
IsSquashAfter flag: If the instruction has IsSquashAfter flag true means that whenever this instruction come to the head of the ROB all the instruction except this in ROB will be flushed.
Below is the table of all Syscall Micro-Ops and flags value(IsNonSpeculative, IsSerializeAfter, IsSquashAfter in order):

Serial no.	MicroOp	Flags
1	limm t1, ”(uint64_t)(-1)”, dataSize=8	nnn
2	rdip rcx	nnn
3	rflags t2	nnn
4	limm t3, ”∼RFBit”, dataSize=8	nnn
5	and r11, t2, t3, dataSize=8	nnn
6	rdval t3, star	nnn
7	srli t3, t3, 32, dataSize=8	nnn
8	andi t3, t3, 0xFC, dataSize=1	nnn
9	wrsel cs, t3	yyn
10	wrbase cs, t0, dataSize=8	yyy
11	wrlimit cs, t1, dataSize=4	yyy
12	limm t4, val, dataSize=8	nnn
13	wrattr cs, t4	yyy
14	addi t3, t3, 8	nnn
15	wrsel ss, t3	yyn

16	wrbase ss, t0, dataSize=8	yyn
17	wrlimit ss, t1, dataSize=4	yyn
18	limm t4, val, dataSize=8	nnn
19	wrattr ss, t4	yyn
20	rdval t7, lstar, dataSize=8	nnn
21	wrip t0, t7, dataSize=8	nnn
22	rdval t3, sf_mask, dataSize=8	nnn
23	xor t3, t3, t1, dataSize=8	nnn
24	and t3, t3, r11, dataSize=8	nnn
25	wrflags t3, t0	yyn

y: flag is true
n: flag is false
Our focus remains on the MicroOps for which these flags are true. Serial No. for these MicroOps are 9, 10, 11, 13, 15,16, 17, 19, 25.

4.1 Program

The program we use to run in gem5 is here.

4.2 O3PipeView Utility Output

Gem5 provides a utility which takes the output of debug-flag named O3PipeView and converts it into a view of a pipeline in a processor.
In the output of the utility:
f = fetch stage
d = decode stage
n = rename stage
p = dispatch stage
i = issue stage
c = complete stage
r = retire stage

```
[..r.....................................................................fdn.ic.]-( 16907516560000) 0x555555554718.0 MOV_R_I
[..r.....................................................................fdn.ic.]-( 16907516560000) 0x55555555471f.0 SYSCALL_64
[..r.....................................................................fdn.ic]-( 16907516560000) 0x55555555471f.1 SYSCALL_64
[ic.r....................................................................fdn.p.]-( 16907516560000) 0x55555555471f.2 SYSCALL_64
[...r....................................................................fdn.ic]-( 16907516560000) 0x55555555471f.3 SYSCALL_64
[.ic.r...................................................................fdn.p.]-( 16907516560000) 0x55555555471f.4 SYSCALL_64
[....r...................................................................fdn.ic]-( 16907516560000) 0x55555555471f.5 SYSCALL_64
[c...r...................................................................fdn.pi]-( 16907516560000) 0x55555555471f.6 SYSCALL_64
[ic..r...................................................................fdn.p.]-( 16907516560000) 0x55555555471f.7 SYSCALL_64
[.......ic.r.............................................................fdn.p.]-( 16907516560000) 0x55555555471f.8 SYSCALL_64
[............p.ic.r.....................................................fdn..]-( 16907516560000) 0x55555555471f.9 SYSCALL_64
[.....................fdn.p.ic.r.....................................]-( 16907516640000) 0x55555555471f.10 SYSCALL_64
[..................................fdn.ic.r.....................]-( 16907516640000) 0x55555555471f.11 SYSCALL_64
[..................................fdn.p.....ic.r...............]-( 16907516640000) 0x55555555471f.12 SYSCALL_64
[.................................................fdn.ic.r.....]-( 16907516640000) 0x55555555471f.13 SYSCALL_64
[.................................................fdn.p.....ic.r.]-( 16907516640000) 0x55555555471f.14 SYSCALL_64
[.................................................fdn.............p.ic.r.....]-( 16907516640000) 0x55555555471f.15 SYSCALL_64
[.................................................fdn.....................p.ic.r.]-( 16907516640000) 0x55555555471f.16 SYSCALL_64
[.ic.r............................................fdn.....................]-( 16907516640000) 0x55555555471f.17 SYSCALL_64
[.p.....ic.r......................................fdn.....................]-( 16907516640000) 0x55555555471f.18 SYSCALL_64
[.............ic..r...............................fdn.....................]-( 16907516640000) 0x55555555471f.19 SYSCALL_64
[.............pic.r...............................fdn.....................]-( 16907516640000) 0x55555555471f.20 SYSCALL_64
[..................fdn.ic.r.....................................................]-( 16907516720000) 0x55555555471f.21 SYSCALL_64
[..................fdn.pic.r.....................................................]-( 16907516720000) 0x55555555471f.22 SYSCALL_64
[..................fdn.p.ic.r.....................................................]-( 16907516720000) 0x55555555471f.23 SYSCALL_64
[..................fdn.p.......ic.r.....................................................]-( 16907516720000) 0x55555555471f.24 SYSCALL_64
[...................fdn..............ic.r.....................................]-( 16907516720000) 0xffffffff81600010.0 SWAPGS
[...................fdn..............ic.r.....................................]-( 16907516720000) 0xffffffff81600010.1 SWAPGS
[...................fdn..............p.....ic.r.....................]-( 16907516720000) 0xffffffff81600010.2 SWAPGS
[...................fdn.............................p.ic.r.....................]-( 16907516720000) 0xffffffff81600010.3 SWAPGS
[...................fdn.....................................ic.r.....]-( 16907516720000) 0xffffffff81600013.0 MOV_M_R
```

Our syscall address is 0x55555555471f.

4.3 Performance and security analysis

- The design is highly sequential and has strict limitations.
- The first eight MicroOps are executed through pipelining.
- Serialization begins when the ninth MicroOp is executed and the flag effects are involved.
- Speculation occurs at the wrip.
- The SquashAfter flag eliminates user space instructions after altering the CS register.
- It takes 111 cycles to complete, from the initial instruction's commit to the final instruction's commit.
- It is compatible with all kernels and operates accurately.

5 Syscall variants and their analysis

We have tried many different versions of syscalls and analysed their performance and security implication.

5.1 Version 1

In this version IsNonSpeculative and IsSerialisingAfter flag is false for all MicroOps.

5.1.1 Updated MicroOps Description

Serial no.	MicroOp	Flags
1	limm t1, ”(uint64_t)(-1)”, dataSize=8	nnn
2	rdip rcx	nnn
3	rflags t2	nnn
4	limm t3, ”∼RFBit”, dataSize=8	nnn
5	and r11, t2, t3, dataSize=8	nnn
6	rdval t3, star	nnn
7	srli t3, t3, 32, dataSize=8	nnn
8	andi t3, t3, 0xFC, dataSize=1	nnn
9	wrsel cs, t3	nnn
10	wrbase cs, t0, dataSize=8	nny
11	wrlimit cs, t1, dataSize=4	nny
12	limm t4, val, dataSize=8	nnn
13	wrattr cs, t4	nny
14	addi t3, t3, 8	nnn
15	wrsel ss, t3	nnn
16	wrbase ss, t0, dataSize=8	nnn
17	wrlimit ss, t1, dataSize=4	nnn
18	limm t4, val, dataSize=8	nnn
19	wrattr ss, t4	nnn
20	rdval t7, lstar, dataSize=8	nnn
21	wrip t0, t7, dataSize=8	nnn
22	rdval t3, sf_mask, dataSize=8	nnn
23	xor t3, t3, t1, dataSize=8	nnn
24	and t3, t3, r11, dataSize=8	nnn

25	wrflags t3, t0	nnn

5.1.2 O3 PipeView Utility Output

```
[.............................................fdn.ic...r.....................]-( 17350510000000) 0x555555554718.0 MOV_R_I
[.............................................fdn.ic...r.....................]-( 17350510000000) 0x55555555471f.0 SYSCALL_64
[..............................................fdn.ic..r.....................]-( 17350510000000) 0x55555555471f.1 SYSCALL_64
[..............................................fdn.p.ic....r.................]-( 17350510000000) 0x55555555471f.2 SYSCALL_64
[..............................................fdn.ic......r.................]-( 17350510000000) 0x55555555471f.3 SYSCALL_64
[..............................................fdn.p..ic...r.................]-( 17350510000000) 0x55555555471f.4 SYSCALL_64
[..............................................fdn.ic......r.................]-( 17350510000000) 0x55555555471f.5 SYSCALL_64
[..............................................fdn.pic.....r.................]-( 17350510000000) 0x55555555471f.6 SYSCALL_64
[..............................................fdn.p.ic....r.................]-( 17350510000000) 0x55555555471f.7 SYSCALL_64
[..............................................fdn.p..ic...r.................]-( 17350510000000) 0x55555555471f.8 SYSCALL_64
[...............................................fdn.ic.....r.................]-( 17350510000000) 0x55555555471f.9 SYSCALL_64
[.....................................................................fdn...ic.r.....]-( 17350510000000) 0x55555555471f.10 SYSCALL_64
[.ic.r.....................................................................fdn]-( 17350510000000) 0x55555555471f.11 SYSCALL_64
[.pic..r...................................................................fdn]-( 17350510000000) 0x55555555471f.12 SYSCALL_64
[.........fdn.ic.r.....................................................]-( 17350510080000) 0x55555555471f.13 SYSCALL_64
[.........fdn.pic.r....................................................]-( 17350510080000) 0x55555555471f.14 SYSCALL_64
[.........fdn.ic..r....................................................]-( 17350510080000) 0x55555555471f.15 SYSCALL_64
[.........fdn.ic..r....................................................]-( 17350510080000) 0x55555555471f.16 SYSCALL_64
[.........fdn.ic..r....................................................]-( 17350510080000) 0x55555555471f.17 SYSCALL_64
[.........fdn.pic.r....................................................]-( 17350510080000) 0x55555555471f.18 SYSCALL_64
[.........fdn.ic..r....................................................]-( 17350510080000) 0x55555555471f.19 SYSCALL_64
[.........fdn.pic.r....................................................]-( 17350510080000) 0x55555555471f.20 SYSCALL_64
[..........f.dn.ic.r...................................................]-( 17350510080000) 0x55555555471f.21 SYSCALL_64
[..........f.dn.pic.r..................................................]-( 17350510080000) 0x55555555471f.22 SYSCALL_64
[..........f.dn.p.ic.r.................................................]-( 17350510080000) 0x55555555471f.23 SYSCALL_64
[..........f.dn.p..ic.r................................................]-( 17350510080000) 0x55555555471f.24 SYSCALL_64
[...........f.dn.ic...r................................................]-( 17350510080000) 0xffffffff81600010.0 SWAPGS
[...........f.dn.ic...r................................................]-( 17350510080000) 0xffffffff81600010.1 SWAPGS
[...........f.dn.p.......ic.r..........................................]-( 17350510080000) 0xffffffff81600010.2 SWAPGS
[...........f.dn...............p.ic.r..................................]-( 17350510080000) 0xffffffff81600010.3 SWAPGS
```

5.1.3 Performance and security analysis

- Our program was executed accurately.
- After observing the O3PipeView Utility, we noticed pipelining. occurring in some MicroOps related to the SS register.
- Only for MicroOps associated with the CS register, we experience a loss of cycles.
- The total number of cycles taken was 43.
- However, this execution method may not be entirely accurate for all scenarios. For instance, a user space instruction in the ROB may be present, and changing the CS register(sel) could lead to incorrect execution. Additionally, if wrflags are not written, kernel instructions using them could also cause issues. The same applies to wrsel(SS reg).

5.2 Version 2

In this version, IsNonSpeculative, IsSerializeAfter and IsSquashAfter are false for all MicroOps except wrattr MicroOp for which the IsSquashAfter flag is true for our syscall.

5.2.1 Updated MicroOps Description

Serial no.	MicroOp	Flags
1	limm t1, ”(uint64_t)(-1)”, dataSize=8	nnn
2	rdip rcx	nnn
3	rflags t2	nnn
4	limm t3, ”∼RFBit”, dataSize=8	nnn
5	and r11, t2, t3, dataSize=8	nnn
6	rdval t3, star	nnn
7	srli t3, t3, 32, dataSize=8	nnn
8	andi t3, t3, 0xFC, dataSize=1	nnn
9	wrsel cs, t3	nnn
10	wrbase cs, t0, dataSize=8	nnn
11	wrlimit cs, t1, dataSize=4	nnn
12	limm t4, val, dataSize=8	nnn
13	wrattr cs, t4	nny
14	addi t3, t3, 8	nnn
15	wrsel ss, t3	nnn
16	wrbase ss, t0, dataSize=8	nnn
17	wrlimit ss, t1, dataSize=4	nnn
18	limm t4, val, dataSize=8	nnn
19	wrattr ss, t4	nnn
20	rdval t7, lstar, dataSize=8	nnn
21	wrip t0, t7, dataSize=8	nnn
22	rdval t3, sf_mask, dataSize=8	nnn
23	xor t3, t3, t1, dataSize=8	nnn
24	and t3, t3, r11, dataSize=8	nnn
25	wrflags t3, t0	nnn

5.2.2 O3PipeView Utility Output

```
[n.ic...r.....................................................................fd]-( 18501453120000) 0x555555554718.0 MOV_R_I
[n.ic...r.....................................................................fd]-( 18501453120000) 0x55555555471f.0 SYSCALL_64
[dn.ic..r.....................................................................f]-( 18501453120000) 0x55555555471f.1 SYSCALL_64
[dn.p.ic....r.................................................................f]-( 18501453120000) 0x55555555471f.2 SYSCALL_64
[dn.ic......r.................................................................f]-( 18501453120000) 0x55555555471f.3 SYSCALL_64
[dn.p..ic...r.................................................................f]-( 18501453120000) 0x55555555471f.4 SYSCALL_64
[dn.ic......r.................................................................f]-( 18501453120000) 0x55555555471f.5 SYSCALL_64
[dn.pic.....r.................................................................f]-( 18501453120000) 0x55555555471f.6 SYSCALL_64
[dn.p.ic....r.................................................................f]-( 18501453120000) 0x55555555471f.7 SYSCALL_64
[dn.p..ic...r.................................................................f]-( 18501453120000) 0x55555555471f.8 SYSCALL_64
[fdn.ic.....r.................................................................]-( 18501453200000) 0x55555555471f.9 SYSCALL_64
[fdn.ic......r................................................................]-( 18501453200000) 0x55555555471f.10 SYSCALL_64
[fdn.ic......r................................................................]-( 18501453200000) 0x55555555471f.11 SYSCALL_64
[fdn.pic.....r................................................................]-( 18501453200000) 0x55555555471f.12 SYSCALL_64
[..............fdn...ic.r.....................................................]-( 18501453200000) 0x55555555471f.13 SYSCALL_64
[..............fdn...pic.r....................................................]-( 18501453200000) 0x55555555471f.14 SYSCALL_64
[..............fdn...ic..r....................................................]-( 18501453200000) 0x55555555471f.15 SYSCALL_64
[..............fdn...ic..r....................................................]-( 18501453200000) 0x55555555471f.16 SYSCALL_64
[..............fdn...ic..r....................................................]-( 18501453200000) 0x55555555471f.17 SYSCALL_64
[..............fdn...pic.r....................................................]-( 18501453200000) 0x55555555471f.18 SYSCALL_64
[..............fdn...ic..r....................................................]-( 18501453200000) 0x55555555471f.19 SYSCALL_64
[..............fdn...pic.r....................................................]-( 18501453200000) 0x55555555471f.20 SYSCALL_64
[.............f...dn.ic.r.....................................................]-( 18501453200000) 0x55555555471f.21 SYSCALL_64
[.............f...dn.pic.r....................................................]-( 18501453200000) 0x55555555471f.22 SYSCALL_64
[.............f...dn.p.ic.r...................................................]-( 18501453200000) 0x55555555471f.23 SYSCALL_64
[.............f...dn.p..ic.r..................................................]-( 18501453200000) 0x55555555471f.24 SYSCALL_64
[............f..dn.ic....r....................................................]-( 18501453200000) 0xffffffff81600010.0 SWAPGS
[............f..dn.ic....r....................................................]-( 18501453200000) 0xffffffff81600010.1 SWAPGS
[............f..dn.p........ic.r...............................................]-( 18501453200000) 0xffffffff81600010.2 SWAPGS
[............f..dn................p.ic.r.....................................]-( 18501453200000) 0xffffffff81600010.3 SWAPGS
```

5.2.3 Performance and security analysis

- Our program was executed successfully.
- Upon observing the O3PipeView Utility, we discovered additional pipelining.
- We experienced a performance loss only when dealing with WrAttr.
- The total number of cycles taken was 22.
- However, this method of execution may not be entirely accurate for all scenarios. For instance, if a user space instruction is in the ROB and we change the CS register, it could result in accessing a user-space address in kernel mode. A malicious user could exploit this to access kernel addresses from userspace without a segfault. Additionally, there are implications for wrsel and wrflags as well

5.3 Version 3

In this version, all flags of WrAttr are preserved And IsNonSpeculative, IsSerializingAfter and IssquashAfter flag is removed from rest of the MicroOps.

5.3.1 Updated MicroOps Description

Serial no.	MicroOp	Flags
1	limm t1, ”(uint64_t)(-1)”, dataSize=8	nnn
2	rdip rcx	nnn
3	rflags t2	nnn
4	limm t3, ”∼RFBit”, dataSize=8	nnn
5	and r11, t2, t3, dataSize=8	nnn
6	rdval t3, star	nnn
7	srli t3, t3, 32, dataSize=8	nnn
8	andi t3, t3, 0xFC, dataSize=1	nnn
9	wrsel cs, t3	nnn
10	wrbase cs, t0, dataSize=8	nnn
11	wrlimit cs, t1, dataSize=4	nnn
12	limm t4, val, dataSize=8	nnn
13	wrattr cs, t4	yyy
14	addi t3, t3, 8	nnn
15	wrsel ss, t3	nnn
16	wrbase ss, t0, dataSize=8	nnn
17	wrlimit ss, t1, dataSize=4	nnn
18	limm t4, val, dataSize=8	nnn
19	wrattr ss, t4	nnn
20	rdval t7, lstar, dataSize=8	nnn
21	wrip t0, t7, dataSize=8	nnn
22	rdval t3, sf_mask, dataSize=8	nnn
23	xor t3, t3, t1, dataSize=8	nnn
24	and t3, t3, r11, dataSize=8	nnn
25	wrflags t3, t0	nnn

5.3.2 O3PipeView Utility Output

```
[.................fdn.ic...r.....................................................]-( 18849404080000) 0x555555554718.0 MOV_R_I
[.................fdn.ic...r.....................................................]-( 18849404080000) 0x55555555471f.0 SYSCALL_64
[..................fdn.ic..r.....................................................]-( 18849404080000) 0x55555555471f.1 SYSCALL_64
[..................fdn.p.ic....r.................................................]-( 18849404080000) 0x55555555471f.2 SYSCALL_64
[..................fdn.ic......r.................................................]-( 18849404080000) 0x55555555471f.3 SYSCALL_64
[..................fdn.p..ic...r.................................................]-( 18849404080000) 0x55555555471f.4 SYSCALL_64
[..................fdn.ic......r.................................................]-( 18849404080000) 0x55555555471f.5 SYSCALL_64
[..................fdn.pic.....r.................................................]-( 18849404080000) 0x55555555471f.6 SYSCALL_64
[..................fdn.p.ic....r.................................................]-( 18849404080000) 0x55555555471f.7 SYSCALL_64
[..................fdn.p..ic...r.................................................]-( 18849404080000) 0x55555555471f.8 SYSCALL_64
[...................fdn.ic.....r.................................................]-( 18849404080000) 0x55555555471f.9 SYSCALL_64
[...................fdn.ic......r................................................]-( 18849404080000) 0x55555555471f.10 SYSCALL_64
[...................fdn.ic......r................................................]-( 18849404080000) 0x55555555471f.11 SYSCALL_64
[...................fdn.p.........ic.r...........................................]-( 18849404080000) 0x55555555471f.12 SYSCALL_64
[.....................................fdn...ic.r...............................]-( 18849404080000) 0x55555555471f.13 SYSCALL_64
[.....................................fdn...pic.r..............................]-( 18849404080000) 0x55555555471f.14 SYSCALL_64
[.....................................fdn...ic..r..............................]-( 18849404080000) 0x55555555471f.15 SYSCALL_64
[.....................................fdn...ic..r..............................]-( 18849404080000) 0x55555555471f.16 SYSCALL_64
[.....................................fdn...ic..r..............................]-( 18849404080000) 0x55555555471f.17 SYSCALL_64
[.....................................fdn...p......ic.r........................]-( 18849404080000) 0x55555555471f.18 SYSCALL_64
[.....................................fdn...ic........r........................]-( 18849404080000) 0x55555555471f.19 SYSCALL_64
[.....................................fdn...pic.......r........................]-( 18849404080000) 0x55555555471f.20 SYSCALL_64
[.....................................f...dn.ic......r........................]-( 18849404080000) 0x55555555471f.21 SYSCALL_64
[.....................................f...dn.pic.....r........................]-( 18849404080000) 0x55555555471f.22 SYSCALL_64
[.....................................f...dn.p.ic....r........................]-( 18849404080000) 0x55555555471f.23 SYSCALL_64
[.....................................f...dn.p..ic...r........................]-( 18849404080000) 0x55555555471f.24 SYSCALL_64
[.....................................f...dn.ic....r........................]-( 18849404080000) 0xffffffff81600010.0 SWAPGS
[.....................................f...dn.ic.....r.......................]-( 18849404080000) 0xffffffff81600010.1 SWAPGS
[.....................................f...dn.p.........ic.r.................]-( 18849404080000) 0xffffffff81600010.2 SWAPGS
[.....................................f...dn.................p.ic.r.........]-( 18849404080000) 0xffffffff81600010.3 SWAPGS
```

5.3.3 Performance and security analysis

- Our program was executed accurately.
- Compared to the previous case, we experienced a performance loss only
- when dealing with WrAttr due to the introduction of additional restrictions such as setting flags IsSpeculative and IsSerializingAfter to true.
- The total number of cycles taken was 31.
- However, this method of execution may not be entirely accurate for all scenarios. For instance, if a user space instruction is in the ROB and we change the CS register (selector), the user space could utilize it. It can also occur that we have not modified the WrFlags, and we are using WrFlags in kernel mode, leading to inaccuracies.

5.4 Version 4

Unnecessary MicroOps are removed from the syscall instruction and all the flags of all MicroOps are preserved.

5.4.1 Updated MicroOps Description

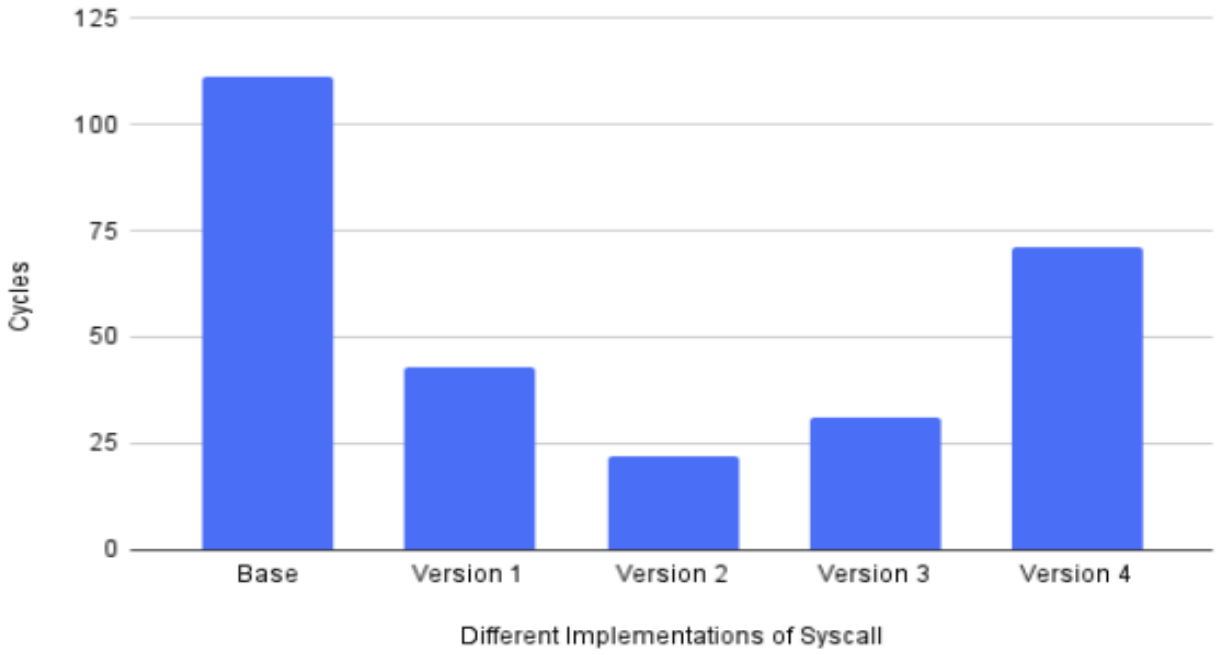
Serial no.	MicroOp	Flags
1	limm t1, ”(uint64_t)(-1)”, dataSize=8	nnn
2	rdip rcx	nnn
3	rflags t2	nnn
4	limm t3, ”∼RFBit”, dataSize=8	nnn
5	and r11, t2, t3, dataSize=8	nnn
6	rdval t3, star	nnn
7	srli t3, t3, 32, dataSize=8	nnn
8	andi t3, t3, 0xFC, dataSize=1	nnn
9	wrsel cs, t3	yyn
10	limm t4, val, dataSize=8	nnn
11	wrattr cs, t4	yyy
12	addi t3, t3, 8	nnn
13	wrsel ss, t3	yyn
14	limm t4, val, dataSize=8	nnn
15	wrattr ss, t4	yyn
16	rdval t7, lstar, dataSize=8	nnn
17	wrip t0, t7, dataSize=8	nnn
18	rdval t3, sf_mask, dataSize=8	nnn
19	xor t3, t3, t1, dataSize=8	nnn
20	and t3, t3, r11, dataSize=8	nnn
21	wrflags t3, t0	yyn

5.4.2 Performance and security analysis

- Our program was executed accurately.
- Despite not being pipelined, our implementation is correct for all kernels that do not use segmentation, resulting in a loss of performance.
- The total number of cycles taken was 71.

- This method of execution is accurate for all processors that do not support segmentation since most modern operating systems use page tables for translations. Segment register base and limit are only utilized for backward compatibility.

6 Conclusion



The security and performance tradeoff is a well-known concept in computer science, especially when it comes to designing computer systems, such as operating systems and applications. In general, the more secure a system is, the slower it will perform. Conversely, the faster a system performs, the less secure it may be.

To achieve better performance, many systems implement techniques such as caching, pipelining, and speculative execution. While these techniques can lead to better performance, they can also introduce security vulnerabilities. For instance, speculative execution can lead to side-channel attacks, where an attacker can potentially access sensitive information through unintended channels.

To avoid these type of attacks and still gets the syscall performance we have to do something more which is we have to design a kernel based on the syscall instruction implementation. We can design kernels which will make use of Version 3 implementation of Syscall instruction and didn't leak anything. Version 4 Syscall Instruction can be used on a system which doesn't support backward

compatibility to segmentation which was used for translation for 32-bit architectures.

7 References

- Gem5 GitHub: https://github.com/gem5/gem5/tree/stable/src
- Linux Kernel Github: https://github.com/torvalds/linux
- Gem5 Documentation: https://www.gem5.org/documentation/
- Intel Software Developer Manual: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
- Extended ASM documentation: https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html