FAST SYSTEM CALL

Performance and Security Implications

Project Team: Manish and Sarthak Rout

Supervised by Prof. Debadatta Mishra

Introduction

- The idea of “system call” which exposes services provided by the operating system to the user. They are critical for the user to perform privileged operations.
- Earlier implementations used INT 0x80 instruction to enter the kernel. This was slow as it involves reading from memory.
- Later, the fast system call was introduced which only manipulated some general purpose and special registers to switch to kernel mode. This was lighter and faster.
- It involved finding the kernel address to jump to from LSTAR MSR registers, modifying the CS and SS selector registers from STAR MSR.

Reverse Engg. the Syscall Instruction

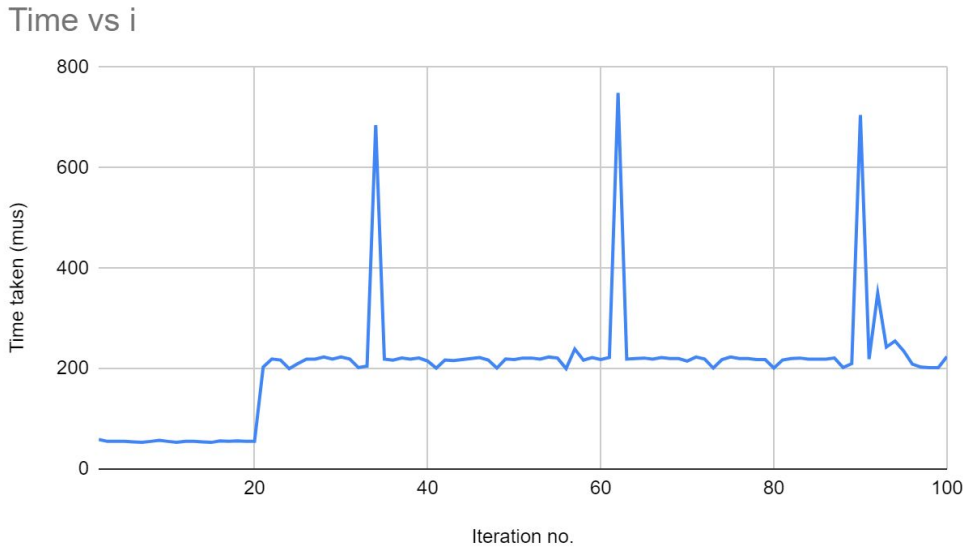
- We want to establish the behavior and interaction of syscall with neighbouring instructions in speculative and non-speculative mode.
- In particular, in speculative mode, we want to know whether user-space (PC + 4) instructions are fetched/executed, or are kernel-space instructions are fetched/executed or if it stalls.
- We use a code snippet to train the branch predictor on a syscall instruction (which is present in a if block) and investigate the case when the predictor fails the first time.
- We use timing measurements on data blocks to find out whether any particular load instruction (in user-space or kernel-space) was even speculatively executed.

Pseudocode for code snippet

```
global buffer
make_syscall(cond){
    flush buffer
    if cond {
        execute syscall instruction // can access buffer from kernel mode
        // can access buffer here
    }
}

main(){

loop over i from 1 to 100,
    make_syscall(i <= 20 );
    measure time taken to access buff using RDTSC
}
```

User Mode Speculation: Results

```
global buffer
make_syscall(cond){
    flush buffer
    if cond {
        execute syscall
instruction
        access buffer here
    }
}
main(){

loop over i from 1 to 100,
    make_syscall(i <= 20 );
    measure time taken to
access buff
}
```



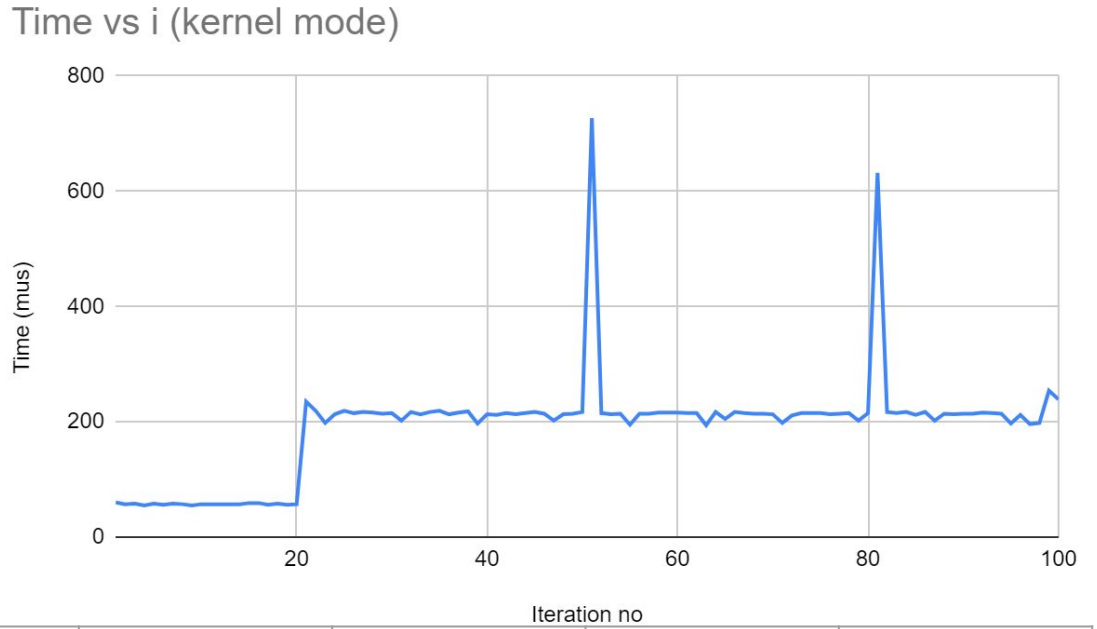
18	19	20	21	22	23
56	55	55	203	219	217

Kernel Mode Speculation: Accessing the buffer from kernel

```
SYM_CODE_START(entry_SYSCALL_64)
    cmpq $780, %rax  // We set rax (syscall number) to 780 which is
unused
    jne label1      // We jump to label if rax != 780 else fall through
    movq (%rsi), %r9    // We access the buffer as it is stored in rsi in
    the user snippet
label1,
    UNWIND_HINT_ENTRY
    ENDBR

    swapgs
    /* tss.sp2 is scratch space. */
    movq    %rsp, PER_CPU_VAR(cpu_tss_rw + TSS_sp2)
    SWITCH_TO_KERNEL_CR3 scratch_reg=%rsp
    movq    PER_CPU_VAR(cpu_current_top_of_stack), %rsp
```

Kernel Mode Speculation: Results

```
global buffer
make_syscall(cond){
    flush buffer
    if cond {
        execute syscall
instruction
        // access buffer in
entry_64.S

}
main(){

loop over i from 1 to 100,
    make_syscall(i <= 20 );
    measure time taken to
access buff
}
```

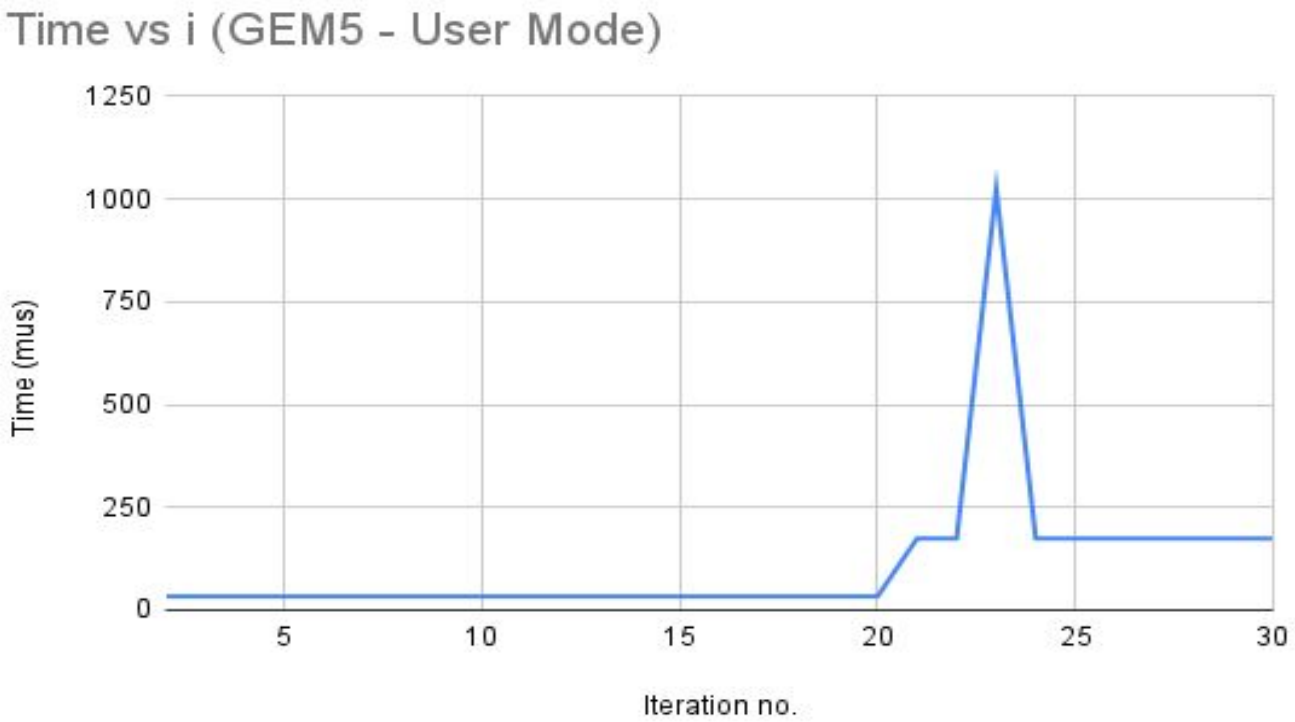


18	19	20	21	22	23
58	56	57	235	219	198

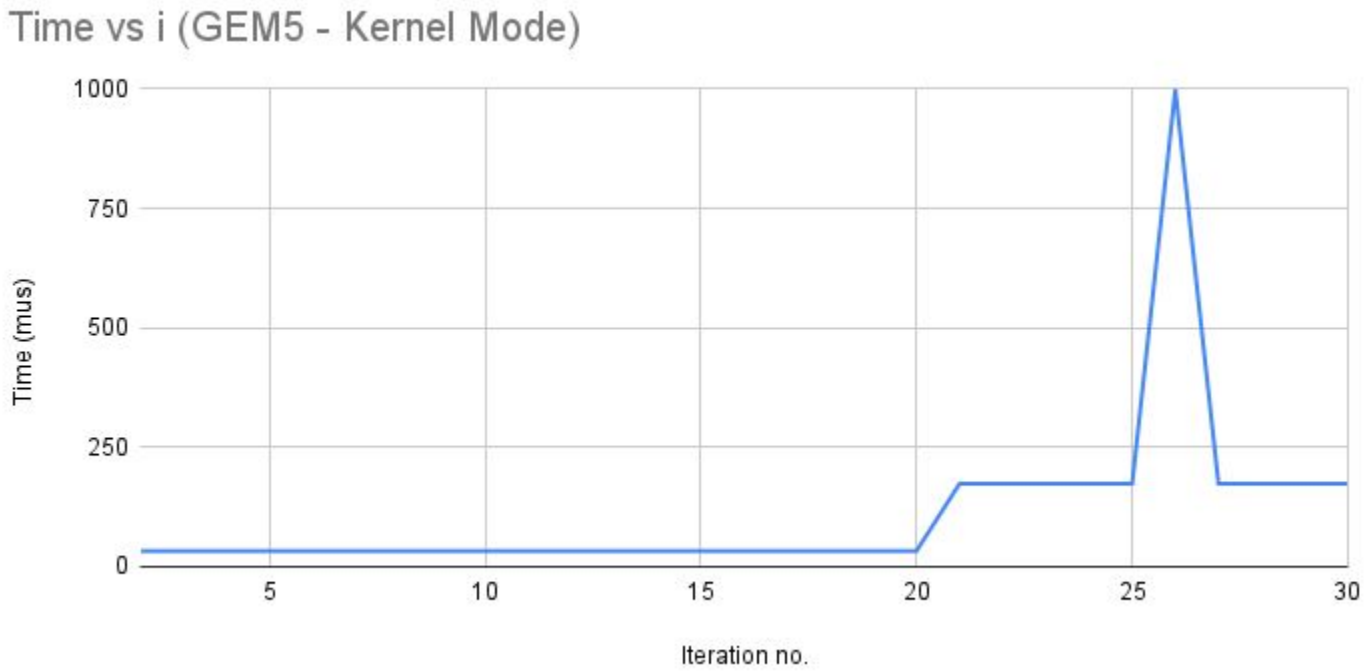
Gem5 Experiments

- We decided to validate the same experiments on Gem5 simulator which simulates both hardware and software.
- We used our modified kernel and code snippet and set up a workload which boots with KVM CPU (for faster experiments) and later switches to O3 CPU (for detailed out-of-order simulations). It uses ClassicCache instead of the default RubyCache to support *clflush* instructions.
- The experiments in Gem5 were consistent with real machine experiments
- The buffer was not fetched from user-space or kernel-space which implies that syscall completely forbids executing any instruction speculatively in user mode or kernel mode.

Gem5 Experiments: Results (User Mode Speculation)



Gem5 Experiments: Results (Kernel Mode Speculation)



Deep Dive into Syscall Instruction in Gem5

Key Points to Note,

1. Flags, IsNonSpeculative
   IsSerializeAfter
   IsSquashAfter
2. Total Microops: 25
3. Instruction to focus,
   Wrsel - (CS/SS reg)
   Wrbase - (CS/SS reg)
   Wrlimit - (CS/SS reg)
   Wrattr - (CS/SS reg)
   Wrip
   Wrflags

```
limm t1, "(uint64_t)(-1)", dataSize=8   nnn
rdip rcx                                nnn
rflags t2                               nnn
limm t3, "~RFBit", dataSize=8           nnn
and r11, t2, t3, dataSize=8             nnn
rdval t3, star                          nnn
srli t3, t3, 32, dataSize=8             nnn
andi t3, t3, 0xFC, dataSize=1           nnn
wrsel cs, t3                            yyn
wrbase cs, t0, dataSize=8               yyy
wrlimit cs, t1, dataSize=4              yyy
limm t4, val, dataSize=8                nnn
wrattr cs, t4                           yyy
addi t3, t3, 8                          nnn
wrsel ss, t3                            yyn
wrbase ss, t0, dataSize=8               yyn
wrlimit ss, t1, dataSize=4              yyn
limm t4, val, dataSize=8                nnn
wrattr ss, t4                           yyn
rdval t7, lstar, dataSize=8             nnn
wrip t0, t7, dataSize=8                 nnn
rdval t3, sf_mask, dataSize=8           nnn
xor t3, t3, t1, dataSize=8              nnn
and t3, t3, r11, dataSize=8             nnn
wrflags t3, t0                          yyn
```

O3pipeView Util Output

```
[..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
```

Base Syscall Performance and Security Analysis

- Highly serializing and restrictive in nature
- First 8 MicroOps execute in pipelined mode.
- In 9 MicroOps effects of flags came into play and serialisation start
- Speculation is happening at wrip
- SquashAfter flags insures the removal of userspace instruction after changing the CS reg.
- Takes 111 cycles (diff between first Insts commit and last Insts commit)
- Correct for all kernels

Version 1

IsNonSpeculative and IsSerialisingAfter flag is false for all MicroOps

```
limm t1, "(uint64_t)(-1)", dataSize=8   nnn
rdip rcx                                nnn
rflags t2                               nnn
limm t3, "~RFBit", dataSize=8           nnn
and r11, t2, t3, dataSize=8             nnn
rdval t3, star                          nnn
srli t3, t3, 32, dataSize=8             nnn
andi t3, t3, 0xFC, dataSize=1           nnn
wrsel cs, t3                            nnn
wrbase cs, t0, dataSize=8               nny
wrlimit cs, t1, dataSize=4              nny
limm t4, val, dataSize=8                nnn
wrattr cs, t4                           nny
addi t3, t3, 8                          nnn
wrsel ss, t3                            nnn
wrbase ss, t0, dataSize=8               nnn
wrlimit ss, t1, dataSize=4              nnn
limm t4, val, dataSize=8                nnn
wrattr ss, t4                           nnn
rdval t7, lstar, dataSize=8             nnn
wrip t0, t7, dataSize=8                 nnn
rdval t3, sf_mask, dataSize=8           nnn
xor t3, t3, t1, dataSize=8              nnn
and t3, t3, r11, dataSize=8             nnn
wrflags t3, t0                          nnn
```

O3pipeView Util 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
```

Version 1: Performance and Security Implication

- Correctly executed for our program
- Now we can see in the O3PipeView Util that some pipelining start to happening in SS reg MicroOps
- Only for CS reg registers MicroOps we are losing out some cycles
- Total Cycles Taken: 43 Cycles
- This is not perfectly correct for all execution because it may happen that some user space instruction is in ROB and we have changed the CS Reg(sel) which can lead to wrong execution or it may also happen that *wrflags* are not written and kernel instructions is using it. Same can happen for *wrsel*(SS reg)

Version 2

IsNonSpeculative, IsSerializeAfter and IsSquashAfter are false for all MicroOps except wrattr MicroOp for which IsSquashAfter flag is true for our syscall

```
limm t1, "(uint64_t)(-1)", dataSize=8    nnn
rdip rcx                                 nnn
rflags t2                                nnn
limm t3, "~RFBit", dataSize=8            nnn
and r11, t2, t3, dataSize=8              nnn
rdval t3, star                           nnn
srli t3, t3, 32, dataSize=8              nnn
andi t3, t3, 0xFC, dataSize=1            nnn
wrsel cs, t3                             yyn
wrbase cs, t0, dataSize=8                yyy
wrlimit cs, t1, dataSize=4               yyy
limm t4, val, dataSize=8                 nnn
wrattr cs, t4                            yyy
addi t3, t3, 8                           nnn
wrsel ss, t3                             yyn
wrbase ss, t0, dataSize=8                yyy
wrlimit ss, t1, dataSize=4               yyy
limm t4, val, dataSize=8                 nnn
wrattr ss, t4                            yyy
rdval t7, lstar, dataSize=8              nnn
wrip t0, t7, dataSize=8                  nnn
rdval t3, sf_mask, dataSize=8            nnn
xor t3, t3, t1, dataSize=8               nnn
and t3, t3, r11, dataSize=8              nnn
wrflags t3, t0                           yyn
```

O3pipeView Util 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
```

Version 2: Performance and Security Implication

- Correctly executed for our program
- Now we can see in the O3PipeView Util that some more pipelining start to happening
- Only on WrAttr we are losing out some performance
- Total Cycles Taken, 22 Cycles
- This is not perfectly correct for all execution because it may happen that some user space instruction is in ROB and we have changed the CS Reg which can lead to access of user-space address in kernel mode or some malicious user can exploit that to access kernel addr from userspace without segfault and there are implication on wrsel and wrflags also.

Version 3

All flags of WrAttr are preserved
And IsNonSpeculative, IsSerializingAfter and IssquashAfter flag is removed from rest of the MicroOps.

```
limm t1, "(uint64_t)(-1)", dataSize=8   nnn
rdip rcx                                nnn
rflags t2                               nnn
limm t3, "~RFBit", dataSize=8           nnn
and r11, t2, t3, dataSize=8             nnn
rdval t3, star                          nnn
srli t3, t3, 32, dataSize=8             nnn
andi t3, t3, 0xFC, dataSize=1           nnn
wrsel cs, t3                            nnn
wrbase cs, t0, dataSize=8               nnn
wrlimit cs, t1, dataSize=4              nnn
limm t4, val, dataSize=8                nnn
wrattr cs, t4                           yyy
addi t3, t3, 8                          nnn
wrsel ss, t3                            nnn
wrbase ss, t0, dataSize=8               nnn
wrlimit ss, t1, dataSize=4              nnn
limm t4, val, dataSize=8                nnn
wrattr ss, t4                           nnn
rdval t7, lstar, dataSize=8             nnn
wrip t0, t7, dataSize=8                 nnn
rdval t3, sf_mask, dataSize=8           nnn
xor t3, t3, t1, dataSize=8              nnn
and t3, t3, r11, dataSize=8             nnn
wrflags t3, t0                          nnn
```

O3pipeView Util 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
```

Version 3: Performance and Security Implication

- Correctly executed for our program
- Only on *WrAttr* we are losing out some performance than previous case as we have introduced more restriction by making flags IsSpeculative and IsSerializingAfter true.
- Total Cycles Taken: 31 Cycles
- This is not perfectly correct for all execution because it may happen that some user space instruction is in ROB and we have changed the CS Reg(selector) which can be used by the user space. It can also happen than we have not changed the *WrFlags* and we are using *WrFlags* in kernel mode.

Version 4

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

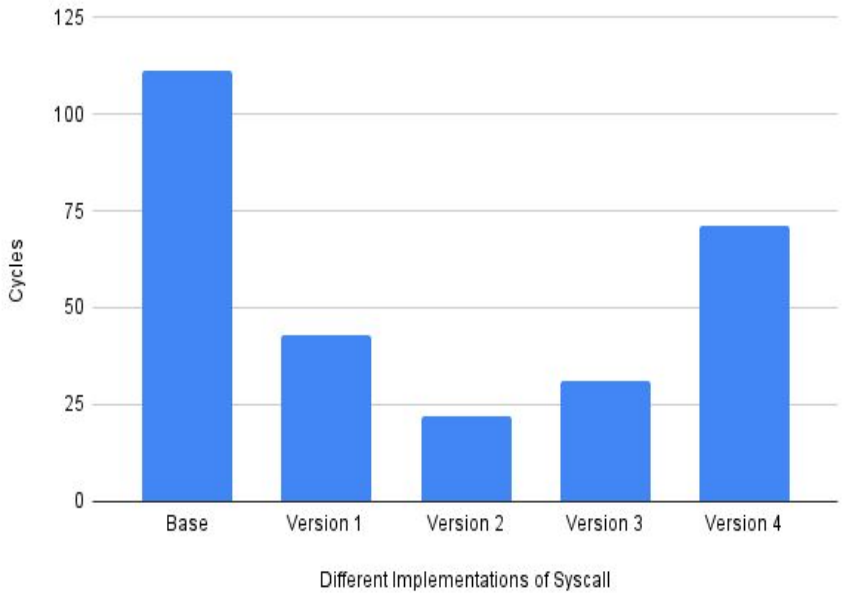
```
limm t1, "(uint64_t)(-1)", dataSize=8   nnn
rdip rcx                                nnn
rflags t2                               nnn
limm t3, "~RFBit", dataSize=8           nnn
and r11, t2, t3, dataSize=8             nnn
rdval t3, star                          nnn
srli t3, t3, 32, dataSize=8             nnn
andi t3, t3, 0xFC, dataSize=1           nnn
wrsel cs, t3                            yyn
limm t4, val, dataSize=8                nnn
wrattr cs, t4                           yyy
addi t3, t3, 8                          nnn
wrsel ss, t3                            yyn
limm t4, val, dataSize=8                nnn
wrattr ss, t4                           yyy
rdval t7, lstar, dataSize=8             nnn
wrip t0, t7, dataSize=8                 nnn
rdval t3, sf_mask, dataSize=8           nnn
xor t3, t3, t1, dataSize=8              nnn
and t3, t3, r11, dataSize=8             nnn
wrflags t3, t0                          yyn
```

Version 4: Performance and Security Implication

- Correctly executed for our program
- We are losing out performance by not making it pipelined but our implementation is correct for all kernels which don’t use Segmentation
- Total Cycles Taken: 71 Cycles
- This is correct for all the processor which don’t support Segmentation (As nowadays most OSes use page table for translations). Segment register base and limit are only for backward compatibility.

Conclusion

- This is well known tradeoff between security and performance.
- If we can design our specific processor for our OS than we can make use of Version3/Version4 to correctly implement syscall.



Thank You

We thank Professor Debadatta Mishra for giving us the opportunity to dive deep into the functioning of system calls in the UG project. We are very grateful for his timely help and guidance.