|  |  |  |
| --- | --- | --- |
| **OS [kernel mode]** | **Hardware** | **Process [User mode]** |
|  |  | Process A - Simple addtion program  000. pushq  %rbp  001.   movq    %rsp, %rbp  002.   movl    $5, -8(%rbp)  003.   movl    -8(%rbp), %eax  004.   addl    $5, %eax |
|  | **Timer Interrupt**  save reg(A) to k-Stack(A)  move to Kernel mode  move to trap handler |  |
| Handle the timer Trap  Call **switch()** routine  save reg(A) to PCB(A)  restore reg(B) from PCB(B)  switch to k-Stack(B)  return from trap(into B) |  |  |
|  | restore regs(B) from k-stack(B)  move to user mode  jump to B’s PC |  |
|  |  | Process B - read and add program  000. pushq  %rbp  001.    movq    %rsp, %rbp  002.    subq    $16, %rsp  003.    movq    %fs:40, %rax  004.    movq    %rax, -8(%rbp)  005.    xorl    %eax, %eax  006.    leaq    -16(%rbp), %rax  007.    movq    %rax, %rsi  008.    leaq    .LC0(%rip), %rdi  009.    movl    $0, %eax  **010.    call    \_\_isoc99\_scanf@PLT**  .  .  [The scanf() function in the C library invokes the read() system-level function call during it’s execution - int 0x80] |
|  | **System call interrupt - read()**  save reg(B) to k-Stack(B)  move to kernel mode  move to Trap handler |  |
| Handle the read Trap  Call **switch()** routine  save reg(B) to PCB(B)  restore reg(A) from PCB(A)  switch to k-Stack(A)  return from trap(into A) |  |  |
|  | restore regs(A) from k-stack(A)  move to user mode  jump to A’s PC |  |
|  |  | Process - A  005.   movl    %eax, -4(%rbp)  006.   movl    -4(%rbp), %eax  007.   popq    %rbp  008.   **ret** |
| Handle the exit Trap  Call the **switch()** routine  save reg(A) to PCB(A)  restore reg(B) from PCB(B)  switch to k-Stack(B)  return from trap(into B) |  |  |
|  |  | Process B - read and add program  011.    movl    -16(%rbp), %eax  012.    addl    $5, %eax  013.    movl    %eax, -12(%rbp)  014.    movl    -12(%rbp), %eax  015.    movq    -8(%rbp), %rdx  016.    xorq    %fs:40, %rdx  017.    je  .L3  **018.    call    \_\_stack\_chk\_fail@PLT**  019. .L3:  020.    leave  021.   ret |
| Handle the Trap  Call the **switch()** routine  save reg(B) to PCB(B)  restore reg(C) from PCB(C)  switch to k-Stack(C)  return from trap(into C) |  |  |

**The xv6 Context Switch Code**

1 # void swtch(struct context \*\*old, struct context \*new);

2 #

3 # Save current register context in old

4 # and then load register context from new.

5 .globl swtch

6 swtch:

7 # Save old registers

8 movl 4(%esp), %eax # put old ptr into eax

9 popl 0(%eax) # save the old IP

10 movl %esp, 4(%eax) # and stack

11 movl %ebx, 8(%eax) # and other registers

12 movl %ecx, 12(%eax)

13 movl %edx, 16(%eax)

14 movl %esi, 20(%eax)

15 movl %edi, 24(%eax)

16 movl %ebp, 28(%eax)

17

18 # Load new registers

19 movl 4(%esp), %eax # put new ptr into eax

20 movl 28(%eax), %ebp # restore other registers

21 movl 24(%eax), %edi

22 movl 20(%eax), %esi

23 movl 16(%eax), %edx

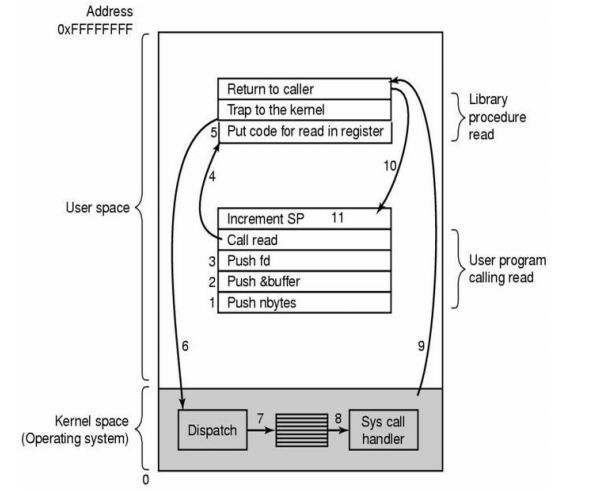
24 movl 12(%eax), %ecx

25 movl 8(%eax), %ebx

26 movl 4(%eax), %esp # stack is switched here

27 pushl 0(%eax) # return addr put in place

28 ret # finally return into new ctxt



**Linux Kernel code for synchronous IO read()**

static ssize\_t new\_sync\_read(struct file \*filp, char \_\_user \*buf, size\_t len, loff\_t \*ppos)

{

struct kiocb kiocb;

struct iov\_iter iter;

ssize\_t ret;

init\_sync\_kiocb(&kiocb, filp);

kiocb.ki\_pos = (ppos ? \*ppos : 0);

iov\_iter\_ubuf(&iter, ITER\_DEST, buf, len);

ret = call\_read\_iter(filp, &kiocb, &iter);

BUG\_ON(ret == -EIOCBQUEUED);

if (ppos)

\*ppos = kiocb.ki\_pos;

return ret;

}

**Linux kernel code for exit()**

void \_\_noreturn do\_exit(long code)

{

struct task\_struct \*tsk = current;

int group\_dead;

WARN\_ON(irqs\_disabled());

synchronize\_group\_exit(tsk, code);

WARN\_ON(tsk->plug);

kcov\_task\_exit(tsk);

kmsan\_task\_exit(tsk);

coredump\_task\_exit(tsk);

ptrace\_event(PTRACE\_EVENT\_EXIT, code);

user\_events\_exit(tsk);

validate\_creds\_for\_do\_exit(tsk);

io\_uring\_files\_cancel();

exit\_signals(tsk); /\* sets PF\_EXITING \*/

/\* sync mm's RSS info before statistics gathering \*/

if (tsk->mm)

sync\_mm\_rss(tsk->mm);

acct\_update\_integrals(tsk);

group\_dead = atomic\_dec\_and\_test(&tsk->signal->live);

if (group\_dead) {

/\*

\* If the last thread of global init has exited, panic

\* immediately to get a useable coredump.

\*/

if (unlikely(is\_global\_init(tsk)))

panic("Attempted to kill init! exitcode=0x%08x\n",

tsk->signal->group\_exit\_code ?: (int)code);

#ifdef CONFIG\_POSIX\_TIMERS

hrtimer\_cancel(&tsk->signal->real\_timer);

exit\_itimers(tsk);

#endif

if (tsk->mm)

setmax\_mm\_hiwater\_rss(&tsk->signal->maxrss, tsk->mm);

}

acct\_collect(code, group\_dead);

if (group\_dead)

tty\_audit\_exit();

audit\_free(tsk);

tsk->exit\_code = code;

taskstats\_exit(tsk, group\_dead);

exit\_mm();

if (group\_dead)

acct\_process();

trace\_sched\_process\_exit(tsk);

exit\_sem(tsk);

exit\_shm(tsk);

exit\_files(tsk);

exit\_fs(tsk);

if (group\_dead)

disassociate\_ctty(1);

exit\_task\_namespaces(tsk);

exit\_task\_work(tsk);

exit\_thread(tsk);

/\*

\* Flush inherited counters to the parent - before the parent

\* gets woken up by child-exit notifications.

\*

\* because of cgroup mode, must be called before cgroup\_exit()

\*/

perf\_event\_exit\_task(tsk);

sched\_autogroup\_exit\_task(tsk);

cgroup\_exit(tsk);

/\*

\* FIXME: do that only when needed, using sched\_exit tracepoint

\*/

flush\_ptrace\_hw\_breakpoint(tsk);

exit\_tasks\_rcu\_start();

exit\_notify(tsk, group\_dead);

proc\_exit\_connector(tsk);

mpol\_put\_task\_policy(tsk);

#ifdef CONFIG\_FUTEX

if (unlikely(current->pi\_state\_cache))

kfree(current->pi\_state\_cache);

#endif

/\*

\* Make sure we are holding no locks:

\*/

debug\_check\_no\_locks\_held();

if (tsk->io\_context)

exit\_io\_context(tsk);

if (tsk->splice\_pipe)

free\_pipe\_info(tsk->splice\_pipe);

if (tsk->task\_frag.page)

put\_page(tsk->task\_frag.page);

validate\_creds\_for\_do\_exit(tsk);

exit\_task\_stack\_account(tsk);

check\_stack\_usage();

preempt\_disable();

if (tsk->nr\_dirtied)

\_\_this\_cpu\_add(dirty\_throttle\_leaks, tsk->nr\_dirtied);

exit\_rcu();

exit\_tasks\_rcu\_finish();

lockdep\_free\_task(tsk);

do\_task\_dead();

}

**Local Timer Interrupt in Linux Kernel code**

static void local\_apic\_timer\_interrupt(void)

{

struct clock\_event\_device \*evt = this\_cpu\_ptr(&lapic\_events);

/\*

\* Normally we should not be here till LAPIC has been initialized but

\* in some cases like kdump, its possible that there is a pending LAPIC

\* timer interrupt from previous kernel's context and is delivered in

\* new kernel the moment interrupts are enabled.

\*

\* Interrupts are enabled early and LAPIC is setup much later, hence

\* its possible that when we get here evt->event\_handler is NULL.

\* Check for event\_handler being NULL and discard the interrupt as

\* spurious.

\*/

if (!evt->event\_handler) {

pr\_warn("Spurious LAPIC timer interrupt on cpu %d\n",

smp\_processor\_id());

/\* Switch it off \*/

lapic\_timer\_shutdown(evt);

return;

}

/\*

\* the NMI deadlock-detector uses this.

\*/

inc\_irq\_stat(apic\_timer\_irqs);

evt->event\_handler(evt);

}

**Deals with math errors in linux kernel code**

static void math\_error(struct pt\_regs \*regs, int trapnr)

{

struct task\_struct \*task = current;

struct fpu \*fpu = &task->thread.fpu;

int si\_code;

char \*str = (trapnr == X86\_TRAP\_MF) ? "fpu exception" :

"simd exception";

cond\_local\_irq\_enable(regs);

if (!user\_mode(regs)) {

if (fixup\_exception(regs, trapnr, 0, 0))

goto exit;

task->thread.error\_code = 0;

task->thread.trap\_nr = trapnr;

if (notify\_die(DIE\_TRAP, str, regs, 0, trapnr,

SIGFPE) != NOTIFY\_STOP)

die(str, regs, 0);

goto exit;

}

/\*

\* Synchronize the FPU register state to the memory register state

\* if necessary. This allows the exception handler to inspect it.

\*/

fpu\_sync\_fpstate(fpu);

task->thread.trap\_nr = trapnr;

task->thread.error\_code = 0;

si\_code = fpu\_\_exception\_code(fpu, trapnr);

/\* Retry when we get spurious exceptions: \*/

if (!si\_code)

goto exit;

if (fixup\_vdso\_exception(regs, trapnr, 0, 0))

goto exit;

force\_sig\_fault(SIGFPE, si\_code,

(void \_\_user \*)uprobe\_get\_trap\_addr(regs));

exit:

cond\_local\_irq\_disable(regs);

}