



Error Sensitivity of Linux on PowerPC (G4) & Pentium (P4)

W. Gu, Ravi K. Iyer
Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign



Motivation

- Linux kernel responds to transient errors that impact kernel code, kernel data, kernel stack, and processor system registers, and
- How processor hardware architecture (instruction set architecture and register set) impacts kernel behavior in the presence of errors
- Approach: compare Linux behavior under errors on two different platforms
 - Intel Pentium 4 (P4) running RedHat Linux 9.0
 - Motorola PowerPC (G4) running YellowDog Linux 3.0.
- Important in:
 - establishing benchmarking procedure for analyzing and comparing different platforms
 - facilitating analysis of *costs–reliability–performance* tradeoffs in selecting a computing platform



Approach

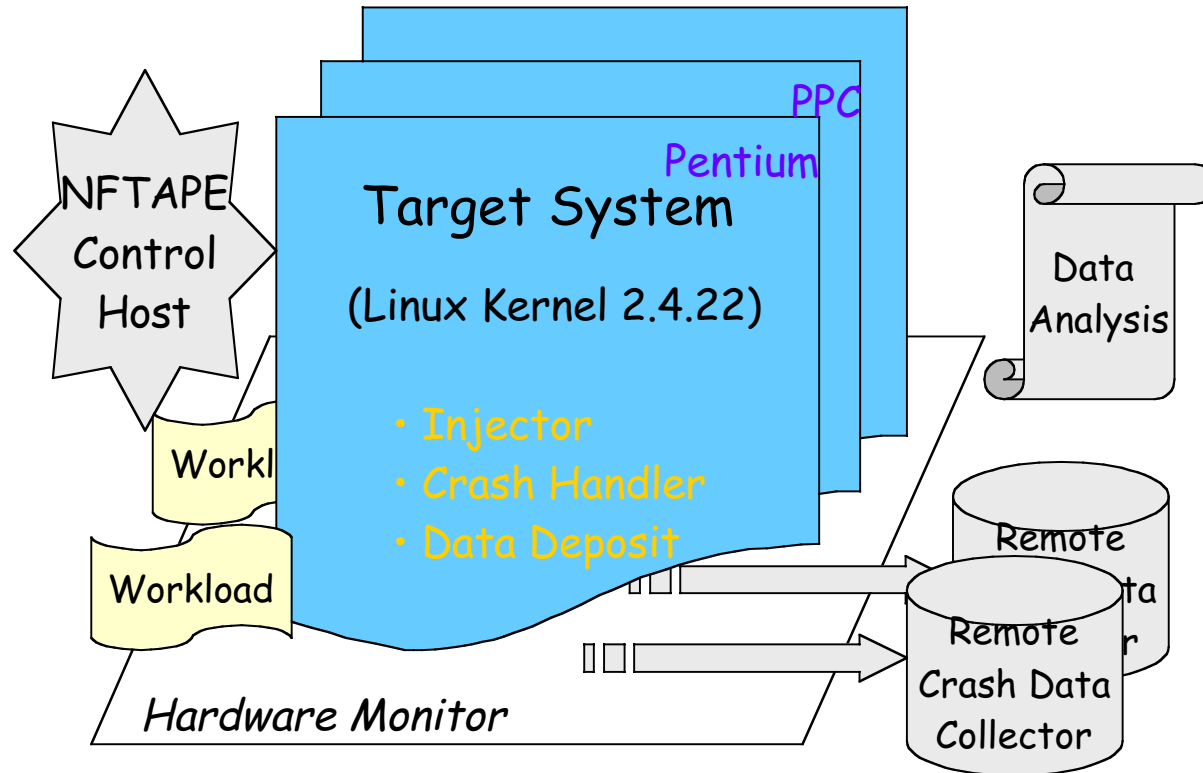
- Automated error injection into the instruction stream of the kernel code
 - Over 115,000 errors injected into the kernel code, data, stack, and CPU registers.
- Error model
 - Transients emulated by injecting single bit errors into the kernel and CPU registers
 - Error origin not presumed – injections reflect the ultimate fault impact on the system behavior
 - Error location pre-selected based on profiling



Major Findings

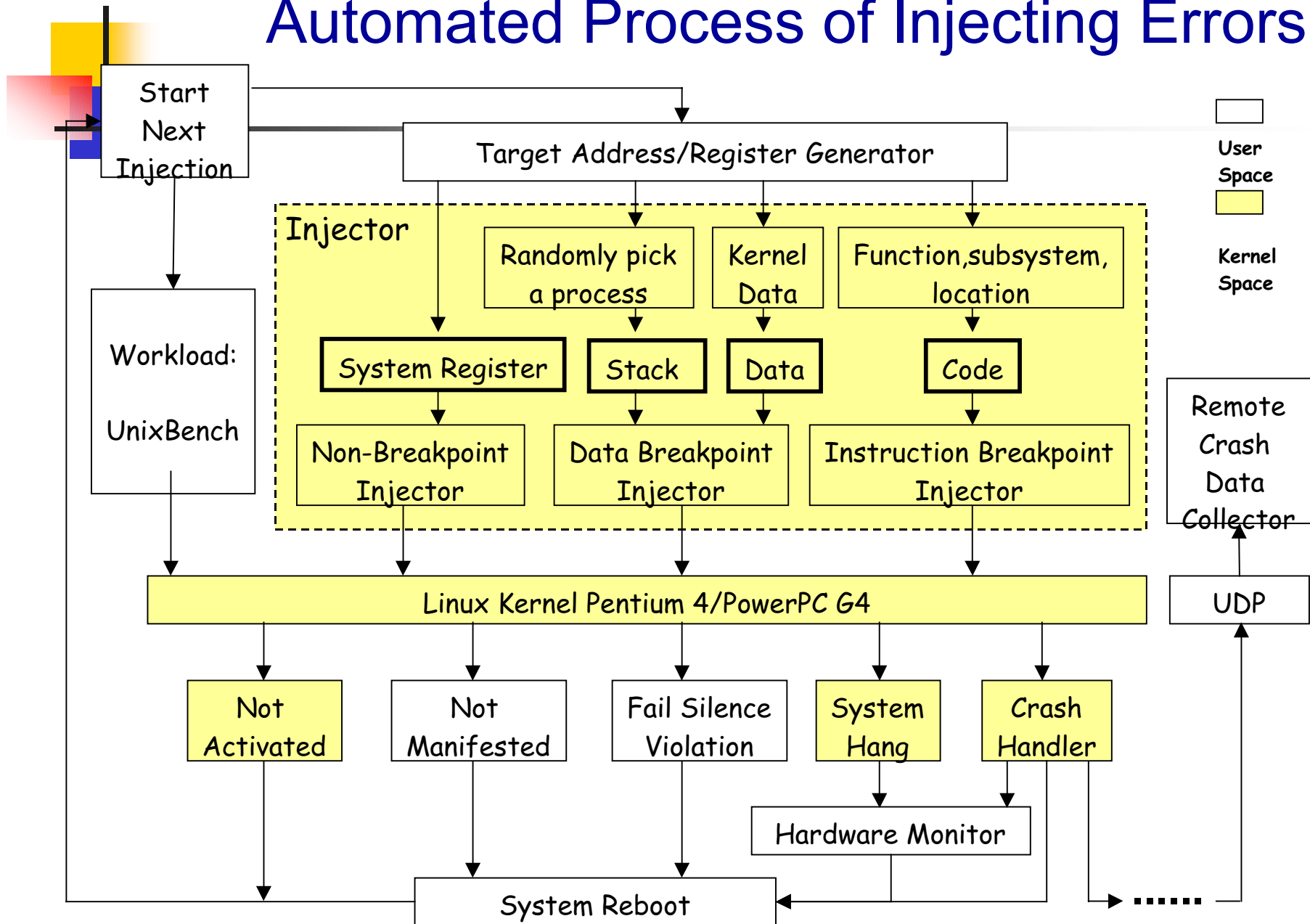
- Error activations similar for both processors – manifestation percentages for the Pentium P4 twice as high.
- Significant difference between the two processors in manifestation percentages for
 - Stack errors: 56% for P4 and 21% for G4.
 - Kernel data: 66% for P4 and 21% for G4
- Variable length instruction format on P4 allows a bit error to alter a single instruction into a sequence of multiple valid instructions.
 - Can lead to poorer diagnosability
 - Can reduce crash latency (fail fast)
- Less compact fixed 32-bit data and stack access on the G4 can contribute to error masking, i.e., sparsity of the data can mask errors
- More optimized access patterns on P4 can increase the chances that an error is detected at the time the data item is accessed/used.
- Important for performance/reliability tradeoff.

Experimental Setup



Processor	Hardware			System Software			Injection Tool
	Desktop	CPU Clock	Memory	Distribution	Linux Kernel	Compiler	
Intel Pentium 4	Dell	1.5GHz	256MB	RedHat 9.0	2.4.22	GCC 3.2.2	NFTAPE
Motorola PowerPC G4	Apple	1.0GHz	256MB	YellowDog 3.0	2.4.22		

Automated Process of Injecting Errors





Error Injections and Outcome Categories

- Single-bit errors are injected into:
 - instructions of the target kernel functions,
 - stack of the corresponding kernel process,
 - kernel data structures,
 - CPU's system registers.

Outcome Category	Description
Activated	The corrupted instruction/data is executed/used.
Not Manifested	The corrupted instruction/data is executed/used, however it does not cause a visible abnormal impact on the system.
Fail Silence Violation	Either operating system or application erroneously detects the presence of an error or allows incorrect data/response to propagate out.
Crash	Operating system stops working, e.g., bad trap or system panic. Crashes are divided into two groups: (i) <i>Crash – cause known</i> : a detailed dump information collected and (ii) <i>Crash – cause unknown</i> : no dump information collected
Hang	System resources are exhausted resulting in a non-operational system, e.g., deadlock.

Statistics on Error Activation and Failure Distribution

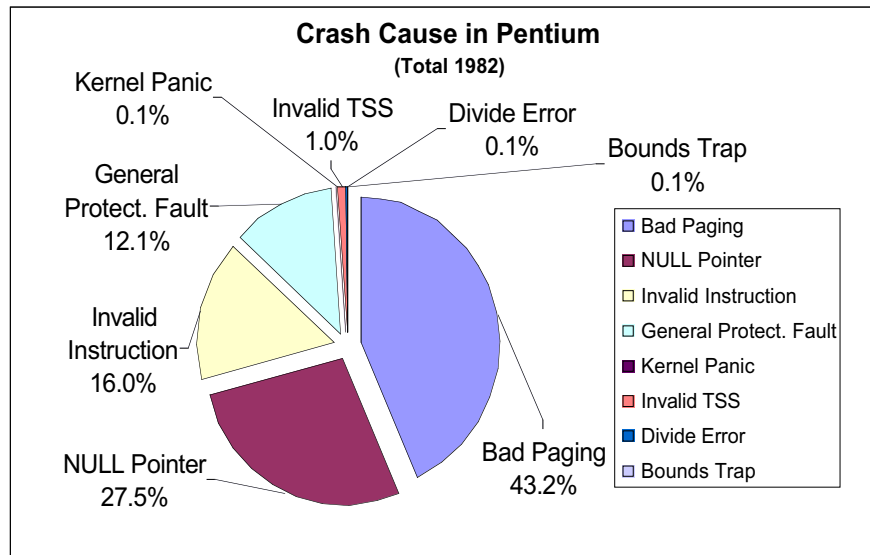
Intel Pentium 4 Campaign	Injected	Error Activated (Percentage)	Activated			
			Not Manifested	Fail Silence Violation	Known Crash	Hang/Unknown Crash
Stack	10143	2973(29.3%)	1305(43.9%)	0(0%)	1136(38.2%)	532(17.9%)
System Register	3866	N/A	3459(89.5%)	0(0%)	305(7.9%)	102(2.6%)
Data	46000	226(0.5%)	77(34.1%)	0(0%)	96(42.5%)	53(23.4%)
Code	1790	982(54.9%)	308(31.4%)	13(1.3%)	455(46.3%)	216(22.0%)
Total	61799					

Motorola PPC G4 Campaign	Injected	Error Activated (Percentage)	Activated			
			Not Manifested	Fail Silence Violation	Known Crash	Hang/Unknown Crash
Stack	3017	1203(29.3%)	949(78.9%)	0(0%)	172(14.3%)	84(7.0%)
System Register	3967	N/A	3774(95.1%)	0(0%)	69(1.7%)	124(3.1%)
Data	46000	704(1.5%)	551(78.3%)	7(1.0%)	55(7.8%)	91(12.9%)
Code	2188	1415(64.7%)	580(41.0%)	33(1.5%)	576(40.7%)	226(16.0%)
Total	55172					

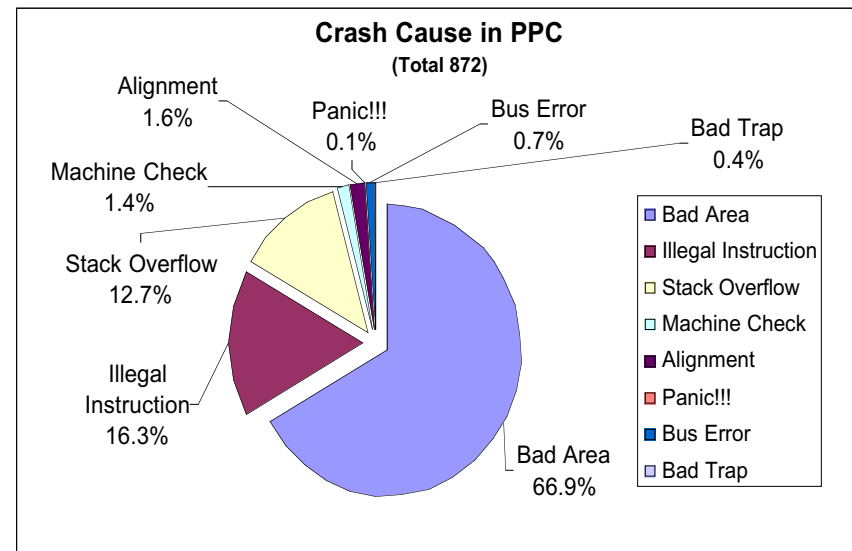
Distributions of Crash Causes

Linux kernel 2.4.22

Intel Pentium 4



Motorola PPC G4



- NULL Pointer: NULL pointer de-reference;
- Bad Paging: Other bad paging except NULL pointer;
- General Protection Fault: Exceeding segment limit;
- Kernel Panic: Operating system detects an error;
- Invalid TSS: Selector, or code segment is outside table limit;
- Bounds Trap: Bounds checking error.

- Bad Area: Bad paging access including NULL pointer;
- Stack Overflow: Stack pointer of a process is out of range;
- Machine Check: Errors on the processor-local bus;
- Alignment: Load/Store operands are not word-aligned;
- Bus Error: Protection faults;
- Bad trap: Unknown exceptions.

Stack Injection: An Example of Error Propagation Due to Undetected Stack Overflow

mm/page_alloc.c <__free_pages_ok>: ... ①
 c013ec65: 8d 65 f4 lea 0xffffffff4(%ebp),%esp }
 c013ec68: 5b pop %ebx
 c013ec69: 5e pop %esi
 c013ec6a: 5f pop %edi
 c013ec6b: 5d pop %ebp
 c013ec6c: c3 ret

Original Code

mm/page_alloc.c <__free_pages_ok>: ...
 c013ec65: 8d 64 f4 5b lea 0x5b(%esp,%esi,8),%esp
 c013ec69: 5e pop %esi
 c013ec6a: 5f pop %edi
 c013ec6b: 5d pop %ebp
 c013ec6c: c3 ret

Corrupted Code

② Return Address (in the stack) pattern:

c0119cb2 c0107784 c010799a c0108067 c0119cb2 c0107784 c010799a c0108067 c0119cb2 c0107784 c010799a c0108067
 c0119cb2 c0107784 c010799a c0108067 c0119cb2 c0107784 c010799a c0108067 c0119cb2 c0107784 c010799a c0108067
 c0119cb2 c0107784 c010799a c0108067 c0119cb2 ...

net/core/skbuff.c <alloc_skb>: ...
 c02abf1b: 8b 8a e0 7a 43 c0 mov 0xc0437ae0(%edx),%ecx
 c02abf21: 31 c0 xor %eax,%eax
 c02abf23: 39 d9 cmp %ebx,%ecx
 c02abf25: 74 27 je c02abf4e <alloc_skb+0xae>
 c02abf27: 89 c8 mov %ecx,%eax
 c02abf29: 8b 08 mov (%eax),%ecx



③ Unable to handle kernel paging request at virtual address 170fc2a5

%eax=0x170fc2a5

Crash Latency=13116444

Date Injections – Example Crash due to Illegal Instruction (P4)

C code to use the data:

```
static inline void spin_unlock(  
    *lock)  
{  
    #if SPINLOCK_DEBUG  
        if (lock > magic !=  
            SPINLOCK_MAGIC)  
            BUG();  
        if (!spin_is_locked(lock))  
            BUG();  
    #endif  
    __asm__ __volatile__(  
        spin_unlock_string  
    );
```

<sys_ioctl>:

Corresponding Assembly:

```
c015852d: 85 c0  
c015852f: 79 07  
c0158531: 6a 42  
c0158533: e8 78 6a fc ff  
c0158538: 48  
c0158539: 85 c0  
c015853b: 89 42 1c  
c015853e: 79 2e  
c0158540: 81 3d c4 5b 37 c0 ad  
c0158547: 4e ad de  
c015854a: 74 08  
c015854c: 0f 0b  
...
```

Pentium 4 kernel Data Section:

```
c0375bc0 <kernel_flag_cacheline>:  
c0375bc0: 01 00  
c0375bc2: 00 00  
c0375bc4: ad  
c0375bc5: 4e One bit flip in data  
c0375bc6: ad changes 4E to 0E  
c0375bc7: de 00
```

2 **cmpl** generates wrong flags because of bad data

```
test    %eax,%eax  
jns    c0158538 <sys_ioctl+0x2d8>  
push   $0x42  
call   c011efb0 <__out_of_line_bug>  
dec    %eax  
test   %eax,%eax  
mov    %eax,0x1c(%edx)  
jns    c015856e <sys_ioctl+0x30e>  
cmpl  $0xdead4ead,0xc0375bc4  
  
je     c0158554 <sys_ioctl+0x2f4>  
ud2a
```

3 **Illegal Instruction**

Example of Consequences of Kernel Stack Injection (P4)

```
#define TASK_STOPPED 8
int kupdate(void *startup)
{ ...
  for (;;) {
    /* update interval */
    if (interval) {
      tsk->state = TASK_INTERRUPTIBLE;
      schedule_timeout(interval);
    } else {
      tsk->state = TASK_STOPPED;
      schedule(); /* wait for SIGCONT */
    }
    /* check for sigstop */
    if (signal_pending(tsk)) {
      int stopped = 0; ...
    }
    sync_old_buffers();
    run_task_queue(&tq_disk);
  }
}
```

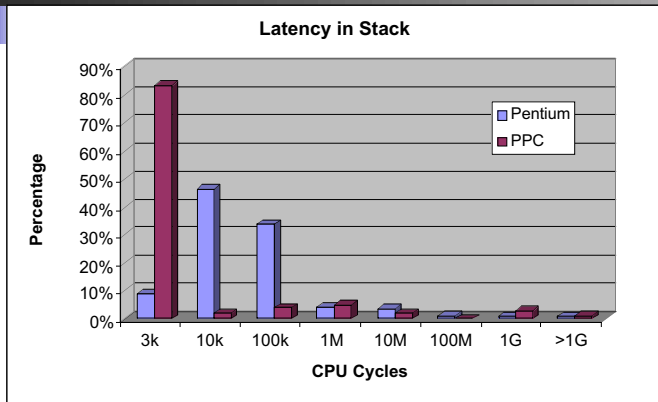
Crash latency is 12864 cycles

Machine Code	Assembly	①
8b 45 e0	mov 0xffffffff0(%ebp), %eax	
c7 00 08 00 00	movl \$0x8, (%eax)	
00		
e8 12 e3 fc ff	call c011ad00 <schedule>	
8b 55 e0	mov 0xffffffff0(%ebp), %edx	
8b 4a 08	mov 0x8(%edx), %ecx	③
85 c9	test %ecx, %ecx	
0f 84 ca 00 00	je c014cac6<kupdate+0x1f6>	
00		

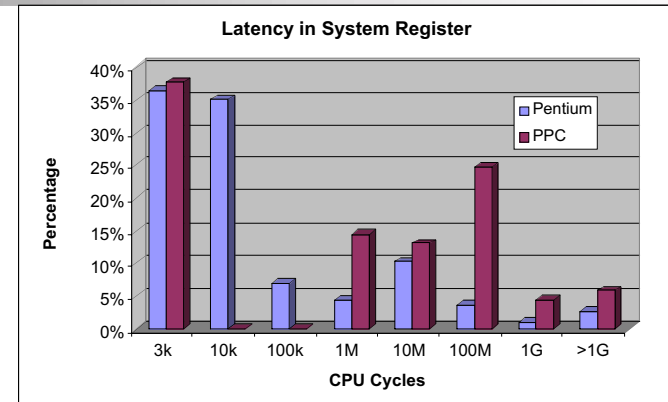
Unable to handle kernel NULL pointer at virtual address 00000008

- ① Get address of data structure tsk from stack at location of 0xffffffff0(%ebp). But One bit flip in stack alters the addr saved in EAX.
- ② Using wrong EAX to set "tsk->state."
- ③ Crash because EDX coming from stack now is 0.

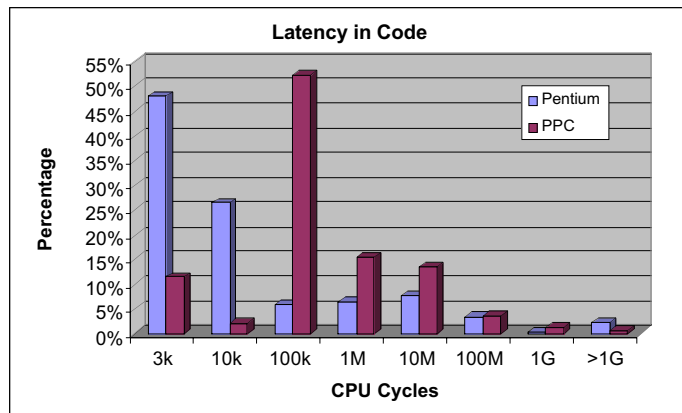
Distribution of Cycles-to-crash



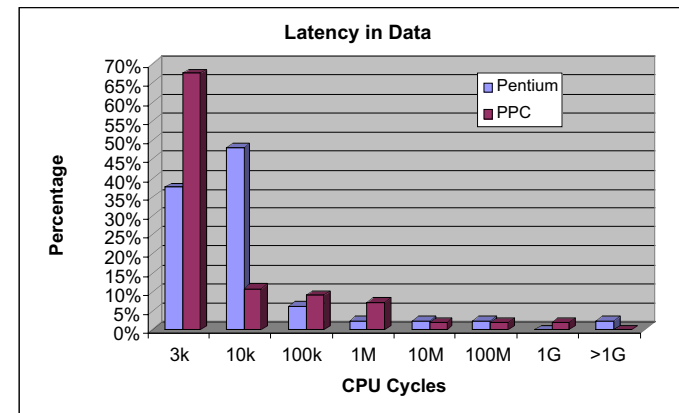
(A) Stack Error Injection



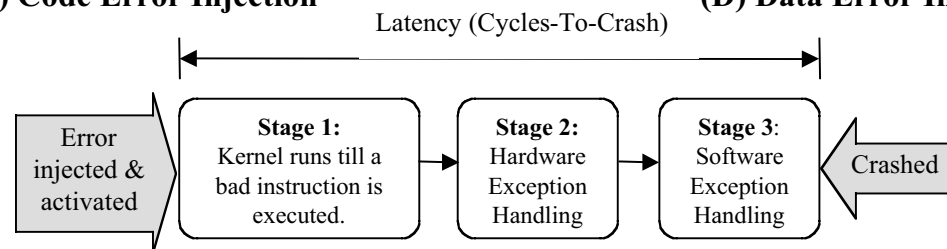
(B) Register Error Injection



(C) Code Error Injection



(D) Data Error Injection





Crash Latency – Major Results

- 80% of stack errors
 - short-lived (less than 3,000 cycles) on G4
 - longer crash latency (3,000 to 100,000 cycles) on P4
 - Reason a disparity in the way the two platforms handle exceptions, e.g., kernel on the G4 provides quick detection of stack overflow errors, on P4 converts stack overflow events into other types of exceptions (e.g., bad paging)
- Opposite trend in crash latency of errors impacting the kernel code
 - On P4 – 45% of errors are short-lived (less than 3,000 cycles),
 - On G4 – 50% of errors have latency between 10,000 and 100,000 cycles.
 - Dissimilarity due to the differences in the number of general-purpose registers provided by the two processors (32 on the G4; 8 on the P4).

Example of Variation in Crash Latency (G4)

- A bit error in the `sys_read()` function (kernel code on G4).
- The error transforms the `mflr` instruction to `lhax`.
- The system crashes (kernel access of a bad area) due to an illegal address generated by `lhax r0, r8, r0` (`gpr8 + gpr0`).

```
c0048fac <sys_read>:
c0048fac:      94 21 ff e0    stwu  r1,-32(r1)
c0048fb0:      7c 08 02 a6    mflr  r0      } Original Code

c0048fac <sys_read>:
c0048fac:      94 21 ff e0    stwu  r1,-32(r1)
c0048fb0:      7c 08 02 ae    lhax  r0,r8,r0
```

One bit flipped → "kernel access of bad area"

One bit flip *changes* original code
"mflr r0(copy the contents of the LR to r0)"
TO
"lhax r0,r8,r0(load half word algebraic indexed)"
 $gpr0 \leq (gpr8 + gpr0)_{16}$

Depending on the contents of `gpr0` and `gpr8`,
crash latency varies from 1285 cycles
(209 instructions) to 424256 cycles
(230958 instructions)



Conclusions

- Less dense data access of the stack and data segment makes G4 platform less sensitive to errors.
- Although a similar number of errors are activated on both platforms, a much smaller number manifests on G4 than on P4 platform.
- More optimized (from 1-4 bytes) access patterns on the P4 ensure that if an error location is accessed it is more likely to lead to problems.
- Variable length instruction format on P4 allows a bit error to alter a single instruction into a sequence of multiple valid instructions.
 - Can lead to poorer diagnosability
 - Can reduce crash latency (fail fast)
- Enhanced interaction between the hardware and the operating system can improve error detectability and diagnosability.
 - e.g., stack overflow detection, could be added by extending the semantics of PUSH and POP instructions on P4 to enable checking for a memory access beyond the currently allocated space for the stack.