1

Virtual Pages – partitions of virtual memory into fixed size blocks

- page hit: allocated in DRAM

- page fault: unallocated page (not yet created) or allocated (not cached in DRAM) \*not SRAM cache

swapping/paging: (disk to memory and back)

-every time you bring in a page, you nee to bring one out

thrashing: result of if working set doesn’t fit into memory

process of converting – address translation

DRAM misses are expensive, 10x slower than SRAM. disks are 100,000x slower than DRAM

DRAM always uses write-back instead of write-through, because of large access time of disk

Multi-Core (cont)

Concurrency: logical control flows overlap in time

useful in:

- accessing slow i/o devices – applications can overlap useful work with i/o requests

- interacting with humans – multitasking, every time user requests action, a concurrent logical flow is created to run

- reducing latency by deferring work – …by deferring other operations and performing them concurrently.

- servicing multiple network clients

- computing in parallel on multicore machines

Building concurrent programs:

- processes – each logical control flow is a process that is scheduled and maintained by the kernel

- i/o multiplexing – applications schedule their own logical flows. logical flows are modeled as state machines that the main program explicitly transitions from state to state as a result of data arriving on file descriptors.

- threads – logical flows that run in the context of a single process are scheduled by the kernel. “Hybrid of other two approaches”, scheduled by kernel like process flows, and sharing same virtual address space like i/o multiplexing flows.

Threading:

race condition: if they run at the same time, they compete for memory read/writes

RISC vs CISC

RISC:

-Reduced instruction set computer

-Less instr. & equal efficiency

-smaller hardware

-less power

e.g. MIPS, SPARC, PowerPC, ARM

- no instruction with long execution time

- all instructions same length (4bytes in MIPS)

- arithmetic, logical operations only on reg

- no condition codes

- many more registers (MIPS – 32+2hi/lo)

- register intensive procedure linkage

- reg’s used for some arg’s and return addr’s

CISC:

-complex instruction set computer

-e.g. IA32 x64-64

-backwards compatibility

- some instructions with long execution time

- variable length encodings IA32 can range from 1-15 bytes

- multiple formats for specifying operands

- arithmetic and logical operations an be applied to both memory and register operands

implementation artifacts hidden from machine level programs

- condition codes (flags)

- stack intensive procedure linkage

Thwarting Buffer Overflow Attacks:

1. stack randomization – make position in stack vary from one program run to another

security monoculture – systems vulnerable to same strain of virus (diversity = harder for hackers to attack)

2. stack corruption detection

- “call \_\_stack\_chk\_fail”

3. limit executable code regions

----------------------------------------

MIPS32 Instructions.

d = double word, w = word

h = halfword, hu = halfword unsign

b = byte, bu = byte unsigned

|  |  |
| --- | --- |
| add $d, $s, $t | $d = $s + $t |
| addu $d, $s, $t | $d = $s + $t uns. |
| sub $d, $s, $t | $d = $s - $t |
| subu (unsigned |  |
| addi $t, $s, C | $t = $s + C-sind |
| addiu $t, $s, C | same ^ |
| mult $s, $t | LO = (($s \* $t) << 32) >> 32;  Hi = ($s \* $t) >> 32; |
| div $s, $t | LO = $s/ $t  HI = $s % $t |
| divu $s, $t | unsigned |
| ld $t, C($s) | $t=Mem[$s+c] |
| lw $t, C($s) | same ^ |
| lh $t, C($s) | signed same ^ |
| lhu | unsigned same^ |
| lb, lbu |  |
| sd $t, C($s) | Mem[$s+C]= $t |
| sw, sh, sb | same ^ |
| lui $t, C –load upper immed. | $t = C << 16 |
| mfhi $d | $d = HI (reg) |
| mflo $d | $d = LO (reg) |
| and, andi, or, ori | xor, nor |
| slt $d, $s, $t | $d = ($s < $t) |
| slti $t, $s, C | $t = ($s < C) |
| sll $t, $s, C | $t = $s << C(log) |
| srl $t, $s, C | $t = $s >> C(log) |
| sra $t, $s, C | arithmetic |
| beq $s, $t, C | if ($s==$t) go to PC+4+4\*C |
| bne $s, $t, C | if ($s!=$t) go to PC+4+4\*C |
| j C | jump |
| jr $s | goto add $s |
| jal C | jump and link |
| move $rt, $rs | R[rt] = R[rs] |

Classes of exceptions (abrupt change in control flow):

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Cause | Async/Sync | Return behavior |
| Interrupt | Signal from i/o device | Async | Always returns to next instr. |
| Trap | Intentional exception | Sync | Always returns to next instr. |
| Fault | Potentially recoverable error | Sync | Might return to current instr. |
| Abort | Nonrecoverable error | sync | Never returns |

Interrupt ex: network packet arriving. “ctrl z” on linux

Trap ex: exit(0)

Fault ex: an instruction references a virtual address whose corresponding physical page is not in memory, and must be retrieved from disk. (page fault, segmentation fault)

Abort ex: SRAM or DRAM bit corrupted

IA32 has 256 exceptions.

Multi-Core:

Microprocessors have become smaller, denser, and more powerful.

Why multicore?

Limits of Moore’s

1. Power density

Parallelism saves power

Using additional cores:

-Increase density (= more transistors =more capacitance)

-Can increase cores (2x) and performance (2x)

-Increase cores (2x), but decrease frequency (1/2) same performance at ¼ power

2. Hidden Parallelism Tapped Out

- ½ due to transistor density

- ½ due to architecture changes (ILP)

- superscalar were state of the art

-multiple instruction issue

- dynamic scheduling: hardware discovers parallelism between instructions

- speculative execution: look past predicted branches

- non-blocking caches: multiple outstanding memory operations

- these sources have been used up

3. Chip Yield

- manufacture costs and yield problems limit use of density

- Moore’s (Rock’s) 2nd law: fabrication costs up

-Yield (%usable chips) drops

-parallelism can help

Revolution is Happening Now

-chip density is increasing 2x every 2 years

-clock speed isn’t

-# of processor cores may double instead

- little to no hidden parallelism to be found

- parallelism must be exposed to and manage by software