Consider the following program.

#include<stdio.h>

int a[16] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

int result;

int main(){

register int i, sum=0;

for(i=0;i<16;i++) sum += a[i];

result = sum;

printf("a is %p, result is %d\n",a,result);

return 0;

}

When compiled with gcc –S –O on titan1, the part of the assembly file containing the loop is:

main:

.LFB23:

.cfi\_startproc

movl $a, %eax

movl $a+64, %edx

movl $0, %ecx

.L2:

addl (%rax), %ecx

addq $4, %rax

cmpq %rdx, %rax

jne .L2

A trace of the memory accesses can be obtained by the lackey tool in valgrind:

% valgrind --tool=lackey --trace-mem=yes --log-file=looptrace.out ./a.out

a is 0x601060, result is 136

The first four iterations of the loop produce the following trace of memory accesses. The virtual addresses (in 8-digit hexadecimal format) are tagged with the access type: instruction fetches (“I”), loads (“L”), and stores (“S”). The number after a comma is the number of bytes accessed. Since this trace was produced on an x86, the instruction bytes can vary from 1 to 7. (E.g., “I 00400555,2” means an instruction was fetched from address 0x00400555, the instruction was two bytes in length, and the PC will be updated to 00400555 + 2 = 00400557 for the next instruction fetch.) The data accesses in this section of the trace are all loads (“L”) and are all words (4 bytes). (The loads access below are indented more than what lackey produces.)

I 00400555,2

L 00601060,4

I 00400557,4

I 0040055b,3

I 0040055e,2

I 00400555,2

L 00601064,4

I 00400557,4

I 0040055b,3

I 0040055e,2

I 00400555,2

L 00601068,4

I 00400557,4

I 0040055b,3

I 0040055e,2

I 00400555,2

L 0060106c,4

I 00400557,4

I 0040055b,3

I 0040055e,2

1. If the pages are 4 KiB in size, what is the virtual page number on which the instructions are found?

Virtual Address = PFN | Offset

# of bits in offset = log2(4 KiB) = 12 bits

PFN is field left of the 12-bit offset.

PFN = 400

2. Do you see spatial locality in the instruction fetches? If so, explain.

Yes, the instruction addresses are nearby: offsets within pages 555, then 557, then 55b, then 55e

3. Do you see temporal locality in the instruction fetches? If so, explain.

Yes, the instructions in the loop are being used over and over again. For example, the offset of 555 is repeated on each loop iteration.

4. Suppose an application is assigned four frames of physical memory (PFNs 1-4), and these page frames are initially empty. The application then references pages in the following sequence (using letters for VPN references in the manner of the textbook). Using the diagram format of the textbook, show how the system would fault pages into the four frames of physical memory using an ascending placement policy when empty page frames exist and then the FIFO replacement policy once there are no more empty page frames. (The textbook shows the VPN letter to designate a miss and shows a + to designate a hit.)

VPN references: A C B D B A E F B F A G E F A

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | C | B | D | B | A | E | F | B | F | A | G | E | F | A |
| 1 | A |  |  |  |  | + | E |  |  |  |  |  | + |  |  |
| 2 |  | C |  |  |  |  |  | F |  | + |  |  |  | + |  |
| 3 |  |  | B |  | + |  |  |  | + |  | A |  |  |  | + |
| 4 |  |  |  | D |  |  |  |  |  |  |  | G |  |  |  |

5. Repeat but use the LRU replacement policy.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | C | B | D | B | A | E | F | B | F | A | G | E | F | A |
| 1 | A |  |  |  |  | + |  |  |  |  | + |  |  |  | + |
| 2 |  | C |  |  |  |  | E |  |  |  |  | G |  |  |  |
| 3 |  |  | B |  | + |  |  |  | + |  |  |  | E |  |  |
| 4 |  |  |  | D |  |  |  | F |  | + |  |  |  | + |  |

6. True LRU is expensive in time and/or extra hardware to implement, and thus several replacement policies have been proposed that can approximate LRU with minimum extra cost in time and hardware. Describe at least one possible implementation of true LRU and then explain why it is expensive.

Skipped in Class