ECS 150 Homework 4

Hardy Jones
999397426
Professor Levitt
Winter 2015

1 Let's assume we have the following pages for the elements of each array:

elements			A	B
1	-	100	1	11
101	-	200	2	12
201	-	300	3	13
301	-	400	4	14
401	-	500	5	15
501	-	600	6	16
601	-	700	7	17
701	-	800	8	18
801	-	900	9	19
901	-	1000	10	20

(a) The page reference string looks similar to

1 , 11 , 10 , 1 , 11 , 10 , ...repeated 98 more times.
2 , 12 , 9 , 2 , 12 , 9 , ...repeated 98 more times.
3 , 13 , 8 , 3 , 13 , 8 , ...repeated 98 more times.
4 , 14 , 7 , 4 , 14 , 7 , ...repeated 98 more times.
5 , 15 , 6 , 5 , 15 , 6 , ...repeated 98 more times.
6 , 16 , 5 , 6 , 16 , 5 , ...repeated 98 more times.
7 , 17 , 4 , 7 , 17 , 4 , ...repeated 98 more times.
8 , 18 , 3 , 8 , 18 , 3 , ...repeated 98 more times.
9 , 19 , 2 , 9 , 19 , 2 , ...repeated 98 more times.
10 , 20 , 1 , 10 , 20 , 1 , ...repeated 98 more times.

(b) i. Using FIFO, nearly every time we access a new page it is a page fault. The reason for so many page faults is every three sequential page accesses are different, with one exception. When we access 5, 15, 6, 6, 16, 5, there is one group of three where we access the same page twice. This is the only instance where we do not page fault.
We can compute this value simply: in each row we perform three page accesses 100 times, we have 10 rows, minus one access that doesn't miss.
So the number of page faults is $3 \cdot 100 \cdot 10 - 1 = 2999$.

ii. Using MIN, for each row, we can keep one page in memory, use the register to hold the addition temporarily, and swap between the two other pages. With the first row as an example, we have 1, 11, 10, 1, 11, 10. If we keep page 1

in memory, we would just swap out 11 and 10. So we miss the first access to page 1, then every access to pages 11 and 10. So each row has $1+2\cdot 100 = 201$ misses. Again, we do not have a miss when we switch from row 5 to row 6. So the number of page faults is $(1 + 2 \cdot 100) \cdot 10 - 1 = 2000$.

2 §4.11 Since there are 32 bits in an address, and we have 20 of the bits taken already, the offset has 12 bits. So each page is 2^{12} bytes.

We can calculate the number of pages as $2^9 \cdot 2^{11} = 2^{20}$.

3 §4.12 We need to make some assumptions.

First, we assume that what is written after "Call a procedure at 5120, stacking the return address" are the actual instructions at address 5120. Otherwise, it is all but impossible to infer a valid answer from this question. Next, we assume an x86-like semantics for immediate constants. Finally, we assume pages are numbered from 0.

Since the program is at address 1020 and the instructions are each 4 bytes, we can enumerate some of the addresses.

Address	Page	Instruction
1020	1	Load word 6144 into register 0
1024	2	Push register 0 onto the stack
1028	2	Call a procedure at 5120, stacking the return address
5120	10	Subtract the immediate constant 16 from the stack pointer
5124	10	Compare the actual parameter to the immediate constant 4
5128	10	Jump if equal to 5152

Now, each of the operands need to be accounted for.

Word 6144 is on page 12.

The stack in the second instruction is still at 8192 which is on page 16, however, when we need to push on to the stack, we must move the pointer down 4 bytes to 8188, which is on page 15.

When we stack the return address of 5120, we move the stack pointer down to 8184, which is still on page 15.

When we compare the subtraction to 4, we need to access the stack pointer minus 16 which is 8168, and is still on page 15.

When we test to jump, we need to access 5152, which is on page 10.

So our page reference string should read.

1, 12, 2, 15, 2, 10, 15, 10, 10, 15, 10, 10

4 §4.13 Since we know the address size, we only need to know one of two things:

- The sum $a + b + c$
- d

If we know $a + b + c$ then the page size is 2^{a+b+c}.

If we know d then the page size is 2^{32-d}

5 §4.18 (a) NRU will replace page 0, as page 0 is the only page in class 0.

(b) FIFO will replace page 2, as page 2 is the oldest page.

(c) LRU will replace page 1, as page 1 has the oldest last reference.

(d) Second chance will replace page 0.

It sees page 2 is the oldest but has its R bit set, so page 2 is given a new loaded value of the current time and its R bit cleared. The algorithm continues and finds page 0 as the next oldest. Since page 0 does not have its R bit set, this page is removed.

6 §5.16 This has the advantage that for small files only one seek is necessary to read the data, rather than one seek for the i-node and another seek for the data.

7 §5.15

- The linked list uses D bits for each free block, so the size is $F \cdot D$ bits. The bitmap uses one bit per block, so the size is B bits.

  So the linked list will use less space when $F \cdot D < B$
- For $D = 16$, we can manipulate the equation:

$$\begin{aligned} F \cdot 16 &< B \\ \frac{F}{B} &< \frac{1}{16} \\ \frac{F}{B} &< 0.0625 \end{aligned}$$

So the linked list uses less space when there is less than 6.25% disk space left.