CS 130 Homework 2

▯▯▯

TOTAL POINTS

100 / 100

QUESTION 1

1 Q1 **20 / 20**

✓ **- 0 pts** Correct

- 10 pts wrong FIFO case

- 10 pts wrong LRU case

- 4 pts No algorithm detail.

QUESTION 2

2 Q2 **20 / 20**

✓ **- 0 pts** Correct

- 16 pts Click here to replace this description.

- 17 pts Click here to replace this description.

- 10 pts Click here to replace this description.

- 7 pts Click here to replace this description.

- 9 pts Click here to replace this description.

- 2 pts Click here to replace this description.

- 14 pts Click here to replace this description.

- 4 pts Click here to replace this description.

- 13 pts Click here to replace this description.

- 6 pts Click here to replace this description.

- 20 pts Click here to replace this description.

QUESTION 3

3 Q3 **20 / 20**

✓ **+ 5 pts** Correct Read Performance

✓ **+ 5 pts** Correct Write Performance

✓ **+ 5 pts** Correct Space Overhead

✓ **+ 5 pts** Correct Reliability

+ 0 pts No answer

QUESTION 4

4 Q4 **20 / 20**

✓ **- 0 pts** Correct

- 10 pts (b) wrong answer

- 20 pts (a) wrong answer and (b) wrong answer

- 1 pts overlap case is wrong

- 2 pts (b) lack clear result

- 8 pts (a) lack details (b) lack details

QUESTION 5

5 Q5 **20 / 20**

✓ **- 0 pts** Correct

- 20 pts Wrong.

CS 130 Operating Systems - Homework Assignment 2

(Due at 11:59 pm on 6 December 2019)

1. [20 points]

> If FIFO page replacement is used with four page frames and eight pages, how many page faults will occur with the reference string 0172327103 if the four frames are initially empty? Now repeat this problem for LRU. For each page replacement algorithm, please list page frames that are referenced. (Case for each page algorithm is worth 10 points)

- FIFO page replacement:

```
Access      Page fault      Pages in memory         Evict frame
0           true            0                       -
1           true            0,1                     -
7           true            0,1,7                   -
2           true            0,1,7,2                 -
3           true            1,7,2,3                 0
2           false           1,7,2,3                 -
7           false           1,7,2,3                 -
1           false           1,7,2,3                 -
0           true            7,2,0,3                 1
3           false           7,2,0,3                 -
```

  - Totally, 4 + 2 = 6 page faults occurred.

- LRU page replacement:

```
Access      Page fault      Pages in memory         Evict frame
0           true            0                       -
1           true            0,1                     -
7           true            0,1,7                   -
2           true            0,1,7,2                 -
3           true            1,7,2,3                 0
2           false           1,7,2,3                 -
7           false           1,7,2,3                 -
1           false           1,7,2,3                 -
0           true            1,7,2,0                 3
3           true            1,7,3,0                 2
```

  - Totally, 5 + 2 = 7 page faults.

1 Q1 **20 / 20**

✓ **- 0 pts** Correct

- 10 pts wrong FIFO case

- 10 pts wrong LRU case

- 4 pts No algorithm detail.

2. [20 points]

> Suppose that we are using byte addressing. Byte addressing refers to hardware architectures which support accessing individual bytes of data rather than only larger units called words. In other words, each 32-bit address corresponds to one byte data in memory. (Hint: Consider the following: #blocks, block size, index size, tag size, bits/block, bits in cache):

> (a) (10 points) How many total bits are needed for a direct-mapped cache with 128 KBytes of data and one word blocks? (Suppose a 32-bit address is used and one word is 4 bytes);

- Block size = 1 word = 2^2 bytes
- Data size = 128 KBytes = 2^17 Bytes
- # of blocks = 2^17/2^2 = 2^15
- Each block has 32 bits of data + tag + valid_bit
- tag + valid_bit = 1 + (32 - 15 - 2) = 16
- Total bits = 2^15*(16 + 32) = 48*2^15 bits = 192 KBytes

> (b) (10 points) How many total bits are needed for a direct-mapped cache with 64 KBytes of data and 4 word blocks? (Suppose a 32-bit address is used and one word is 4 bytes).

- Block size = 4 words = 2^4 bytes
- Data size = 64 KBytes = 2^6 * 2^10 = 2^16 bytes
- # of blocks = 2^16 / 2^4 = 2^12
- Each block has 128 bits of data + tag + valid_bit
- tag + valid_bit = 1 + (32 - 12 - 4) = 17
- Total bits = 2^12*(128 + 17)bit = 145*2^12 bits = 72.5 KBytes

3. [20 points]

> Compare RAID level 0 through 5 with respect to read performance, write performance, space overhead, and reliability.

- Read performance
  - All RAID levels(0-5) allow parallel reading of single request. Morever, RAID level 1 allows two read requests in parallel, because it stores the same data twice in different disks.
- Write performance
  - All RAID levels(0-5) have similar write performance.
- Space overhead
  - RAID 0: 0% overhead
  - RAID 1: 100% overhead, for save same data twice.
  - RAID 2: For 32 bit data word & 6 parity drives: 18.75%.
  - RAID 3: For 32 bit data word: 3.13%.
  - RAID 4 & 5: For 33 drives: 3.13%.
- Reliability
  - RAID 0: No reliability, Once data is incorrect, can't detect & can't recover.
  - RAID 1: Once data is incorrect, can recover data from another mirror disk.
  - RAID 2: Can detect & recover single random bit error in a word.
  - RAID 3,4,5: Can detect single random bit error.

2 Q2 20 / 20

✓ - **0 pts** Correct

- **16 pts** Click here to replace this description.
- **17 pts** Click here to replace this description.
- **10 pts** Click here to replace this description.
- **7 pts** Click here to replace this description.
- **9 pts** Click here to replace this description.
- **2 pts** Click here to replace this description.
- **14 pts** Click here to replace this description.
- **4 pts** Click here to replace this description.
- **13 pts** Click here to replace this description.
- **6 pts** Click here to replace this description.
- **20 pts** Click here to replace this description.

2. [20 points]

> Suppose that we are using byte addressing. Byte addressing refers to hardware architectures which support accessing individual bytes of data rather than only larger units called words. In other words, each 32-bit address corresponds to one byte data in memory. (Hint: Consider the following: #blocks, block size, index size, tag size, bits/block, bits in cache):

> (a) (10 points) How many total bits are needed for a direct-mapped cache with 128 KBytes of data and one word blocks? (Suppose a 32-bit address is used and one word is 4 bytes);

- Block size = 1 word = 2^2 bytes
- Data size = 128 KBytes = 2^17 Bytes
- # of blocks = 2^17/2^2 = 2^15
- Each block has 32 bits of data + tag + valid_bit
- tag + valid_bit = 1 + (32 - 15 - 2) = 16
- Total bits = 2^15*(16 + 32) = 48*2^15 bits = 192 KBytes

> (b) (10 points) How many total bits are needed for a direct-mapped cache with 64 KBytes of data and 4 word blocks? (Suppose a 32-bit address is used and one word is 4 bytes).

- Block size = 4 words = 2^4 bytes
- Data size = 64 KBytes = 2^6 * 2^10 = 2^16 bytes
- # of blocks = 2^16 / 2^4 = 2^12
- Each block has 128 bits of data + tag + valid_bit
- tag + valid_bit = 1 + (32 - 12 - 4) = 17
- Total bits = 2^12*(128 + 17)bit = 145*2^12 bits = 72.5 KBytes

3. [20 points]

> Compare RAID level 0 through 5 with respect to read performance, write performance, space overhead, and reliability.

- Read performance
  - All RAID levels(0-5) allow parallel reading of single request. Morever, RAID level 1 allows two read requests in parallel, because it stores the same data twice in different disks.
- Write performance
  - All RAID levels(0-5) have similar write performance.
- Space overhead
  - RAID 0: 0% overhead
  - RAID 1: 100% overhead, for save same data twice.
  - RAID 2: For 32 bit data word & 6 parity drives: 18.75%.
  - RAID 3: For 32 bit data word: 3.13%.
  - RAID 4 & 5: For 33 drives: 3.13%.
- Reliability
  - RAID 0: No reliability, Once data is incorrect, can't detect & can't recover.
  - RAID 1: Once data is incorrect, can recover data from another mirror disk.
  - RAID 2: Can detect & recover single random bit error in a word.
  - RAID 3,4,5: Can detect single random bit error.

3 Q3 **20** / **20**

- ✓ + **5 pts** Correct Read Performance
- ✓ + **5 pts** Correct Write Performance
- ✓ + **5 pts** Correct Space Overhead
- ✓ + **5 pts** Correct Reliability
- + **0 pts** No answer

4. [20 points]

> A client makes remote method invocations to a server. The client takes 5 milliseconds to compute the arguments for each request, and the server takes 10 milliseconds to process each request. The local operating system processing time for each send or receive operation is 0.5 milliseconds, and the network time to transmit each request or reply message is 3 milliseconds. Marshalling or unmarshalling takes 0.5 milliseconds per message. Calculate the time taken by the client to generate and return from two requests:

> (a) (10 points) if it is single-threaded;

- For each request

```
T_req_total = t_compute_args + t_server_process + 2*t_marshal +
2*t_unmarshal + 2*t_internet + 2*(OS_send + OS_receive)
        = 5 + 10 + 2*0.5 + 2*0.5 + 2*3 + 2*(0.5 + 0.5)
        = 25ms
```

- Totally, t = 2*t_req_total = 50ms

> (b) (10 points) if it has two threads that can make requests concurrently on a single processor. You can ignore context-switching time.

- Client:
  - First, client calculate first req's args, marshal args, and OS send req to server (t <= 5 + 0.5 + 0.5 = 6)
  - Second, client calculate second req's args, marshal args, and OS send req to server (6 < t <= 6 + 5 + 0.5 + 0.5 = 12)
  - Then, client wait until req1 is finished (12 < t <= 21 + 3 = 24)
  - Then, client receive first req's result, and unmarshal it (24 < t <= 24 + 0.5 + 0.5 = 25)
  - Then, client receive second req's result, and unmarshal it (33 + 3 = 36 < t <= 36 + 0.5 + 0.5 = 37)
- Server:
  - At t = 6 + 3 = 9, receive req1
  - (Req 1) OS_receive, unmarshal, server_process, marshal, OS_send (9 < t <= 9 + 0.5 + 0.5 + 10 + 0.5 + 0.5 = 21)
  - (Req 2) OS_receive, unmarshal, server_process, marshal, OS_send (21 < t <= 21 + 0.5 + 0.5 + 10 + 0.5 + 0.5 = 33)
- Totally, use 37ms to process 2 requests
- Morever, if `marshal + OS_send` can overlap with `unmarshal + OS_receive`, the time can be deducted to 36ms.

4 Q4 **20 / 20**

- ✓ **- 0 pts** Correct
- **- 10 pts** (b) wrong answer
- **- 20 pts** (a) wrong answer and (b) wrong answer
- **- 1 pts** overlap case is wrong
- **- 2 pts** (b) lack clear result
- **- 8 pts** (a) lack details (b) lack details

5. [20 points]

> Why is there no open or close operation in the interface to the flat file service or the directory service.
> What are the differences between our directory service Lookup operation and the UNIX open?

- Why no open/close flat file /dir?
  - Both services are stateless. The design of `open` or `close` interface for flat file is unnecessary.
- Difference between `Lookup` and `UNIX open`
  - `Lookup` is that, given a simple **file name** in specified directory, and return a `UFID` corresponds to it.
  - If we want to look up a pathname, we must call `Lookup` many times.
  - `UNIX open` is that, given a **path name**, and returns the file descriptor for the file or directory.

5 Q5 **20 / 20**

✓ **- 0 pts** Correct

- 20 pts Wrong.