PRACTICE SHEET 2: CS 170

1) Synchronization

Hunter High School in New York City was for many years a school for gifted girls. In 1974, the school was forced by a Court order to admit boys for the first time, becoming a co-ed school for gifted girls and boys. Unfortunately, there was no money for building renovations, and **there was only one student bathroom**. Your task is to help the school, which must guarantee that the following rules are enforced:

1. If a girl is in the bathroom, other girls may enter, but no boys

2. If a boy is in the bathroom, other boys may enter, but no girls

3. If the bathroom is empty, either a boy or a girl may enter, but not both.

A sign is nailed to the door, with a sliding arrow. At any given time, it points to either "`Empty`", "`Girls Present`", or "`Boys Present`".

Write code sketches for two processes, `Boy` and `Girl`, that follow the given rules and guarantee that a boy and girl are never in the bathroom at the same time. You can use any synchronization primitive that you feel is useful (e.g., semaphores, monitors).

Global variables:

```
semaphore g_mutex(1), b_mutex(1), bathroom(1);
unsigned int girls_inside = 0, boys_inside = 0;
enum DoorSign { Empty, Girls_Present, Boys_Present } sign = Empty;
```

Code:

```
void Girl()
{
      g_mutex.down();
      // first girl wants to enter
      if (girls_inside == 0) {
            bathroom.down();
            sign = Girls_Present;
      }
      girls_inside++;
      g_mutex.up();

      use_bathroom();

      g_mutex.down();
      girls_inside--;
      if (girls_inside == 0) {
            // last girl is leaving
            bathroom.up();
            sign = Empty;
      }
      g_mutex.up();
}
```

```
void Boy()
{
      b_mutex.down();
      // first boy wants to enter
      if (boys_inside == 0) {
            bathroom.down();
            sign = Boys_Present;
      }
      boys_inside++;
      b_mutex.up();

      use_bathroom();

      b_mutex.down();
      boys_inside--;
      if (boys_inside == 0) {
            // last boy is leaving
            bathroom.up();
            sign = Empty;
      }
      b_mutex.up();
}
```

2) Disks

Suppose that we build a disk subsystem to handle a high rate of I/O by coupling many disks together. Properties of this system are as follows:

- Uses 4TiB disks that rotate at 10,000 RPM, have a data transfer rate of 40 MBytes/s (for each disk), and have a 5ms average seek time, 4 KiByte sector size
- Has a SCSI interface with a 2ms controller command time
- The file system has a 32 KiByte block size
- Has a total of 20 disks

Each disk can handle only one request at a time, but each disk in the system can be handling a different request. The data is not striped (all I/O for each request has to go to one disk). *Note: Sizes are in powers of 2, bandwidths are in powers of 10.*

Problem 2.a: What is the average time to retrieve a single disk sector from a random location on a single disk, assuming no queuing time? What is the achievable bandwidth if all requests are for random sectors on one disk?

Service Time = controller + seek time + rotational delay + transfer =
2ms + 5ms + 1⁄2 × (60000 ms/min)/10000 R/min + (4096 bytes/40×10^6 bytes/s)×10^3ms/s =
2ms + 5ms + 3ms + 0.1024ms = 10.1 ms

BW = (4096 bytes/10.1ms) ×1000ms/s= 405.5 KB/s

Problem 2.b: Suppose we consider block-sized requests instead of sector-sized requests. How does the bandwidth calculated in (2.a) improve? *Hint: you should be able to reuse most parts of (2.a).*

Only the transfer time changes. So, service time =
10ms + (32768 bytes/40×10^6 bytes/s)×10^3 ms/s = 10.8 ms

BW = (32768 bytes/10.8ms)×1000ms/s = 3.034 MB/s

Problem 2.c: Give one advantage and one disadvantage to using 32 KiB blocks for the filesystem instead of the native 4KiB sector size.

Advantage: Higher BW off disk
Disadvantages: More fragmentation for small files

Problem 2.d: What is the average number of I/Os per second (IOPS) that the whole disk system can handle (assuming that I/O requests are 32KiB at a time, evenly distributed among the drives, and uncorrelated with one another)?

IOPS = 20 × IOPS(for 1 disk) = 20 × (1/10.8ms) × 10^3ms/s = 1852 IOPS

Problem 2.e: Now, suppose that we decide to improve the system by using new, better disks. For the same total price as the original disks, you can get 12 disks that have 1 TiB each, rotate at 12000 RPM, transfer at 50MB/s and have a 4ms seek time. What is the average unloaded *service time* to read a block from a single disk?

Service Time = 2ms + 4ms + 1⁄2 × (60000 ms/min)/12000 R/min + 32768/(50×10^6bytes/s) × 10^3ms/s =
2ms + 4ms + 2.5ms + 0.65536ms = 9.16ms

Problem 2.f: What is the average number of IOPS in the new system?

IOPS = 12 × (1/9.16ms) × 10^3ms/s = 1310 IOPS

3) Page Replacement

A process is allocated 5 physical page frames. Below you find the sequence of pages that this process accesses (the reference string).

Reference string: 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5

Problem 3.a: Assume that the operating system uses the FIFO (first in, first out) page replacement algorithm. After every page access, show which pages are present in the physical memory.

1	1	1	1	1	1	1	1	2	3
	2	2	2	2	2	2	2	3	4
		3	3	3	3	3	3	4	5
			4	4	4	4	4	5	6
			5	5	5	5	5	6	7

4	4	4	5	5	5	5	5	6	7
5	5	5	6	6	6	6	6	7	8
6	6	6	7	7	7	7	7	8	9
7	7	7	8	8	8	8	8	9	4
8	8	8	9	9	9	9	9	4	5

Problem 3.b: Assume that the operating system uses the LRU (least recently used) page replacement algorithm. After every page access, show which pages are present in the physical memory.

1	2	3	4	5	3	4	1	6	7
	1	2	3	4	5	3	4	1	6
		1	2	3	4	5	3	4	1
			1	2	2	2	5	3	4
				1	1	1	2	5	3

8	7	8	9	7	8	9	5	4	5
7	8	7	8	9	7	8	9	5	4
6	6	6	7	8	9	7	8	9	9
1	1	1	6	6	6	6	7	8	8
4	4	4	1	1	1	1	6	7	7

4) Virtual Memory

Note: This is a fairly difficult question, but a great exercise to really understand paging. I took this question from a CS162 midterm exam at UC Berkeley.

Consider a multi-level paging-based memory management scheme using the following format for virtual addresses (18 bits virtual addresses):

Virtual Page # (4 bits)	Virtual Page # (5 bits)	Offset (9 bits)

Problem 4.a: If the physical address space is 16 bits, what will X and Y be in the following format?

Physical Page # (____*7* bits)	Offset (____*9* bits)

Problem 4.b: How many PTEs are in the first level page table (page directory)? The second level (page table)?

First Level: 2^4 = *16 PTE's*
Second Level: 2^5 = *32 PTE's*

Problem 4.c: Page table entries (PTE) are 16 bits in the following format, stored in big-endian form in memory (that is, the MSB -- the most significant byte -- is the first byte in memory):

Physical Page #	Unused (3)	Writable	Kernel	Dirty	Use	Directory	Valid

Using the scheme above, and the physical memory table on the next page, translate the following addresses. Assume that the Page Table Pointer points to **0x3000.** Intermediate page table entries (the entries in the page directory) should have the directory bit set. If you encounter an error, write **"Error"** in the Translated Physical address box instead of an address.

Virtual Address	**Translated Physical Address**
0x1024F (example)	*"Error"*
0x0442F	*0x562F*
0x0842D	*0x982D*
0x0CF1A	*"Error"*

(See explanation for the correct solution below)

Page Table Pointer: **0x3000**

Physical Memory

Address	**+0**	**+1**	**+2**	**+3**	**+4**	**+5**	**+6**	**+7**	**+8**	**+9**	**+A**	**+B**	**+C**	**+D**	**+E**	**+F**
0x0000	00	2A	A3	32	4A	BC	CD	DE	A1	A4	A3	AB	BC	A1	A3	3A
0x0010	AA	BB	CC	DD	EE	FF	11	22	33	44	55	66	77	88	99	00
…																
0x2000	A1	00	A3	00	A5	00	A7	00	A9	00	AB	00	AD	00	AF	00
0x2010	00	B1	00	B3	00	B5	00	B7	00	B9	00	BB	00	BD	00	BF
…																
0x3000	20	00	42	03	F0	03	60	00	20	03	F0	00	00	08	42	10
0x3010	00	12	00	14	00	16	00	18	42	12	42	16	42	18	42	04
…																
0x4200	12	32	00	54	56	01	78	02	9A	AB	03	CD	DE	04	32	00
0x4210	12	32	A3	A2	A1	DA	DD	1E	75	12	91	23	37	12	81	7C
…																
0x6000	DE	00	32	00	9A	AB	03	CD	56	01	78	02	12	32	00	54
0x6010	37	12	81	7C	75	12	91	23	A1	DA	DD	1E	12	32	A3	A2
…																
0xF000	A2	A1	FD	EF	98	01	CD	2A	56	14	32	12	65	54	42	32
0xF010	23	12	82	32	12	33	01	23	45	54	AB	CD	EA	12	32	12
…																

Explanation for solution given above

0x0442F*: first 4 bits are "0001" so we look at index 1 in "0x3000" which is "0x4203" (in the new memory table). Last 2 bits are "11" so it is valid and a directory which is fine. Top 7 bits are taken and we 0 out the offset so we go to address "0x4200." The next 5 bits are "00010" so we look at the index 2, which is "0x5601." This has last bit as 1, which is valid, so we take the top 7 bits of "0x5601" and add that to the offset "0x02F" which gets us "0x562F."*

0x0842D: *first 4 bits are "0010" so we look at index 2 in "0x3000" which is "0xF003". This has last 2 bits "11" which is fine. Top 7 bits are taken with 0 offset, so we got to "0xF000". We are looking for index 2 because next 5 bits are "00010", so we get "0x9801". This has valid bit, so it is good. we take top 7 bits here and add it to the offset, so we get "0x982D".*

0x0CF1A: *Look at index 3 because first 4 bits are "0011" and we find "0x6000". Looking at the last 2 bits, we realize this will cause some kind of error.*