Solution

Problem 1: (21 points)

1. [1]8k/8*1024 [2] 2^(27-13) = 2^14 [3] 12

2. [1]0006 [2]2 [3]001 [4]N
   [5]N [6]66 [7]CCABC

   [1]033E [2]2 [3]0CF [4]Y
   [5]N [6]7F [7]FF234

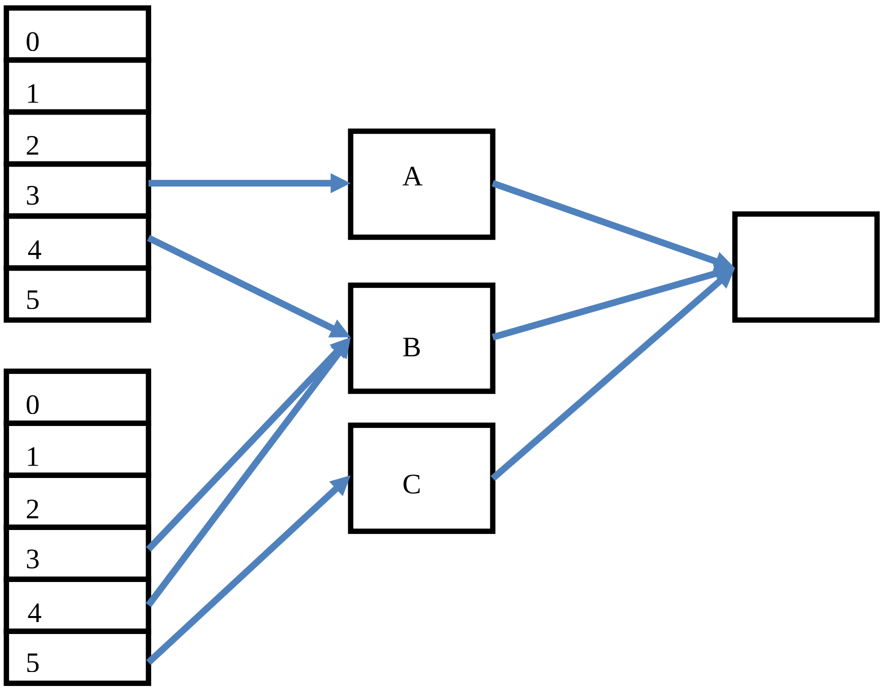
3. if we use virtual address to access the cache, then we need to flush the cache evey time when we switch the process.(any other reasonable answer can also get points)

Problem 2: (16 points)

1. [1] Y [2] N [3] Y [4] N
   [5] N [6] N

2. [1] ics2014i
   [2] ics2014ii
   [3] ics2014iic

3.



Problem 3: (15 points)

1. **(possible answer)**

[1]	a	[2]	b	[3]	b
[4]	c	[5]	c	[6]	a
[7]	1	[8]	0	[9]	0

2. The pthread_join() function only waits for the thread's termination and reaps the resource. It doesn't block the execution of other threads. Other threads may continue to print 0/1/2 after the first execution of do2() and before the pthread_join of the first-finished do2()'s thread.

   All possible output strings besides the correct one:

   ```
   0120\n12\n\n
   01201\n2\n\n
   012012\n\n\n
   ```

   (Any 2 of them are enough to acquire the full score.)

Problem 4: (5 points)

It is possible. Executions that can cause deadlock:

```
(1)   Thread1: P(a) [waits for b]
      Thread2: P(c) P(b) V(c) [waits for a]
(2)   Thread1: P(a) P(b) V(a) [waits for c]
      Thread2: P(c) [waits for b]
```

(Any 1 of them is enough to acquire the full score.)

Problem 5: (18 points)

```
1. serverfd = Open_client("59.78.3.103",6666);*/

2. sprintf(send_buf,"LOGIN %d",student_id);
   send(serverfd,send_buf,strlen(send_buf),0)

3. recv(serverfd,recv_buf,1024,0);
   if(strcmp(recv_buf,"LOGIN ERR") == 0) {
       printf("%s\n",recv_buf);
       exit(0);
    }

4. sprintf(send_buf,"SCORE %d %d %d %d",score1,score2,score3,sc
   ore4);
   send(serverfd,send_buf,strlen(send_buf),0);


5. recv(serverfd,recv_buf,1024,0);

   printf("%s\n",recv_buf); exit(0);
```

Problem 6: (25 points)

1. (10’)

  1) MAP_PRIVATE | MAP_ANON

  2) D

Reason: If child process is executed first, writing to idx will cause COW. If parent process is executed first, the invocation to waitpid (i.e. call waitpid in assembly) will modify the user stack, which in turn triggering COW.

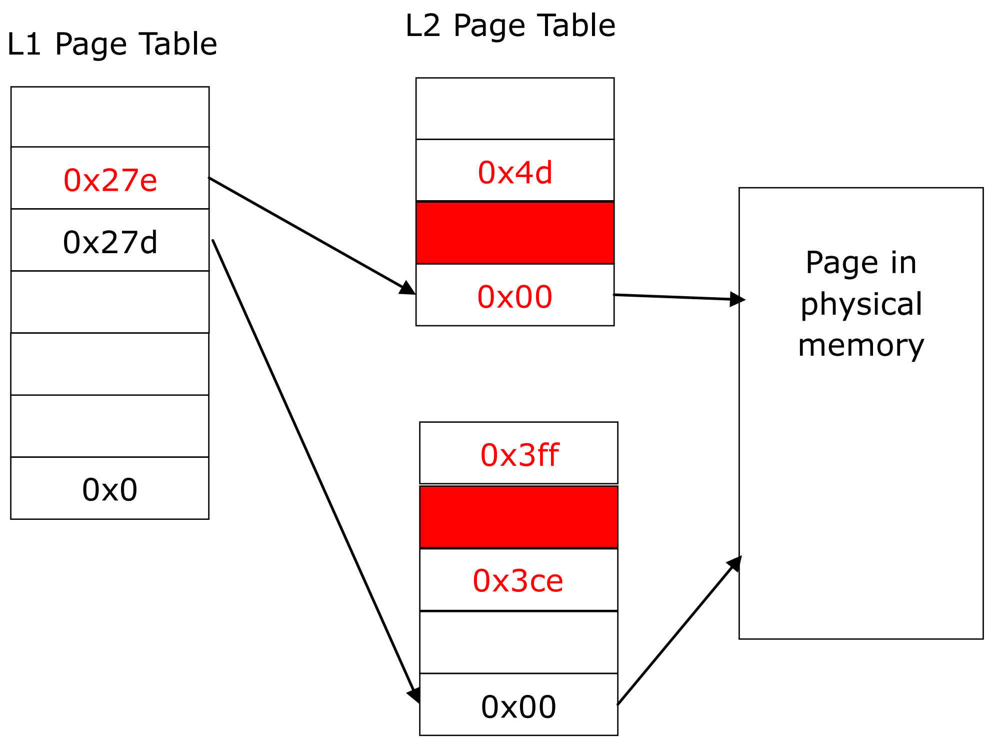
  3) 2014

Since sarr is a shared object, the modification on sarr in the child process is visible to the parent process.

2. (12’)

1) [1]	262144(256K)	[2]	27e
[3]	0x9f80dfff	[4]	0x9f84dfff

2)



3) 128

First, 64 page faults will occur during writing to sarr. Then 64 page faults will occur when modifying parr.

3. (3’) Shared objects can be used for communication (message passing) between processes.