B Solution

Problem 1: (22 points)

1.	[1]	20	[2]	11	[3]	12	[4]	2KB
2.	[1]	0F80	[2]	0	[3]	3E0	[4]	Y
	[5]	N	[6]	62	[7]	31510		
	[8]	29E	[9]	2	[10]	A7	[11]	N
	[12]	-	[13]	-	[14]	-		

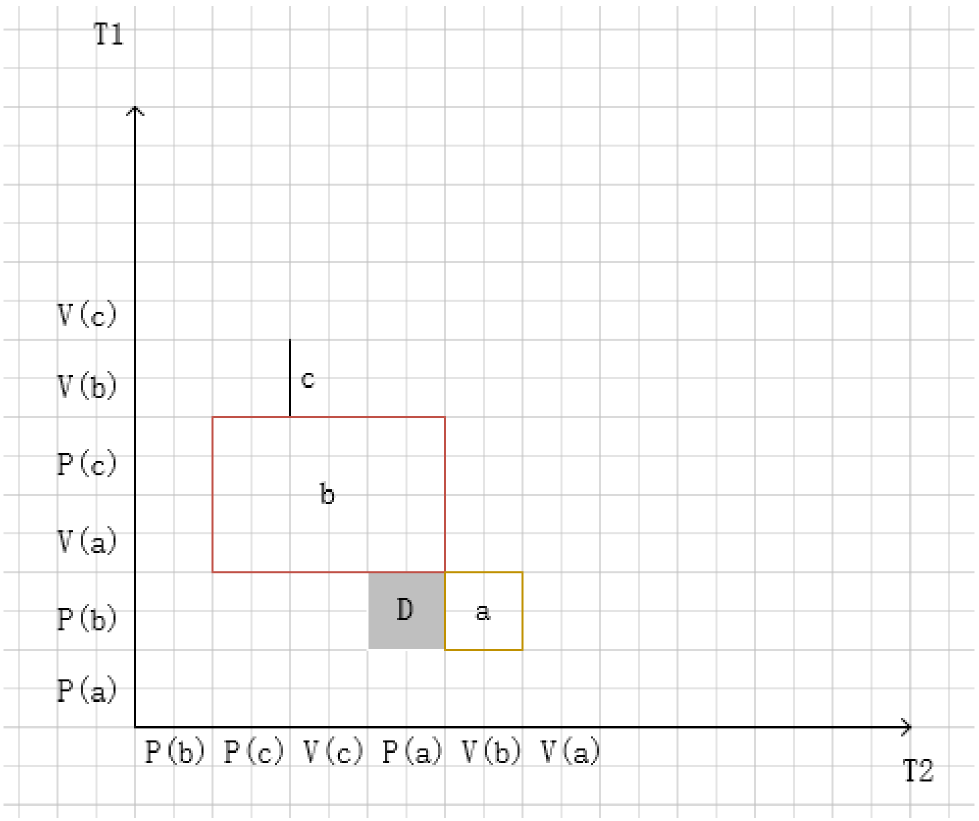
Problem 2: (12 points)

1. Case1: After T1 finish step1 and T3 finish step3

   Case2: After T1 finish step1, T3 finish step1, and T2 finish step3, they will enter deadlock.

2. It may cause deadlock, a possible execution flow is:

   P1(a)->P3(b)->P3(c)->V3(c)->deadlock



Problem 3: (14 points)

1. [1] P(&a);   [2] V(c);

   [3] P(&b);   [4] V(c);

   [5] P(c); P(c); P(c);   [6] V(a); V(b); V(b);

   [7] 1   [8] 2   [9] 0

2. When we have more than 1 thread job2, we can't make sure three P(c) are executed atomically and program will get stuck if more than 1 thread job2 require semaphore "c" at the same time.
We can add another semaphore sem_t mutex initialized as 1. Then we can make sure three P(c) are executed atomically.
[5] will be changed into P(mutex); P(c); P(c); P(c); V(mutex);

Problem 4: (18 points)

```
[1] int clientfd = Open_clientfd(argv[1], server_port);
    Rio_writen(clientfd, argv[3], strlen(argv[3]));
    Rio_writen(clientfd, "\r\n", 2);

[2] rio_t rio;
    Rio_readinitb(&rio, clientfd);
    char buf[MAXLINE];
    int file_found = 0;
    while (1) {
      Rio_readlineb(&rio, buf, MAXLINE);
      long int length = atoi(buf);
      if (length == 0) {
          if (!file_found) {
              report_not_find_file(argv[3]);
              Close(clientfd);
              Close(openfd);
              return 1;
          }
          Rio_readnb(&rio, buf, 2);
          break;
      } else {
          file_found = 1;
      }

      ssize_t bytes_remained = length;
      while (bytes_remained != 0) {
          ssize_t bytes_toread = bytes_remained > MAXLINE ?
                                 MAXLINE : bytes_remained;
          ssize_t sz = Rio_readnb(&rio, buf, bytes_toread);
          Rio_writen(openfd, buf, sz);
          bytes_remained -= sz;
      }
      Rio_readnb(&rio, buf, 2);
   }
    Close(clientfd);
```

Problem 5: (24 points)

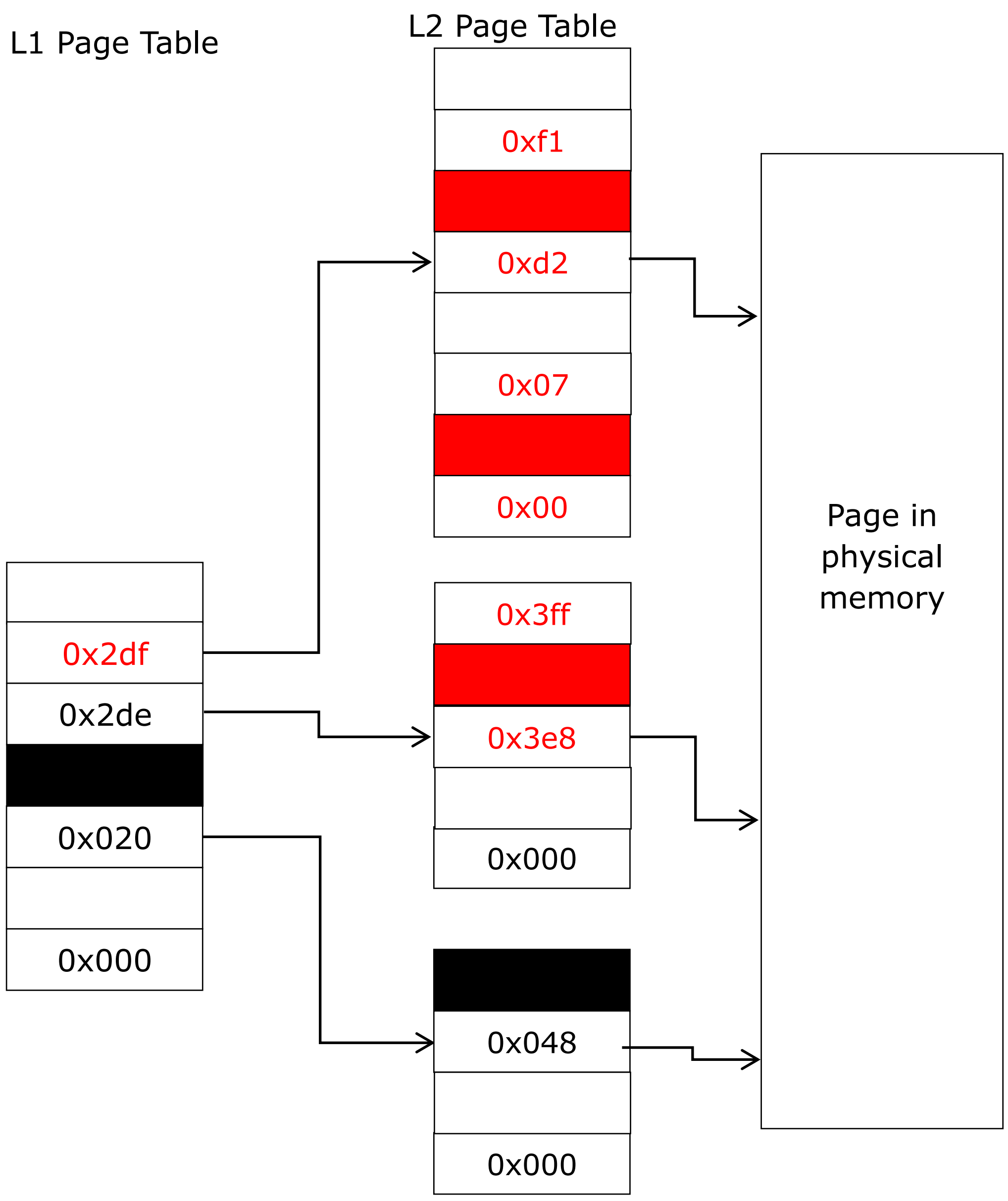
1. [1] 128000 [2] 0xb7c073ff
   [3] 0xb7cf13ff [4] 0x2df

2. 96, 32
Reason: Because mmap will not copy data from disk to memory. To fill the page table at the first access, any read to sbuf and pbuf will cause page fault after mmap. And the write to pbuf will cause COW for it's a private mapping. And the content of array sbuf and pbuf are in 32 pages.

3. 32, 32
After fork, all the pages are set as readonly and all the write to memory sbuf will cause COW. And the content of array pbuf are in 32 pages.

4. [1] '1' [2] '1'
   [3] '2' [4] '1'

5.



Problem 6 Concurrent Hash Table: (17 points)

```
1.  [1] int bucket = FetchAndAdd(&next_bkt);
        while (bucket < htb_original.num_bucket) {
          list_t list = htb_original.lists[bucket];
          node_t *node = list.head;
          while (node) {
             Hash_Insert(&htb_rehashed, node->key);
             node = node->next;
          }
          bucket = FetchAndAdd(&next_bkt);
       }
```