CS323 Exercises – Solutions
Week 4
15 March 2019

Problem 1: Synchronization

A) A block may be allocated multiple times.

B)

```
bitmap *bm;
pthread_mutex_t *bitmap_mutex;
…
int take_free_entry() {
    for (int i = 0; i < MAXBLOCKS; i++) {
        if (bitmap_read(bm, i) == 0) {
            bitmap_set(bm, i);
            return i;
        }
    }

    return -1;
}

int alloc_entry() {
    pthread_mutex_lock(bitmap_mutex);

    int index = take_free_entry();

    pthread_mutex_unlock(bitmap_mutex);
    return index;
}

void free_entry(int index) {
    pthread_mutex_lock(bitmap_mutex);

    bitmap_clear(bm, index);

    pthread_mutex_unlock(bitmap_mutex);
}
```

C) One possible improvement is to replace the global bitmap mutex with atomic bitmap operations

```
bitmap *bm;
…
int take_free_entry() {
    for (int i = 0; i < MAXBLOCKS; i++) {
        // atomically set a bit and check its old value
        if (bitmap_test_and_set(bm, i) == 0) {
            return i;
        }
    }
```

```
    return -1;
}

int alloc_entry() {
    int index = take_free_entry();

    return index;
}

void free_entry(int index) {
   // atomically clear a bit
   bitmap_test_and_clear(bm, index);
}
```

Problem 2: Synchronization

```
ListenerThread {
      for( i=0; i<MAX_THREADS; i++ ) thread[i] = pthread_create(WorkerThread);
      forever {
            receive(request)
            pthread_mutex_lock(queuelock)
            while(avail == N) pthread_cond_wait(notfull, queuelock)
            put request in queue
            avail++
            pthread_cond_signal(notempty, queuelock)
            pthread_mutex_unlock(queuelock)
      }
}

WorkerThread {
      forever {
            pthread_mutex_lock(queuelock)
            while(avail <= 0) pthread_cond_wait(notempty, queuelock)
            take request out of queue
            avail--
            pthread_cond_signal(notfull, queuelock)
            pthread_mutex_unlock(queuelock)
            read file from disk
            send(reply)
      }
}
```

Problem 3: Synchronization

Using a mutex over the entire table is undesirable since it would unnecessarily restrict concurrency. Such a design would only permit a single insert, lookup or delete operation to be performed at any given time, even if they are to different hash bins.

Using a mutex over each element in the doubly linked list would permit the greatest concurrency, but a correct, deadlock-free implementation has to ensure that all elements involved in a delete or insert operation, are acquired in a well-defined order.

Using a mutex over each hash bin is a compromise between the two solutions – it permits more concurrency than solution 1, and is easier to implement correctly than solution 2.