CS323 – Exercises
Week 4
15 March 2019

Problem 1: Synchronization

Suppose you have a memory allocator that allocates `MAXBLOCKS` blocks of memory, each of fixed size `BLOCKSIZE`. The allocator simply maintains a `bitmap[0..MAXBLOCKS-1]` indicating that a block of memory is allocated or free. The allocator has two methods:

- *`int alloc_entry()`* - requests allocation of one block of memory and returns the index of that block in bitmap (-1 if none available).
- *`void free_entry(int entry)`* - requests deallocation of `entry` (you can assume that all entries passed to this call were previously allocated).

The relevant code for the allocator is:

```
bitmap *bm;
…
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() {
    int index = take_free_entry();
    return index;
}

void free_entry(int entry) {
    bitmap_clear(bm, entry);
}
```

The allocator had been used in a single-threaded application, but now it is going to be used in a multithreaded application.

A) Do you see a potential problem with the use of the allocator in a multithreaded application? Describe the problem.

B) Can you come up with a quick fix using `Pthreads` primitives? Maximum five additional lines of code.

C) Can you find a better solution by using the atomic bitmap functions:

- `bool bitmap_test_and_set(bitmap *bm, int index)`: sets the bit located at `index` and returns its old value
- `bool bitmap_test_and_clear(bitmap *bm, int index)`: clears the bit located at `index` and returns its old value?

Problem 2: Synchronization

In class we looked at how to multithread a web server with a listener thread and a number of worker threads, communicating through a queue synchronized by `pthreads` locks and condition variables. The way we did this, the queue could grow in an unbounded fashion, if requests come in faster than the server can handle them.
Can you modify the solution given in class so that the queue can contain only a fixed maximum of requests and the listener thread blocks if it is full?

Problem 3: Synchronization

Suppose you need to design a data structure for efficient dictionary lookup in a multithreaded application. The design uses a hash table that consists of an array of pointers each corresponding to a hash bin. The array has 5001 elements, and a hash function takes an item to be searched and computes an entry between 0 and 5000. The pointer at the computed entry is either null, in which case the item is not found, or it points to a doubly linked list of items that you would search sequentially to see if any of them matches the item you are searching for. There are three functions defined on the hash table:

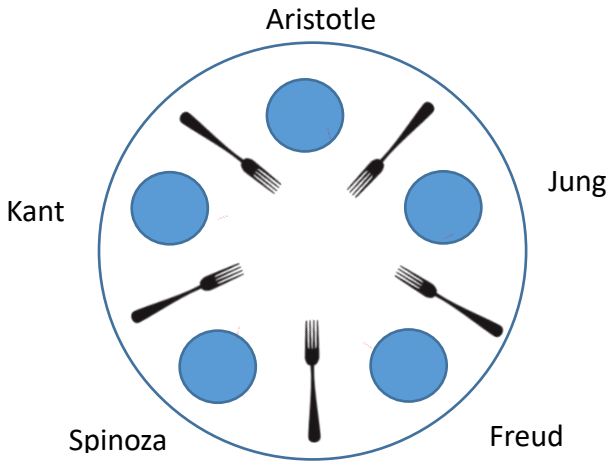
- insertion - if an item is not there already,
- lookup - to see if an item is there,
- deletion - to remove an item from the table.

Describe and comment on the different locking alternatives.
(*Hint: Consider using mutex on the entire table, each hash bin, each element or a combination*)

Extra reading: Dining philosophers problem

The dining philosophers problem is a classic synchronization problem which is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.

Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and five bowls of rice as shown in the below figure.



At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from his left and one from his right side. When a philosopher wants to think, he keeps down both chopsticks at their original place. A deadlock could potentially occur if all the philosophers reached for the left (or right) chopstick at the same time. Can you think of a solution that prevents deadlock?

You can read more about the Dining philosophers problem in the presentation posted on Moodle and in the book Operating Systems: Three Easy Pieces.