1. Consider the parallel-sum problem. Use semaphores to synchronize. Assume that all array-indexing is one-based, and the size of the array is $n = 2^t$.

```
shared int A[n];
semaphore s[n] = {1, 1, . . . , 1};

for j = 1, 2, . . . , t do:
        for i = 1, 2, . . . , n, do:
                if ( i % 2^j == 0), then:
                        wait(s[i]);
                        A[i] = A[i] + A[i – 2^(j – 1)];
                        if (i + 2^j < n) signal(s[i + 2^j]);
```

This solution sends some redundant (although harmless) signals in the last line. To avoid these, you can add another condition $((i + 2^j) \% \ 2^{j+1} == 0)$.

2. [Generalization of lab assignment LA5] Consider the reader-writer problem with designated readers. There are *n* reader processes, where *n* is known beforehand. There are one or more writer processes. Items are stored in a buffer of unlimited capacity. Every item is written by a writer and is designated for a particular reader. Solve this problem so that no process makes any busy wait.

```
semaphore rw_mutex = 1;
semaphore r_mutex[n] = {0, 0, . . . , 0};

reader (i)
{
        wait(r_mutex[i]);
        while (true) {
                wait(rw_mutex);
                Read and remove one item from buffer, that is meant for the i-th reader;
                signal(rw_mutex);
                wait(r_mutex[i]);
        }
}

writer ()
{
        while (true) {
                Generate item for reader i;
                wait(rw_mutex);
                Write (item, i) to buffer;
                signal(rw_mutex);
                signal(r_mutex[i]);
        }
}
```

3. [*Starvation-free reader-priority reader-writer problem*] Implement under the assumption that the semaphore queues are FIFO queues.

```
shared int read_count = 0;
semaphore rw_mutex = 1;
semaphore r_mutex = 1;
semaphore q_mutex = 1;

reader ()
{
        wait(q_mutex);
        wait(r_mutex);
        ++read_count;
        if (read_count == 1) wait(rw_mutex);
        signal(q_mutex);
        signal(r_mutex);

        read();

        wait(r_mutex);
        – –read_count();
        if (read_count == 0) signal(rw_mutex);
        signal(r_mutex);
}

writer ()
{
        wait(q_mutex);
        wait(rw_mutex);
        signal(q_mutex);

        write();

        signal(rw_mutex);
}
```

4. [Sleeping barber problem]

```
shared light_in_the_waiting_room = green;
shared chairs_in_the_waiting_room = all_empty;

barber ()
{
        while (true) {
                inspect the waiting room;
                if (there are no customers), then
                        set the status light of waiting room to green (available);
                        sleep until woken up by a customer;
                set the status light of waiting room to red (busy);
                serve the next customer;

        }
}

customer ()
{
        enter the waiting room;
        if the status light is red {
                if all of the n chairs in the waiting room are occupied, then leave;
                occupy an empty chair in the waiting room;
                sleep until woken up by the barber;
        }
        enter barber's room;
        wake up the barber if sleeping;
        have hair-cut and leave;
}
```

(a) Where are race conditions possible?

(i) For occupying empty chairs

(ii) Barber sleeping. Two (or more) new customers come at the same time. Both see the status light green and enter barber's room.

(iii) Barber finishes a hair-cut, inspects the waiting room, finds nobody. Barber is preempted. A new customer comes, sees the red light, and sleeps. Barber is rescheduled, sets the status light to green, and sleeps.

(b) Solve using semaphores.

```
semaphore barber_mtx = 0;
semaphore chair_mtx = 1;
shared int no_of_empty_chairs = n;
semaphore customer_mtx = 0;

barber ()
{
        while (true) {
                wait(customer_mtx);
                wait(chair_mtx);
                ++no_of_empty_chairs;
                signal(barber_mtx);
                signal(chair_mtx);

                hair_cut();
        }
}

customer ()
{
        wait(chair_mtx);
        if (no_of_empty_chairs == 0) {
                signal(chair_mtx);
        } else {
                – –no_of_empty_chairs;
                signal(customer_mtx);
                signal(chair_mtx);
                wait(barber_mtx);

                have_hair_cut();
        }
}
```