Automatic Biryani Serving

Implementation

For tackling this question, I maintain arrays of robots, tables and students. The program behaves exactly as described in the problem pipeline. Each student, robot and table is a thread.

Robots endlessly create biryani whenever they have all their vessels, students arrive and get a slot on some table, which serves biryani from it's own container until it becomes empty, and then waits until it refills it's container.

Synchronization is achieved by the use of mutex locks. Each table has it's own mutex lock.

Robots iterate over all tables (after acquiring the lock for the same). If it finds some table whose container is empty, it fills that table's container with one of it's vessels.

Students iterate over all the tables (after acquiring the lock for the same). If they find a table which has some open slots left, it goes to that table and occupies that slot. After this, it eats and leaves.

Tables give out slots from the container they have until they can give. After that, they wait for some robot to fill it's slot.

Assumptions

Large input is to be avoided, since a large number of threads cause a problem.

Code snippets

Robot

Robot's thread:

```
while(1) {
    Vessels * v = (Vessels *) malloc(sizeof(Vessels));
    v->count = randRange(1, 10);
    v->capacity = randRange(25, 50);

    r->vessels = v;

    sleep(randRange(2, 5)); // preparing r vessels
    biryani_ready((Robot *)a);
}
```

Robot waiting to fill a container on some table:

```
while(r->vessels->count > 0) {
    for(int i = 0; i < N; i++) {
        pthread_mutex_lock(&tables[i]->mutex);

        if(!tables[i]->container) {
            Container * c = (Container *) malloc(sizeof(Container));
            c->capacity = r->vessels->capacity;
            r->vessels->count--;

            tables[i]->container = c;
        }

        pthread_mutex_unlock(&tables[i]->mutex);
    }
}
```

Table

Table's thread:

```
while(1) {
    while(!t->container);

    printf("Serving table %d entering serving phase\n", t->uid);

    while(t->container->capacity > 0)
        ready_to_serve_table(t, randRange(1,
            min(10, t->container->capacity)));

    t->container = 0;

    printf("Serving table %d is empty, waiting to be refilled\n", t->uid);
}
```

Table serving some students:

```
void ready_to_serve_table(Table * t, int slots) {
    printf("Serving table %d is ready to serve with %d slots\n",
            t->uid, slots);
    pthread_mutex_lock(&t->mutex);

    t->slots = slots;
    t->container->capacity -= slots;
    pthread_mutex_unlock(&t->mutex);

    while(t->slots);
}
```

Students

Students looking for slots on table:

```
bool gotSlot = false;

while(!gotSlot) {
    for(int i = 0; i < N; i++) {
        pthread_mutex_lock(&tables[i]->mutex);

        if(tables[i]->slots > 0) {
            gotSlot = true;

            printf("Student %d assigned to a slot on the serving table "
                    "%d and waiting to be served\n",
                    s->uid, i);

            tables[i]->slots--;

            pthread_mutex_unlock(&tables[i]->mutex);
            break;
        }

        pthread_mutex_unlock(&tables[i]->mutex);
    }
}
```