STDISCM

Problem Set 2

Slides by Taro(Enzo Arkin Panugayan)

Deadlocks

Possible problems that can arise for Deadlocks

Healer 1 goes to Dungeon 1
Tank 1 goes to Dungeon 1
DPS 1 goes to Dungeon 2
DPS 2 goes to Dungeon 1
DPS 3 goes to Dungeon 1

No other available players

None of them get to play since
Dungeon 1 has 4/5
Dungeon 2 has 1/5

Starvation

Possible problems that can arise for Starvation

Healer 1 queues at 9:15:03
Healer 2 queues at 9:15:04

Healer 2 was served first before Healer 1

Healer 3 queues at 9:15:07

Healer 3 was served first before Healer 1

Healer 1 was never served

Implementation

Helpers with Lock Guards

Queue-based algo
- Allows FCFS
  - No Starvation
- Easily insert new players

```
// ================ DPS QUEUE HELPERS ================
void add_dps(int id) {
    lock_guard<mutex> guard(dps_mutex);
    dps_queue.push(id);
}

int get_dps_size() {
    lock_guard<mutex> guard(dps_mutex);
    return (int)dps_queue.size();
}

void create_dps() {
    lock_guard<mutex> guard(dps_mutex);
    if (count[2] < 99999) {
        count[2]++;
        dps_queue.push(count[2]);
    } else {
        count[2] = -1;
    }
}

int get_dps() {
    lock_guard<mutex> guard(dps_mutex);
    if (!dps_queue.empty()) {
        int id = dps_queue.front();
        dps_queue.pop();
        return id;
    }
    return -1;
}

int get_dps_at(int idx) {
    lock_guard<mutex> guard(dps_mutex);
    if (idx >= 0 && idx < (int)dps_queue.size()) {
        queue<int> temp = dps_queue;
        for (int i = 0; i < idx; i++) {
            temp.pop();
        }
        return temp.front();
    }
    return -1;
}
```

Implementation

If there is not enough players
- Tank = 0
- Healer = 0
- DPS < 3

Start generating players

```
void create_dps_thread() {
    while (!is_done) {
        for(int i = 0; i < rand_range(1,60); i++){
            create_dps();
        }
        this_thread::sleep_for(chrono::seconds(rand_range(t3_global, t4_global)));
    }
}

void queue_watcher_thread() {

    while (!is_done) {
        if (!generators_started &&
            (
                get_tank_size() == 0
                || get_healer_size() == 0
                || get_dps_size() < 3
            )
        ) {
            thread(create_tank_thread).detach();
            thread(create_healer_thread).detach();
            thread(create_dps_thread).detach();

            generators_started = true;
        }

        this_thread::sleep_for(chrono::seconds(10));
    }
}
```

Implementation

Dungeon treated as a
Free Frame List

To at least attempt to balance

(Some dungeons take longer than others)

```
// ---------- dungeon helpers ----------
int get_most_available_dungeon(){
    lock_guard<mutex> guard(dungeon_mutex);
    if(!dungeon_free_list.empty()){
        int id = dungeon_free_list.front();
        dungeon_free_list.pop();
        dungeon_in_progress_list[id] = 1;
        return id;
    }
    return -1;
}

void dungeon_free(int instance_id, int time_served_seconds) {
    lock_guard<mutex> guard(dungeon_mutex);
    dungeon_served_time[instance_id] += time_served_seconds;
    dungeon_party_list[instance_id] = "EMPTY";
    dungeon_time_list[instance_id] = 0;
    dungeon_in_progress_list[instance_id] = 0;
    dungeon_free_list.push(instance_id);
}
```

Implementation

Match Thread checks first if there are enough players

- Tank >= 1
- Healer >= 1
- DPS >= 3

Then check if there is free dungeon

If yes, tell dungeon to get players

```
// ---------- scheduler (match_thread) ----------
void match_thread() {

    while (!is_done) {
        int tanks = get_tank_size();
        int healers = get_healer_size();
        int dpss = get_dps_size();

        if (tanks >= 1 && healers >= 1 && dpss >= 3) {
            int instance_id = -1;
            {
                lock_guard<mutex> guard(dungeon_mutex);
                if (!dungeon_free_list.empty()) {
                    instance_id = dungeon_free_list.front();
                    dungeon_free_list.pop();
                    dungeon_in_progress_list[instance_id] = 1;
                }
            }
            if (instance_id >= 0) {
                dungeon_assign(instance_id, rand_range(t1_global, t2_global));
            }
        }
        else{
            if (!bonus_global) {
                bool any_active = false;
                {
                    lock_guard<mutex> guard(dungeon_mutex);
                    for (int status : dungeon_in_progress_list) {
                        if (status == 1) {
                            any_active = true;
                            break;
                        }
                    }
                }
                if (!any_active) {
                    this_thread::sleep_for(chrono::seconds(1));
                    is_done = true;
                }
            }
        }
    }
}
```

Implementation

Dungeon takes players(FCFS)

Getters have Mutex Locks

Thread to run the dungeon uptime

```
void dungeon_assign(int instance_id, int assigned_time_seconds) {
    int tank_id = get_tank();
    int healer_id = get_healer();
    int d1 = get_dps();
    int d2 = get_dps();
    int d3 = get_dps();

    {
        lock_guard<mutex> guard(dungeon_mutex);
        stringstream ss;
        ss << "T" << setw(5) << setfill('0') << tank_id
           << " H" << setw(5) << setfill('0') << healer_id
           << " D" << setw(5) << setfill('0') << d1
           << " D" << setw(5) << setfill('0') << d2
           << " D" << setw(5) << setfill('0') << d3;
        dungeon_party_list[instance_id] = ss.str();
        dungeon_time_list[instance_id] = assigned_time_seconds;
        dungeon_served_count[instance_id] += 1;
    }

    // Launch dedicated instance thread for this dungeon
    thread([instance_id, assigned_time_seconds]() {
        int remaining = assigned_time_seconds;
        while (remaining > 0) {
            {
                lock_guard<mutex> guard(dungeon_mutex);
                dungeon_time_list[instance_id] = remaining;
            }
            this_thread::sleep_for(chrono::seconds(1));
            remaining--;
        }
        dungeon_free(instance_id, assigned_time_seconds);
    }).detach();
}
```

Summary

Helpers with Mutex Lock Guards
- So I can just call the helper functions

Non Preemptive First Come First Serve basis for Players
- Dungeon can’t suddenly end mid game (Shit game if oo)
- FCFS to make sure no Starvation for Players

Free Frame List for Dungeons
- Just to attempt to balance
- FIFO resource allocation

Others

Input and Output is just based on my old CSOPESY Marquee Console Code

Config and Input Validation is AI Generated Code I used for STDISCM PSET1

Display only shows a certain number of dungeons at a time

You can press up and down arrow keys to scroll through the dungeons