Politechnika Wrocławska

Raport z Projektu 2

Przedmiot: Systemy Operacyjne 2

Kyrylo Semenchenko 273004
Informatyka Techniczna I stopnia
Wydział Informatyki i Telekomunikacji

17 czerwca 2025

Spis treści

1 Wstęp

W ramach drugiego projektu z przedmiotu *Systemy operacyjne* opracowano w języku C++ symulację funkcjonowania restauracji, w której występują równocześnie różne procesy takie jak przyjmowanie klientów, obsługa kelnerska, przygotowywanie dań w kuchni, dostawy składników oraz generowanie logów. Program wykorzystuje wielowątkowość, złożone struktury danych, operacje na współdzielonych zasobach oraz mechanizmy synchronizacji, takie jak muteksy i zmienne warunkowe. Celem projektu było praktyczne zastosowanie wiedzy z zakresu zarządzania współbieżnością i zasobami w systemach wielowątkowych.

2 Architektura programu

Program został zorganizowany wokół struktury `Restaurant`, która pełni funkcję centralnego stanu aplikacji i zawiera wszystkie istotne dane dotyczące stolików, kolejki klientów, zapasów składników, przepisów, zamówień oraz konfiguracji symulacji.

Główne komponenty programu

- **Struktura `Table`** — reprezentuje stolik w restauracji. Każdy stolik posiada ID, rozmiar, stan zajętości, identyfikator przypisanej grupy klientów oraz listę zamówień. Dostęp do danych chroniony jest lokalnym muteksem.
- **Struktura `CustomerGroup`** — reprezentuje grupę klientów. Przechowuje ID grupy, jej rozmiar oraz listę zamówionych dań.
- **Struktura `Order`** — opisuje pojedyncze zamówienie: danie, ID grupy, ID stolika oraz czas przygotowania.
- **Struktura `Restaurant`** — agreguje wszystkie dane i kolejki, m.in.:
  - wektory stolików (`tables`),
  - kolejki: oczekujących klientów, zamówień w kuchni, gotowych zamówień,
  - magazyn składników i przepisy (`ingredients_stock`, `recipe`),
  - parametry symulacji (liczba kelnerów, czas dostawy, czasy przygotowania dań, itp.),
  - mechanizmy synchronizacji: liczne muteksy, zmienna warunkowa do kuchni.

Wątki

Program uruchamia szereg wątków:

- `customer_arrival` — generuje nowe grupy klientów i próbuje ich posadzić przy stoliku lub dodaje do kolejki oczekujących.
- `waiter` — każdy kelner obsługuje stoliki, zbiera zamówienia i dostarcza gotowe dania.
- `kitchen` — przetwarza zamówienia, sprawdza dostępność składników, przygotowuje dania i przesyła do kolejki gotowych zamówień.
- `ingredient_delivery` — okresowo uzupełnia zapasy składników w magazynie.

- `logger` — co sekundę zbiera dane i zapisuje log symulacji do pliku CSV.

Każdy wątek działa na współdzielonych zasobach przy pomocy zabezpieczeń synchronizacyjnych (np. `std::mutex`, `std::condition_variable`), aby zapobiegać konfliktom danych.

3 Uruchamianie programu

Aby uruchomić program, należy przygotować plik konfiguracyjny `config.json`, który powinien zawierać wszystkie wymagane dane wejściowe symulacji, zgodnie z poniższą strukturą:

- Lista stolików z ich ID i rozmiarami (`tables`),
- Liczba kelnerów (`num_waiters`),
- Menu wraz z czasem przygotowania oraz składnikami (`menu`),
- Zakres rozmiarów grup klientów (`group_size_range`),
- Zakres opóźnień między pojawieniem się kolejnych grup (`arrival_delay_range_ms`),
- Czas trwania symulacji (`simulation_time`),
- Stan początkowy magazynu składników (`ingredients_stock`),
- Parametry dostaw składników (`delivery_interval_ms`, `delivery_amount`).

Po uruchomieniu programu wczytywana jest konfiguracja, a następnie startowane są wszystkie wątki symulacji. Logi są zapisywane automatycznie do pliku `restaurant_log.csv` w katalogu zbudowanym przez CMake.

4 Mechanizmy prewencji konfliktów wątków

W projekcie symulacji działania restauracji, ze względu na współbieżny charakter aplikacji (wiele wątków: klienci, kelnerzy, kuchnia, dostawy), zastosowano szereg mechanizmów synchronizacji w celu uniknięcia konfliktów i zapewnienia poprawności danych.

- **Mutexy (`std::mutex`)** – zastosowane do ochrony współdzielonych zasobów przed równoczesnym dostępem. Przykładowo:
  - `restaurant.tables_mutex` – chroni listę stolików,
  - `table.mutex` – chroni stan pojedynczego stolika,
  - `restaurant.kitchen_mutex` – ochrona kolejki zamówień w kuchni,
  - `restaurant.ingredient_mutex` – chroni stan magazynu składników.
- **Zmienna warunkowa (`std::condition_variable`)** – używana w kuchni do oczekiwania na pojawienie się nowych zamówień. Wątek kucharza czeka efektywnie (bez aktywnego sprawdzania), aż do powiadomienia go przez kelnera o dodaniu zamówienia.
- **Lokalne blokady (`std::lock_guard` i `std::unique_lock`)** – zapewniają RAII dla mutexów i bezpieczne zwalnianie blokad, nawet w przypadku wyjątków.

- **Unikanie wyścigów przy alokacji ID grup** – za pomocą osobnego mutexa (`restaurant.group_id_mutex`) zapewniono, że tylko jeden wątek może jednocześnie zwiększyć wartość `next_group_id`.
- **Requeue zamówień w przypadku braku składników** – jeśli kuchnia nie może przygotować zamówienia z powodu braku składników, zamówienie trafia z powrotem do kolejki. Zastosowano lekkie opóźnienie (200 ms), by uniknąć zapętlenia i nadmiernego obciążenia CPU.
- **Thread-safe kolejki FIFO** – kolejki takie jak `waiting_queue`, `kitchen_queue` i `completed_orders` są osłonięte mutexami, co zapobiega uszkodzeniu danych przy jednoczesnym zapisie i odczycie.

Zastosowanie tych mechanizmów pozwoliło osiągnąć stabilną i przewidywalną symulację, nawet przy intensywnym obciążeniu wielowątkowym. Wykorzystanie standardowych narzędzi biblioteki STL zwiększyło bezpieczeństwo implementacji oraz czytelność kodu.

5 Testy

Do testów przygotowano zestaw plików `config.json`, różniących się kluczowymi parametrami: liczbą kelnerów, liczbą i wielkością stolików, czasem dostawy składników, składem magazynu, czasem przygotowania dań oraz zakresem liczby klientów. Każda konfiguracja była analizowana pod kątem:

- efektywności obsługi klientów (np. brak kolejek oczekujących),
- optymalnego wykorzystania stolików i zasobów kuchennych,
- wpływu opóźnień dostaw i niedoboru składników na liczbę odrzuconych zamówień,
- obciążenia kelnerów.

Zostało przeprowadzonych 6 testów z różnymi zestawami parametrów:

1. Krótki interwał dostawy (3000 ms), małe grupy klientów (1–3), niewielka liczba składników — system radził sobie dobrze, występowała niewielka kolejka oczekujących.
2. Duże grupy klientów (5–8) i tylko dwa stoliki — obserwowano częstsze kolejki i opóźnienia w obsłudze.
3. Mała liczba kelnerów (1) i duży ruch klientów — kuchnia miała zaległości, jednak nie doszło do blokad.
4. Duża liczba składników i długi czas dostawy (8000 ms) — system miał zapasy i funkcjonował płynnie mimo rzadkich dostaw.
5. Zwiększenie liczby kelnerów do 4 przy tych samych warunkach — zauważalny wzrost płynności obsługi i redukcja liczby oczekujących.
6. Maksymalne wartości klientów i minimalne zasoby — system utrzymał stabilność, ale wiele grup nie zostało obsłużonych na czas.

Wyniki testów zostały zapisane w plikach CSV generowanych automatycznie przez program w pliku `restaurant_log.csv`. Ze względu na dużą liczbę zmiennych i parametrów, dane są trudne do przedstawienia w formie tekstowej w sprawozdaniu, jednak można się z nimi zapoznać w załączniku.

System zachowywał się zgodnie z przewidywaniami. Nie wystąpiły błędy krytyczne ani konflikty współbieżności prowadzące do awarii aplikacji. Wszystkie wątki zakończyły działanie poprawnie, a zasoby zostały skutecznie zarządzane.

Możliwe usprawnienia:

- Dodanie dynamicznego dostosowywania liczby kelnerów do aktualnego obciążenia,
- Zaawansowana analiza logów z wizualizacją (np. wykresy Gantta, średni czas oczekiwania),
- Obsługa rezerwacji oraz kolejki priorytetowej dla dużych grup.

6 Wyniki symulacji

Tabela 1 przedstawia średnie wyniki dla kilku różnych konfiguracji symulacji, przy próbkowaniu wykonywanym co sekundę. Dane zostały wygenerowane przy użyciu skryptu napisanego w języku Python.

Można zauważyć, że największy wpływ na przebieg symulacji miało zatrzymanie dostaw składników do kuchni. Prowadziło to do wydłużenia kolejek, zwiększenia średniego czasu oczekiwania gości, a także wydłużenia czasu przygotowania dań.

Warto również podkreślić, że pomimo stosunkowo małej liczby kelnerów, nie stanowili oni wąskiego gardła w obsłudze — w większości przypadków byli dostępni i nie przyczyniali się do opóźnień.

Dla zwiększenia dokładności wyników, możliwe jest zastosowanie większej częstotliwości próbkowania oraz wydłużenie całkowitego czasu trwania symulacji.

Nr	**Avg Queue Size** (groups)	**Avg Wait Time** (s)	**Avg Meal Time** (s)	**Peak Queue** (groups)	**Kitchen Queue** Size (orders)	**Waiters** (active)
1	1.40	8.29	13.91	5	9.19	2.0
2	0.61	3.82	19.21	3	11.05	2.0
3	6.79	26.62	17.41	17	12.45	2.0
4	2.43	11.12	N/A	5	12.33	2.0
5	0.55	N/A	N/A	3	9.55	1.0

Tabela 1: Restaurant Queue and Service Statistics

7 Podsumowanie

Program spełnił wszystkie założone cele projektowe. Udało się zrealizować realistyczną symulację pracy restauracji z wieloma wątkami, współdzielonymi strukturami danych i synchronizacją. System był odporny na przeciążenia oraz nieprzewidziane kombinacje parametrów konfiguracyjnych. Wyniki testów potwierdzają poprawność implementacji oraz stabilność programu.

Projekt stanowi dobrą bazę do dalszego rozwijania symulacji oraz potencjalnej rozbudowy o bardziej zaawansowane mechanizmy planowania i statystyki.

8 Literatura

1. Silberschatz A., Galvin P.B., Gagne G., *Operating System Concepts*, Wiley, 10th Edition.
2. Butenhof D., *Programming with POSIX Threads*, Addison-Wesley.
3. Stroustrup B., *The C++ Programming Language*, Addison-Wesley, 4th Edition.

9 Kod źródłowy

9.1 Kod źródłowy pliku głównego w C++

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <queue>
#include <random>
#include <chrono>
#include <algorithm>
#include <sstream>
#include <iomanip>
#include <nlohmann/json.hpp>

using json = nlohmann::json;
using namespace std::chrono;

// Random number generation
std::random_device rd;
std::mt19937 gen(rd());

// Restaurant data structures
struct Table {
    int id;
    int size;
    bool is_occupied;
    int group_id;
    std::vector<std::string> orders;
    std::mutex mutex;

    Table() : id(0), size(0), is_occupied(false), group_id(0) {}

    Table(int id_, int size_) : id(id_), size(size_), is_occupied(false),
        group_id(0) {}

    Table(const Table &) = delete;

    Table &operator=(const Table &) = delete;

```

```
    Table(Table &&other) noexcept
            : id(other.id),
              size(other.size),
              is_occupied(other.is_occupied),
              group_id(other.group_id),
              orders(std::move(other.orders)) {
        // Mutex is default-constructed
    }

    Table &operator=(Table &&other) noexcept {
        if (this != &other) {
            id = other.id;
            size = other.size;
            is_occupied = other.is_occupied;
            group_id = other.group_id;
            orders = std::move(other.orders);
            // Mutex is default-constructed
        }
        return *this;
    }
};

struct Order {
    int group_id;
    int table_id;
    std::string dish;
    milliseconds prep_time;
};

struct CustomerGroup {
    int id;
    int size;
    std::vector<std::string> orders;
};

// Global state
struct Restaurant {
    std::map<std::string, int> ingredients_stock;
    std::map<std::string, std::vector<std::string>> recipe;
    std::mutex ingredient_mutex;
    int delivery_interval_ms;
    int delivery_amount;
    std::vector<Table> tables;
    std::mutex tables_mutex;
    int num_waiters;
    std::vector<std::string> menu_items;
    std::vector<milliseconds> prep_times;
    int group_size_min, group_size_max;
    int arrival_delay_min_ms, arrival_delay_max_ms;
    int simulation_time;
    std::queue<CustomerGroup> waiting_queue;
    std::queue<Order> kitchen_queue;
    std::queue<Order> completed_orders; // New queue for prepared orders
    std::mutex queue_mutex;
    std::mutex kitchen_mutex;
    std::mutex completed_orders_mutex; // Mutex for completed_orders
    std::condition_variable kitchen_cv;
    bool running;
```

```
    std::mutex running_mutex;
    int next_group_id;
    std::mutex group_id_mutex;
};

// Read configuration from JSON file
void load_config(const std::string &filename, Restaurant &restaurant) {
    std::ifstream file(filename);
    json j;
    file >> j;

    restaurant.next_group_id = 1;
    restaurant.running = true;

    // Load tables
    restaurant.tables.clear();
    restaurant.tables.reserve(j["tables"].size());
    for (const auto &table: j["tables"]) {
        restaurant.tables.emplace_back(table["id"], table["size"]);
    }

    // Load waiters
    restaurant.num_waiters = j["num_waiters"];

    // Load menu
    for (const auto &item: j["menu"]) {
        restaurant.menu_items.push_back(item["name"]);
        restaurant.prep_times.push_back(milliseconds(item["prep_time_ms"]))
            ;
    }

    // Load group size range
    restaurant.group_size_min = j["group_size_range"]["min"];
    restaurant.group_size_max = j["group_size_range"]["max"];

    // Load arrival delay range
    restaurant.arrival_delay_min_ms = j["arrival_delay_range_ms"]["min"];
    restaurant.arrival_delay_max_ms = j["arrival_delay_range_ms"]["max"];

    // Load simulation time
    restaurant.simulation_time = j["simulation_time"];
    // Load recipes
    for (const auto &item: j["menu"]) {
        std::string dish = item["name"];
        restaurant.recipe[dish] = item["ingredients"].get<std::vector<std::
            string>>();
    }

    // Load initial ingredient stock
    restaurant.ingredients_stock = j["ingredients_stock"].get<std::map<std
        ::string, int>>();

    // Delivery config
    restaurant.delivery_interval_ms = j["delivery_interval_ms"];
    restaurant.delivery_amount = j["delivery_amount"];

}

```

```cpp
// Generate random number in range
int random_int(int min, int max) {
    std::uniform_int_distribution<> dis(min, max);
    return dis(gen);
}

// Generate random customer group
CustomerGroup generate_group(Restaurant &restaurant) {
    std::lock_guard<std::mutex> lock(restaurant.group_id_mutex);
    CustomerGroup group;
    group.id = restaurant.next_group_id++;
    group.size = random_int(restaurant.group_size_min, restaurant.
       group_size_max);
    int num_orders = group.size;
    for (int i = 0; i < num_orders; ++i) {
        int menu_idx = random_int(0, restaurant.menu_items.size() - 1);
        group.orders.push_back(restaurant.menu_items[menu_idx]);
    }
//    std::cout << "[Debug] Generated group " << group.id << " with size "
   << group.size << "\n";
    return group;
}

// Try to seat a group at a table
bool seat_group(Restaurant &restaurant, CustomerGroup &group) {
    std::lock_guard<std::mutex> tables_lock(restaurant.tables_mutex);
    for (auto &table: restaurant.tables) {
        std::lock_guard<std::mutex> lock(table.mutex);
        if (!table.is_occupied && table.size >= group.size) {
            table.is_occupied = true;
            table.group_id = group.id;
            table.orders = group.orders;
//            std::cout << "[Debug] Seated group " << group.id << " at
   table " << table.id << "\n";
            return true;
        }
    }
//    std::cout << "[Debug] Group " << group.id << " added to waiting queue
   \n";
    return false;
}

// Customer arrival thread
void customer_arrival(Restaurant &restaurant) {
    while (true) {
        {
            std::lock_guard<std::mutex> lock(restaurant.running_mutex);
            if (!restaurant.running) break;
        }

        CustomerGroup group = generate_group(restaurant);
        bool seated = seat_group(restaurant, group);
        if (!seated) {
            std::lock_guard<std::mutex> lock(restaurant.queue_mutex);
            restaurant.waiting_queue.push(group);
        }

```

```
205         int delay = random_int(restaurant.arrival_delay_min_ms, restaurant.
                arrival_delay_max_ms);
206         std::this_thread::sleep_for(milliseconds(delay));
207     }
208 }
209
210 // Waiter thread
211 void waiter(Restaurant &restaurant, int waiter_id) {
212     while (true) {
213         {
214             std::lock_guard<std::mutex> lock(restaurant.running_mutex);
215             if (!restaurant.running) break;
216         }
217
218         // Check for orders to take
219         {
220             std::lock_guard<std::mutex> tables_lock(restaurant.tables_mutex
                    );
221             for (auto &table: restaurant.tables) {
222                 std::lock_guard<std::mutex> lock(table.mutex);
223                 if (table.is_occupied && !table.orders.empty()) {
224                     std::vector<Order> orders;
225                     for (const auto &dish: table.orders) {
226                         int menu_idx = std::find(restaurant.menu_items.
                                begin(), restaurant.menu_items.end(), dish) -
227                                         restaurant.menu_items.begin();
228                         orders.push_back({table.group_id, table.id, dish,
                                restaurant.prep_times[menu_idx]});
229                     }
230                     table.orders.clear();
231                     {
232                         std::lock_guard<std::mutex> kitchen_lock(restaurant
                                .kitchen_mutex);
233                         for (const auto &order: orders) {
234                             restaurant.kitchen_queue.push(order);
235 //                              std::cout << "[Debug] Waiter " << waiter_id
    << " submitted order " << order.dish << " for group " << order.group_id
    << "\n";
236                         }
237                         restaurant.kitchen_cv.notify_one();
238                     }
239                     continue;
240                 }
241             }
242         }
243
244         // Check for completed orders to deliver
245         std::vector<Order> to_deliver;
246         {
247             std::unique_lock<std::mutex> lock(restaurant.
                    completed_orders_mutex);
248             while (!restaurant.completed_orders.empty() && to_deliver.size
                    () < 3) {
249                 to_deliver.push_back(restaurant.completed_orders.front());
250                 restaurant.completed_orders.pop();
251             }
252         }
253         if (!to_deliver.empty()) {
```

```
254             std::lock_guard<std::mutex> tables_lock(restaurant.tables_mutex
                    );
255             for (const auto &order: to_deliver) {
256                 for (auto &table: restaurant.tables) {
257                     std::lock_guard<std::mutex> lock(table.mutex);
258                     if (table.group_id == order.group_id && table.id ==
                            order.table_id) {
259                         // Simulate delivery and eating
260                         std::this_thread::sleep_for(milliseconds(500));
261                         table.is_occupied = false;
262                         table.group_id = 0;
263 //                          std::cout << "[Debug] Waiter " << waiter_id << "
    delivered " << order.dish << " to group " << order.group_id << " at
    table " << table.id << "\n";
264                         // Try to seat a waiting group
265                         CustomerGroup group;
266                         bool has_group = false;
267                         {
268                             std::lock_guard<std::mutex> queue_lock(
                                    restaurant.queue_mutex);
269                             if (!restaurant.waiting_queue.empty()) {
270                                 group = restaurant.waiting_queue.front();
271                                 restaurant.waiting_queue.pop();
272                                 has_group = true;
273                             }
274                         }
275                         if (has_group && table.size >= group.size) {
276                             table.is_occupied = true;
277                             table.group_id = group.id;
278                             table.orders = group.orders;
279 //                              std::cout << "[Debug] Seated waiting group "
    << group.id << " at table " << table.id << "\n";
280                         }
281                         break;
282                     }
283                 }
284             }
285         }
286
287         std::this_thread::sleep_for(milliseconds(500));
288     }
289 }
290
291 // Kitchen thread
292 void kitchen(Restaurant &restaurant) {
293     while (true) {
294         std::unique_lock<std::mutex> lock(restaurant.kitchen_mutex);
295         restaurant.kitchen_cv.wait(lock, [&] {
296             return !restaurant.kitchen_queue.empty() || !restaurant.running
                    ;
297         });
298         {
299             std::lock_guard<std::mutex> running_lock(restaurant.
                    running_mutex);
300             if (!restaurant.running && restaurant.kitchen_queue.empty())
                    break;
301         }
302         if (!restaurant.kitchen_queue.empty()) {
```

```
303             Order order = restaurant.kitchen_queue.front();
304             restaurant.kitchen_queue.pop();
305             lock.unlock();
306             bool can_prepare = true;
307             {
308                 std::lock_guard<std::mutex> lock(restaurant.
                        ingredient_mutex);
309                 for (const auto &ingredient: restaurant.recipe[order.dish])
                        {
310                     if (restaurant.ingredients_stock[ingredient] <= 0) {
311                         can_prepare = false;
312                         break;
313                     }
314                 }
315                 if (can_prepare) {
316                     for (const auto &ingredient: restaurant.recipe[order.
                            dish]) {
317                         restaurant.ingredients_stock[ingredient]--;
318                     }
319                 }
320             }
321 
322             if (can_prepare) {
323                 std::this_thread::sleep_for(order.prep_time);
324                 std::lock_guard<std::mutex> completed_lock(restaurant.
                        completed_orders_mutex);
325                 restaurant.completed_orders.push(order);
326             } else {
327                 // requeue order if ingredients missing
328                 std::lock_guard<std::mutex> kitchen_lock(restaurant.
                        kitchen_mutex);
329                 restaurant.kitchen_queue.push(order);
330                 std::this_thread::sleep_for(milliseconds(200)); // avoid
                        tight loop
331             }
332 
333             {
334                 std::lock_guard<std::mutex> completed_lock(restaurant.
                        completed_orders_mutex);
335                 restaurant.completed_orders.push(order);
336 //                std::cout << "[Debug] Kitchen completed order " << order.
    dish << " for group " << order.group_id << "\n";
337             }
338         }
339     }
340 }
341 
342 void ingredient_delivery(Restaurant &restaurant) {
343     while (true) {
344         {
345             std::lock_guard<std::mutex> lock(restaurant.running_mutex);
346             if (!restaurant.running) break;
347         }
348 
349         {
350             std::lock_guard<std::mutex> lock(restaurant.ingredient_mutex);
351             for (auto &pair: restaurant.ingredients_stock) {
352                 pair.second += restaurant.delivery_amount;
```

```
353             }
354 //              std::cout << "[Delivery] Ingredients restocked\n";
355         }
356 
357         std::this_thread::sleep_for(milliseconds(restaurant.
                delivery_interval_ms));
358     }
359 }
360 
361 void logger(Restaurant &restaurant) {
362     auto start = std::chrono::system_clock::now();
363     std::ofstream log_file("restaurant_log.csv", std::ios::app);
364 
365     // Write CSV header
366     log_file << "Timestamp,Tables,Kitchen Queue,Completed Orders,Waiting
            Queue,Ingredients,Waiters\n";
367 
368     while (true) {
369         std::stringstream tables_ss, kitchen_ss, completed_ss, waiting_ss,
                ingredients_ss;
370         auto now = std::chrono::system_clock::now();
371         auto timestamp = std::chrono::system_clock::to_time_t(now);
372 
373         // Tables status
374         std::cout << "Tables:\n";
375         tables_ss << "\"";
376         {
377             std::lock_guard<std::mutex> tables_lock(restaurant.tables_mutex
                    );
378             for (auto &table : restaurant.tables) {
379                 std::lock_guard<std::mutex> lock(table.mutex);
380                 std::cout << "  Table " << table.id << " (size " << table.
                        size << "): "
381                           << (table.is_occupied ? "Occupied (Group " + std
                              ::to_string(table.group_id) + ")" : "Free");
382                 tables_ss << "Table " << table.id << " (size " << table.
                        size << "): "
383                           << (table.is_occupied ? "Occupied (Group " + std
                              ::to_string(table.group_id) + ")" : "Free");
384                 if (!table.orders.empty()) {
385                     std::cout << ", Orders: ";
386                     tables_ss << ", Orders: ";
387                     for (const auto &order : table.orders) {
388                         std::cout << order << " ";
389                         tables_ss << order << " ";
390                     }
391                 }
392                 std::cout << "\n";
393                 tables_ss << ";";
394             }
395         }
396         tables_ss << "\"";
397 
398         // Kitchen Queue
399         std::cout << "Kitchen Queue: ";
400         kitchen_ss << "\"";
401         {
402             std::lock_guard<std::mutex> lock(restaurant.kitchen_mutex);
```

```
            auto temp = restaurant.kitchen_queue;
            while (!temp.empty()) {
                auto order = temp.front();
                std::cout << order.dish << "(Group " << order.group_id << ") ";
                kitchen_ss << order.dish << "(Group " << order.group_id << ") ";
                temp.pop();
            }
        }
        std::cout << "\n";
        kitchen_ss << "\"";

        // Completed Orders
        std::cout << "Completed Orders: ";
        completed_ss << "\"";
        {
            std::lock_guard<std::mutex> lock(restaurant.completed_orders_mutex);
            auto temp = restaurant.completed_orders;
            while (!temp.empty()) {
                auto order = temp.front();
                std::cout << order.dish << "(Group " << order.group_id << ") ";
                completed_ss << order.dish << "(Group " << order.group_id << ") ";
                temp.pop();
            }
        }
        std::cout << "\n";
        completed_ss << "\"";

        // Waiting Queue
        std::cout << "Waiting Queue: ";
        waiting_ss << "\"";
        {
            std::lock_guard<std::mutex> lock(restaurant.queue_mutex);
            auto temp = restaurant.waiting_queue;
            while (!temp.empty()) {
                auto group = temp.front();
                std::cout << "Group " << group.id << "(size " << group.size << ") ";
                waiting_ss << "Group " << group.id << "(size " << group.size << ") ";
                temp.pop();
            }
        }
        std::cout << "\n";
        waiting_ss << "\"";

        // Ingredient Stock
        std::cout << "Ingredients Stock:\n";
        ingredients_ss << "\"";
        {
            std::lock_guard<std::mutex> lock(restaurant.ingredient_mutex);
            for (const auto &pair : restaurant.ingredients_stock) {
                std::cout << "  " << pair.first << ": " << pair.second << "\n";
```

```
                ingredients_ss << pair.first << ":" << pair.second << ";";
            }
        }
        ingredients_ss << "\"";

        // Waiter count
        std::cout << "Waiters: " << restaurant.num_waiters << " active\n";
        std::string waiters_str = std::to_string(restaurant.num_waiters) +
            " active";

        // Write to CSV
        log_file << std::put_time(std::localtime(&timestamp), "%Y-%m-%d %H
            :%M:%S") << ","
                 << tables_ss.str() << ","
                 << kitchen_ss.str() << ","
                 << completed_ss.str() << ","
                 << waiting_ss.str() << ","
                 << ingredients_ss.str() << ","
                 << "\"" << waiters_str << "\"\n";
        log_file.flush();

        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
}

int main() {
    Restaurant restaurant;
    load_config("../config.json", restaurant);
    std::thread customer_thread(customer_arrival, std::ref(restaurant));
    std::vector<std::thread> waiter_threads;
    std::thread delivery_thread(ingredient_delivery, std::ref(restaurant));
    for (int i = 1; i <= restaurant.num_waiters; ++i) {
        waiter_threads.emplace_back(waiter, std::ref(restaurant), i);
    }
    std::thread kitchen_thread(kitchen, std::ref(restaurant));
    std::thread logger_thread(logger, std::ref(restaurant));

    customer_thread.join();
    for (auto &t: waiter_threads) t.join();
    kitchen_thread.join();
    logger_thread.join();
    delivery_thread.join();

    return 0;
}
```

Listing 1: Przykładowy kod w C++

9.2 Kod źródłowy w Pythonie pliku queue_stats.py

```
import pandas as pd, re, datetime as dt, os

FILE_PATH = os.getenv('CSV_PATH', 'restaurant_log6.csv')

df = pd.read_csv(FILE_PATH)

```

```python
def count_waiting_groups(cell: str) -> int:
    if pd.isna(cell) or not str(cell).strip():
        return 0
    return len(re.findall(r'Group \d+\(size \d+\)', str(cell)))

def count_kitchen_orders(cell: str) -> int:
    if pd.isna(cell) or not str(cell).strip():
        return 0
    return len(re.findall(r'[A-Za-z]+\(Group \d+\)', str(cell)))

# Average waiting queue size
waiting_group_counts = df['Waiting Queue'].apply(count_waiting_groups)
avg_queue_size = waiting_group_counts.mean()

# Peak waiting queue size
peak_queue_size = waiting_group_counts.max()

# Average kitchen queue size
kitchen_queue_counts = df['Kitchen Queue'].apply(count_kitchen_orders)
avg_kitchen_queue_size = kitchen_queue_counts.mean()

# Average number of active waiters
def parse_waiters(cell: str) -> int:
    if pd.isna(cell) or not str(cell).strip():
        return 0
    m = re.match(r'(\d+)', str(cell).strip())
    return int(m.group(1)) if m else 0

avg_active_waiters = df['Waiters'].apply(parse_waiters).mean()

# Tracking waiting and meal times
queue_entry = {}   # group_id -> (timestamp, group_size)
seating_time = {}  # group_id -> (timestamp, group_size)

wait_seconds_total = 0
wait_persons_total = 0

meal_wait_seconds_total = 0
meal_wait_persons_total = 0

for _, row in df.iterrows():
    ts = pd.to_datetime(row['Timestamp'])

    # groups currently in waiting queue
    waiting_cell = str(row['Waiting Queue'])
    for grp_id, size in re.findall(r'Group (\d+)\(size (\d+)\)',
        waiting_cell):
        gid = int(grp_id); sz = int(size)
        if gid not in queue_entry:
            queue_entry[gid] = (ts, sz)

    # groups currently seated
    tables_cell = str(row['Tables'])
    for grp_id in re.findall(r'Group (\d+)', tables_cell):
        gid = int(grp_id)
        if gid in queue_entry:
            start_ts, sz = queue_entry.pop(gid)
            wait_seconds_total += (ts - start_ts).total_seconds() * sz
```

```
            wait_persons_total += sz
            seating_time[gid] = (ts, sz)

    # completed orders
    completed_cell = str(row['Completed Orders'])
    for grp_id in re.findall(r'Group (\d+)', completed_cell):
        gid = int(grp_id)
        if gid in seating_time:
            seat_ts, sz = seating_time.pop(gid)
            meal_wait_seconds_total += (ts - seat_ts).total_seconds() * sz
            meal_wait_persons_total += sz

avg_wait_queue_time = (wait_seconds_total / wait_persons_total) if
    wait_persons_total else None
avg_meal_wait_time = (meal_wait_seconds_total / meal_wait_persons_total) if
    meal_wait_persons_total else None

stats = {
    'Average queue size (groups)': round(avg_queue_size, 2),
    'Average time a person waited in queue (seconds)': round(
        avg_wait_queue_time, 2) if avg_wait_queue_time is not None else 'N/A
        ',
    'Average time a person waited for a meal (seconds)': round(
        avg_meal_wait_time, 2) if avg_meal_wait_time is not None else 'N/A',
    'Peak queue size (groups)': int(peak_queue_size),
    'Average kitchen queue size (orders)': round(avg_kitchen_queue_size, 2)
        ,
    'Average active waiters': round(avg_active_waiters, 2)
}

if __name__ == '__main__':
    for k, v in stats.items():
        print(f"{k}: {v}")
```

Listing 2: Przykładowy kod w Pythonie