Raport z Projektu - System Obsługi Lotniska

Aleksander Dygoń 151856

1. Założenia projektowe

Projekt implementuje system symulujący działanie lotniska z następującymi głównymi komponentami:

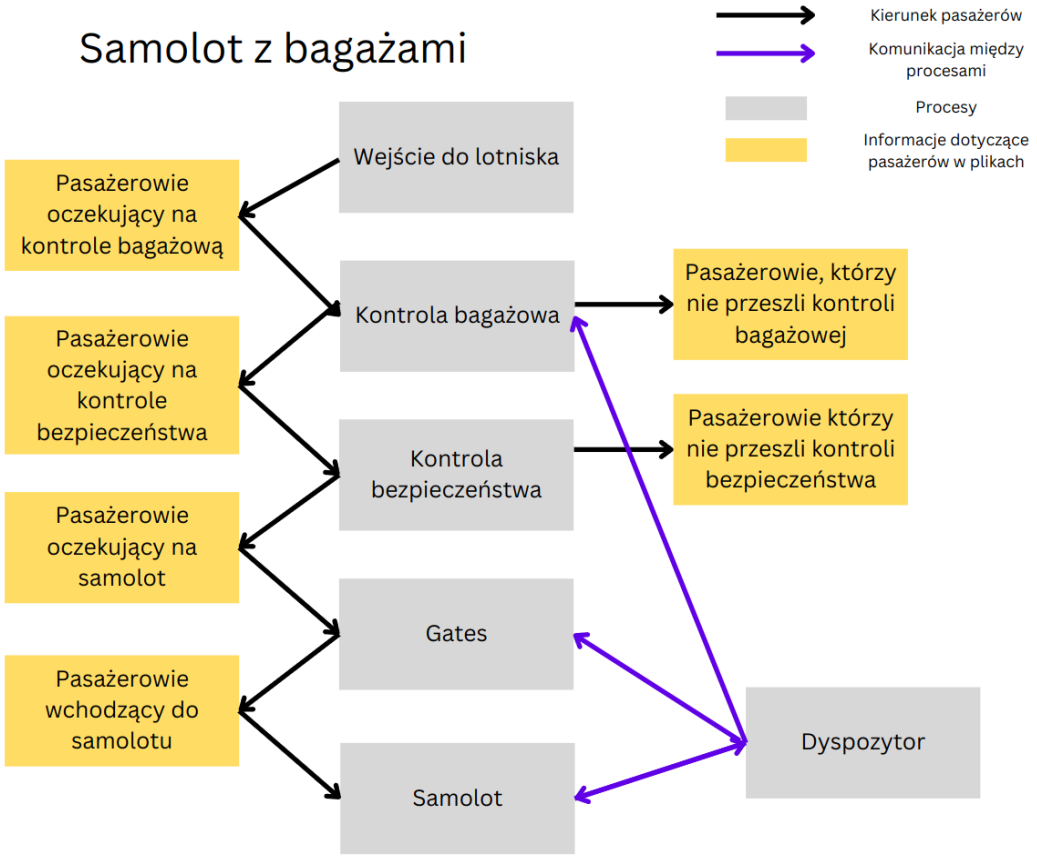
- Generator pasażerów
- Kontrola biletowo-bagażowa
- Kontrola bezpieczeństwa
- System zarządzania bramkami (gates)
- Dyspozytor lotów
- Proces obsługi samolotu

1.1 Główne wymagania funkcjonalne:

- Odprawa biletowo-bagażowa z limitem wagowym
- 3 równoległe stanowiska kontroli bezpieczeństwa
- Maksymalnie 2 osoby tej samej płci na stanowisku
- Limit 3 przepuszczeń w kolejce dla każdego pasażera
- System obsługi VIP
- Zarządzanie schodami pasażerskimi o ograniczonej pojemności
- Koordynacja lotów przez dyspozytora
- Zarządzanie flotą samolotów

2. Implementacja wymagań obowiązkowych

Diagram komunikacji i zależności w projekcie



2.1 Dokumentacja przypadków użycia

Projekt zawiera szczegółową dokumentację w postaci komentarzy w kodzie oraz struktury modułowej reprezentującej poszczególne komponenty systemu. Główne przypadki użycia są zaimplementowane w oddzielnych modułach:

- ./src/generator.py - generowanie pasażerów
- ./src/luggageControl.py - kontrola bagażowa
- ./src/securityControl.py - kontrola bezpieczeństwa
- ./src/gate.py - obsługa bramek
- ./src/dispatcher.py - zarządzanie lotami
- ./src/airplane.py - obsługa samolotu
- ./src/consts.py - stałe oraz konfiguracja programów

2.2 Walidacja danych

Walidacja została zaimplementowana w module utils.py w funkcji validate_config()

2.3 Obsługa błędów

Obsługa błędów systemowych jest zaimplementowana w module utils.py w funkcji handle_system_error()

2.4 Minimalne prawa dostępu

Projekt implementuje minimalne prawa dostępu dla tworzonych struktur:

- Kolejek
- Plików

3. Realizacja wymagań

a. Tworzenie i obsługa plików

- Tworzenie pliku
- Zapis do pliku
- Odczyt z pliku

b. Tworzenie procesów

- Tworzenie
- Kończenie

c. Obsługa sygnałów

- Obsługa przerwania

d. Synchronizacja procesów

- Synchronizacja przez pliki

e. Kolejki komunikatów

- Tworzenie kolejek
- Dodawanie do kolejki
- Pobieranie z kolejki

f. Pamięć współdzielona

- Tworzenie pamięci współdzielonej
- Modyfikacja pamięci współdzielonej

4. Problemy i rozwiązania

4.1 Synchronizacja dostępu do plików

Problem: Równoległy dostęp do plików przez różne procesy.
Rozwiązanie: Wykorzystanie mechanizmu fcntl do blokowania i odblokowywania plików

4.2 Koordynacja procesów

Problem: Koordynacja wielu procesów i przekazywanie sygnałów.
Rozwiązanie: Wykorzystanie kolejek komunikatów:

- Tworzenie kolejek
- Dodawanie do kolejki
- Pobieranie z kolejki

5. Testy

5.1 Struktura testów

Projekt zawiera kompleksowy zestaw testów podzielony na dwie główne kategorie:

1. Testy jednostkowe poszczególnych komponentów:

- ./tests/test_generator.py - testy generatora pasażerów
  - TEST_1: Wygenerowanie oraz sprawdzanie poprawności generowanych danych
- ./tests/test_luggageControl.py - testy kontroli bagażowej
  - TEST_1: Kontrola pasażera z prawidłową wagą bagażu
  - TEST_2: Kontrola pasażera z nadwagą bagażu podręcznego
- ./tests/test_securityControl.py - testy kontroli bezpieczeństwa
  - TEST_1: Kontrola pojedyńczego pasażera bez niebezpiecznych przedmiotów
  - TEST_2: Kontrola pasażera z niebezpiecznym przedmiotem
  - TEST_3: Kontrola wielu pasażerów
  - TEST_4: Weryfikacja zasady zachowania płci na stanowisku
  - TEST_5: Weryfikacja przepuszczeń pasażerów
  - TEST_6: Weryfikacja obsługi VIP
- ./tests/test_gate.py - testy systemu bramek
  - TEST_1: Sprawdzenie odlotu samolotu przy idealnych warunkach
  - TEST_2: Sprawdzenie odlotu samolotu przy braku pasażerów do zapełnienia samolotu
  - TEST_3: Sprawdzenie odlotu samolotu przy nadmiarze pasażerów oczekujących na odlot

2. Test end-to-end:

- ./tests/test_e2e.py - test integracyjny sprawdzający przepływ pasażerów przez cały system
  - TEST_1: Przepływ pasażerów przez cały system lotniskowy. Sprawdzenie wydajności kontroli bagażowej oraz bezpieczeństwa. Synchronizacja wielu odlotów oraz powrót samolotów na lotnisko.

5.2 Zakres testów

Testy jednostkowe

Każdy moduł ma dedykowany zestaw testów sprawdzających:

- Poprawność walidacji danych
- Obsługę skrajnych przypadków

Test end-to-end

Test ./tests/test_e2e.py weryfikuje:

- Pełny przepływ pasażera przez system
- Integrację wszystkich komponentów
- Poprawność synchronizacji procesów

6. Elementy wyróżniające projekt

- Zaawansowany system logowania i statystyk:
- Moduł stats.py zbierający kompleksowe statystyki
- System logowania z timestampami
- Testy end-to-end
- Wielopoziomowa walidacja.

7. Wnioski

Projekt spełnia wszystkie wymagane funkcjonalności i implementuje zaawansowane mechanizmy synchronizacji i komunikacji między procesami. Szczególnie warte uwagi są:

1. Kompleksowa obsługa błędów i walidacja danych
2. Efektywna synchronizacja procesów
3. Modułowa struktura kodu
4. Rozbudowany system logowania i statystyk

Podczas implementacji szczególnie wymagające okazały się:

- Koordynacja wielu procesów
- Zapewnienie spójności danych przy równoległym dostępie

- Implementacja systemu priorytetów dla pasażerów (limitu przepuszczeń)

8. Włączenie symulacji

8.1 Wymagania

- Python 3.x

8.2 Uruchomienie

```
cd src && python3 main.py
```

8.3 Zebranie statystyk

Pliki ze statystykami zostaną zapisane w katalogu
`./stats/stats_{SIMULATE_START_TIMESTAMP}.json`

```
# Z katalogu src
python3 stats.py
```