Laboratorium 6

Lotnisko

Ćwiczenie 6.1

Hangar na lotnisku może przechowywać *POJEMNOSC_HANGARU* samolotów. Samolot wykonuje następujące czynności przylatuje do lotniska, ląduje na pasie startowym (jest tylko jeden), kołuje do hali przylotów, udaje się do hangaru, kołuje do hali odlotów i startuje. Użyj semaforów do rozwiązania tego zadania. Samolotów jest *LICZBA_SAMOLOTOW*. Stałe *POJEMNOSC_HANGARU* oraz *LICZBA_SAMOLOTOW* są definiowane przez użytkownika. Uzupełnij program przedstawiony na listingu 6.1 tak aby lotnisko funkcjonowało prawidłowo (w hangarze nie może znajdować się więcej niż *POJEMNOSC_HANDARU* samolotów, na pasie startowym w danym momencie może znajdować się tylko 1 samolot)

Listing 6.1

```
#include <thread>
#include <mutex>
#include <random>
#include <iostream>
#include <set>
#include <vector>
#include <thread>

using namespace std;

const int POJEMNOSC_HANGARU=5;
const int LICZBA_SAMOLOTOW=10;

random_device rd;
minstd_rand generator(rd());
uniform_int_distribution<int> zakres(1, 3);

class Hangar;
class Samolot;

class Lotnisko {
public:

    Lotnisko(Hangar &h) : pasWolny(true), hangar(h) {
    }
    void laduj(Samolot &s);
    void kolujDoHangaru(Samolot &s);
    void opuscHangar(Samolot &s);
    void startuj(Samolot &s);
private:
    bool pasWolny;
    Hangar &hangar;
};

class Samolot {
public:

    Samolot(string n, Lotnisko &l) : nazwa(n), lotnisko(l) {
    }
```

```
    string nazwa;

    void operator()() {
        this_thread::sleep_for(chrono::seconds(zakres(generator)));
        lotnisko.laduj(*this);
        // postoj przy hali przylotow
        this_thread::sleep_for(chrono::seconds(zakres(generator)));
        lotnisko.kolujDoHangaru(*this);
        // postoj w hangarze
        this_thread::sleep_for(chrono::seconds(zakres(generator)));
        lotnisko.opuscHangar(*this);
        // postoj przy hali odlotow
        this_thread::sleep_for(chrono::seconds(zakres(generator)));
        lotnisko.startuj(*this);
    }

private:
    Lotnisko &lotnisko;
};

class Hangar {
public:

    Hangar(int n) : rozmiarHangaru(n) {
    }

    void parkuj(Samolot &s) {
        if (samoloty.size() < rozmiarHangaru) {
            samoloty.insert(s.nazwa);
        } else {
            cout << "Hangar pelny\n";
        }
    }

    void opusc(Samolot &s) {
        if (samoloty.count(s.nazwa) == 1) {
            samoloty.erase(s.nazwa);
        } else {
            cout << "W hangarze nie ma tego samolotu\n";
        }
    }

private:
    unsigned rozmiarHangaru;
    set<string> samoloty;
};

void Lotnisko::laduj(Samolot &s) {
    if (pasWolny) {
        pasWolny = false;
        string info = "Samolot " + s.nazwa + " laduje\n";
        cout << info;
        this_thread::sleep_for(chrono::seconds(zakres(generator)));
        info = "Samolot " + s.nazwa + " wyladowal\n";
        pasWolny = true;
    } else {
        cout << "Zderzenie na pasie startowym\n";
    }
}
```

```
void Lotnisko::kolujDoHangaru(Samolot &s) {
    hangar.parkuj(s);
}

void Lotnisko::opuscHangar(Samolot &s) {
    hangar.opusc(s);
}

void Lotnisko::startuj(Samolot &s) {
    if (pasWolny) {
        pasWolny = false;
        string info = "Samolot " + s.nazwa + " startuje\n";
        cout << info;
        this_thread::sleep_for(chrono::seconds(zakres(generator)));
        info = "Samolot " + s.nazwa + " wystartowal\n";
        pasWolny = true;
    } else {
        cout << "Zderzenie na pasie startowym\n";
    }
}

int main(int argc, char** argv) {
    Hangar hangar(POJEMNOSC_HANGARU);
    Lotnisko lotnisko(hangar);

    vector<thread> watki;

    for (int i = 0; i < LICZBA_SAMOLOTOW; ++i) {
        string napis = "Rownolegle Linie Lotnicze "+to_string(i);
        watki.emplace_back(Samolot(napis, lotnisko));
    }

    for (thread &w: watki) w.join();

    return EXIT_SUCCESS;
}
```