# Systemy operacyjne 2

**Projekt**

**Etap 2**

**252737 Andrzej Olszewski**

# 1. Opis problemu oraz jego rozwiązania

Wykonany przeze mię program jest implementacją oraz rozwiązanie problemu ucztujących filozofów zwanym też problemem pięciu filozofów. Został on wymyślony w 1965 roku przez Edsgera Dijkstrę. Opis problemu:

Pięciu filozofów siedzi przy stole i każdy wykonuje jedną z dwóch czynności – albo je, albo rozmyśla. Stół jest okrągły, przed każdym z nich znajduje się miska ze spaghetti, a pomiędzy każdą sąsiadującą parą filozofów leży widelec, a więc każda osoba ma przy sobie dwie sztuki – po swojej lewej i prawej stronie. Ponieważ jedzenie potrawy jest trudne przy użyciu jednego widelca, zakłada się, że każdy filozof korzysta z dwóch. Dodatkowo nie ma możliwości skorzystania z widelca, który nie znajduje się bezpośrednio przed daną osobą. Problem ucztujących filozofów jest czasami przedstawiany przy użyciu ryżu, który musi być jedzony dwiema pałeczkami, co lepiej obrazuje sytuację. Filozofowie nigdy nie rozmawiają ze sobą, co stwarza zagrożenie zakleszczenia w sytuacji, gdy każdy z nich zabierze lewy widelec i będzie czekał na prawy (lub na odwrót).

Rozwiązanie (poprzez hierarchię zasobów)

Jednym z rozwiązań tego problemu jest możliwe przez ustalenie hierarchii dla widelców, którymi posługują się filozofowie. i wprowadzenie zasady, że kolejność dostępu do widelców jest ustalona przez ten porządek, a ich zwalnianie następuje w odwrotnej kolejności. Ponadto dwa widelce, które nie są ze sobą w relacje nie mogą zostać użyte przez jednego filozofa w tym samym czasie.

## 2. Budowa programu

Program składa się z 2 klas:

- Philosopher
- Fork

**Klasa Philosopher** jest to klasa symulująca zachowanie filozofa.

Opis wybranych pól i metod klasy:

- name – pole typu string przechowujące imię filozofa
- appetite – pole typu int mówiące o tym, ile musi zjeść filozof, aby się najeść
- food_eaten – pole typu int mówiące o tym, ile już zjadł filozof
- left_fork, right_fork – pole przechowujące referencje do klasy Fork
- stopped – pole typu bool mówiące o tym czy filozof skończył jeść
- eat – metoda na początku przypisuje referencje do obiektów klasy Fork do pól left_fork, right_fork, a następnie za pomocą funkcji randomize wyznacza czas potrzebny na jedzenie
- think -metoda za pomocą funkcji randomize wyznacza czas przez który filozof będzie myślał.

```
class Philosopher {

    string name;
    int appetite;
    int food_eaten;
    Fork &left_fork;
    Fork &right_fork;
    thread td;
    mutex &g_lock;
    mt19937 rng{random_device{}()};
    bool stopped;

public:
    Philosopher(string, int, Fork &, Fork &, mutex &);
    ~Philosopher();
    void eat();
    void think();
    void randomize(int, int);
    void print(const string &);
    void stop();
    void run();
    void join();
};
```

**Klasa Fork**

```
class Fork {
    string id;
public:
    mutex mtx;
    Fork();
};
```

**Główna funkcja programu –** funkcja main na początku inicjalizuje tablicę z widelcami oraz filozofami. Podczas gdy wątki się wykonują funkcja nasłuchuje czy ktoś nie przerwał programu (np. poprzez skrót klawiszowy ***CTRL+C***). W momencie, gdy wszystkie wątki wykonają się, program kończy pracę.

```
int main() {
    cout << "Dinner is served! (Ctl+C to end)" << endl;

    for (auto &fork: forks)
        fork = new Fork();

    cout << "\nPhilosophers: " << endl;
    for (int i = 0; i < names.size(); i++) {
        int rc = (i + 1) % (int)names.size();
        philosophers[i] = new Philosopher(colors[i] + names[i] +
colors[5], 5, *forks[i], *forks[rc],
                                          global_lock);
    }
    cout << endl << endl;

    signal(SIGINT, quit);

    for (Philosopher *p: philosophers)
        p->join();

    return 0;
}
```

# 3. Sposób kompilacji

Zaprezentowano sposób kompilacji i uruchomienia przy pomocy cmake w wersji 3.16.3
Na początku tworzymy katalog na zbudowanie pliki a następnie otwieramy go

***mkdir [nazwa_katalogu]***
***cd [nazwa_katalogu]***

Uruchamiamy program cmake na katalogu rodzicielskim

***cmake ..***

Uruchamiamy program make

***make***

Po wykonaniu powyższych czynności można uruchomić program:

***./Etap2***

## 4. Przykładowe wyniki działania programu

Przykładowe uruchomienie programu

```
andrzej@DESKTOP-P2L11KG:~/SO2/project/systemy-operacynne-projekt/Etap2/build-dir$ ./Etap2
Dinner is served! (Ctl+C to end)

Philosophers:
Plato has joined
Socrates has joined
Kant has joined
Descartes has joined
Confucius has joined


Plato is thinking
Socrates is thinking
Kant is thinking
Descartes is thinking
Confucius is thinking
Socrates is eating (1/5)
Descartes is eating (1/5)
```

Przykładowe zakończenie programu bez zatrzymywania programu

```
Kant is thinking
Socrates is eating (5/5)
Confucius is thinking
Descartes is eating (5/5)
Socrates is finished
Plato is eating (5/5)
Descartes is finished
Kant is eating (5/5)
Plato is finished
Confucius is eating (5/5)
Kant is finished
Confucius is finished
```

Przykładowe zakończenie programu poprzez przerwanie programu

```
Confucius is eating (1/5)
Socrates is thinking
Descartes is eating (1/5)
Plato is eating (2/5)
Confucius is thinking
^C
Dinner is over. Letting philosophers finish up...
Socrates is finished
Descartes is finished
Kant is eating (2/5)
Kant is finished
Plato is finished
Confucius is finished
```