

Lista 1

Termin wykonania: 2025-04-06

Prezentacje rozwiązań dla podanych zadań wykonywać z wykorzystaniem programu `tmux`, jak opisano na Liście 0 w Zadaniu 1.

W prezentacji powinny się pojawić kolejno:

- Komentarz postaci `# imię nazwisko numer indeksu`
- polecenie postaci `svn export URL_katalogu_z_rozwiazaniami_listy`
- polecenie postaci `cd wyeksportowana_ścieżka`
- w podkatalogu z każdym zadaniem, kompilacja i uruchomienie programów dla wymaganych testów.

Zadania polegają na implementacji, w językach Ada i Go, symulacji i zapisywaniu historii działania systemu, w którym k obiektów (*podróżników*) przemieszcza się po dwuwymiarowej kratce (*planszy*) o rozmiarach $m \times n$.

Pola planszy mają współrzędne postaci (x, y) , gdzie $x \in \{0, \dots, m - 1\}$ i $y \in \{0, \dots, n - 1\}$.

Plansza ma *topologię torusa*, tzn. pole (x, y) ma czterech sąsiadów:

- w poziomie: $((x - 1) \bmod m, y)$, $((x + 1) \bmod m, y)$, oraz
- w pionie: $(x, (y - 1) \bmod n)$, $(x, (y + 1) \bmod n)$.

Podróżnik może wykonywać *kroki*. W jednym kroku podróżnik przemieszcza się na jedno z sąsiednich pól.

Każdy podróżnik symulowany jest przez osobny wątek programu.

Poleceniem: `svn export https://repo.cs.pwr.edu.pl/info/PW/`
pobrać katalog **PW** z repozytorium `https://repo.cs.pwr.edu.pl/info/`.

W podkatalogu `./PW/ada/tasks` znajdzie się przykładowy program `travelers.adb` w Adzie symulujący działanie takiego systemu.

Parametry programu zdefiniowane są przy pomocy stałych, żeby nie trzeba było ich wczytywać:
`-- Travelers moving on the board`

```
Nr_Of_Travelers : constant Integer :=15; -- k
```

```
Min_Steps : constant Integer := 10 ;
```

```
Max_Steps : constant Integer := 100 ;
```

```
Min_Delay : constant Duration := 0.01;
```

```
Max_Delay : constant Duration := 0.05;
```

-- 2D Board with torus topology

```
Board_Width  : constant Integer := 15;  -- m
Board_Height : constant Integer := 15;  -- n
```

Program demonstruje symulację systemu działającego zgodnie z następującymi zasadami:

- Wszyscy podróżnicy pojawiają się na planszy na dowolnie wylosowanych polach.
- Następnie, każdy podróżnik niezależnie,
 - wykonuje losową liczbę, pomiędzy `Min_Steps` a `Max_Steps`, następujących działań:
 - odczekuje przez losowy czas pomiędzy `Min_Delay` a `Max_Delay` sekund,
 - przemieszcza się na losowo wybrane sąsiednie pole (wykonuje jeden krok),
 - po czym przekazuje odpowiednio sformatowaną historię swojej podróży do wątku `Printer`.

Program wypisuje na standardowe wyjście następujący ciąg wierszy:

- pierwszy wiersz postaci:

```
-1 Nr_Of_Travelers Board_Width Board_Height
```

gdzie *Nazwa_parametru* jest wartością liczbową danego parametru,
- kolejne wiersze opisują zdarzenia wykonania kroków i są postaci:

```
Time_Stamp Id X Y Symbol
```

gdzie kolejne pozycje są
 - liczbami oznaczającymi: czas zdarzenia, identyfikator i nowe współrzędne (X,Y) podróżnika, oraz
 - symbolem podróżnika w czasie zdarzenia.

Program można skompilować w podkatalogu: `./PW/ada/tasks` poleceniem:

```
gnatmake travelers.adb
```

a następnie uruchomić poleceniem:

```
./travelers > out
```

Do dynamicznego odtwarzania tego ciągu, przygotowany został skrypt

`display-travel.bash` w sąsiednim podkatalogu `../.. /bash-tools`, który tu można uruchomić poleceniem:

```
bash ../.. /bash-tools/display-travel.bash out
```

Skrypt po uruchomieniu wyświetli:

```
STEP = 0  TIME = -1
```

TO STEP?

Zadanie 1. (za 10 punktów)

Dla outputu wygenerowanego poleceniem (na przykład):

umieścić w prezentacji polecenia:

importowanie SVN

Zadanie 2. (za 10 punktów)

Zaimplementować w języku Ada modyfikację systemu z programu `travelers.adb` polegającą na tym, że na jednym polu może przebywać w danej chwili tylko jeden podróżnik. Wątek podróżnika, który chce przemieścić się na sąsiednie pole, musi mieć zagwarantowane, że nikogo tam nie ma i nikt jednocześnie z nim na to pole nie wejdzie. Taka synchronizacja nie może w żaden sposób wpływać na współbieżne działanie wątków, których nie dotyczy. Dla każdego pola można zaimplementować osobny wątek lub zmienną typu `protected`, pilnującą aby na danym polu nie pojawił się więcej niż jeden podróżnik. Zastosować `terminate` aby program mógł się zakończyć, gdy wszyscy podróżnicy zakończą działanie.

Jeśli chodzi o podróżników, to mogą oni utknąć w cyklicznym oczekiwaniu ("deadlock"). Należy więc wykorzystać odpowiednio dobrany `Timeout` (uwzględniając parametr `Max_Delay`) w taki sposób, że jeśli podróżnik nabierze podejrzeń, że utknął w deadlocku, to zamienia swój symbol na małą literę, pozostawia swój ostatni ślad w miejscu, gdzie utknął, wysyła swoją historię do wątku `Printer` i kończy działanie.

W sprawozdaniu przygotować (w taki sposób jak w poprzednim zadaniu) pliki historii symulacji przy różnym zatłoczeniu planszy, zmieniając parametr `Nr_Of_Travelers`.

Zadanie 3. (za 10 punktów)

Wykonać wersję Zadania 2 w języku Go.

Zadanie 4. (za 5 punktów)

Przerobić program w Adzie z Zadania 2, tak aby wszystkie parametry:

- `Nr_Of_Travelers`,
- `Board_Width`,
- `Board_Height`,

były równe 15, podróżnik o identyfikatorze i startował na pozycji (i, i) , na przekątnej. Na początku każdy podróżnik o identyfikatorze parzystym wybiera sobie losowy kierunek ruchu w pionie, a podróżnik o identyfikatorze nieparzystym – w poziomie, a następnie już za każdym razem wybiera ten sam kierunek.

Zaobserwować parę symulacji. Zachować w prezentacji dwie lub trzy przykładowe, szczególnie jeśli uda się wyłapać wystąpienie deadlocku.

Zadanie 5. (za 5 punktów)

Wykonać odpowiednik Zadania 5, przerabiając program w Go z Zadania 3.