Lista 4 (24-25-2)

Termin wykonania: 2025-06-15

Zadanie 1. (za 10 punktów)

Podobnie jak w Zadaniu 1 z Listy 3, zaimplementować w Adzie demonstrację algorytmu Szymańskiego, opisanego na stronie:
https://en.wikipedia.org/wiki/Szyma%C5%84ski%27s_algorithm.
Należy oprzeć się na pseudokodzie przedstawionym na stronie i przedstawić jego działanie przy pomocy większej liczby stanów (wierszy) zawierających w nazwie wartość zmiennej flagowej `f[i]`. Zauważmy, że gdy proces jest w protokole wejściowym to ta zmienna może mieć wartości 1, 2, 3, 4. Aby uzyskać więcej wierszy, wystarczy zmienić typ `Process_State` następująco:

```
type Process_State is (
  Local_Section,
  Entry_Protocol_1,
  Entry_Protocol_2,
  Entry_Protocol_3,
  Entry_Protocol_4,
  Critical_Section,
  Exit_Protocol
  );
```

`Exit_Protocol` polega jedynie na czekaniu aż można zmienić `f[i]` z 4 na 0.
Można wykorzystać atomowe operacje na zmiennych `f[i]`.
Zrozumieć jak, przy pomocy pętli, zaimplementować `await(all ... )` oraz `await(any ... )`.

Zadanie 2. (za 10 punktów)

Rozwiązać Zadanie 1 w języku Go.

Zadanie 3. (za 10 punktów)

Poprawić implementację pakietu monitora w Adzie dostępną w repozytorium https://github.com/motib/concurrent-distributed w podkatalogu https://github.com/motib/concurrent-distributed/tree/main/Ada (również na slajdach do wykładu z pewnymi komentarzami). Przy tej implementacji instrukcja typu `Condition.Wait` poprzedzana jest opuszczeniem monitora (`Monitor.Leave`), co powoduje krótki odstęp, w którym inny proces teoretycznie może uruchomić swoją procedurę monitora i wykonać `Signal` na tej samej zmiennej typu `Condition`.

Aby temu zapobiec, można zastąpić `Wait'Count` w tasku `Condition` przez lokalną zmienną (np. `My_Count`) zliczającą "rezerwacje kolejki do `Wait`", zwiększaną w dodatkowo zdefiniowanym entry (np. `Pre_Wait`), które - podobnie jak `Signal` i `Waiting` - musi być akceptowane także w pętli wewnątrz `accept Wait`. (Również w tej pętli `accept Signal` powinno zmniejszać `My_Count`.) Wtedy ustawianie się w kolejce do `C` wymaga sekwencji trzech instrukcji:

```
C.Pre_Wait;
Monitor.Leave;
C.Wait;
```

Można ją opakować w procedurę typu: `procedure Wait(C: in out Condition)`.

Następnie należy zadbać aby task `Monitor` oraz taski typu `Condition` kończyły się po zakończeniu tasków, które z nich korzystają. (W tasku `Monitor` - opakować każdy `accept` w `select` z dodatkową gałęzią `terminate` a w każdym `select` w tasku typu `Condition` - dodać gałąź `terminate`.)
Wykorzystując `mutex_template.adb` z Listy 3, użyj swojego pakietu monitora do zaimplementowania demonstracji działania algorytmu czytelników i pisarzy dla 10 czytelników i 5 pisarzy. Każdy czytelnik ma mieć symbol `R` a każdy pisarz - symbol `W`. Każdy pisarz i czytelnik ma przemieszczać się w osobnej kolumnie między wierszami o etykietach dla stanów:

```
type Process_State is (
  Local_Section,
  Start,
  Reading_Room,
  Stop
  );
```

gdzie `Start` i `Stop` to odpowiednio wchodzenie i wychodzenie z czytelni.

(Na potrzeby `gnatmake`, można zmienić duże litery na małe w nazwach plików poleceniem:
`for X in $(ls *.ad?); do mv ${X} ${X,,[[:alpha:]]}; done`
)

Zadanie 4. (za 10 punktów)

Zaimplementuj własny monitor i wykorzystaj do implementacji algorytmu czytelników i pisarzy w języku Go. (Zauważ, że w "Stateful Goroutines", zamiast natychmiast odesłać odpowiedź przez kanał zawarty w prośbie `Wait`, możesz ten kanał zapamiętać w lokalnej kolejce i przy obsłudze prośby `Signal` wysyłać odpowiedź do pierwszego w tej kolejce.)