Zadania przygotowujące do egzaminu z Programowania Współbieżnego

Zadanie 1 (komunikacja synchroniczna, egz. 2021)

Procesy `M[w : 0 .. W - 1][k : 0 .. K - 1]` ($W > 0, K > 0$), wraz z procesem pomocniczym `Z`, implementują rozproszoną macierz wartości typu `porcja`. Procesy `U[n : 0 .. N-1]` ($N > 0$) wykonują operacje na tej macierzy.

Proces `M[w][k]` przechowuje porcję z wiersza `w` i kolumny `k`. Wartość początkową dostaje jako wynik funkcji:

```
porcja początkowa(int w, int k);
```

Proces `U[n]` odczytuje porcję z wiersza `w` i kolumny `k`. wykonując instrukcje:

```
send M[w][k].daj(n);
receive odczytana;
```

Proces `U[n]` zleca przesunięcie cykliczne zawartości wiersza `w` o `p` pozycji, dla $0 < p < K$, wykonując instrukcję:

```
send Z.przesuńWiersz(n, w, p);
```

Efektem jest przeniesienie porcji w wierszu `w`, dla każdego `k` od `0` do `K - 1` z kolumny `k` do kolumny `(k + p) % K`.

Proces `U[n]` zleca przesunięcie cykliczne zawartości kolumny `k` o `p` pozycji, dla $0 < p < W$, wykonując instrukcję:

```
send Z.przesuńKolumnę(n, k, p);
```

Efektem jest przeniesienie porcji w kolumnie `k`, dla każdego `w` od `0` do `W - 1` z wiersza `w` do wiersza `(w + p) % W`.

Polecenie:

Zdefiniuj w notacji Rendezvous procesy `Z` i `M[w][k]`.

Założenia:

Maksymalna liczba porcji przechowywanych na raz przez proces jest stała, niezależna od sumy `N + W + K`.

Górne ograniczenie na rozmiar pomocniczych struktur danych procesu jest funkcją liniową sumy `N + W + K`.

Odczytywane porcje powinny być zgodne ze scenariuszem, w którym operacje na macierzy są wykonywane sekwencyjnie i niepodzielnie.

Efektywność rozwiązania będzie miała istotny wpływ na ocenę. Tam, gdzie to możliwe, operacje na macierzy powinny być wykonywane współbieżnie. Koszt przesunięcia elementów powinien być mniejszy niż liniowy względem ich liczby.

Zadanie 2 (komunikacja synchroniczna, kol. 2020)

Na ogrodzeniu dużego terenu jest rozmieszczonych `N` czujników ponumerowanych od `1` do `N`, gdzie `N` jest parzyste. Każdy z czujników może wywołać predefiniowaną funkcję `sprawdź()`, która przekaże w wyniku `0`, jeśli wykryto uszkodzenie fragmentu ogrodzenia, za które odpowiada dany czujnik, wpp przekaże `1`.

Proces koordynator co jakiś czas wywołuje funkcję `kontrola(int d)`, która: informuje czujniki o tym, że należy wykonać sprawdzenie, czeka na informacje o uszkodzeniach i wypisuje na standardowe wyjście w dowolnej kolejności numery czujników w zależności od parametru `d` $\in$ `{1,2,3}`:

- dla `d=1` numery wszystkich czujników, które wykryły uszkodzenie,
- dla `d=2` numery tych czujników, które wykryły uszkodzenie i których prawy sąsiad wykrył uszkodzenie,
- dla `d=3` numery tych czujników, które wykryły uszkodzenie i których obydwaj sąsiedzi wykryli uszkodzenie.

Przykład: `N=10000` oraz czujniki `1`, `2`, `5` i `10000` wykryły uszkodzenie. Dla `d=1` wynikiem jest zbiór `{1,2,5,10000}`, dla `d=2`: `{1,10000}`, a dla `d=3`: `{1}`.

Procesy komunikują się *synchronicznie* korzystając z operacji `send`, `receive` oraz `select`. Każdy czujnik może komunikować się z każdym innym i z koordynatorem. Koordynator nie musi wysyłać ani odbierać wiadomości bezpośrednio do/od każdego czujnika.

Napisz treść procesu `Czujnik(int id)` oraz funkcję `kontrola(int d)` koordynatora.

Zadbaj o poprawność i wydajność rozwiązania przyjmując, że liczba uszkodzeń jest znacznie mniejsza niż `lg(N)`. Za rozwiązanie naiwne będzie można uzyskać co najwyżej połowę punktów.

Zadanie 3 (komunikacja synchroniczna, egz. 2019)

Przetwarzanie obrazów to także obszar zastosowania programowania współbieżnego. Na przykład, żeby wygładzić obraz, piksel na pozycji `x[i][j]` zastępuje się średnią ważoną obliczaną w następujący sposób:

```
x'[i][j] = (4*x[i][j] + x[i+1][j] + x[i][j+1] + x[i-1][j] + x[i][j-1])/8
```

Algorytm wygładzania jest iteracyjny. W każdej iteracji następuje sprawdzenie, czy wartość bezwzględna różnicy pomiędzy wartością `x[i][j]` a `x'[i][j]` przekracza zadane `epsilon`. Jeśli istnieje piksel, dla którego `epsilon` jest przekroczony, to wykonywana jest kolejna iteracja algorytmu dla całego obrazu.

Napisz funkcję `double smooth(int i, int j, double x, double epsilon)`, która dla piksela o współrzędnych `0` $\leqslant$ `i,j` $\leqslant$ `N-1` o wartości początkowej `x` i danego `epsilon` policzy oraz przekaże jako wynik wartość końcową. Za wartości nieistniejących sąsiadów należy przyjąć `0` (np. `x[-1][0] = 0`). Funkcja jest wykonywana dla każdego piksela osobno przez procesory karty graficznej `p[i][j]`, gdzie `0` $\leqslant$ `i,j` $\leqslant$ `N-1`. Procesory komunikują się synchronicznie korzystając z operacji `send`, `receive` oraz `select`.

Zadbaj o poprawność i wydajność rozwiązania.

Zadanie 4 (przestrzeń krotek, egz. 2021)

Równanie różniczkowe Laplace'a opisuje różne zjawiska w procesach fizycznych. Analiza takich procesów często sprowadza się do zdyskretyzowania równania i numerycznego rozwiązania powstałego układu równań. Jedną z używanych metod numerycznych jest metoda kolejnych nadrelaksacji. Przyjmijmy, że równanie jest zdyskretyzowane na siatce `K x K` punktów.

Algorytm numeryczny przybliżający rozwiązanie równania działa następująco dla każdego punktu siatki:

- liczy średnią wartość istniejących sąsiednich punktów siatki (górnego, dolnego, prawego i lewego sąsiada) - np. jeżeli punkt ma 4 sąsiadów to:
  `tmp = (x[i+1][j] + x[i][j+1] + x[i-1][j] + x[i][j-1])/4`,
  a dla punktu `(0,0)`: `tmp = (x[1][0] + x[0][1])/2`
- nadaje nową wartość punktowi równą średniej ze średniej policzonej w poprzednim punkcie oraz starej wartości punktu używając zadanego współczynnika relaksacji `A` jako wagi:
  `x'[i][j] = A*x[i][j] + (1-A)*tmp`

Algorytm jest iteracyjny. W każdej iteracji następuje sprawdzenie, czy wartość bezwzględna różnicy pomiędzy wartością `x[i][j]` a `x'[i][j]` przekracza zadane `epsilon`. Jeśli istnieje punkt, dla którego `epsilon` jest przekroczony, to wykonywana jest kolejna iteracja algorytmu dla wszystkich punktów.

Rozważamy współbieżny program wyznaczający na podstawie danych początkowych przybliżenie rozwiązania równania. Program korzysta z przestrzeni krotek. Składa się z procesu:

```
process Główny() {
    zapiszDane();
    tsPut("START");
    tsFetch("STOP");
    odczytajWynik();
}
```

oraz `N`>1 procesów `Pomocniczy(int id)`, gdzie `id` to unikatowy numer procesu, od `1` do `N`.

Proces `Główny` posługuje się reprezentacją wartości w punkcie za pomocą krotki w formacie `"%d %d %f"`, z numerem wiersza i numerem kolumny tego punktu oraz jego wartości.

Funkcja `zapiszDane()` zapisuje w przestrzeni krotki, które reprezentują dane początkowe. W chwili wykonania funkcji `odczytajWynik()`, w przestrzeni powinny znajdować się tylko krotki, które reprezentują wynik i ewentualnie stała liczba krotek, niezależna od `N`, `K` ani od liczby iteracji algorytmu.

Proces `Pomocniczy` zna wartości `N`, `K`, `A` oraz `epsilon`. Dozwolone jest korzystanie z pomocniczych krotek, ale nie wolno tworzyć dodatkowych procesów. Wielkość krotki oraz rozmiar danych procesu `Pomocniczy` powinnien być stały.

Należy zdefioniować proces `Pomocniczy` dbając o poprawność i efektywność rozwiązania.

Zadanie 5 (przestrzeń krotek, kol. 2020)

Gra w Życie (ang. *Life*) to automat komórkowy, czyli symulacja życia komórek, w dwuwymiarowym świecie.

Komórki są uporządkowane w wiersze i kolumny, numerowane liczbami całkowitymi. Przyjmujemy, że komórka w wierszu `w` i kolumnie `k` sąsiaduje z ośmioma innymi komórkami, które mają numer wiersza od `w - 1` do `w + 1` a numer kolumny od `k - 1` do `k + 1`.

Komórka może być w jednym z dwóch stanów: żywa lub martwa. Łączny stan wszystkich komórek nazywamy generacją.

Symulacja zaczyna się od generacji zerowej. Na podstawie niej liczymy następne. W następnej generacji komórka będzie żywa wtedy i tylko wtedy, gdy:

- w bieżącej generacji jest żywa i ma dwóch lub trzech żywych sąsiadów, albo
- w bieżącej generacji jest martwa i ma trzech żywych sąsiadów.

Rozważamy współbieżny program wyznaczający, na podstawie generacji zerowej, generację numer `G` $\geqslant$ `0`.

Program korzysta z przestrzeni krotek. Składa się z procesu:

```
process Główny() {
    zapiszDane();
    tsPut("START");
    tsFetch("STOP");
    odczytajWynik();
}
```

oraz `N` $>$ `1` procesów `process Pomocniczy(int id) { ... }`, gdzie `id` to unikatowy numer procesu, od `0` do `N - 1`.

Proces `Główny` posługuje się reprezentacją komórki żywej za pomocą krotki w formacie `"%d %d"`, z numerem wiersza i numerem kolumny tej komórki.

Funkcja `zapiszDane()` zapisuje w przestrzeni krotki, które reprezentują komórki żywe w generacji zerowej.

W chwili wykonania funkcji `odczytajWynik()`, w przestrzeni powinny być tylko krotki, które reprezentują komórki żywe w generacji numer `G`.

Polecenie: zdefiniuj proces `Pomocniczy`.

Założenia:

- Proces `Pomocniczy` zna wartości `N` i `G`.
- Współrzędne żywych komórek, liczba żywych komórek w jednej generacji oraz wartości `N` i `G` mieszczą się w zakresie typu `int`.
- Wolno korzystać z pomocniczych krotek i definiować pomocnicze funkcje. Nie wolno definiować dodatkowych procesów.
- Niech `R` będzie sumą liczby żywych komórek w aktualnej generacji oraz wartości `N` i `G`. Liczba krotek w przestrzeni powinna być ograniczona przez funkcję zależącą liniowo od `R`. Wielkość krotki oraz rozmiar danych procesu `Pomocniczy` powinnien być stały, niezależny od `R`.

Zadanie 6 (przestrzeń krotek, egz. 2019)

W systemie działają procesy zlecające prace do wykonania oraz procesy obsługujące zlecenia. Każde zlecenie składa się z jednoznacznego identyfikatora (typu `integer`) oraz (dużych) danych (typu `Data_t`). Każde zlecenie musi być wykonane przez `K` procesów, gdzie `K` jest nieparzyste i mniejsze od liczby procesów obsługujących. Procesy niezależnie wykonują operację `long calculate(Data_t data)`. Ostatecznym wynikiem przetworzenia zlecenia jest mediana (wartość środkowa) z uzyskanych `K` wartości.

Procesy zlecające wykonują: `tsPut("Order %d %D", orderId, data)`, a następnie odbierają wynik poprzez `tsFetch("Order %d ?li", orderId, &median)`.

Procesy obsługujące przy wyborze następnego zlecenia powinny preferować zlecenia, których realizacja już się rozpoczęła. Należy przyjąć, że dane są naprawdę duże, a w związku z tym liczba transmisji danych (typu `Data_t`) z i do przestrzeni krotek powinna być jak najmniejsza. Można korzystać z przesyłania tablic np. `tsPut("%d[%li]", tabsize, tab)`.

Napisz treść procesu realizującego zlecenia. Po zakończeniu pracy w przestrzeni krotek nie powinny pozostawać żadne krotki (przy założeniu, że wyniki będą odbierane przez procesy zlecające).