Skład zespołu:
Krzysztof Kwaśniewski (lider)
Mateusz Jankowski
Miłosz Truszkowski
Łukasz Reszka

Data przekazania:
26.04.2022

Temat zadania:
Napisać wieloprocesowy system realizujący komunikację w języku komunikacyjnym Linda.

Treść zadania:
W16 – zrealizować system nie jako wieloprocesowy lecz jako wielowątkowy, do synchronizacji należy wykorzystać obiekty mutex i cond, synchronizacja powinna być możliwie „drobnoziarnista” (tj. nie jedna prosta sekcja krytyczna na każdej operacji komunikacyjnej).

Interpretacja treści zadania:
Powyżej określone zadanie rozumiemy jako potrzebę utworzenia biblioteki, dzięki której będziemy mogli realizować komunikację między wątkami w standardzie Linda. Oprócz samej biblioteki, konieczne będzie utworzenie oprogramowania do przeprowadzania testów., Aplikacja testowa będzie zrealizowana jako responsywny terminal, za pomocą którego będzie można umieszczać i pobierać krotki z przestrzeni. Polecenia będzie można wydawać przy użyciu konkretnych komend.
Komendy:

- output(*wzorzec*) *identyfikator wątku*
- input(*wzorzec*, timeout) *identyfikator wątku*
- read(*wzorzec*, timeout) *identyfikator wątku*
- create_thread *identyfikator wątku*

Konieczne będzie określanie, który wątek umieszcza, pobiera lub czyta dane, gdyż terminal ma symulować działanie realnego programu korzystającego z naszej biblioteki.

```
wzorzec         = {typ_danych, ":", wartość, ";"};
typ_danych      = "int" | "string" | "float";
wartość         = int | float | string;
int             = cyfra, {cyfra};
float           = int,".", int;
string          = "'", {znak}, "'";
```

timeout - maksymalny czas oczekiwania w milisekundach, typ uint;

Opis funkcjonalny:

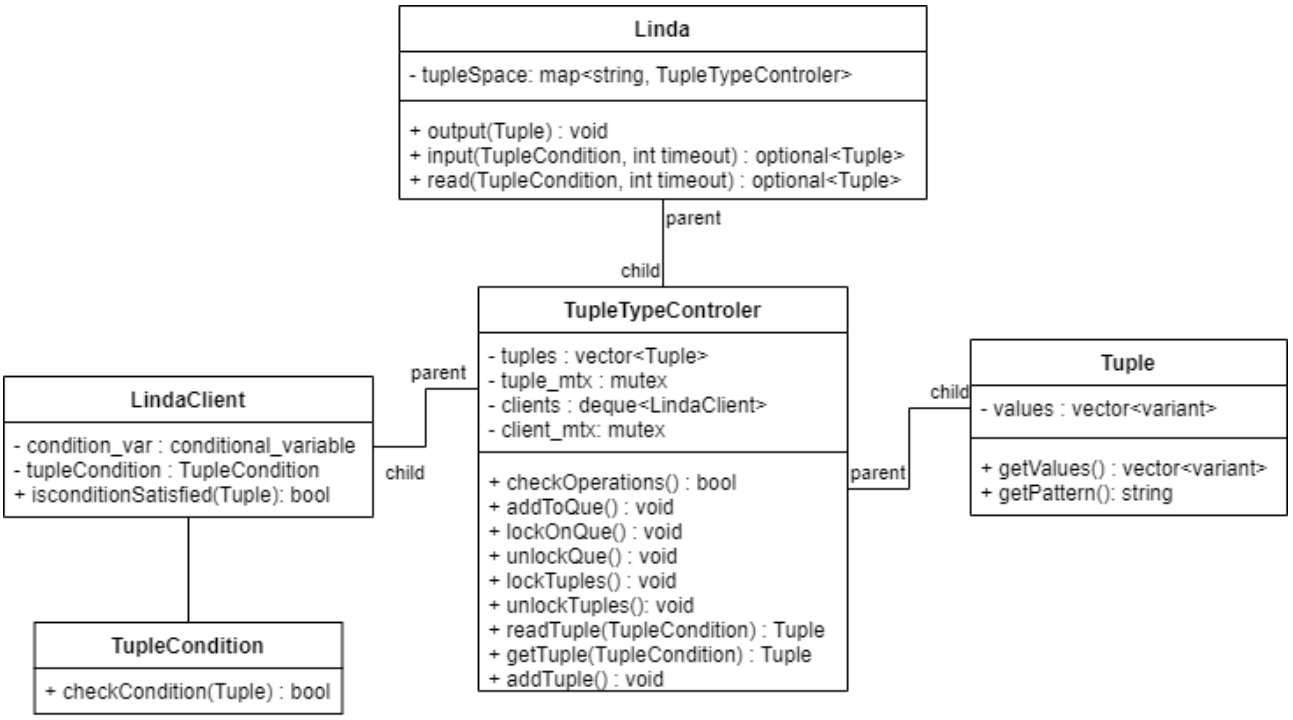
- Możliwość utworzenia obiektu wspólnej przestrzeni krotek zgodnej ze standardem języka Linda w programie za pomocą konstruktora obiektu Linda. Na tym obiekcie możemy wołać funkcję :
    - **output(*krotka*)** - funkcja output umieszcza krotkę w przestrzeni.

- **input(*wzorzec*, timeout)** - funkcja input pobiera i atomowo usuwa krotkę z przestrzeni, przy czym wybór krotki następuje poprzez dopasowanie wzorca-krotki. Wzorzec jest krotką, w której dowolne składniki mogą być niewyspecyfikowane: „*” (podany jest tylko typ) lub zadane warunkiem logicznym. Możliwe warunki: ==, <, <=, > , >=.
- **read(*wzorzec*, timeout)** - operacja read działa tak samo jak input, lecz nie usuwa krotki z przestrzeni.

Założenia niefunkcjonalne:

- Maksymalny rozmiar krotki (ilość typów danych we wzorcu) będzie podawany przy tworzeniu obiektu Linda.
- Maksymalna ilość krotek w przestrzeni będzie podawana przy tworzeniu obiektu Linda.
- Dla typu danej float nie istnieje warunek ==.
- Dla danych string warunki: ==, <, <=, > , >= będą leksykograficznym porównaniem stringów.
- Żądania będą szeregowane, tak by nie dochodziło do zagłodzenia wątku.

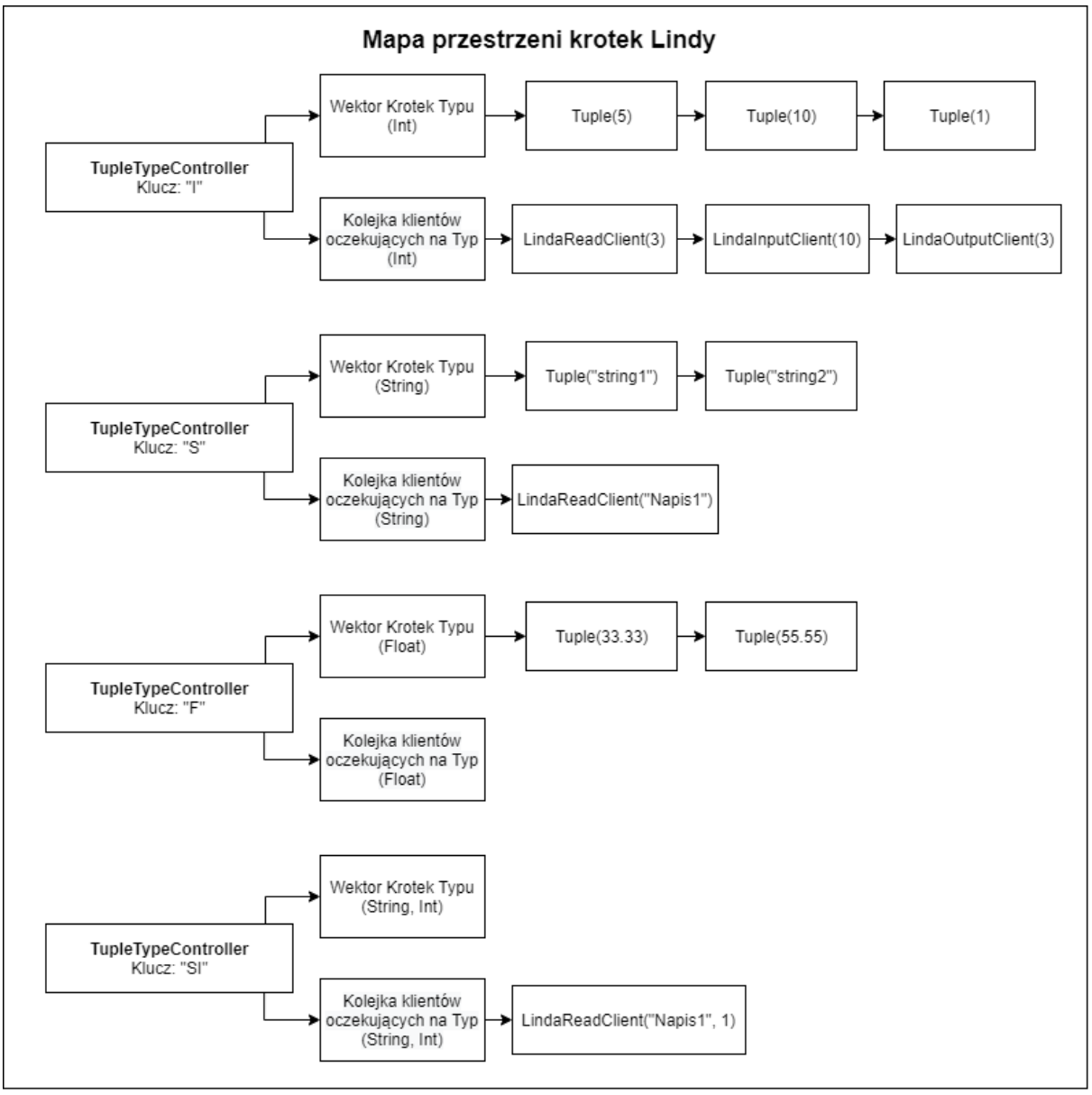
Wstępny szkic architektury i struktur danych:



Nadrzędną strukturą będzie mapa (tupleSpace), w której kluczem będzie “typ” (string) krotki (przykładowo dla krotki z typami String Int Float, wartość klucza będzie wynosić “SIF” - pierwsze litery typów danych), zaś wartością będzie obiekt kontrolujący pojedynczy typ krotki (TupleTypeController). Obiekt ten będzie w sobie przchowywał wektor krotek danego typu oraz kolejkę klientów (LindaClient) oczekujących na operację na danym typie krotki. Klient zawiera warunek, który chce sprawdzić, a dodatkowo zmienną warunkową, która umożliwi

odblokowanie danego klienta, w momencie gdy będzie on mógł wykonać określoną przez siebie operację. Typ operacji zostanie zaimplementowany przy użyciu polimorfizmu.

Poniżej znajduje się przykładowy schemat jak może wyglądać przestrzeń krotek:



Typ krotki jest umieszczany w mapie w momencie, gdy przychodzi pierwsza operacja dla danego typu krotki. Jeżeli jest to output, to wstawi on element do wektora krotek. Inna operacja będzie go mogła potem przeczytać lub pobrać. Jeżeli jako pierwsza przyjdzie operacja input lub read, to zawiesi się ona w oczekiwaniu na krotkę spełniającą warunki. Typ krotki jest usuwany z mapy, jeżeli nie pozostanie żadna operacja do wykonania w kolejce i w wektorze krotek nie ma ani jednego elementu.

Metody komunikacji:
Komunikacja między wątkami będzie odbywać się poprzez obiekt przestrzeni krotek, który będzie przekazywany przez referencję do funkcji, którą będzie wykonywał wątek. Sam

obiekt przestrzeni krotek (Lindy) będzie odpowiedzialny za synchronizację i odpowiednią obsługę wywoływanych operacji w wątkach.

Metody synchronizacji i realizacja współbieżności:

W ramach synchronizacji wyróżniamy 3 byty używające lindę:

- producentów - funkcja output
- czytelników - funkcja read
- konsumentów - funkcja input (konsumenci i czytelnicy określani są mianem Klientów)

W ramach testów programu korzystać będziemy z wątków POSIX.

Synchronizację oraz sekcję krytyczną będziemy realizować poprzez:

- zmienne warunkowe
- mutexy

  które są ściśle powiązane z wątkami POSIX.

Synchronizacja będzie odbywać się na poziomie obiektu typu “TupleTypeControler”. Kontrolowany będzie dostęp do jego wektora krotek. Wektor krotek będzie sekcją krytyczną, obsługiwaną przy użyciu mutexa. Poprzez sekcję krytyczną będzie również kontrolowany dostęp do kolejki obiektów typu “LindaClient”, które to niejako reprezentują zawieszony na warunku wątek kliencki lub wątek producenta czekający na dostęp do sekcji krytycznej.

Wątek który chcę przeczytać lub skonsumować krotkę z Lindy, przy wejściu będzie sprawdzał czy w kolejce oczekujących są jakieś operację, które mogą się wykonać(jeśli tak to odblokuję pierwszą) i czy nie są spełnione warunki jego wykonania. Jeśli suma logiczna tych dwóch sprawdzeń będzie prawdziwa to dany wątek dodaje się do kolejki oczekujących wątków i zawiesza na zmiennej warunkowej połączonej z mutexem kolejki. Jeśli zaś kolejka będzie pusta i jego warunek będzie spełniony lub jeśli po wcześniejszym zawieszeniu zostanie obudzony przez inny wątek, zajmuję sekcję krytyczną dla wektora krotek i konsumuję/czyta krotkę. Następnie sprawdza czy może obudzić inną krotkę, zwalnia blokady i wychodzi z krotką. Ważne jest również to że po obudzeniu sprawdza czy nie został obudzony po przez timeout. Jeśli tak to wychodzi z funkcji bez przeczytania krotki.

Wątek producenta zachowuję się podobnie z tym że nie sprawdza czy spełnione są warunki jego wykonania ponieważ zawsze one będą spełnione z racji tego że jest producentem. Powodem dla którego blokuję się on przy sprawdzaniu czy inny wątek nie może się wykonać jest to że nie chcemy zagłodzić konsumentów poprzez ciągłe dodawanie się wątków producentów i nie odblokowywanie konsumentów/czytelników najszybciej jak tylko się da. Wątek ten nie posiada timeoutu. Jeśli zawiesi się w kolejce oczekujących to jest pewne że w momencie kiedy wątki które przed nim przyszły nie mają spełnionych warunków zostanie wykonany. Mutex na sekcję krytyczną obecnie nie spełnia większej roli, ale w momencie kiedy chcielibyśmy modyfikować projekt będzie kluczowy dlatego jest obecny w naszym projekcie.

Zarys implementacji:
Biblioteka zostanie napisana przy użyciu języka C++. Tak jak wyżej było wspomniane będziemy korzystać z POSIXowych zmiennych warunkowych, mutexów i wątków. Poniżej prezentujemy pseudokod który opisuje planową realizację synchronizacji wątków na poziomie pojedynczego typu krotki.

Linda.output():
semop(down, kolejka)
semop(down, sk)
sprawdź czy mogą wykonać operację występującą w kolejce (odblokowujemy pierwszą możliwą operację w kolejce, której warunek jest spełniony)
semop(up, sk)
jeżeli tak to:
  dodaj się do kolejki wszystkich operacji
  zawieś się na zmiennej warunkowej -> z wykorzystaniem semafora kolejki
  usuń się z kolejki

semop(down, sk)
wstaw do sekcji krytycznej krotkę spełniającą nasz warunek
sprawdź czy mogą wykonać się inne operacje (odblokowujemy pierwszą możliwą operację w kolejce, której warunek jest spełniony)
semop(up, kolejka)
semop(up, sk)

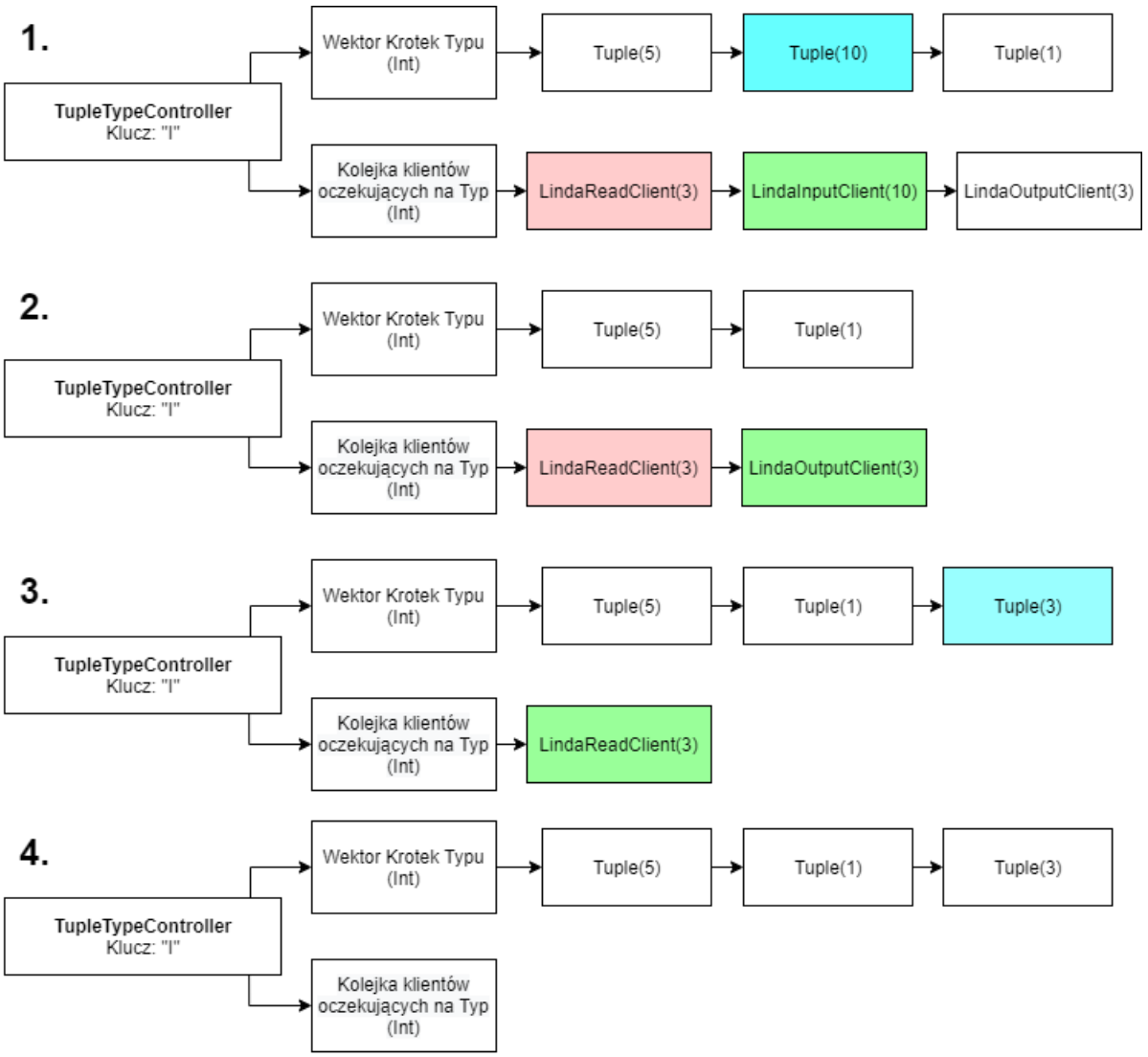
Linda.input():
semop(down, kolejka)
semop(down, sk)
sprawdź czy mogą wykonać operację występującą w kolejce (odblokowujemy pierwszą możliwą operację w kolejce, której warunek jest spełniony)
semop(up, sk)
jeżeli tak lub jeśli jego warunek nie jest spełniony to:
  dodaj się do kolejki wszystkich operacji
  zawieś się na zmiennej warunkowej -> z wykorzystaniem semafora kolejki + TIMEOUT
  (sam zakłada się semafor na kolejkę znowu)
  usuń się z kolejki
  if(timeout)
    semop(up, kolejka)
    return NULL

semop(down, sk)
pobierz i usuń z sekcji krytycznej krotkę spełniającą nasz warunek
sprawdź czy mogą wykonać się inne operacje (odblokowujemy pierwszą możliwą operację w kolejce, której warunek jest spełniony)
semop(up, kolejka)
semop(up, sk)
return krotka

Linda.read():
semop(down, kolejka)
semop(down, sk)
sprawdź czy mogą wykonać operację występującą w kolejce (odblokowujemy pierwszą możliwą operację w kolejce, której warunek jest spełniony)
semop(up, sk)
jeżeli tak lub jeśli jego warunek nie jest spełniony to:
  dodaj się do kolejki wszystkich operacji
  zawieś się na zmiennej warunkowej -> z wykorzystaniem semafora kolejki + TIMEOUT
  (sam zakłada się semafor na kolejkę znowu)
  usuń się z kolejki
  if(timeout)
    semop(up, kolejka)
    return NULL

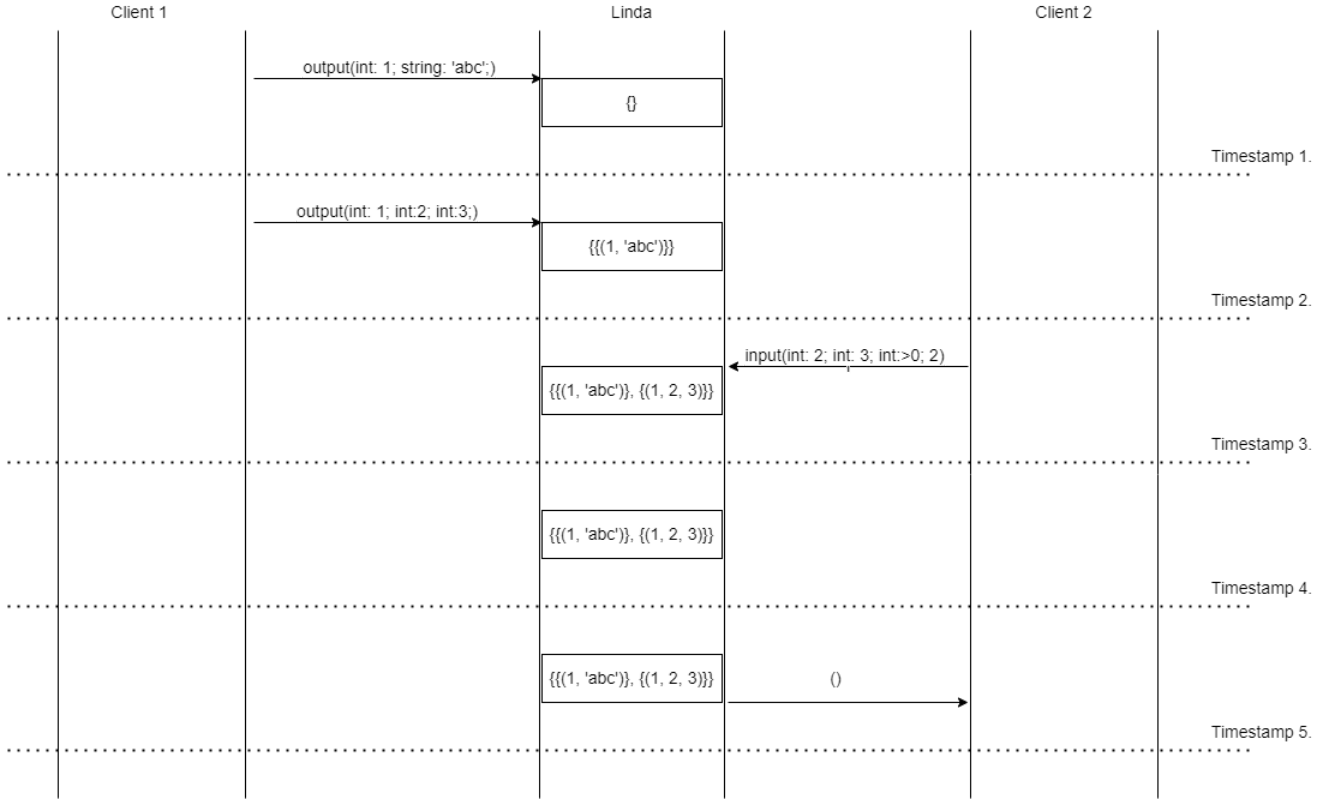
semop(down, sk)
przeczytaj z sekcji krytycznej krotkę spełniającą nasz warunek
sprawdź czy mogą wykonać operację występującą w kolejce (odblokowujemy pierwszą możliwą operację w kolejce, której warunek jest spełniony)
semop(up, kolejka)
semop(up, sk)
return krotka

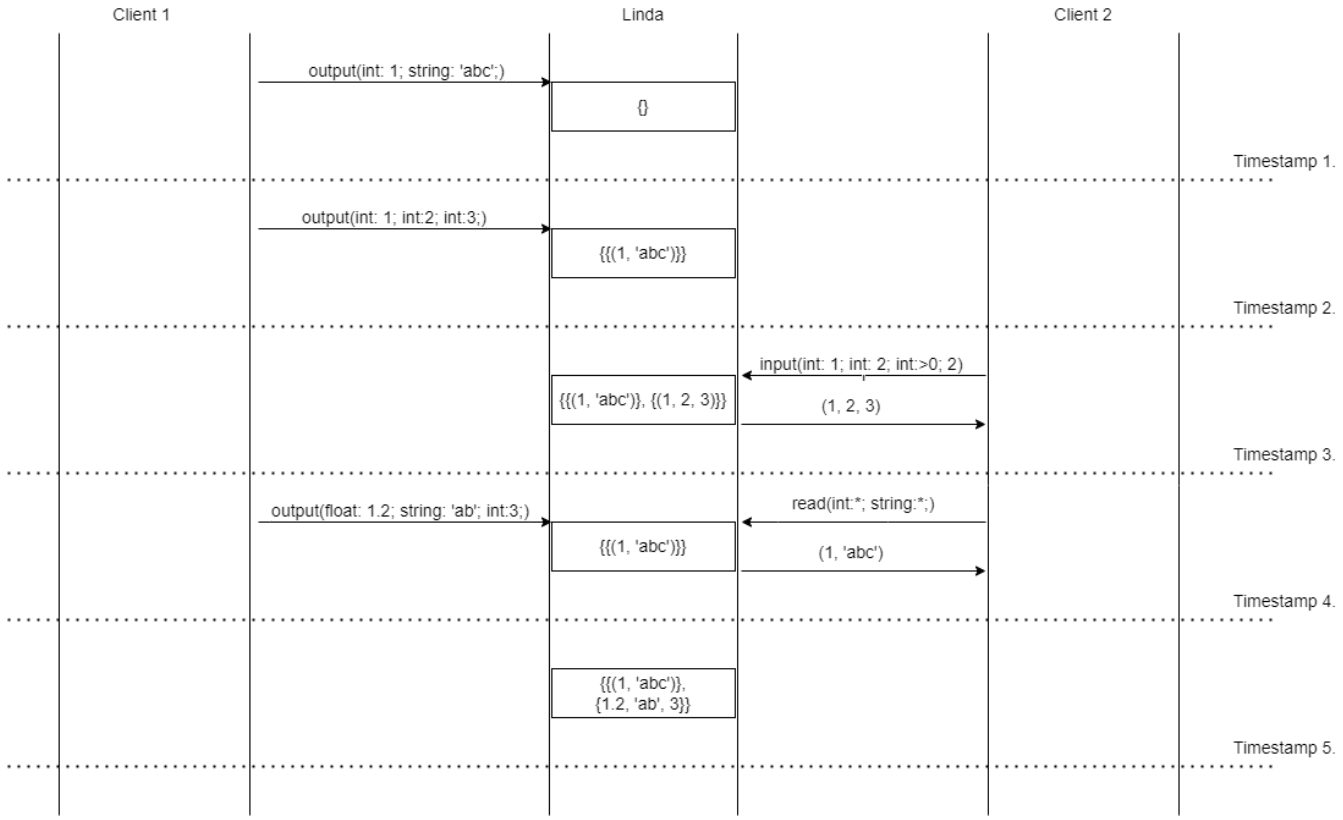
Szeregowanie kolejności wykonywania zapytań:
Tak jak zostało to opisane powyżej, wykonywać będzie się zawsze pierwsza w kolejce operacja, której warunek jest spełniony. Najłatwiej będzie można to przedstawić na przykładzie, na którym pokazane są kolejne stany TupleTypeControllera:



1. W kolejce oczekujących mamy 3 operacje. Pierwszą w kolejce jest LindaReadClient(3), jednak nie może on wykonać swojej operacji (został zaznaczony na czerwono), ponieważ w wektorze nie ma krotki spełniającej jego warunek. Drugi w kolejce jest LindaInputClient(10). Może on wykonać operację (zaznaczenie na zielono), gdyż w kolejce jest Tuple(10). Usuwa on z wektora zaznaczoną na turkusowo krotkę i usuwa siebie z kolejki oczekujących klientów.
2. LindaReadClient(3) nadal nie może się wykonać, ponieważ nadal nie ma w wektorze krotki spełniającej jego warunek. Kolejny jest LindaOutputClient(3), którego zadaniem jest umieszczenie krotki, dlatego nie istnieje dla niego żaden warunek sprawdzający, więc może się on wykonać. Umieszcza on krotkę w wektorze i usuwa się z kolejki oczekujących.
3. LindaReadClient(3) w końcu może przeczytać oczekiwaną krotkę, umieszczoną przez opisany wyżej Output. LindaReadClient czyta zaznaczoną na turkusowo krotkę, a następnie usuwa się z kolejki. Nie usuwa krotki, gdyż operacja read pozostawia krotkę w wektorze.
4. Wszystkie operacje zostały wykonane, dlatego kolejka jest pusta. W wektorze krotek typu Int nadal znajdują się 3 krotki.

Przykładowe schematy przedstawiające sekwencję wykonywanych operacji:





Sposób testowania:
Proces testowania będzie przebiegać z wykorzystaniem zaimplementowanego interpretera poleceń. Wpisując w terminal polecenia read(), input() i output() będzie można określić zachowanie wątków biorących udział w teście. Po wykonaniu odpowiedniego polecenia rozpoczynającego test, wątki odpowiadające poszczególnym funkcjom języka komunikacyjnego Linda będą zwracać odpowiednie informacje składające się na wynik testu. Będzie on porównywany ze spodziewanym wynikiem dla danego przypadku testowego. Testy te będą wykonywane manualnie. Oprócz testów manualnych prawdopodobnie zostaną utworzone proste jednowątkowe testy jednostkowe (żeby działały deterministycznie), które będą testować prawidłową kolejność wykonywania operacji - np. read(), output() lub input(), output().