UXP1A – Dokumentacja Wstępna

Oskar Bartosz, Paweł Müller, **Jakub Robaczewski (lider zespołu)**, Łukasz Topolski

Temat

Napisać wieloprocesowy system realizujący komunikację w języku komunikacyjnym Linda. W uproszczeniu Linda realizuje trzy operacje:

- `output(krotka)`
- `input(wzorzec-krotki, timeout)`
- `read(wzorzec-krotki, timeout)`

Komunikacja między-procesowa w Lindzie realizowana jest poprzez wspólną dla wszystkich procesów przestrzeń krotek. Krotki są arbitralnymi tablicami dowolnej długości składającymi się z elementów 3 typów podstawowych: string, integer, float. Przykłady krotek: `(1, "abc", 3.1415, "d"), (10, "abc", 3.1415)` lub `(2, 3, 1, „Ala ma kota")`. Funkcja output umieszcza krotkę w przestrzeni. 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. Przyjąć warunki: ==, <, <=, >, >=. Przykład: `input (integer:1, string:*, float:*, string:"d")` – pobierze pierwszą krotkę z przykładu wyżej zaś: input `(integer:>0, string:"abc", float:*)` drugą. Operacja read działa tak samo jak input, lecz nie usuwa krotki z przestrzeni. Operacje read i input zawsze zwracają jedną krotkę (choć pasować może więcej niż jedna). W przypadku gdy wyspecyfikowana krotka nie istnieje operacje `read` i `input` zawieszają się do czasu pojawienia się oczekiwanej danej.

- W15 - plików z mechanizmami zajmowania rekordów (np. jeden plik z rekordami zajmowanymi przez określone procesy)

Interpretacja treści zadania

Tworzymy bibliotekę C++ umożliwiającą działanie na krotkach w przestrzeni, która jest systemem plików. Biblioteka realizuje 3 funkcje użytkownika:

- `output()` - umieszczający krotkę w przestrzeni
- `input()` - czytający krotkę i usuwający z przestrzeni
- `read()` - czytający krotkę

Komendy mogą być wywoływane z różnych programów dzielących tę samą przestrzeń. Musimy zapewnić bezpieczeństwo odczytu i zapisu danych dla różnych procesów wraz z odpowiednią ochroną przed wyścigami.

Dodatkowe założenia:

- Maksymalna długość krotki (liczba elementów) wynosi 7
- Maksymalna długość elementu typu `string` wynosi 200
- Przestrzeń składa się z dwóch plików: jeden przechowujący krotki i drugi przechowujący wzorce

Krótki opis funkcjonalny

Funkcja `input()`**:**

- Proces otwiera i blokuje zapis i odczyt całego pliku z krotkami
- Następnie sprawdza czy jest krotka pasująca do jego wzorca
- Przypadek 1.: istnieje krotka pasująca do wzorca:
    - Proces usuwa tę krotkę z pliku, defragmentuje go, zdejmuje blokadę, zamyka plik i zwraca odczytaną krotkę
- Przypadek 2.: nie ma poszukiwanej krotki:
    - Proces tworzy własny semafor IPC

- Następnie zapisuje wzorzec i identyfikator semafora do pliku z wzorcami (na czas zapisu blokuje odczyt i zapis końca pliku)
- Potem, z wykorzystaniem wcześniej stworzonego semafora, zawiesza się na co najwyżej czas zdeterminowany przez timeout i czeka na pojawianie się krotki pasującej (operacja ta zrealizowana będzie poprzez wykorzystanie funkcji **`semtimedop()`**)
- Gdy zostanie dodana pasująca krotka, proces usuwa semafor, otwiera plik z krotkami, odczytuje, usuwa i zwraca pasującą krotkę
- Gdy minie czas określony przez timeout, proces zwraca błąd

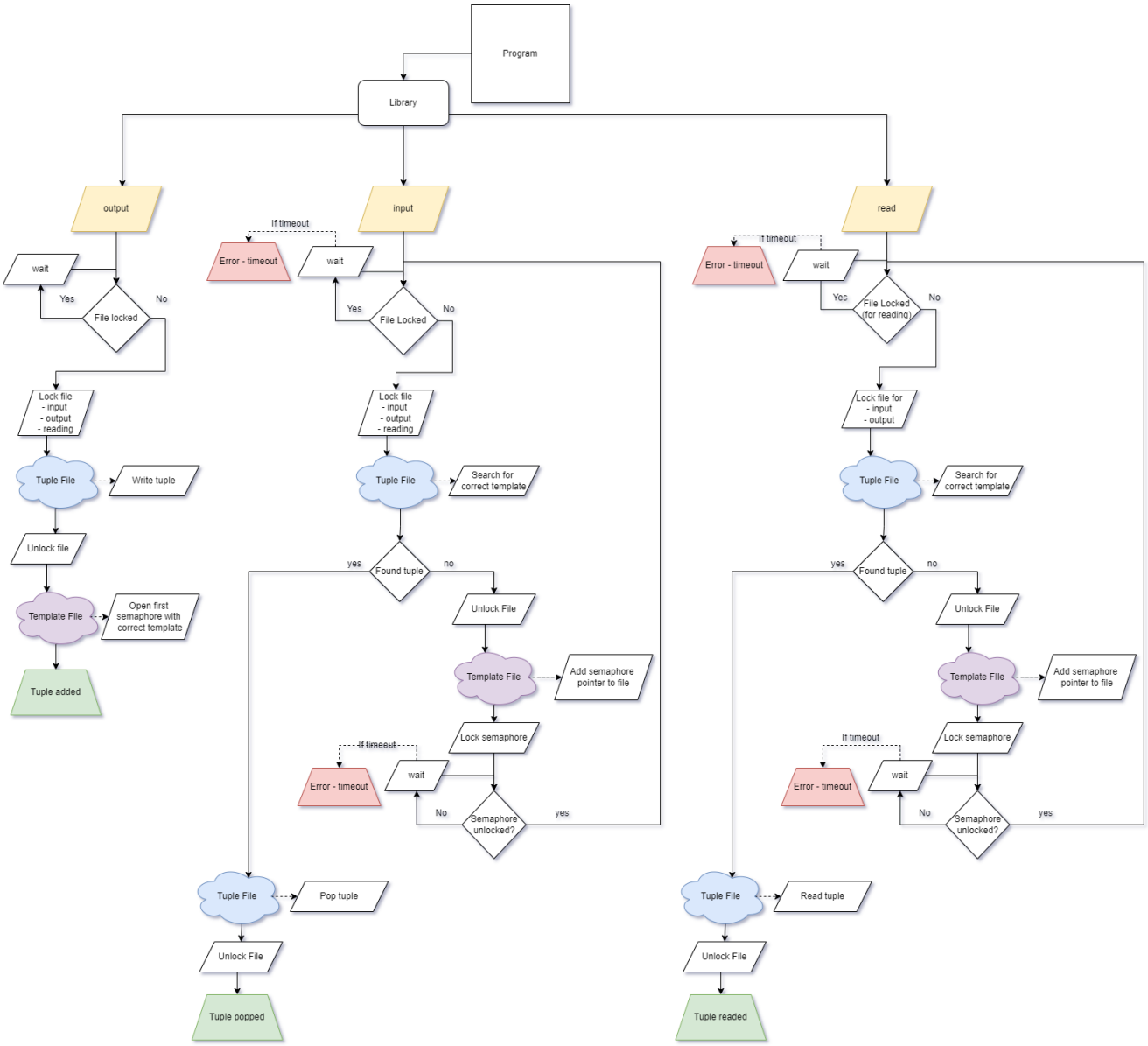
Funkcja `read()`:

- Różni się od poprzedniej tym, że nie usuwa krotki z przestrzeni oraz że blokuje tylko zapis całego pliku

Funkcja `output()`:

- Proces otwiera plik z krotkami oraz blokuje zapis i odczyt na końcu pliku
- Następnie zapisuje na końcu pliku krotkę
- Po zapisie otwiera plik z wzorcami (wtedy nakłada blokadę na zapis i odczyt całego pliku ze wzorcami)
- Proces sprawdza, czy istnieje wzorzec pasujący do krotki
  - Jeśli tak, to proces usuwa znaleziony wzorzec, defragmentuje plik i podnosi semafor odpowiedniego procesu
- Na koniec zdejmuje blokady i zamyka pliki

Rozszerzenie opisu funkcjonalnego



Struktury danych

```
enum Operator {
    LESS, MORE, EQ_LESS, EQ_MORE, EQUAL, ANY
};
enum Type {
    INT, FLOAT, STR             // 0, 1, 2
};
class EntityPattern {
    Type type;                  // Type - string, integer, float
    int* value;                 // Wskazanie na miejsce w pamięci
    int size;                   // Rozmiar zawartości
    Operator op;                // Operator porównania
};
class TuplePattern {
    std::vector <EntityPattern> entities;          // zawartość wzorca
    int* semaphore;             // Wskazanie na semafor
};
class Entity {
    Type type;      // Type - string, integer, float
    int* value;     // Wskazanie na miejsce w pamięci
    int size;       // Rozmiar zawartości
    int compare(EntityPattern other);
};
class Tuple {
    std::vector<Entity> entities;          // zawartość krotki
    int compare(TuplePattern other);
};
```

Struktura plików

Plik przechowujący krotki

Będzie bazować na formacie CSV, każda krotka będzie zapisana w osobnej linii.

Każda kolumna będzie przechowywać wartość odpowiedniego elementu krotki, gdzie wartości typu `string` będą przechowywane w cudzysłowie, np.:

```
1, -15.5, "Słoń", 42
"mordoklejka", 144, 40.4, 1234567
9.15, "orogeneza", 9000
```

Plik przechowujący wzorce

Również zrealizowany w formacie CSV, każdy wzorzec będzie zapisany w osobnej linii.

Kolumny będą oznaczać kolejno:

Wskazanie na semafor (adres), a następnie dla każdego elementu krotki: typ, wartość i operator; np.:

```
0xAABB1984, int:600, float:*, string:"Simba", float:<3.14
0xAABB1988, string:"Powiat łękołody", float:>12345.6789
0xAABB1992, string:"Wzgórza Norylska", float:13.30
```

Podział na moduły:

- Krotki
  - Implementacja krotek, wzorców i związanych z nimi metod
- LindaCommand
  - Implementacja głównych operacji Lindy
- Moduł testujący
  - Zawiera pliki pozwalające na sprawdzenie działania biblioteki w tym przypadków granicznych: usunięcia nieistniejącej krotki, jednoczesnego dodawania i usuwania.

Opis klas i ich najważniejszych funkcji

- **`enum Operator`** - Służy do łatwego opisywania porównań między elementami krotek.
- **`enum Type`** - Opisuje typ pola w krotce.
- **`class EntityPattern`** - Wzorzec elementu krotki. Przechowuje informację o tym jakiego pola szukamy na podstawie typu, wartości i operatora porównania.
- **`class Entity`** – Element krotki, które potrafi porównać się z wzorcem
  - **`int compare(EntityPattern other)`** - Porównuje ze sobą pole wzorca wywołując metody klasy comparator
- **`class TuplePattern`** - Wzorzec krotki. Posiada wektor Pól oraz semafor sterujący odczytem z pliku.
- **`class Tuple`** - Reprezentacja krotki.
  - **`int compare(TuplePattern other)`** - Porównuje ze sobą wzorzec krotki wywołując metody klasy comparator
- **`class Comparator`** - Porównuje ze sobą pola oraz krotki na podstawie ich typu i operatora
  - **`bool compare(Entity ee, EntityPattern ep)`** - Porównywanie pól
  - **`bool compare(Tuple tt, TuplePattern tp)`** - Porównywanie krotek
  - **`bool isInt(int value, int pattern_value, Operator op)`** - Implementacja porównywań poszczególnych typów danych
  - (razem z **`isFloat`**, **`isString`**)
- **`class Parser`** - Zadaniem parsera jest sprowadzenie krotki do postaci CSV i odwrotnie, podczas zapisywania i czytania z pliku.
  - **`void toCSV(std::vector[char]* bytes, Tuple t)`** – Zamienia krotkę do zapis w pliku CSV
  - **`int parseInt(std::vector[char] bytes)`**
  - **`float parseFloat(std::vector[char] bytes)`**
  - **`std::string parseString(std::vector[char] bytes)`**
- **`class Defragmenter`** - Klasa nadzorująca pliki. Do jej zadań należy wywoływanie metody czyszczącej pliki po użyciu ich przez funkcję input.
  - **`int defragment(const char * file_descriptor)`** - Usuwa puste miejsca i defragmentuje wybrany plik. Jest wywoływana po usunięciu krotki z pliku.
- **`class Linda`** – Klasa nadzorująca wykonanie programu stanowiąca interfejs między użytkownikiem a funkcjonalnością programu. Jej zadaniem jest głównie implementacja metod pozwalających na przeszukiwanie pliku w poszukiwaniu pasujących krotek, dodawanie nowych i porównywanie ich
  - **`Tuple* input(TuplePattern tp, int timeout)`** - Input opisany w zadaniu.
  - **`Tuple* read(TuplePattern tp, int timeout)`** - Read opisany w zadaniu.
  - **`int output(Tuple t)`** - Output, zwraca kod błędu.
  - **`Tuple* searchForMatchingTuple(TuplePattern tp)`** - Celem tej funkcji jest analiza krotek zapisanych w pliku oraz dopasowywanie jej do wzorca. W wypadku zwrócenia pustego wskaźnika krotka nie istnieje w pliku.

Koncepcja realizacji współbieżności:

Komunikacja między procesami odbywa się na zasadzie zakładania blokad na plik za pomocą funkcji `fcntl()`. Procesy realizujące usuwanie danych zakładają blokady na cały plik (czytanie i pisanie), zaś te dodające tylko na określoną liczbę bajtów, którą dodają na końcu pliku (czytanie i pisanie). Czytanie wymaga założenie blokady na zapis, by nie doszło do wyścigu pomiędzy procesami czytającymi i zapisującymi.

Zarys koncepcji implementacji (język, biblioteki, narzędzia, etc.):

Implementacja realizowana będzie w języku C++, przy wybranym kompilatorze Clang11. Używane będą biblioteki standardowe języka, z emfazą na biblioteki `std::thread` i `std::fstream`. Do produkcji i testowania oprogramowania wybraliśmy narzędzie WSL z wersjami Ubuntu 18 oraz Ubuntu 20.

Przestrzeń zostanie zrealizowana jako zbiór plików tekstowych do których dostęp będzie miał proces obsługujący bibliotekę. Krotka będzie wpisem w osobnej linii, a w każdym pliku krotki będą posegregowane odpowiednio względem ilości elementów i czasu dodania.

Interfejs użytkownika oraz testy:

System zostanie zaimplementowany jako biblioteka języka C++. Oprócz tego dostarczymy narzędzie do testów, które będzie przyjmować dane w postaci:

```
lindaTest -i pattern -t timeout <- dla funkcji input
lindaTest -r pattern -t timeout <- dla funkcji read
lindaTest -o data               <- dla funkcji output
```

Poza testami manualnymi do systemy zostaną stworzone testy automatyczne w formie programu testowego, który będą sprawdzać poprawność działania i warunki brzegowe np.:

- Poprawne wyszukiwanie krotki
- Sprawdzenie, czy rozmiar pliku zmniejsza się poprawnie przy usuwaniu plików (czy rozmiar pliku nie rośnie w nieskończoność)
- Usunięcie nieistniejącej krotki
- Przeczytanie krotki, następnie dodanie jej i sprawdzenie poprawnego wybudzenia procesu