Sprawozdanie UXP1A

Paweł Wiszenko
Kacper Harasim
Alexis Nowikowski

Temat

Wieloprocesowy system realizujący komunikację w języku komunikacyjnym Linda.

Wariant

Realizacja przestrzeni krotek przy użyciu plików wraz z mechanizmami zajmowania rekordów.

Treść zadania

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 przestrzeni 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)
- (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:*, string:*) drugą. Operacja read działa tak samo jak input, lecz nie usuwa krotki z przestrzeni. Operacje read i input zawsze zwracają jedną krotkę. W przypadku gdy wyspecyfikowana krotka nie istnieje operacje read i input zawieszają się do czasu pojawienia się oczekiwanej danej.

Dodatkowe założenia

- statycznie określony maksymalny rozmiar krotki
- dla danej typu float nie ma warunków ==, >=, <=
- dla danych string warunki ==, <, <=, > , >= należy rozumieć jako leksykograficzne porównanie stringów

Architektura

System Unix

Język

C/C++

Interpretacja zadania

Wykonany zostanie system zarządzania krotkami w oparciu o serializowany dostęp do części pliku. Każda krotka będzie miała dowolną długość (ale nie większą niż z góry zdefiniowany maksymalny rozmiar) i może się składać z danych różnych typów: string, integer, float.
Możemy krotki czytać (przy okazji usuwając ją bądź nie) oraz zapisywać z bazy krotek.
Zapytania będą polegały na:

- W przypadku zapisu - podaniu nowej krotki do zapisania w bazie.
- W przypadku odczytu / usunięcia - podaniu wzorca, który musi spełniać zwrócona krotka. W razie, gdy żadna z obecnych nie spełnia podanego warunku - proces zostaje zawieszony, dopóki do bazy nie zostanie dodana krotka spełniająca wyspecyfikowany warunek.

Zostanie wyspecyfikowany język zapytań, który dla każdego typu zmiennej będzie miał odmienne operatory (np. dla typu float nie ma warunków ==, >= i <=, co jest podane w założeniach projektu).

Opis funkcjonalny

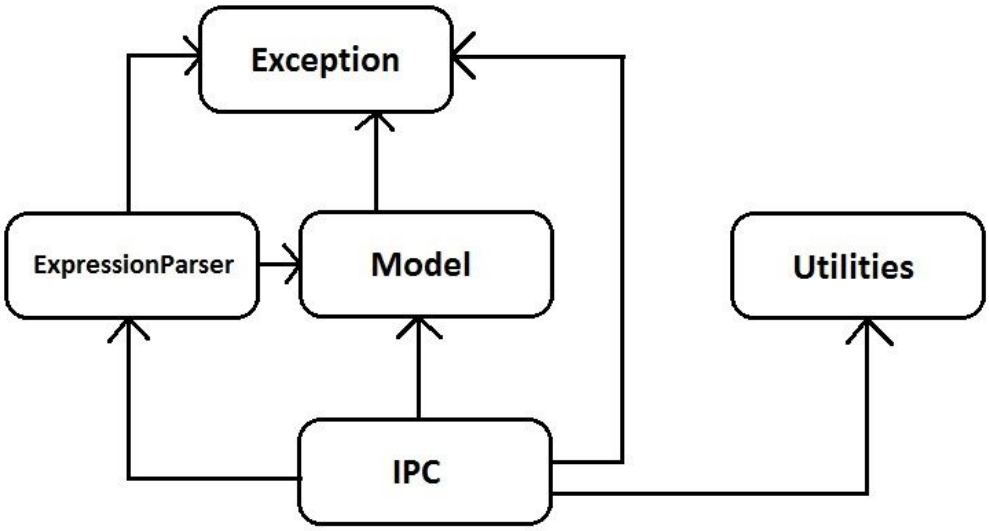
Możliwości funkcjonalne to:

- Połaczenie się do puli krotek
- Rozłączenie się z puli krotek
- Odczytanie krotki z puli bez usuwania jej, funkcja blokująca z zadanym time-outem
- Odczytanie krotki z puli razem z jej usunięciem, funkcja blokująca z zadanym time-outem
- Umieszczenie krotki w puli krotek

Opis modułów

W aplikacji obecne jest 5 modułów, które w projekcie odzwierciedlają katalogi:

- Exception/… - pliki wyjątków,
- ExpressionParser/… - parsery krotek oraz szablonów krotek,
- IPC/… - klasy realizujące komunikację Linda, wstawianie/odczytywanie krotek z puli itp (‘Core’ aplikacji),
- Model/… - klasy/struktury odzwierciedlające obiekty krotek, szablonów krotek, wpisów w pliku z pulą krotek, wpisów w pliku z listą oczekujących procesów na krotkę pasującą do szablonu,
- Utilities/… - funkcjonalność nie związana bezpośrednio z tematem projektu, głównie klasa TimeUtilities.



Opis rozwiązań funkcjonalnych

Architektura rozwiązania

Rozwiązaniem na które zdecydowaliśmy się jest utrzymywanie dwóch plików, które utrzymują cały stan bazy krotek. W jednym pliku są przechowywane krotki, w drugim lista oczekujących procesów.
System nie posiada żadnego procesu zarządczego, który synchronizuje dostęp do danych. Zamiast tego - użytkownicy bazy - osobne procesy - zajmują się również jej zarządzaniem, w tym komunikacją z innymi procesami (choć jest ona niejawna).

Współbieżność
Sposobem komunikacji z plikami jest blokowanie ich rekordów na czas odczytu/zapisu z/do nich danych. System wyszukuje interesującą nas pozycję w pliku i jednocześnie ją blokuje. Blokuje ją do momentu aż zakończy na niej operacje (np. output - wstawienie krotki). Mowa tu o obu plikach, pliku z krotkami i pliku z procesami oczekującymi na krotkę pasującą do szablonu.

Blokowanie
W sytuacji gdy wywołana zostanie operacja read/input z szablonem, dla którego nie ma dostępnej, w chwili wywołania, pasującej krotki, proces doda swój ID procesu i szablon krotki, w postaci rekordu, do pliku z listą oczekujących. Następnie wykona operację *wait* na opuszczonym semaforze, z time-outem, podanym przy wywołaniu funkcji read/input. Semafor ten jest nazwany, nazwa związana jest z ID procesu.

W przypadku kiedy żaden inny proces nie podniesie semafora, funkcja *wait* zwróci błąd pod czasie timeout, a operacja read/input rzuci wyjątek, informujący o tym, że nie została znaleziona krotka.

W przypadku kiedy inny proces umieści krotkę pasującą do wzorca, znajdzie ID procesu oczekujący na krotkę w pliku z listą procesów oczekujących oraz podniesie semafor nazwany na którym wisi oczekujący proces. Wtedy proces oczekujący wie, że została wstawiona pasująca krotka i ją odczyta z pliku.

Format plików
Pierwszym utrzymywanym plikiem jest baza krotek, w której znajdują się wszystkie krotki w systemie w zestandaryzowanej postaci, ograniczone długością do 255 znaków. Jeden rekord składa się z 256 bajtów. Pierwszy bajt to Flagi rekordu, w której jeden bit przechowuje informację o tym, czy rekord jest zajęty czy wolny, pozostałe bity są aktualnie nieużywane. Pozostałe 255 bajtów przechowuje informację o krotce.

Drugim utrzymywanym plikiem jest kolejka procesów oczejkujących na krotki pasujące do szablonów. Jeden rekord składa się z 256 bajtów. Pierwszy bajt to Flagi rekordu, znaczenie Flagi identyczne jak w rekordzie w pliku z krotkami. Kolejne 251 bajtów przechowuje informacje o szablonie krotki, na którą dany proces oczekuje. Ostatnie 4 bajty to ID procesu, który jest właścicielem tego rekordu i oczekuje na krotkę.

Ograniczenia rozmiaru krotki i szablonu krotki spowodowały znaczne uproszczenie implementacji - dzięki temu nie było konieczne zarządzanie przestrzeniami po krotkach usuniętych z systemu, oraz nie mamy problemów z pustymi przestrzeniami w pliku, które nigdy nie zostaną użyte (nie ma fragmentacji w obu plikach, operacja “usunięcia” rekordu w pliku sprowadza się do odznaczenia jednego bitu w Fladze, po odznaczeniu tego bitu, mozna ponownie uzyć miejsce zajmowane przez rekord).

Obsługa wyjątków
Sytuacje wyjątkowe są zgłaszane poprzez rzucanie wyjątków. Wszystkie wyjątki dziedziczą po zdefiniowanej klasie MessageExceptionBase, który dziedziczy po std::exception oraz wymusza podanie komunikatu błędu podczas rzucania. Każdy wyjątek dziedziczący po MessageExceptionBase jest łapany w funkcji *main* programu, która obsługuje polecenia wpisane przez użytkownika z terminala, oraz wywołuje odpowiadające im metody na puli krotek (np. Połącz z pulą krotek, wstaw krotkę, odczytaj krotkę). W przypadku kiedy zostanie złapany taki wyjątek, jego komunikat jest wyświetlany na ekranie konsoli. Program nie kończy w tym przypadku działania.

Opis interfejsu użytkownika

Dostarczony interfejs użytkownika został wykonany jako aplikacja konsolowa, która interpretuje komendy użytkownika.
Po uruchomieniu, wchodzi ona w stan oczekiwania na komendę.
Pozwala ona na wypisanie wszelkich dostępnych komend po wpisaniu jakiegokolwiek znaku do konsoli programu(np. ENTER).
Użytkownik znajdować się może w dwóch stanach: połączony z bazą, oraz rozłączony.

Interfejs dla użytkownika niepołączonego:
Jedyna dostępna komenda to “connect”, która pozwala na dołączenie się do bazy. Wymaga ona podania ścieżek do dwóch plików, które zachowują stan bazy krotek.

Interfejs dla użytkownika połączonego:
Użytkownik połączony ma możliwość rozłączenia się od bazy krotek za pomocą komendy “disconnect”.

W tym stanie ma on dostęp do wszystkich komend dostępnych w bazie krotek:
“output [krotka]” - wstawia krotkę do bazy
“input [wzorzec] [timeout]” - znajduje i usuwa krotkę o wskazanym wzorcu. Jeśli brak takowego w bazie - blokuje przez ilość czasu wyspecyfikowaną w parametrze *timeout*
“read [wzorzec] [timeout]” - znajduje i zwraca krotkę o wskazanym wzorcu. Jeśli brak takowego w bazie - blokuje przez ilość czasu wyspecyfikowaną w parametrze *timeout*

Krotka:
Krotka składa się z zera bądź więcej elementów oddzielonych przecinkami. Elementami są:

- string: łańcuch znaków otoczony cudzysłowiem
- float - liczba zmiennoprzecinkowa, gdzie całości od części oddzielone są kropką
- integer - liczba całkowita.

Przykład: (1, “test”, 3.14)

Wzorzec:
Wzorzec składa się z zera bądź więcej elementów oddzielonych przecinkami. Każdy element wzorca rozpoczyna się od specyfikatora typu: “string”, “float”, “integer” - specyfikują one typ zmiennej której poszukujemy. Następnie, po typie umieszczany być powinien dwukropek, a za nim operator, który specyfikuje zakres wartości z jakimi część wzorca ma być zgodna.

Obsługiwane jest 6 operatorów:

- * operator ANY, mówiący o tym, że może na tym miejscu wystąpić dowolna wartość.
- == porównanie - sprawdza czy elementy są równe. Jako, że dobrą praktyką jest nieporównywanie danych typu zmiennoprzecinkowego - operator ten jest niedostępny dla typu float

- > operator większości
- < operator mniejszości
- >= operator większe-równe, niedostępny dla typu float
- <= operator mniejsze-równe, niedostępny dla typu float.

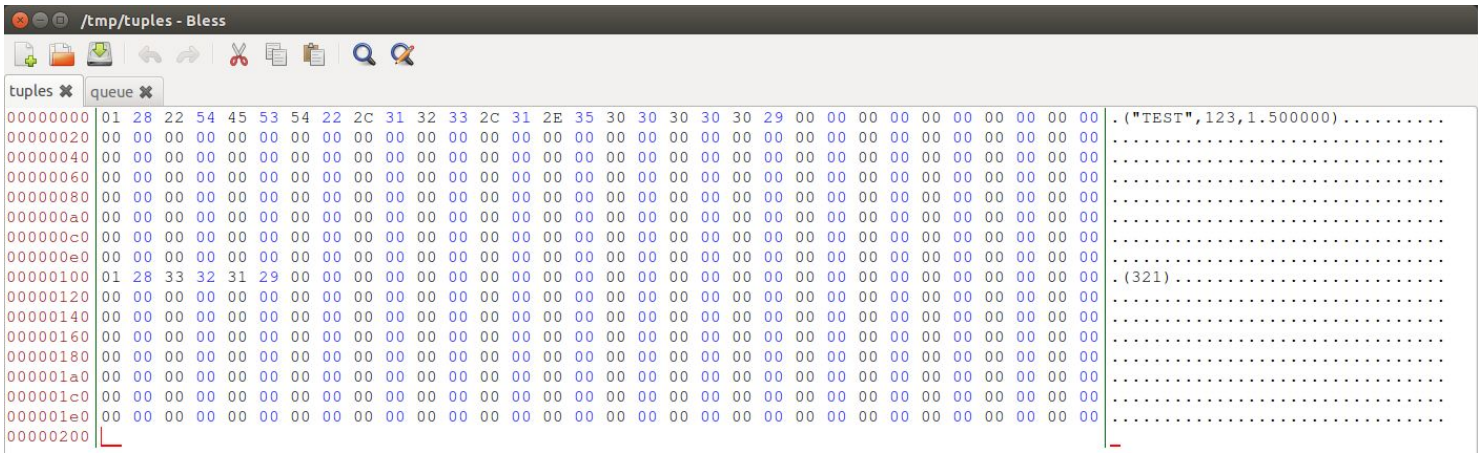
Gdy wyspecyfikowany jest operator typu ***** - po nim element powinien się skończyć, niepotrzebne jest podawanie jakiejkolwiek wartości, jako że wszystkie powinny być przechwytywane.

Po operatorze powinna nastąpić wartość zgodna ze wcześniej wyspecyfikowanym typem: string, float, bądź integer.
Przykład wzorca:
(integer:==3, string:>=”c”, float:>3.14) - wzorzec wyłapujący wszystkie krotki z liczbą rzeczywistą równą 3, stringiem “większym bądź równym” (porównanie leksykograficzne) od “c” oraz wartości typu zmiennoprzecinkowej większej od 3.14.

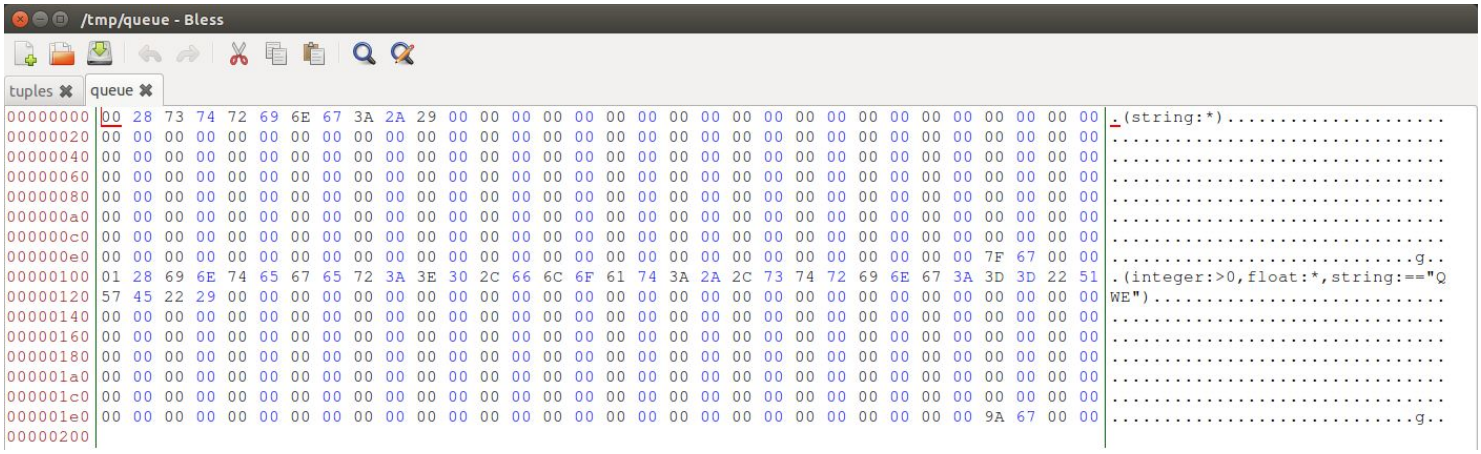
Postać plików konfiguracyjnych

W programie nieobecne są żadne pliki konfiguracyjne. Jedynymi danymi, które powstają w rezultacie działania programu są pliki w których przechowywane są wzorce i kolejka.

Poglądowy plik bazy krotek (zawiera dwie krotki):



Poglądowy plik kolejki procesów (pierwszy rekord nieużywany, drugi uzywany):



Wykorzystane narzędzia

Program napisany został z użyciem języka C++ w wersji 11.
Nie są dołączane żadne dodatkowe biblioteki, poza standardowymi dostępnymi w systemach UNIXowych.
Osiągnięta została kompatybilność pomiędzy systemami Linux i OSX.
Aby to osiągnąć - używany jest port funkcji **sem_timedwait** na OSX, który znaleziony został w ogólnie dostępnych źródłach i dostępny jest do użycia na licencji MIT.
Powodem jest niedostępność tej funkcji w systemie OSX.

Wybranym systemem budowania jest **cmake**.
Używanym IDE był CLion - środowisko od firmy Jetbrains.

Metodyka testów oprogramowania

Przeprowadzone zostały testy dla poszczególnych części programu osobno:

- Klasy, która dopasowuje wzorce do krotek
- parsera krotek
- parsera wzorców
- analizatora leksykalnego

Następnie, wszystkie moduły zostały razem zintegrowane i przeprowadzone zostały manualne testy na systemie uwzględniające współpracę wielu procesów jednocześnie.
Zostały one wykonane z użyciem interpretera komend, uwzględniając analizę pliku na którym pracuje program i na bieżąco analizując jego treść.
Przetestowane zostały rozmaite wzorce, uwzględniając wszystkie operatory.

Sprawdzone zostały standardowe przypadki użycia:

1. Włożenie krotki do systemu i sprawdzenie, czy znajduje się w danym pliku
2. Usunięcie istniejącej krotki z systemu i sprawdzenie, czy flaga w pliku dotycząca jej użycia danego rekordu została odznaczona.

3. Przeczytanie krotki istniejącej w systemie, specyfikując wzorzec(operacja read) i sprawdzenie czy nie została usunięta z pliku (odznaczona jako używana).
4. Przeczytanie krotki istniejącej w systemie, specyfikując wzorzec(operacja input) i sprawdzenie czy została usunięta z pliku (odznaczona jako używana).
5. Przeczytanie krotki, która nie istnieje w systemie, które zaskutkowało zawieszeniem się procesu.
6. Obudzenie procesu A czekającego przez proces B, który wstawił krotkę którą zainteresowany jest A.
7. Scenariusz, w którym dwa procesy chcą wykonać input na krotce, a inny wkłada ją do systemu - pierwszy z nich został obudzony, krotka została usunięta z systemu, a drugi proces pozostał w stanie oczekiwania na krotkę. Nie następuje zagłodzenie procesów.
8. Scenariusze błędów, takie jak błędna długość krotki, źle podane wzorce, czy krotki są sprawdzane przez program, a błędy są sygnalizowane dla użytkownika.