**Opasiak Krzysztof**

**Markiewicz Mikołaj**

**Klara Mateusz**

**Sosnowski Jacek**

**UXP1A**

**Dokumentacja końcowa**

1. **Treść zadania:**

Napisać wieloprocesowy system realizujący komunikację w języku komunikacyjnym Linda przy wykorzystaniu potoków nazwanych i centralnego procesu koordynującego.

1. **Interpretacja treści zadania i założenia:**

Zadanie polega na dostarczeniu biblioteki statycznej umożliwiającej procesom komunikacje zgodną z językiem Linda:

1. Procesy powinny komunikować się poprzez wspólną przestrzeń krotek, którą należy zapewnić.
2. Krotka jest to tabela o dowolnej długości, która może składać się z danych typu string, int oraz float
3. Należy dostarczyć funkcję, która umożliwi umieszczenie danej krotki w przestrzeni krotek.
4. Należy dostarczyć funkcję, która pobierze i w sposób atomowy usunie z przestrzeni, krotkę zgodną z zadanym wzorcem.
5. Wzorzec krotki jest to ciąg znaków, który specyfikuje typu danych poszczególnych pól krotki i może jednocześnie zawierać również następujące warunki logiczne dla danego pola ==, <, <=, >, >=. Wzorzec przyjmuje następujący format <typ danej>:<operator><wartość> lub w przypadku gdy wartość może być dowolna <typ danej>:\*
6. Należy dostarczyć funkcję, która umożliwi analogiczne pobranie krotki, jednak bez usuwania jej z przestrzeni.
7. Próba pobrania krotki nie pasującej do wzorca powinna zakończyć się zawieszeniem procesu do czasu pojawienia się pasującej krotki, lub przekroczenia maksymalnego czasu oczekiwania.

Jako realizację projektu rozumie się dostarczenie:

- biblioteki udostępniającej API do umieszczania i pobierania krotek.

- procesu sterującego – demona, który będzie przechowywał krotki

- zestawu testów

- dokumentacji końcowej.

Założenia:

- maksymalny rozmiar krotki jest określony statycznie i jest on tak dobrany aby zapis do potoku był atomowy.

- dla danej typu float nie ma warunku ==

- dla stringów operacje logiczne rozumiane są jako porównanie leksykograficzne

- w systemie, w którym uruchamiany będzie program istnieje katalog \tmp\linda, a wszystkie procesy korzystające z biblioteki mają prawo zapisu i odczytu do i z tego katalogu

- biblioteka nie implementuje wielowątkowości i wszystkie funkcje należy zewnętrznie synchronizować

- timeout jest to czas liczony od otrzymania przez daemona żądania wydania krotki, jeśli przed jego upływem nie pojawi się krotka spełniająca zadany warunek uznaje się wtedy, że klient powinien zostać o tym poinformowany

1. **Opis funkcjonalności oraz API**

Biblioteka została wykonana przy użyciu metodologii obiektowej i dostarcza następujące API:

class LindaClient

{

public:

//Umieszcza krotkę w przestrzeni

static void push( QVariantList &record );

//Pobiera krotkę I usuwa ja z przestrzeni

static QVariantList pull( QString &format, long timeout );

//Pobiera krotkę z przestrzeni ale jej nie usuwa

static QVariantList preview(QString &format, long timeout );

};

Funkcja push umieszcza krotkę w przestrzeni krotek, funkcja blokująca do czasu zapisania krotki do potoku.

Funkcje pull oraz preview pobierają krotkę z przestrzeni i są blokujące do czasu dostarczenia do procesu zadanej krotki z serwera, lub upłynięcia timeoutu. Timeout podawany jest w milisekundach i jest to czas po jakim klient zostanie odblokowany jeśli krotka nie pojawi się w programie koordynującym. Podanie wartości timeout równej -1 oznacza czekanie w nieskończoność. Ponad to operacja pull atomowo usuwa pobieraną krotkę z przestrzeni krotek.

1. **Podział na moduły**

System składa się z następujących modułów:

- moduł klienta – biblioteka linkowana statycznie, implementująca zaproponowane API. Komunikacja odbywa się poprzez klasę LindaClient posiadająca metody statyczne.



- moduł serwera – program uruchomiony w tle, który sprawuje rolę procesu sterującego, tj. obsługuje wszystkie żądania przesłane przez klientów oraz stanowi magazyn dla krotek.



-moduł współdzielony – statycznie linkowana biblioteka implementująca elementy programu wykorzystywane zarówno w module klienta jak i serwera.



- moduł testujący – stanowi zbiór programów i skryptów przeznaczonych do testowania. Można tu wyróżnić testy jednostkowe oraz testowy program klienta, który wykonuje działania klienta w zależności od danych podanych mu na standardowe wejście. Ponad to w celu automatyzacji testów wykonano skrypty i odpowiednie pliki wejściowe dla programów testowych.

1. **Opis komunikacji pomiędzy modułami.**

W systemie występują dwie strony komunikacji – klient czyli dowolny proces używający implementowanej biblioteki, oraz serwer czyli daemon dostarczony razem z biblioteką, który przechowuje krotki. Komunikacja odbywa się poprzez potoki nazwane i możemy wyróżnić w niej 3 przypadki tj. trzy rodzaje żądań, które mogą być kierowane przez klienta do serwera:

1. Umieszczenie krotki w przestrzeni

Klient zapisuje do potoku serwera dane krotki, która ma zostać zapisana tj. jej format oraz wartości danych. Operacja jest blokująca do czasu, gdy w potoku serwera nie będzie wystarczającej ilości miejsca do zapisania tej komendy. Serwer czyta po kolei komendy z swojego potoku, po czym je wykonuje. Wykonanie tej komendy przez serwer ma dwa warianty:

- jeśli nikt nie czeka na krotkę o takiej zawartości to jest ona dodawana do wielozbioru krotek przechowywanych przez serwer.

- jeśli istnieje proces, który zgłosił żądanie pobrania krotki o takich danych to krotka ta jest przekazywana do tego procesu i zależnie od rodzaju żądania jest ona zapisywana do pamięci lub nie.

1. Pobranie i usunięcie krotki z przestrzeni

Klient zapisuje do potoku serwera wzorzec krotki, którą chce pobrać oraz ścieżkę do pliku jego potoku (tworzenie potoku w chwili pierwszej potrzeby jego użycia) oraz maksymalny czas oczekiwania na krotkę. Serwer po odczytaniu takiego żądania sprawdza dostępność krotki:

- jeśli krotka jest dostępna to jest ona usuwana z przestrzeni, potok klienta jest otwierany, krotka zapisywana, a potok zamykany, jeśli wystąpi błąd, należy zaprzestać zapisu, zamknąć potok, a krotka powinna dalej być przechowywana w pamięci

- jeśli krotka nie jest dostępna, uruchamiany jest timer. Jeśli krotka pojawi się przed upływem czasu, jest ona wysyłana do klienta jak wyżej, jeśli czas upłynie to do klienta wysyłana jest wiadomość o przekroczeniu limitu czasu.

c) Pobranie krotki bez usunięcia jej z przestrzeni – czynność analogiczna do poprzedniej ale krotka nie jest usuwana z przestrzeni krotek.

Ponad to w przypadku kończenia procesu pełniącego role serwera krotek wpisuje on do swojego potoku komunikacyjnego wiadomość, która informuje wątek czytający z potoku o konieczności zakończenia.

//Jacek proszę zweryfikuj protokół komunikacyjny i dopisz strukturę komunikatów

Zatem w całym systemie możemy wyróżnić następujące rodzaje komunikatów o przedstawionej poniżej budowie:

1. umieść krotkę

- nagłówek informujący o typie komunikatu

- ciąg znaków mówiący o typach pól krotki

- pola krotki

1. pobierz krotkę i usuń z przestrzeni

- nagłówek informujący o typie komunikatu

- wyrażenie opisujące warunki, jakie musi spełniać krotka

- maksymalny czas oczekiwania

1. pobierz krotkę

- nagłówek informujący o typie komunikatu

- wyrażenie opisujące warunki, jakie musi spełniać krotka

- maksymalny czas oczekiwania

1. krotka znaleziona

- nagłówek informujący o typie komunikatu

- ciąg znaków mówiący o typach pól krotki

- pola krotki

1. czas upłynął

- nagłówek informujący o typie komunikatu

1. **Opis zaimplementowanego rozwiązania.**

Szczegółowy opis poszczególnych funkcji znajduje się w dokumentacji wygenerowanej przy pomocy programu Doxygen, natomiast w niniejszym punkcie przedstawiono jedynie zarys ogólny implementacji i ważniejszych funkcji

1. **Moduł klienta**

Moduł ten dostarcza interfejs do zapisu i odczytu krotek. Dane dostarczane przez klienta są sprawdzane pod kątem poprawności, a następnie formowana jest wiadomość, która jest wpisywana do potoku serwera. Biblioteka może zgłosić wyjątek spowodowany niepoprawnym formatem danych lub brakiem uruchomionego procesu serwera. W przypadku wiadomości pull i preview klient po wysłaniu komunikatu do serwera oczekuje na utworzonym przez siebie potoku na odpowiedz od serwera. Uproszczony diagram sekwencji dla klienta wygląda następująco:



1. **Moduł serwera**

Moduł jest to program składający się z dwóch wątków.

-Jeden wątek (wątek w którym działa CommandQueue) jest to wątek pomocniczy, który odczytuje z potoku komendy od klientów i przekazaniu jej do wykonania, w związku z czym przez większość czasu wątek ten jest zawieszony na czytaniu z potoku.

-Drugi wątek natomiast zajmuje się faktycznym wykonywaniem zadań zleconych przez klientów oraz pilnuje aby klienci, którym upłynął czas oczekiwania zostali wznowieni i o tym powiadomieni.

Komendy wczytane przez CommandQueue są gromadzone w kolejce zdarzeń zapewnionej przez bibliotekę Qt. W momencie wywołania metod CommandDispatcher’a komendy te są gromadzone w kolejce zdarzeń, a ich wykonanie przenoszone jest do drugiego wątku.

Klasa CommandDispatcher stanowi szkielet całego programu serwera. Przechowuje ona strukturę danych w której przechowywane są krotki oraz żądania klientów. Struktura ta ma budowę słownika, gdzie pojedynczym elementem jest:

Klucz (QString):

String reprezentujący typy danych w krotce np. „ssfii”

Wartość(RecordTable):

Lista rekordów i lista oczekujących na rekordy o tej strukturze

Oczekujący- QList<Demand>

Lista struktur reprezentujących żądania klientów

Rekordy o tej strukturze – Qlist<QVariantList>

Lista rekordow

Każda krotka trafia do odpowiadającej jej wzorcowi tabeli. Przy dodawaniu krotki, sprawdzane jest czy spełnia ona warunki podane przez któregokolwiek z oczekujących na krotkę o takiej strukturze danych. Jeśli tak to jest ona mu przesyłana i w zależności od rodzaju żądania klienta dodawana do tabeli lub nie.

W przypadku dodania żądania na krotkę o zadanych warunkach wyszukiwana jest pozycja według podanego wzorca typów, po czym sprawdzane jest czy w liście rekordów znajduje się jakiś, który spełnia warunki wyspecyfikowane przez klienta, jeśli tak to wysyłany jest rekord, jeśli nie to żądanie dodawane jest do listy żądań oczekujących i uruchamiany jest tiemer, jeśli czas wyspecyfikowany przez klienta dobiegnie końca to żądanie usuwane jest z listy, a do potoku klienta zostaje wpisana specjalna wiadomość informująca o upłynięciu czasu.

Uroszczone działanie serwera przedstawiają poniższe diagramy sekwencji:





1. **Szczegółowy opis interfejsu użytkownika**

Szczegółowy opis interfejsu został wygenerowany z użyciem programu doxygen. Fragment jego dotyczący został przedstawiony poniżej:

//tu będzie to co generuje doxygen o LindaClient

1. **Wykorzystane narzędzia**

W trakcie prac nad projektem wykorzystano następujące narzędzia:

- CMake – automatyczne budowanie

- Doxygen – generacja dokumentacji

- Git – repozytorium

- Eclipse – IDE wykorzystywane do implementacji projektu

- Enterprise Architect – środowisko do projektowania projektów software

- Biblioteka Qt

- Biblioteka Boost

- Google Test Framework – testy jednostkowe

- potoki nazwane – komunikacja miedzy procoesowa

- mechanizm sygnałów – kończenie pracy serwera

1. **Metodyka testowania**

Testy do wykonanego projektu zostały podzielone na kilka etapów:

- pierwszy etap stanowią testy jednostkowe, które sprawdzają podstawową funkcjonalność wybranych klas. W testach tych uczestniczą przede wszystkim klasy odpowiedzialne za sprawdzania zgodności wzorca czy porównywanie warunków wyspecyfikowanych przez użytkownika z danymi faktycznymi.

- drugi etap stanowi program, z prostym interfejsem tekstowym, który wykorzystuje zaimplementowana bibliotekę. Umożliwia on dodanie lub pobranie krotki o zadanym wzorcu, przez co pozwala na przetestowanie w prosty sposób wszystkich scenariuszy, jakie uda się wymyślić.

-trzeci etap są to skrypty wykorzystujące program z ręcznej analizy poprawności z etapu drugiego. Dzięki ich użyciu otrzymujemy powtarzalne testy scenariuszowe. Obecnie testowane są następujące scenariusze:

//Mikołaj dopisz tutaj jakie scenariusze