Marcin Cieślikowski
Marek Jasiński
Paweł Goździkowski

Dokumentacja końcowa – projekt UXP1A

Treść: Napisać program DLM (Distributed Lock Manager) - zarządcy blokad. Celem DLM jest synchronizacja dostępu do współdzielonych zasobów. Aby bezpiecznie korzystać z zasobu należy założyć na niego blokadę. Jeżeli założenie blokady nie jest możliwe proces żądający blokady musi być zawieszony do czasu aż wystąpi timeout lub blokada będzie mogła być przez niego uzyskana.

1) Interpretacja treści zadania

Należy napisać program zarządcy blokad oraz bibliotekę funkcji, dzięki którym procesy działające na współdzielonych zasobach, będą synchronizowane w dostępie do tych zasobów. Komunikacja i synchronizacja pomiędzy zarządcą blokad, a pozostałymi procesami będzie się odbywać poprzez potoki nazwane (FIFO). Procesy w celu uzyskania dostępu do zasobu będą musiały wywołać funkcję biblioteczne. Procesy mogą zakładać różne typy blokad tj. Concurrent Read, Concurrent Write, Protected Read, Protected Write oraz Exclusive. Proces zarządcy blokad rozstrzygał będzie o możliwości założenia blokady przez proces na podstawie aktualnie założonych blokad, zgodnie z macierzą współzależności między typami blokad. W celu uniknięcia zagłodzenia możliwość przydzielenia procesowi zasobu jest rozpatrywana wyłącznie wtedy, gdy inny proces, który wcześniej zgłosił żądanie, nie czeka na ten zasób.

Zasoby są przydzielane wg. poniżej zdefiniowanej macierzy:

Rys. *Macierz koincydencji*

	CR	CW	PR	PW	EX
CR	+	+	+	+	-
CW	+	+	-	-	-
PR	+	-	+	-	-
PW	+	-	-	-	-
EX	-	-	-	-	-

2) Pełen opis funkcjonalny – „black-box"

Proces DLM – przy starcie inicjalizuje potok nazwany (FIFO), który służyć będzie do odbierania komunikatów od procesów. Potok ma stałą i znaną nazwę, oraz ścieżkę dostępu. Jest to /tmp/DLM/DLMfifo. Proces inicjalizuje struktury przechowujące informację o zasobach. Proces zawiesza się w oczekiwaniu na żądania dostępu lub zwolnienia zasobów współdzielonych. Żądania obsługiwane są w kolejności zgłoszeń. Osoba korzystająca z biblioteki musi linować statyczną bibliotekę DLMLib, oraz zaincludować w swoim kodzie plik DLMlib.h. Następnie może w swoim kodzie wołać metody opisane w następnym punkcie. Użytkownik sam przyznaje ID zasobu zasobą do, których chce mieć współbierzny dostęp np. zasobowi /home/mojkatalogdomowy/plik.txt przyznaje

identyfikator 100, teraz sam musi pilnować, aby przed każdym odwołaniem założyć odpowiednią blokadę. DLM jest procesem działającym w przestrzeni użytkownika.

3) Pełen opis biblioteki klienckiej:

DLM_PATH - ścieżka do katalogu w którym będą przechowywane pliki potoków nazwanych (FIFO).

DLM_FIFO_PATH - ścieżka do kolejki DLM.

Typy blokad :

Pełna nazwa blokady	Zdefiniowana nazwa	Kod blokady
Concurrent Read	CR	0
Concurrent Write	CW	1
Protected Read	PR	2
Protected Write	PW	3
Exclusive	EX	4

Kody odpowiedzi:

GRANTED - przydzielono zasób.

TIMEDOUT - wystąpił timeout - nie przydzielono zasobu.

LOCKED - zasób zajęty.

UNLOCKED - zasób został zwolniony.

FREE - zasób jest wolny, zwracany w obsłudze żądania TRYLOCK, która nie przydziela zasobu, a tylko sprawdza, czy zasób jest wolny.

Kody błędów:

EOPENDLMFIFO - błąd otwarcia potoku nazwanego (FIFO) DLM-a.

ECREATEFIFO - bład utworzenia potoku nazwanego - klienta, bądź DLM-a (ten błąd może wynikać z nieposiadania uprawnień do katalogu podanym w DLM_PATH).

EOPENCLIENTFIFO - błąd otwarcia potoku nazwanego klienta.

EWRITE - błąd wysyłania żądania do DLM-a, występuje wtedy, gdy nie uda się wysłać pełnej struktury DLMrequest.

EREAD - błąd czytania z potoku - występuje wtedy, gdy nie uda nam się odczytać pełnej stuktury DLMresponse.

EBADLOCKTYPE - niepoprawna blokada - podana inna blokada niż te zdefiniowane w punkcie o typach blokad.

EBADTIMEOUT - niepoprawny timeout - podana wartość ujemna < -2.

ENOTLOCKED - próba zwolnienia zasobu, który nie został przydzielony.

EAGAIN - próba ponownego zajęcia tego samego zasobu.

- int DLM_lock(int resource_id, int lock_type, long timeout) – wysyła komunikat do zarządcy blokad. Komunikat składa się z identyfikatora zasobu, typu blokady, czasu timeout, identyfikatora procesu (PID). Następnie tworzony jest potok nazwany (FIFO) o nazwie odpowiadającej pid procesu. Proces zawiesza się w oczekiwaniu na komunikat od zarządcy blokad. Typy komunikatów zwrotnych od procesu zarządcy:

Po odebraniu komunikatu proces usuwa potok.

- int DLM_unlock(int resource_id) – wysyła do DLM żądanie zwolnienia zasobu składające się z identyfikatora zasobu oraz identyfikatora procesu (PID). Jako typ blokady przekazywany jest specjalny parametr FREERESOURCE. Parametr timeout nie ma tutaj znaczenia. Następnie tworzony jest potok nazwany (FIFO) o nazwie odpowiadającej pid procesu. Proces zawiesza się w oczekiwaniu na komunikat od zarządcy blokad. Typy komunikatów zwrotnych od procesu zarządcy:

- int DLM_trylock(int resource_id, int lock_type) – wysyła żądanie, w którym parametr timeout = TRYLOCK, gdzie TRYLOCK oznacza zdefiniowaną stałą. DLM odpowiada na takie zgłoszenie możliwie najszybciej. Typy komunikatów zwrotnych procesu zarządcy:

Po odebraniu komunikatu proces usuwa potok.

4) Opis i analiza poprawności stosowanych protokołów komunikacyjnych

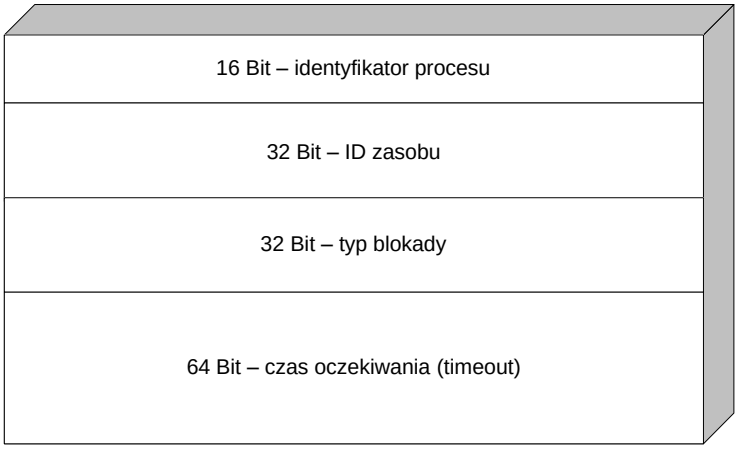
Rysunki przedstawiają strukturę komunikatów przesyłanych do i od DLM. Komunikaty obsługiwane są przez DLM w kolejności zgłaszania, co wynika ze struktury potoku FIFO. Proces oczekujący na odpowiedź od DLM zawiesza się na utworzonym potoku w oczekiwaniu na tę odpowiedź. Po odebraniu wiadomości potok odbiorczy procesu jest kasowany, dzięki czemu zakończone procesy nie powodują pozostania nieużywanych potoków. Nazwa potoku odbiorczego procesu jest ściśle ustalona i zawiera numer identyfikacyjny procesu (PID). Aby komunikaty były przesyłane atomowo muszą być mniejsze niż pojemność bufora kolejki FIFO. Pojemność bufora kolejki w Linuxie to 4096B, a standard POSIX 1-2001 wymaga bufora większego od 512B. Komunikaty do i z DLM mają odpowiednio 18B i 4B, dzięki czemu będą przesyłane atomowo.

Struktura opisująca komunikat pokazany na Rys. 2:

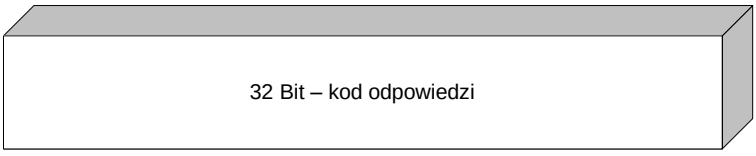
```
typedef struct DLMrequest {
      pid_t pid;
      int resource_id;
      int lock_type;
      long timeout;
} DLMrequest;
```

Struktura opisująca komunikat pokazany na Rys. 3 :

```
typedef struct DLMresponse {
      int response;
} DLMresponse;
```



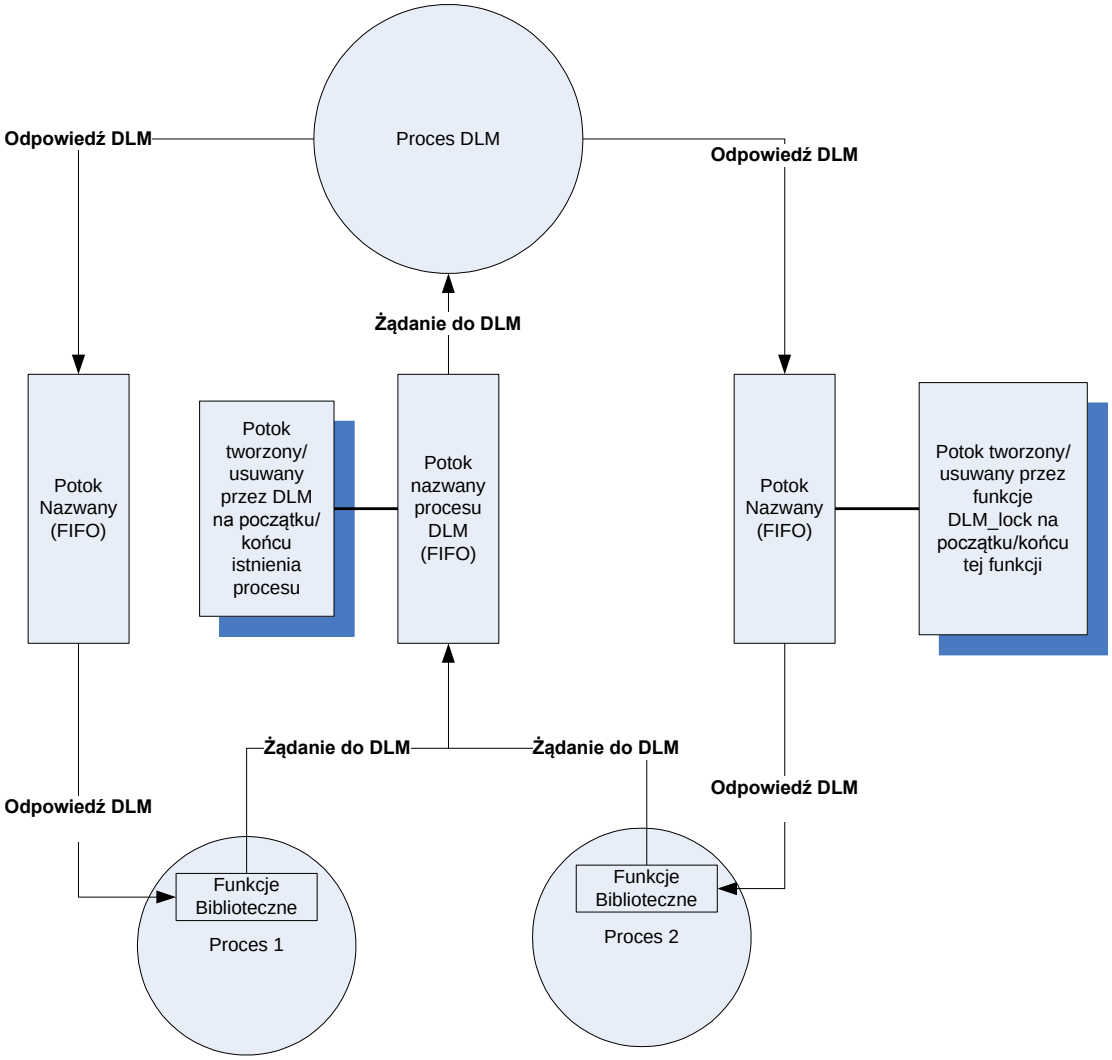
Rys 2. Komunikat do DLM.



Rys 3. Komunikat odpowiedzi DLM.

4) Podział na moduły i struktura komunikacji między nimi

Rysunek przedstawia schemat podziału na moduły oraz strukturę komunikacji pomiędzy nimi.



5) Opis najważniejszych rozwiązań funkcjonalnych (kluczowych funkcji).

Opis struktur danych DLM:

`struct client` – reprezentuje jedno rządanie o zasób klienta, przechowuje typ blokady, oraz pid klienta.

`struct client_timeout` - struktura przechowuje informacje pid klienta, zasób na który czeka klient z timeoutem.

`struct resource_clients` - struktura opisująca jeden zasób - klientów oczekujących na zwolnienie zasobu, oraz klientów, którym przydzielono zasób.

W programie wykorzystane są następujące kontenery STL:

`map<int, resource_clients> resource_map` - kluczem w tej mapie jest id zasobu, wartością jest struktura resource_client opisana powyżej.
`multimap<long, client_timeout> timestamp_map` - kluczem jest czas od początku ery unixa zsumowany z czasem po którym ma nastąpić timeout w milisekundach.

Opis głównych funkcji:

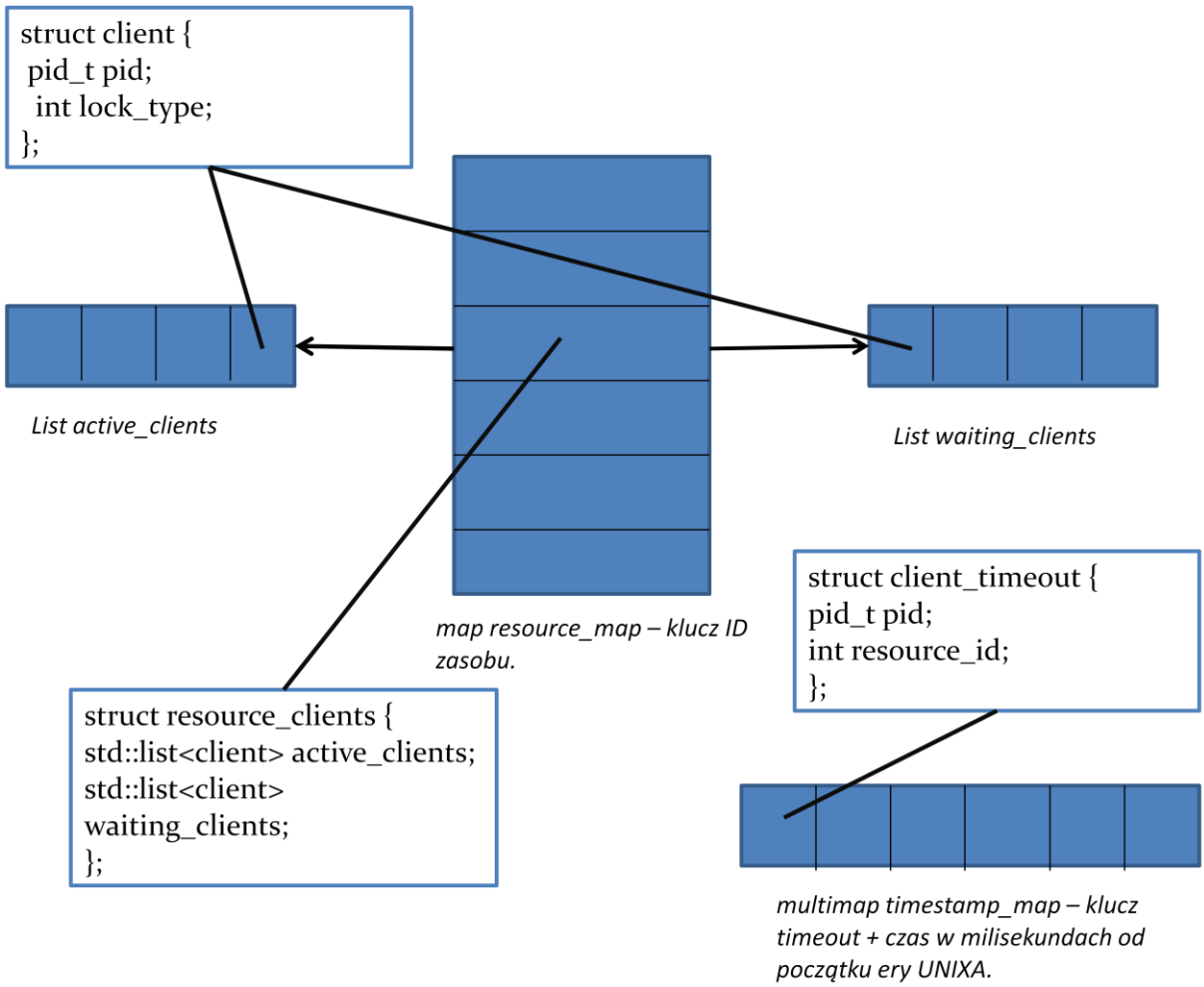
- `void try_grant(map<int, resource_clients>::iterator iter)` - dostaje jako argument wywołania iterator na zasób, który chcemy spróbować przydzielić klientom oczekujących na niego (znajdujących się na liście waiting clients). Pobierany jest typ blokady, którego żąda pierwszy proces na liście oczekujących(waiting_clients), następnie iterujemy po liście klientów, którym przydzielono zasób(active_clients i sprawdzamy, czy typ blokady klienta oczekującego nie koliduje z którymś z aktywnych klientów, jeżeli nie koliduje zasób jest przydzielany (jest przenoszony do listy aktywnych klientów).
- `void timeout_alarm(int)` - jest to funkcja obsługi sygnału SIGALRM. Okres po jakim ma nastąpić sygnał SIGALRM jest ustawiany za pomocą funkcji systemowej ualaram. W funkcji tej w każdym obiegu pętli pobierany jest aktualny czas i sprawdzamy czy element na który wskazuje iterator czy klucz jest większy niż aktualny czas, jeżeli tak to wychodzimy z pętli i ustawiamy nowy timeout(jeżeli lista timeoutów nie jest pusta), w przeciwnym wypadku usuwamy klienta z oczekujących.

Funkcja main programu DLM:

- inicjalizacja programu (w zależności od opcji uruchomienia: otwarcie pliku logu, otwarcie w trybie demona itp).
- otwierana jest kolejka FIFO DLM-a.
- DLM zawiesza się w oczekiwaniu na żadanie klienta.
- DLM na czas obsługi komunikatu blokuje przychodzenie sygnału (nie chcemy, aby została wywołana funkcja obsługi sygnału SIGALRM, w trakcie obsługi jakiegoś komunikatu.
- na podstawie typu blokady i timeout-u podejmowane są następujące akcje:

- zwalnianie zasobu - sprawdzane jest czy klient znajduje się na liście aktywnych, jeżeli znaleziono, to usuwamy z aktywnych i wysyłamy odpowiedź, że udało się zwolnić zasób. W przeciwnym wypadku próbujemy znaleźć go w kolejce oczekujących w przypadku powodzenia usuwamy go z oczekujących i wysyłamy odpowiedź, natomiast jeżeli nie, to wysyłamy odpowiedź o próbie zajęcia nie przydzielonego zasobu. Jeżeli był ostatnim klientem aktywnym i kolejka oczekujących pusta, to usuwamy z mapy zasobów ten zasób.
- żądanie zasobu - sprawdzane jest czy zasób już istnieje, jeżeli nie, to jest tworzony i przydzielany klientowi.Jeżeli już istniał, to sprawdzamy czy klient nie używa już zasobu, następnie sprawdzane jest czy nie znajduje się w kolejce klientów oczekujacych na zasób, jeżeli to wszystko jest spełnione, podejmowana jest odpowiednia akcja w zależności od tego jaki jest timeout:
  - ✓ jeżeli timeout ujemny to sprawdzamy czy możemy przydzielić i wysyłamy odpowiedź (TRYLOCK(-2) nie przydziela zasobu).
  - ✓ jeżeli timeout zerowy, to wywołujemy funkcję try_grant w przeciwnym wypadku wstawiamy do mapy timeoutów klienta(jeżeli aktualnie wstawiony znajduje się na początku multimapy, to ustawiamy czas po którym ma być dostarczony sygnał SIGALRM, na czas timeout-u tego klienta i wywołujemy funkcję try_grant.

- odblokowywane są zablokowane sygnały i następuje próba czytania z kolejki FIFO.



6) Opis wykorzystanych narzędzi:

- biblioteka kliencka DLM jest napisana w języku C i korzysta z podstawowych bibliotek języka C.
- DLM napisany jest w języku C++ (użyte zostały kontenery STL).
- mechanizmy systemu unix/linux (sygnały, kolejki nazwane(FIFO)).

6) Uruchamianie:
- proces DLM można uruchomić z następującymi flagami:
-d - uruchamia w trybie demona.
-l <nazwa_pliku> - logowanie do pliku.
-h wyswietla pomoc programu.

Opcja demona musi być użyta w parze z opcją logowania do pliku.

7) Opis testów i wyników testowania.

Opis testów i wyników testowania

W celu przetestowania programu DLM oraz biblioteki DLMlib zostały utworzone:

1. Program testowy: DLM-client
2. Skrypt uruchamiający przypadki testowe: run_test_cases.sh
3. Skrypty będące przypadkami testowymi: test_case*.sh, gdzie * oznacza numer testu

Ad. 1. Program testowy DLM-client przyjmuje 5 argumentów. Są to kolejno:

- ✓ numer zasobu
- ✓ typ blokady
- ✓ timeout (ms)
- ✓ czas uśpienia (s)
- ✓ liczba iteracji

Dokładny opis programu znajduje się w dokumentacji w standardzie man.

Ad. 2. Argumentem skryptu są przypadki testowe, które mają zostać kolejno uruchomione. Skrypt uruchamia DLM o ile nie jest jeszcze uruchomiony.

Ad. 3. Skrypty będące przypadkami testowymi nie wymagają argumentów. Realizują one konkretne scenariusze:

- test_case0.sh – uruchamia jednego klienta, który 3-krotnie zajmuje i zwalnia zasób stosując typ blokady EX. Test wykazał poprawność komunikacji z DLM.

-test_case1.sh – uruchamia jednocześnie dwóch klientów, którzy 5-krotnie wykonują tą samą czynność. Klienci odwołują się do tego samego zasobu. Jeden z nich zakłada blokadę typu EX, natomiast drugi PW. Czasy timeout to 6s, natomiast czas uśpienia dla każdego z klientów wynosi 5s. Wynik testu: procesy uzyskują zasób naprzemiennie.

- test_case2.sh – uruchamia dwa procesy odwołujące się do tego samego zasobu. Procesy wywołują funkcję try_lock. Wynik testu: zasób jest za każdym razem dostępny, gdyż nikt go nie zajął.

- test_case3.sh – uruchamia dwa procesy odwołujące się do tego samego zasobu. Jeden z nich używa funkcji try_lock z typem blokady EX, natomiast drugi z nich zajmuje zasób z blokadą CW. W wyniku testu proces wywołujący funkcję try_lock otrzymuje komunikat o tym, że zasób jest zajęty lub wolny w zależności od tego, czy drugi proces go zajął.

- test_case4.sh – uruchamia dwa procesy odwołujące się do tego samego zasobu z blokadami PW i parametrem timeout 600ms. Czas uśpienia pomiędzy zajęciem a zwolnieniem wynosi 5s. Zgodnie z oczekiwaniami pojawiają się odpowiedzi od DLM o minięciu czasu timeout.

- test_case5.sh – uruchamia osiem procesów, z czego 3 odwołują się do zasobu o numerze 1000, a 5 do zasobu o numerze 100. Logi wskazują na poprawność działania zarządcy blokad.

Wszystkie wytworzone programy, oraz skrypty zostały przetestowane w laboratorium w sali 9.