Zdalne Kolejki *RQP*

Opis Protokołu — 6 maja 2008

Łukasz Bieniasz-Krzywiec, 235922

* * *

Spis treści

1 Streszczenie

Niniejszy dokument opisuje specyfikację protokołu zdalnych kolejek *RQP*. Składa się z następujących sekcji:

- opis celów protokołu,
- opis założeń,
- opis formatu komunikatów,
- opis stanów.

2 Cele

Głownym celem protokołu jest rozszerzenie możliwości zwykłych uniksowych kolejek komunikatów o funkcję zapisywania danych przez sieć do kolejek na systemach zdalnych.

3 Terminologia

Wykaz używanych w tekście terminów i ich znaczenia:

- *Nadawca* — użytkownik instancji protokołu, który chce przesyłać informacje.
- *Odbiorca* — użytkownik odczytujący informacje z docelowej kolejki zdalnej.
- MCA *nadający* — agent zawiadujący wysyłaniem wiadomości do kolejek zdalnych.
- MCA *odbierający* — agent zawiadujący odbieraniem wiadomości wysyłanych przez *MCA* nadającego oraz umieszczaniem ich w odpowiednich kolejkach zdalnych.
- *Kolejka zdalna* — docelowa kolejka uniksowa.
- *dead.letter.q* — kolejki, do których kierowane są niepoprawnie sformułowane wiadomości lub wiadomości *pewne*, dla których nie ma odbiorcy.
- *DLQ* — skrót od *dead.letter.q*.
- *Wiadomość pewna* — wiadomość, która musi zostać dostarczona.
- *Wiadomość niepewna* — wiadomość, która nie wymaga potwierdzenia dostarczenia.

4 Założenia

4.1 Połączenie

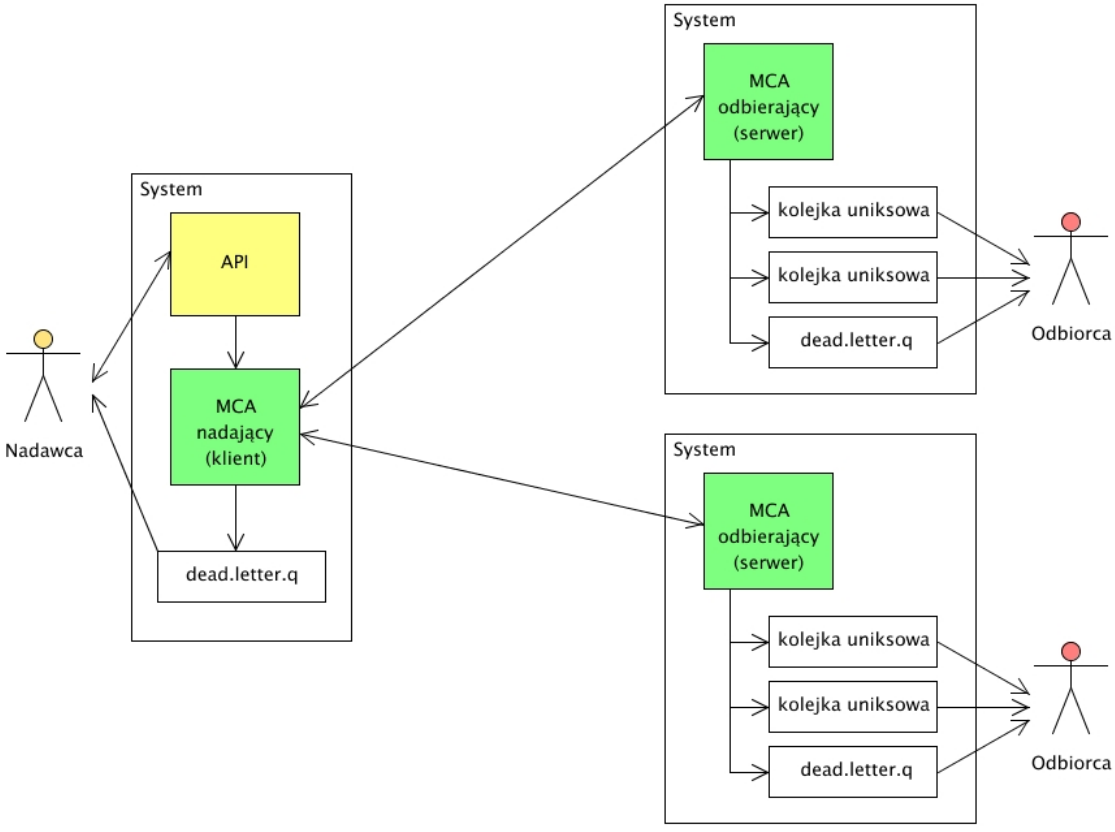
Połączenie składa się z jednego *kanału transmisyjnego*, przez który są przesyłane wiadomości od *MCA* nadającego do *MCA* odbierającego. W jednej chwili *MCA* nadający może utrzymywać wiele połączeń z różnymi *MCA* odbierającymi.

4.2 Powiązania z innymi protokołami

Protokół działa w warstwie aplikacji. Transmisja wiadomości *pewnych* musi być niezawodna, cała komunikacja będzie się zatem opierać na protokole TCP/IP.

4.3 Model komunikacji

Poniżej przedstawiony jest ogólny model *RQP*:



Protokół *RQP* działa w oparciu o model klient-serwer. *MCA* nadający jest klientem, a *MCA* odbierający to serwery nasłuchujące na porcie 6861.

5 Opis formatu komunikatów

Założenia:

- porządek octetów w liczbach: sieciowy,
- sposób interpretacji liczb ze znakiem: jak w arytmetyce uzupełnień do 2.

Typy pomocnicze:

```
KEY_T {
  uint8         key_length;
  octet[length] key;
}
```

`key_length` — pole zawierające długość klucza, długość ta obejmuje tablicę `key`.

```
DATA_T {
  uint32            buf_length;
  octet[buf_length] buf;
}
```

`buf_length` — pole zawierające długość danych, długość ta obejmuje tablicę `buf`.

```
USER_MESSAGE_T {
  uint8   type;
  KEY_T   key;
  DATA_T  data;
  DATA_T  opt;
}
```

Typy komunikatów:

```
KEY_MSG_T {
  uint8 msg_id;
  KEY_T key;
}

NET_MESSAGE_MSG_T {
  uint8           msg_id;
  USER_MESSAGE_T  msg;
  uint32          id;
}

CONFIRMATION_MSG_T {
  uint8   msg_id;
  uint32  id;
}
```

6 Opis wymienianych komunikatów

W protokole używane są następujące komunikaty (w nawiasach podane sa argumenty, a po dwukropku typ komunikatu). Numer porządkowy to jednocześnie `msg_id` przydzielone komunikatowi.

1. `HAS_KEY_MSG(key) : KEY_MSG_T`

   Pytanie o posiadanie kolejki o kluczu `key`.

2. `ACCEPT_KEY_MSG(key) : KEY_MSG_T`

   Twierdząca odpowiedź na pytanie o posiadanie kolejki o kluczu `key`.

3. `REJECT_KEY_MSG(key) : KEY_MSG_T`

   Przecząca odpowiedź na pytanie o posiadanie kolejki o kluczu `key`.

4. `NET_MESSAGE_MSG(type, key, data, opt, id) : NET_MESSAGE_MSG_T`

   Wiadomość `data` o typie `type` (*pewna* lub *niepewna*) i opcjach `opt`, skierowana do kolejki o kluczu `key`. Wartość `id` jest używana do identyfikowania potwierdzeń.

5. `ACCEPT_MESSAGE_MSG(id) : CONFIRMATION_MSG_T`

   Potwierdzene otrzymania wiadomości *pewnej* o identyfikatorze `id` i dostarczenia jej do docelowej kolejki IPC.

6. `REJECT_MESSAGE_MSG(id) : CONFIRMATION_MSG_T`

   Potwierdzene otrzymania wiadomości *pewnej* o identyfikatorze `id` i dostarczenia jej do kolejki dead.letter.q.

`msg_id`	Komunikat
1	`HAS_KEY_MSG(key)`
2	`ACCEPT_KEY_MSG(key)`
3	`REJECT_KEY_MSG(key)`
4	`NET_MESSAGE_MSG(type, key, data, opt, id)`
5	`ACCEPT_MESSAGE_MSG(id)`
6	`REJECT_MESSAGE_MSG(id)`

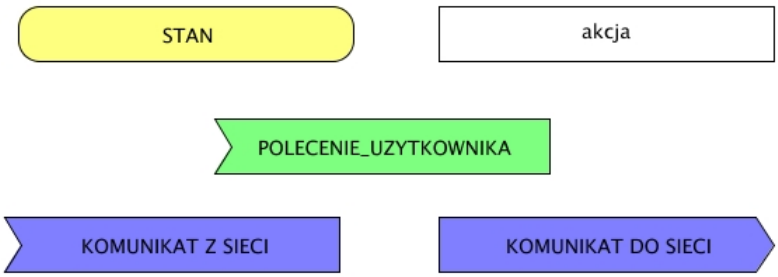
7 Opis stanów

MCA nadający może znajdować się w jednym z trzech stanów:

- `WAIT_MESSAGE_STATE`
- `WAIT_CONNECTION_STATE`
- `WAIT_CONFIRMATION_STATE`

MCA odbierający jest bezstanowy.

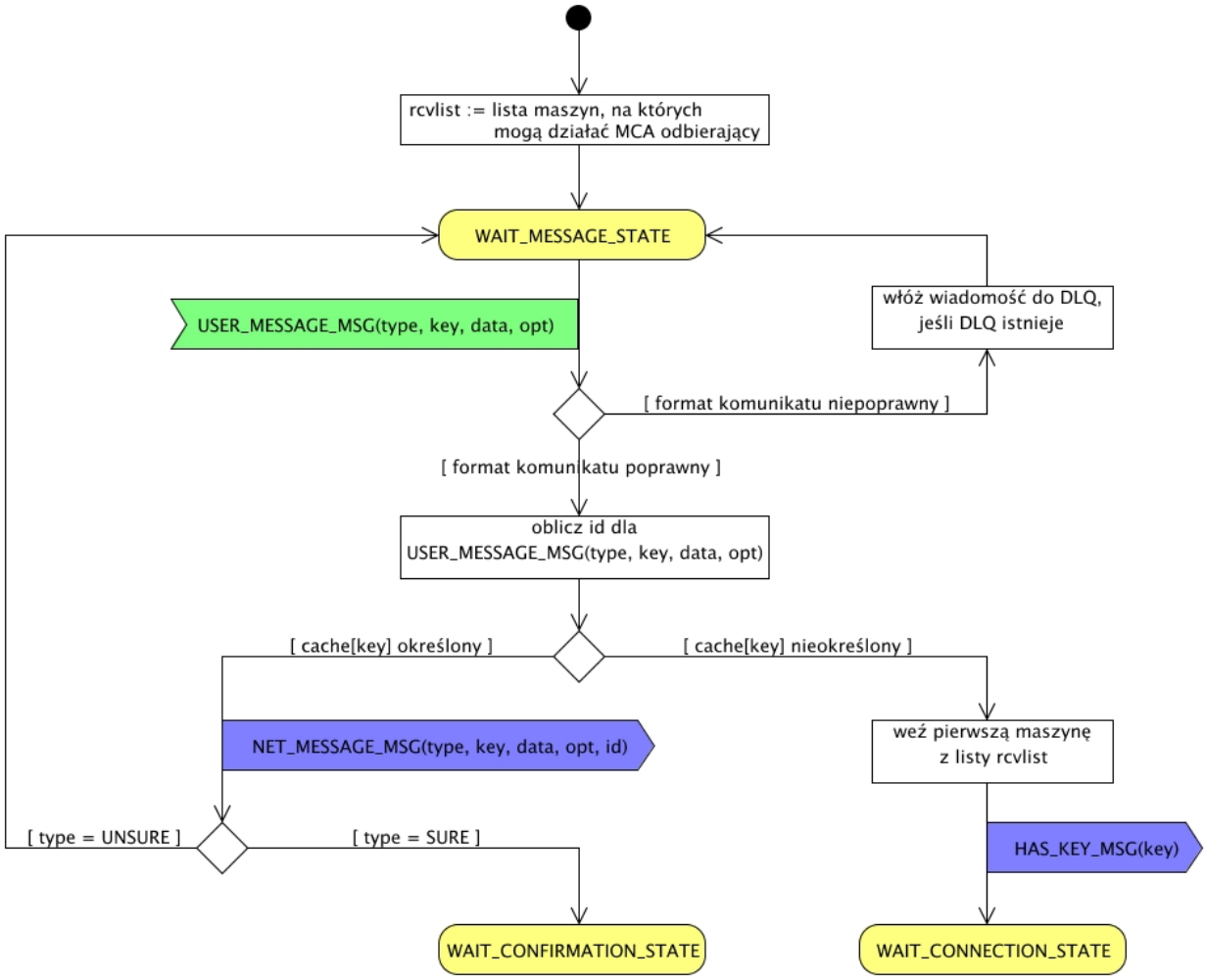
Oznaczenia używane na diagramach:



7.1 *MCA* nadający

7.1.1 `WAIT_MESSAGE_STATE`

W tym stanie *MCA* nadający czeka na zlecenia wysłania wiadomości do jednej z dostępnych kolejek zdalnych. Gdy użytkownik dostarczy takie zlecenie, *MCA* nadający musi określić adresata wiadomości, czyli *MCA* odbierającego dysponującego kolejką IPC o odpowiednim kluczu. Aby ułatwić sobie to zadanie, *MCA* nadający wykorzystuje podręczny `cache` połączeń z *MCA* odbierającymi. `cache` jest dynamiczny i ciągle uaktualniany podczas działania. Przez `cache[key]` oznaczamy informacje potrzebne do odnalezienia połączenia z *MCA* odbierającym posiadającym kolejkę zdalną o kluczu `key`.



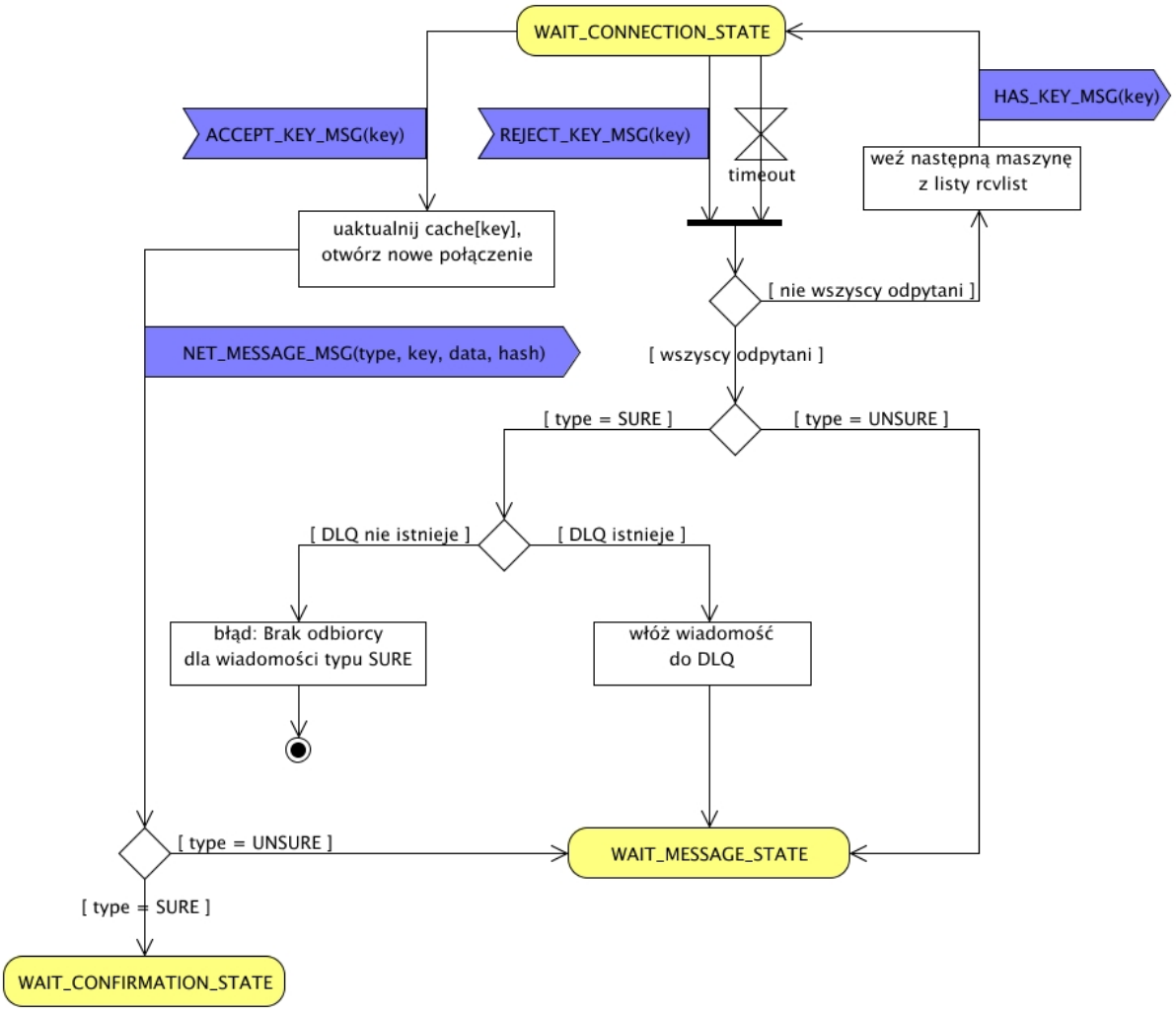
7.1.2 `WAIT_CONNECTION_STATE`

W tym stanie *MCA* nadający czeka na odpowiedź na komunikat `HAS_KEY_MSG(key)`. Zmienne `type`, `key`, `data` oraz `opt` pochodzą ze zlecenia użytkownika. `id` to identyfikator wiadomości obliczony wcześniej.

Akceptowane komunikaty:

- `ACCEPT_KEY_MSG(key)`
- `REJECT_KEY_MSG(key)`

Inne komunikaty są ignorowane.



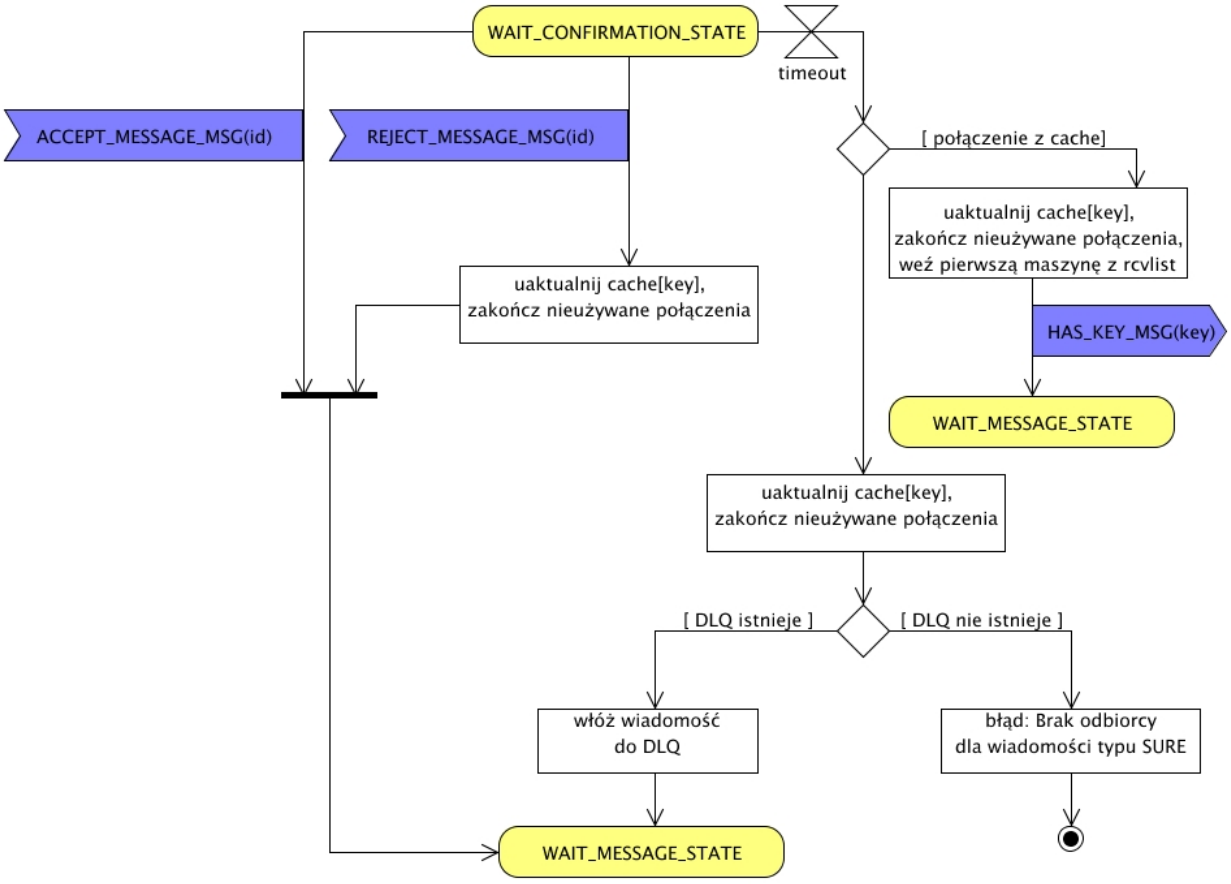
7.1.3 `WAIT_CONFIRMATION_STATE`

W tym stanie *MCA* nadający czeka na potwierdzenie dostarczenia *pewnej* wiadomości `NET_MESSAGE_MSG(type, key, data, opt, id)`. Zmienna `key` pochodzi ze zlecenia użytkownika. `id` to identyfikator wiadomości obliczony wcześniej.

Akceptowane komunikaty:

- `ACCEPT_MESSAGE_MSG(id)`
- `REJECT_MESSAGE_MSG(id)`

Inne komunikaty są ignorowane.



Gdy w dowolnym momencie nie powiedzie się wysłanie komunikatu `NET_MESSAGE_MSG(type, key, data, opt, id)`, to uaktualniamy `cache` (usuwamy informacje o zerwanym połączeniu). Jeśli próbowaliśmy wysłać wiadomość pewną, to wstawiamy ją do *dead.letter.q* lub zgłaszamy bład w przypadku gdy *dead.letter.q* nie istnieje.

Gdy w dowolnym momencie nie powiedzie się wysłanie komunikatu `HAS_KEY_MSG(key)`, to próbujemy odpytać następną maszynę na liście `rcvlist`.

7.2 *MCA* odbierający

Działanie *MCA* odbierającego jest bardzo proste. Czeka on na komunikaty wysyłane przez *MCA* nadających i odpowiednio na nie reaguje.

Akceptowane komunikaty:

- `HAS_KEY_MSG(key)`

MCA odbierający sprawdza czy dysponuje kolejką IPC o kluczu `key`. Jeśli tak - odsyła komunikat `ACCEPT_KEY_MSG(key)`, jeśli nie - odsyła komunikat `REJECT_KEY_MSG(key)`.

- `NET_MESSAGE_MSG(type, key, data, opt, id)`

  Jeśli `type = UNSURE`, to *MCA* odbierający wstawia wiadomość do kolejki IPC o kluczu `key` (o ile taka istnieje). Jeśli `type = SURE`, to *MCA* odbierający sprawdza czy dysponuje kolejką IPC o kluczu `key`. Jeśli tak, to wstawia do niej wiadomość i odsyła komunikat `ACCEPT_MESSAGE_MSG(id)`. Jeśli nie, to sprawdza czy ma dostęp do `dead.letter.q`. Gdy `dead.letter.q` istnieje, to *MCA* odbierający kieruje tam wiadomość i odsyła komunikat `REJECT_MESSAGE_MSG(id)`.

8 Numery

Wykaz używanych w tekście stałych i ich wartości:

- SURE = 1,
- UNSURE = 2,
- TIMEOUT - zalecana wartość to 10 sekund