Unix - architektura, programowanie i administrowanie

Sprawozdanie z projektu "Wieloprocesowy system realizujący komunikację w języku Linda przy pomocy kolejek komunikatów."

Konrad Kaproń
Mateusz Morusiewicz
Marcin Suwała

Spis treści

1. Zadanie projektowe

W ramach projektu została stworzona biblioteka krotek, oraz wykorzystujące ją proste serwer i klient krotek.

Biblioteka umożliwia tworenie krotek złożonych z trzech podstawowych typów: string, integer, float. Dodatkowo, biblioteka umożliwia porównanie krotki ze wzorcem oraz serializację i deserializację krotek.

Serwer krotek zapewnia możliwość przechowania krotek i zapytań o nie, realizuje wysyłanie krotek (output) i odbieranie krotek (input, read). Odbieranie krotek polega na przekazywaniu jednej z krotek, która jest zgodna z podanym wzorcem. Zarówno odbieranie jak i wysyłanie krotek to operacje atomowe i blokujące, które są zrealizowane przy pomocy kolejek komunikatów.

2. API biblioteki

Dostarczane funkcje

Biblioteka krotek udostępnia funkcje zarządzające krotkami:
— Alokacja i dealokacja pamięci na poszczególne składowe,
— Tworzenie krotek na podstawie szablonu,
— Przypisywanie typów i wartości do poszczególnych składowych,
— Porównywanie krotek na podstawie szablonów i wartości,
— Serializację i deserializację krotek.

Obsługa błędów

Biblioteka definiuje następujące błędy:
— `TUPLE_E_OUT_OF_RANGE`: odwołanie do pola krotki poza zadeklarowane,
— `TUPLE_E_INVALID_TYPE`: podany błędny typu,
— `TUPLE_E_INVALID_OP`: podana błędna operacja.

Struktury i deklaracje funkcji biblioteki

Listing 2.1. Struktura krotki

```
typedef struct tuple_element {
    uint16_t type;
    union {
        int i;
        float f;
        char *s;
    } data;
} tuple_element;

typedef struct tuple {
    unsigned nelements;
    struct tuple_element *elements;
} tuple;
```

Listing 2.2. Funkcje tworzące i usuwające krotki

```
tuple *tuple_make(const char *format, ...);
tuple *tuple_make_nelements(unsigned nelements);

void tuple_free(tuple *obj);
```

Listing 2.3. Funkcje pomocnicze dla krotek

```
int tuple_typeof(const tuple *obj, unsigned position);
int tuple_operator(const tuple *obj, unsigned position);
int tuple_compare_to(const tuple* obj, const tuple *blueprint);

int tuple_to_buffer(const tuple *obj, char *buffer, int size);
tuple *tuple_from_buffer(const char *buffer);

char *tuple_error_to_string(int code);
```

Listing 2.4. Funkcje inicjalizujące pola krotek

```
int tuple_set_int(tuple *obj, unsigned position, int input);
int tuple_set_float(tuple *obj, unsigned position, float input);
int tuple_set_string(tuple *obj, unsigned position, char *input);

int tuple_set_int_op(tuple *obj, unsigned position,
                     int input, uint16_t operator);

int tuple_set_float_op(tuple *obj, unsigned position,
                     float input, uint16_t operator);

int tuple_set_string_op(tuple *obj, unsigned position,
                        char *input, uint16_t operator);
```

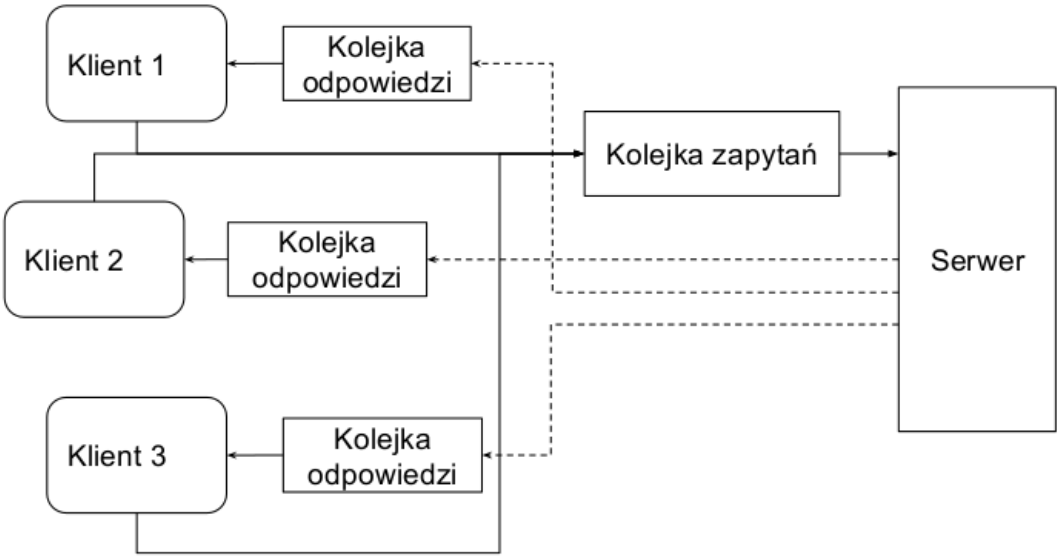
Listing 2.5. Funkcje pobierające pola krotek

```
int tuple_get_int(const tuple *obj, unsigned position,
                  int *output);

int tuple_get_float(const tuple *obj, unsigned position,
                    float *output);

int tuple_get_string(const tuple *obj, unsigned position,
                     char **output);
```

3. Podział na moduły oraz schemat komunikacji



Rys. 3.1. Schemat komunikacji

System podzielony jest na trzy moduły:
1. Bibliotekę,
2. Serwer,
3. Klienta

Serwer oraz klient korzystają z biblioteki do obsługi krotek przy komunikacji. Szczegółowy opis serwera oraz klienta został podany w sekcji "Szczegółowy opis interfejsu użytkownika"

3.1. Protokół

Serwer tworzy kolejkę komunikatów o określonej nazwie ($/Linda-Server$). Klienci wkładają do kolejki zapytania dotyczące krotek. Klient, przed włożeniem zapytania do kolejki serwera, tworzy własną kolejkę o nazwie takiej, jak jego PID. Umożliwi to odebranie odpowiedzi od serwera.

3.2. Zapytanie klienta

Kolejne pola komunikatu klienta:
1. Wielkość struktury pid_t (1B)
2. PID procesu klienta (2-4B)

3. Numer komendy (operacji) (1B)
4. Krotkę po serializacji (pozostałe miejsce)

Maksymalny rozmiar komunikatu jest definiowany przez klienta i jest stały, przyjęte jest 256 bajtów (domyślny rozmiar komunikatu w kolejce systemowej). Zmiana maksymalnego rozmiaru komunikatu wymaga rekompilacji programu.

Numery komend są jak następuje:
— 1: send,
— 2: read,
— 4: get

3.3. Odpowiedź serwera

Kodowanie komunikatu serwera zawiera:
1. Kod wyjścia (1B)
2. Krotka po serializacji (jeśli dotyczy)

Maksymalny rozmiar komunikatu zwrotnego tak jak w zapytaniu klienta. Kod wyjścia to 0, jeśli operacja się powiodła (udało się dodać krotkę przy operacji "send", została dopasowana krotka przy operacji "read" lub "get").

W razie błędu komunikacji z klientem krotka dla operacji "get" krotka zostaje usunięta z serwera.

4. Szczegółowy opis interfejsu użytkownika

Do dyspozycji użytkownika są gotowe funkcje w C służące do obsługi krotek po stronie klienckiej oraz serwerowej. Poza funkcjami dostępne są: gotowy program służący za serwer oraz prosty klient shellowy do odbierania i wysyłania krotek.

4.1. Serwer

Funkcja C tworząca serwer to $voidrun_server(char * server_name)$;, za argument przyjmująca nazwę głównej kolejki systemu do obsługi krotek.
Program *server* umożliwia stworzenie serwera o predefiniowanej nazwie kolejki komunikatów $/Linda - Server$.

4.2. Klient

Funkcja C służąca do wysyłania krotek to $staticvoiddo_request(tuple*obj, intrequest, constchar* server_queue_name)$;. Za argumenty przyjmuje:
— *obj*: wskaźnik do krotki (tworzonej przy pomocy funkcji z biblioteki),
— *request*: operacja do wykonania (send, read, get),
— $server_queue_name$: nazwa kolejki serwera
Program *client* pozwala na wykonywanie operacji na krotkach, łączy się do kolejki komunikatów serwera o nazwie $/Linda - Server$. Pobranie krotki (operacje *read* i *get*) powodują zawieszenie programu do czasu otrzymania odpowiedzi. Przykłady użycia programu:

1. $clientsend - n2 - sLinda - i42$
2. $clientget - n2 - sany - igt40$
3. $clientget - n1 - ieq32$
4. $clientsend - n1 - i32$

Listing 4.1. Użycie programu server

```
client --help
Usage:
./client send|read|get -n [int] [elements]
        Where elements is -i|-f|-s [operator] [value]
        Where operator is eq|lt|le|gt|ge|any
                (only specified if command != send)
        With operator == any value is forbidden
```

5. Logowanie

Wszystkie błędy są logowane przy pomocy funkcji perror, a informacje mniej krytyczne przy pomocy printf. W połączeniu z pracą na pierwszym planie umożliwia to pełną integrację z systemd-journald.

6. Wykorzystane narzędzia

W pracy nad systemem użyto poniższych narzędzi:
— Check - szkielet aplikacji (framework) do testów jednostkowych
— Python3 - do stworzenia programu demonstracyjnego
— Doxygen - do dokumetnacji kodu
— LaTeX - do stworzenia sprawozdania

7. Opis metodyki testów i wyników testowania

Zostały stworzone testy jednostkowe dla funkcji biblioteki krotek (tworzenie krotek, przypisywanie typów i wartości, porównywanie krotek, serializacja i deserlializacja). Testy jednostkowe są przeprowadzane automatycznie przy kompilacji programów.
Dodatkowo napisane zostały testy funkcjonalne programów interfejsu użytkownika *server* i *klient* (przesyłanie krotek, porównywanie krotek, poszczególne operacje na krotkach). Testy funkcjonalne nie są tak dokładne i sformalizowane jak testy jednostkowe, jednak pozwalają na przetestowanie bardziej zaawansowanych funkcjonalności systemu. Testy te są uruchamiane ręcznie, należy też przeanalizować ich wynik.

Przy pomocy testów, zarówno jednostkowych, jak i funkcjonalnych, zostało z powodzeniem wykryte kilka błędów w trakcie tworzenia oprogramowania.