Rozdział 1

Wieloplatformowe powłoki procesów w strukturze MRROC++

1.1 Wprowadzenie

Ramowa struktura programowa MRROC++ do sterowania robotów, stworzona, rozwijana i wykorzystywania w Instytucie Automatyki i Informatyki Stosowanej PW do sterowania głównie robotów przemysłowych typu IRb-6 została zaimplementowana dla systemu operacyjnego czasu rzeczywistego QNX [3, 1]. Sterowanie w sposób ciągły obiektem mechanicznym wymaga gwarancji spełnienia bezwzględnych wymagań czasowych, stąd konieczność stosowania systemu operacyjnego takiej właśnie klasy. W procesie sterowania wykorzystywane są informacje o środowisku dostarczane przez exteroreceptory, jak np. kamery czy systemy rozpoznawania dźwięku. Twórcy systemu QNX położyli nacisk na wydajność implementacji, jednak kwestie związane z obsługą nowoczesnych urządzeń zostały odsunięte na dalsze plan. Z tego właśnie powodu korzystne wydaje się stworzenie jednolitej struktury, która umożliwi rozproszenie systemu MRROC++ na komputery w sieci lokalnej działające pod kontrolą także innych systemów operacyjnych niż QNX. Takie podejście jest uzasadnione, ponieważ w ramach działania systemu robotycznego wykonywanych może być szereg operacji, które nie narzucają ścisłych wymagań co do czasu zakończenia. Przykładami takich zadań mogą być analiza oraz synteza mowy w celu zapewnienia interakcji z użytkownikiem bądź operatorem. Kolejną motywacją dla zapewnienia działania struktury na innych systemach operacyjnych jest dostęp do bardziej zaawansowanych kompilatorów oraz specjalizowanych bibliotek optymalizowanych na konkretny typ procesora, które to narzędzia nie są dostępne dla systemu QNX.

1.2 Implementacja

W ramach opisanych prac zostały zaimplementowane modyfikacje kodu struktury MRROC++, które zapewniają działanie wybranych jej komponentów na systemach operacyjnych klasy UNIX. Jako przykładowa platforma został tutaj wykorzystany system Linux ze względu na bardzo szeroki zakres obsługiwanych urządzeń zewnętrznych typowo wykorzystywanych w robotyce. Wprowadzane zmiany bazują na standardzie POSIX, zatem po niewiel-

kich modyfikacjach możliwe będzie wykorzystywanie także innych systemów operacyjnych. Zastosowanie systemu Windows jako platformy do działania systemu MRROC++ zostanie także przybliżone, ponieważ mimo znaczących różnic między nim, a systmami klasy UNIX, istnieją obecnie już środowiska (biblioteki) zapewniające kompatybilność działania kodu tworzonego z wykorzystaniem standardu POSIX.

Kluczowe dla działania komponentów systemu MRROC++ w ramach innych niż QNX systemów operacyjnych jest rozwiązanie następujących kwestii:

1. przenośność na poziomie kodu źródłowego,
2. zdalne powoływanie procesów na komputerach w sieci lokalnej,
3. możliwość uruchomienia systemu z różnymi parametrami konfiguracji,
4. zapewnienie komunikacji pomiędzy procesami działającymi na różnych systemach operacyjnych,
5. stworzenie gotowych do użycia szablonów komponentów systemu MRROC++.

1.2.1 Przenośność na poziomie kodu źródłowego

Przenośność na tym poziomie oznacza, że ten sam plik z kodem źródłowym może być skompilowany tak dla systemu QNX jak i innych, w tym przypadku dla systemu Linux. Dzięki takiemu rozwiązaniu zmniejsza się ilość kodu, którym trzeba zarządzać i utrzymanie systemu staje się mniej kosztowne. Uzyskanie takiej przenośności polegało na:

1. zastąpieniu wszystkich wywołań funkcji specyficznych dla systemu QNX ich odpowiednikami ze standardu POSIX,
2. wprowadzeniu gdzie to było konieczne dyrektyw kompilacji warunkowej.

Przykładem modyfikacji pierwszego typu było usunięcie wywołań funkcji delay() (dostępnej jedynie w systemie QNX4) jej odpowiednikiem ze standardu POSIX.1-2001 w postaci funkcji usleep():

```
// QNX4
unsigned int delay( unsigned int duration );

// POSIX.1-2001
int usleep(useconds_t usec);
```

Przykładem modyfikacji drugiego typu było wprowadzenie dyrektyw kompilacji warunkowej, które w zależności od docelowego systemu operacyjnego włączają odpowiedni fragment kodu:

```
#if defined(__QNXNTO__)
#include <sys/sched.h>
#include <process.h>
#endif /* __QNXNTO__ */
```

Opisane modyfikacjom umożliwają kompilowanie kodu źródłowego MRROC++ na innych niż QNX systemach. Dzięki temu można także dokonywać kompilacji nowszymi, bardziej restrykcyjnymi wersjami kompilatorów i w ten sposób eliminować błędy jeszcze przed uruchomieniem programu. Przykładowa kompilacja kodu na systemie Linux wymaga ustawienia zmiennych dla programu `make`:

```
~/mrrocpp $ make CC=gcc CXX=g++ LDFLAGS=-lm
```

1.2.2 Zdalne powoływanie procesów

Kluczowa w rozproszonym systemie MRROC++ jest możliwość powoływania procesów na zdalnych komputerach, co w znaczący sposób ułatwia i przyśpiesza uruchamianie jak również testowanie systemu. Do tej pory, w ramach jednolitego środowiska lokalnej sieci komputerów pracujących pod kontrolą systemu QNX do powoływania zdalnych procesów wykorzystywana była bezpośrednio funkcja spawn(). Programy były uruchamiane ze wspólnego katalogu instalacji MRROC++, udostępnianego między lokalnymi komputerami w ramach działania natywnej sieci systemu QNX – QNET.

W celu zachowania tej bardzo przydatnej cechy MRROC++ w środowisku innych niż QNX systemów operacyjnych zostały wprowadzone modyfikacje, których celem jest zachowanie obecnej funkcjonalności. Wykorzystują one narzędzie `rsh` (ang. remote shell), wywodzące się z systemu 4.2BSD i dostępne obecnie dla praktycznie każdego systemu operacyjnego. W celu poprawnego działania tej usługi każdy z węzłów QNX lokalnej sieci musi zostać odpowiednio skonfigurowany, co ogranicza się do zapewnienie automatycznego uruchamiania demona obsługującego tą usługę. W tym celu należy zmodyfikować dwa pliki: w pliku `/etc/inetd.conf` należy odkomentować linie:

```
#
# Shell, login, exec are BSD protocols.
#
shell  stream  tcp     nowait  root    /usr/sbin/rshd     in.rshd
login  stream  tcp     nowait  root    /usr/sbin/rlogind  in.rlogind
exec   stream  tcp     nowait  root    /usr/sbin/rexecd   in.rexecd
```

zaś do pliku `/etc/rc.d/rc.local` dodać uruchomienie procesu serwera nadrzędnego inetd:

```
/usr/sbin/inetd
```

W analogiczny sposób należy zapewnić uruchamianie usługi na innych systemach operacyjnych. Ze względu na wygodę pracy korzystnie jest tak skonfigurować środowisko, aby zdalne powoływanie procesów nie wymagało od operatora systemu MRROC++ podawania haseł logowania przy każdym powołaniu procesu.

W przypadku instalacji QNX w zamkniętym laboratorium, gdzie w praktyce bezpieczeństwo w ramach sieci lokalnej nie jest zagrożone, wystarczające jest zapewnienie istnienia użytkownika o takiej samej nazwie na wszystkich komputerach. Dostęp do takiego konta należy skonfigurować z pustym hasłem (co i tak często ma miejsce).

W przypadku wykorzystywania innych systemów korzystnie jest się posłużyć możliwością autoryzacji przez zdefiniowanie reguł zaufania pomiędzy komputerami. Takiej konfiguracji dokonuje się przez edycję pliku `/etc/hosts.equiv`, przy czym należy skorzystać z dokumentacji danego systemu w celu uzyskania szczegółowych informacji, gdyż mogą występować pewne różnice w zależności od implementacji.

Powoływanie procesów przy użyciu usługi rsh zostało zaimplementowane w metodzie configurator::process_spawn(). Tymczasowo dla zachowania kompatibilności z obecnym systemem zachowano uruchamianie procesów w katalogu instalacji MRROC++. Przykładowe uruchomienie zdalnego procesu realizowane jest przez wywołanie polecenia:

```
rsh yoyek "cd /net/siodmy/home/mrrocpp/bin; \
      UI_HOST=siodmy /net/siodmy/home/mrrocpp/bin/mp_c \
      ..."
```

Wprowadzenie tej zmiany pociągnęło za sobą modyfikacje zabijania procesów. Obecnie stosowane wywołanie systemowe SignalKill() z parametrem określającym identyfikator węzła, na którym pracuje proces zostało zastąpione wywołaniem kill(), które zabija lokalny proces rsh reprezentujący zdalnie wykonywany komponent.

1.2.3 Konfiguracja systemu

Do chwili obecnej konfiguracja systemu była przechowywana w plikach tekstowych, do których każdy z uruchamianych procesów miał bezpośredni dostęp przez wspólny system plików działający ramach sieci QNET. W konfiguracji systemu są przechowywane takie parametry jak np. informacja o trybie testowym działania efektorów, czy nazwy węzłów, na których mają zostać uruchomione poszczególne procesy systemu.

W przypadku uruchamiania procesów MRROC++ poza systemem QNX, a co za tym idzie bez bezpośredniego dostępu do plików konfiguracyjnych musiał zostać zapewniony dostęp do istniejącej konfiguracji. W tym celu został dodany do systemu nowy proces, tzw. serwer konfiguracji o nazwie configsrv. Jest on powoływany przy uruchamianiu interfejsu użytkownika UI i pozostaje aktywny przez cały czas działania systemu.

Dostęp do konfiguracji przez pozostałe procesy systemu odbywa się przy pomocy klasy configurator(), której interfejs został zachowany. Modyfikacji uległy jedynie ciała metod, które zamiast odwoływać się do pliku tekstowego komunikują się z serwerem konfiguracji przy pomocy komunikatów.

Dla przykładu – do odczytania wartości parametru liczbowego `servo_tryb` wywoływana jest metoda
int configurator::return_int_value(const char* _key, const char* __section_name) z argumentami "servo_tryb" i "edp_irp6_on_track".

```
[edp_irp6_on_track]
test_mode=2
node_name=yoyek64
servo_tryb=1
```

1.2.4 Komunikacja międzyplatformowa

Komunikacja między procesami systemu MRROC++ odbywa się przy pomocy tzw. spotkań, czyli blokującego wysyłania komunikatu przez klienta i oczekiwanie na odpowiedź od serwera. W konsekwencji narzuca to sposób konstruowania programów pełniących rolę serwerów jako pętli, w której oczekują one na komunikaty z poleceniami od klientów. Jako przykład takich konstrukcji można podać najniższe warstwy systemu, czyli procesy EDP oraz VSP.

Wykorzystywane w MRROC++ natywne dla QNX funkcje przesyłania komunikatów (tj. MsgSend(), MsgReceive() oraz MsgReply()) wykorzystują jako warstwę transportu sieciowego protokoły QNET. Z tego względu nie jest możliwa obsługa takich pakietów poza systemem QNX i dla realizacji działania MRROC++ w innych środowiskach należało zapewnić inną metodę komunikacji.

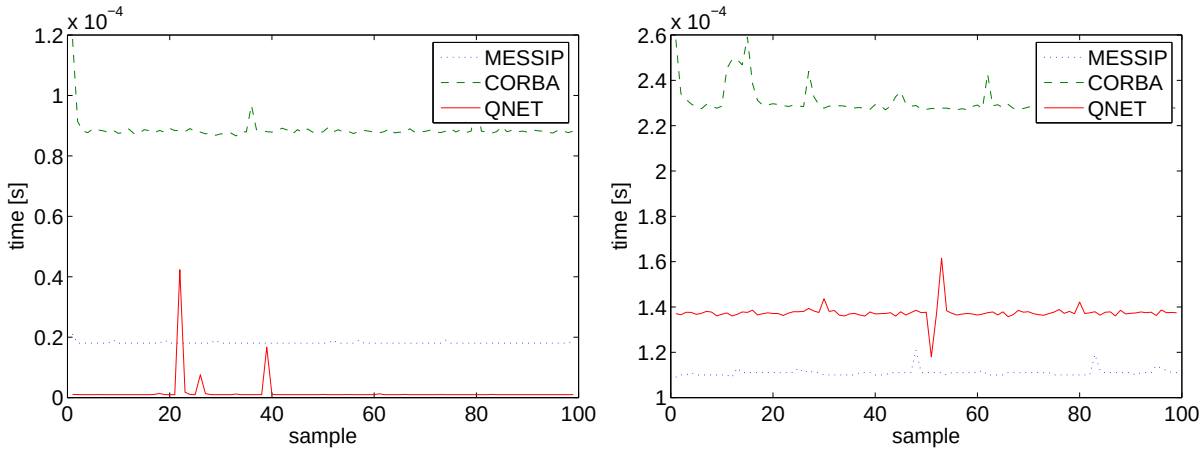
Metoda komunikacji międzyplatformowej zaproponowana w opisywanym rozwiązaniu bazuje na protokole TCP/IP jako najbardziej rozpowszechnionym i obecnie uznawanym jako standard. Na podstawie przeprowadzonych eksperymentów nad dostępnymi narzędziami komunikacji (m.in. CORBA, ICE) została wyłoniona biblioteka messip (Message–Passing over TCP/IP) [6, 2, 5]. W trakcie badań okazało się, że rozbudowane narzędzia do komunikacji w praktyce wnoszą istotny narzut czasowy na wymianę informacji oraz ich architektura (np. request broker dla CORBA) nie jest odpowiednia do tworzenia bardziej skomplikowanych schematów obsługi klientów, jak to ma miejsce np. w różnych wersjach powłok procesów VSP. Wykorzystywana biblioteka messip zapewnia funkcjonalność oryginalnych funkcji komunikacji QNX przy jednoczesnych minimalnych narzutach czasowych. Wysoką wydajność osiągnięto tutaj przez minimalizację wystąpień wywołań systemowych niezbędnych do zarządzania obsługą komunikacji TCP/IP oraz eliminacją dynamicznej alokacji pamięci, która prowadzi do niedeterministycznych czasów wykonania.

W celu zastąpienia natywnych dla systemu QNX funkcji komunikacji zostały wykorzystane ich odpowiedniki, bazujące na protokole TCP/IP (tab. 1.2.4). Funkcje te zostały zdefiniowane w plikach źródłowych w podkatalogu `src/lib/messip/` źródeł MRROC++ i są dostępne przez dołączenie pliku źródłowego `"messip/messip.h"`

Funkcja QNX	Funkcja messip
name_attach()	messip_channel_create()
name_detach()	messip_channel_delete()
name_open()	messip_channel_connect()
name_close()	messip_channel_disconnect()
MsgSend()	messip_send()
MsgReceive()	messip_receive()
MsgReply()	messip_reply()

Tabela 1.1: Odpowiedniki funkcji z QNX do międzyplatformowej komunikacji międzyprocesowej

Zarządzenie przestrzenią nazw połączeń komunikacyjnych realizowane jest przez proces menadżera nazw messip_mgr, którego swoją funkcjonalnością odpowiada procesowi serwera nazw gns z systemu QNX. Proces menadżera nazw uruchamiany jest automatycznie



Rysunek 1.1: Czasy komunikacji między procesami dla różnych mechanizmów komunikacji. Po lewej komunikacja w obrębie jednej maszyny, po prawej w obrębie LAN

przy starcie interfejsi użytkownika UI. Informacja o węźle, na którym jest uruchomiony przekazywana jest do wszystkich procesów MRROC++ jako zmienna środowiskowa `UI_HOST` i ustawiana w trakcie ich zdalnego powoływania przez mechanizm rsh. Dopiero w dalszej kolejności uruchamiany jest serwer konfiguracji, który także korzysta z opisanych mechanizmów komunikacji. Cały proces startowania został zawarty w skrypcie uruchomieniowym `bin/cfg.sh` (tab. 1.2).

Poniżej zamieszono wykresy ilustrujące porównanie czasów komunikacji między procesowej z wykorzystaniem natywnego protokołu QNET (dla maszyn pracujących pod kontrolą systemu QNX), biblioteki MESSIP oraz systemu CORBA w implementacji ACE+TAO (dla maszyn pracujących pod kontrolą systemu Linux) (rys. 1.1) [5]. Testy przeprowadzono na komputerach z procesorem klasy AMD 1.8GHz i lokalnej sieci ethernet 100mbit/sek, zaś przesyłane komunikaty miały rozmiar kilkudziesięciu bajtów (typowy dla systemu MRROC++). Wyraźny determinizm czasów komunikacji dla systemu Linux został osiągnięty przez uruchomienie procesów z wykorzystaniem szeregowania czasu rzeczywistego (ang. real time scheduler). Co znaczące - komunikacja przy zastosowaniu biblioteki MESSIP na komputerach Linux okazuje się być bardziej wydajna, niż protokół QNET w systemie czasu rzeczywistego QNX.

1.2.5 Przenośne szablony procesów MRROC++

Opisane modyfikacje i mechanizmy zostały wykorzystane do stworzenia wieloplatformowej powłoki procesu **VSP** pracującej w trybie interaktywnym (jako najczęściej wykorzystywanej) [4]. Działanie powłoki zostało przetestowane i zweryfikowane na przykładowym czujniku generującym puste dane. Stosując omówione wcześniej metody możliwe jest skompilowanie i uruchomienie procesu na systemie Linux.

```
#!/bin/sh

./messip_mgr &
MESSIP_PID=$!

export UI_HOST=‘hostname‘

sleep 1

PWD=‘pwd‘
CONFIG=‘cat ../configs/default_file.cfg‘
./configsrv ‘hostname‘ ‘dirname ${PWD}‘/ ${CONFIG} &
CONFIGSRV_PID=$!

sleep 1

echo CONFIGSRV_PID=${CONFIGSRV_PID}
echo MESSIP_PID=${MESSIP_PID}

./ui

kill ${MESSIP_PID}
kill ${CONFIGSRV_PID}
```

Tabela 1.2: Skrypt uruchomieniowy z menadżerem nazw i serwerem konfiguracji

1.3 Podsumowanie

Opisane zostały zrealizowane modyfikacje ramowej struktury programowej MRROC++ niezbędne do uruchamiania jej komponentów (procesów) na innych niż QNX systemach operacyjnych. Zbadana została efektywność implementacji, a jej skuteczność potwierdzona implementacją przykładowego procesu dla systemu Linux.

Omówione modyfikacje są kolejnym, istotnym krokiem w kierunku pełnej integracji pakietów MRROC++ oraz Player/Stage, które obecnie są wykorzystywane do sterowania manipulatorami oraz robotami mobilnymi w ramach prac laboratoryjnych i naukowo badawczych zespołu.

Rozdział 2

Eksperymenty

Bibliografia

[1] QNX Neutrino reference manual, 2006.

[2] Michi Henning. A new approach to object-oriented middleware. IEEE Internet Computing, 8(1):66–75, 2004.

[3] Sacha K. Laboratorium systemu QNX. Oficyna Wydawnicza Politechniki Warszawskiej, 1995.

[4] T. Kornuta. Szablony do przetwarzania danych sensorycznych w układach sterowania robotów. Praca inżynierska, Warszawa, 2003.

[5] D. Schmidt. Tao: a high-performance orb endsystem architecture for real-time corba.

[6] Olivier Singla. MESSIP: Message-Passing over TCP/IP.