ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1995

Seria: INFORMATYKA z.28 Nr kol. 1270

Wiesław BARCIKOWSKI

Robert KOŚLA

SYMULACJA ZDARZEŃ DYSKRETNYCH W CZASIE Z WYKORZYSTANIEM MOŻLIWOŚCI SYSTEMU PRZETWARZANIA ROZPROSZONEGO QNX

Streszczenie. W artykule przedstawiono podsystem umożliwiający realizację eksperymentów symulacyjnych w rozproszonym środowisku systemu operacyjnego QNX. Elementy służące do budowy eksperymentów symulacyjnych (procesy, kolejki, generatory liczb losowych, itp.) dostępne są w postaci pakietu SimPack stanowiącego rozszerzenie standardowej biblioteki języka C. Pakiet SimPack pozwala wykorzystać w eksperymencie symulacyjnym dostępne w systemie rozproszonym zasoby: procesory, pamięci i terminale. Dzięki temu można skrócić czas badań symulacyjnych oraz prowadzić eksperymenty z udziałem operatorów oddziałujących interakcyjnie na badany system. Istotna jest również łatwość budowania i badania modeli symulacyjnych systemów fizycznych, które z natury są rozproszone.

DISCRETE EVENT SIMULATION IN DISTRIBUTED PROCESSING SYSTEM QNX

Summary. The paper presents distributed simulation subsystem basing on QNX operating system. A special packet called SimPack containing elements to build simulation experiments (processes, queues, random generators, etc.) is implemented as an extension to standard library of C language. SimPack allows using in simulation experiments resources of distributed system: processors, memories and terminals. As a result, it is possible to shorten simulation time and to do experiments with human operators interactively working with examined system. Easiness in building and testing of simulation models of phisical systems, which are naturally distributed is an additional benefit of presented subsystem.

DIE SIMULATION DER IN DER ZEIT DISKRETEN EREIGNISSE MIT DER AUSNUTZUNG DER MOEGLICHKEITEN DER VERTEILTEN QNX-DATENVERARBEITUNGSYSTEM

Zusammenfassung. Im dem Artikel ist das Untersystem beschrieben, das die Simulationsexperimente in verteilen QNX- Betriebsystemumwelt ermöglicht. Die Bauelemente für die Simulationexperimente (Processen, Warteschlangen, usw.) sind als das SimPack-Paket erhältlich, das die Erweiterung der Standard-Bibliothek der C-Sprache ist. Das SimPack erlaubt in der Simulationsexperimente die freien Systemkomponenten dem o.a. System, wie: Processoren, Speichern und Terminale, ausnutzen. Im Rezultat, es kann mann: die Zeit der Simulationsuntersuchungen verkürzen und die Experimente mit den Operateuren die inteactiv mit dem System Arbeiten, führen. Sehr wichtig ist auch die Leichtigkeit der Bau und der Untersuchungen der Simulationsmodelle der physikalschen Systeme, die von Natur verteilte sind.

1. Wprowadzenie

Zrozumienie mechanizmów występujących w otaczającym środowisku pozwala poznać sposób funkcjonowania systemów wykorzystujących te mechanizmy i w rezultacie umożliwia poprawianie oraz dostosowanie systemów do wciąż rosnących wymagań ludzi. Do badania zachowania obiektów i systemów wykorzystuje się wiele metod. W ostatnich czasach coraz częściej stosowaną metodą badania systemów jest symulacja komputerowa. Wraz z rozwojem systemów komputerowych staje się możliwe prowadzenie coraz bardziej złożonych eksperymentów symulacyjnych przy malejących kosztach wykorzystywanego sprzętu. Dalszy wzrost możliwości systemu komputerowego wykorzystywanego do realizacji eksperymentów może być uzyskany bez potrzeby jego fizycznej rozbudowy dzięki zastosowaniu mechanizmów przetwarzania rozproszonego.

Rozproszona symulacja komputerowa polega na fizycznym rozproszeniu procesów symulacyjnych w rozproszonym systemie komputerowym. Wykorzystuje ona dostępne zasoby systemu rozproszonego: procesory, pamięci, urządzenia zewnętrzne. Najprostszym systemem rozproszonym może być lokalna sieć komputerowa, pod warunkiem że będzie zarządzana systemem operacyjnym wyposażonym w mechanizmy przetwarzania rozproszonego. W takie możliwości wyposażony jest system operacyjny QNX, w którym zaimplementowany został podsystem symulacji rozproszonej opisany w tym artykule.

2. Pojęcia i mechanizmy wykorzystywane w symulacji rozproszonej

W ostatnich czasach w modelowaniu zdarzeń dyskretnych w czasie oprócz klasycznych języków symulacyjnych takich jak: GASP, SIMSCRIPT, SIMULA czy GPSS coraz częściej wykorzystuje się konwencjonalne, sekwencyjne języki programowania takie jak Pascal lub C. Podstawową zaletą tych języków w porównaniu z typowymi językami symulacyjnymi jest ich powszechna znajomość wśród programistów oraz większa przenoszalność programów między różnymi platformami sprzętowymi. Do symulacji wykorzystywane są również języki programowania współbieżnego takie jak Ada lub Concurrent C. Są one co prawda stosowane niezbyt szeroko, ale dzięki dostarczeniu obiektu typu proces, który jest bardzo zbliżony do pojęcia procesu w symulacji, są lepiej przystosowane do symulacji zdarzeń dyskretnych w czasie. Dostosowując zatem konwencjonalny język programowania do wymagań stawianych przez symulację, należy w nim zaimplementować takie elementy jak: proces, kolejka czy zarządca procesów (scheduler). Ponieważ implementacja tych elementów w postaci nowych konstrukcji językowych jest z zasadzie niemożliwa, dokonuje się tego poprzez rozbudowę funkcji bibliotecznych. Korzysta się przy tym z mechanizmów, jakie może dostarczyć system operacyjny. Jednym z częściej ostatnio wykorzystywanych mechanizmów w symulacji zdarzeń dyskretnych w czasie jest mechanizm przesyłania komunikatów (messages). Żaden z języków symulacyjnych wymienionych uprzednio nie dostarcza tego mechanizmu jako prostej konstrukcji językowej. W celu dostarczenia tych mechanizmów zaprojektowano pakiet symulacyjny SAMOA dla języka ADA. Innym rozwiązaniem jest środowisko symulacyjne zaprojektowane przez Kaubischa i Hoare'a dla języka CSP.

Podstawowymi konstrukcjami wykorzystywanymi w symulacji opartej na wymianie komunikatów są konstrukcje umożliwiające:

1) tworzenie i kasowanie procesu (create i terminate);

2) wysłanie komunikatu do procesu (send);

3) oczekiwanie na komunikat przez określony czas (wait).

Eksperymenty symulacyjne oparte na wymianie komunikatów mogą być zapisane w sposób bardziej naturalny. Program wykorzystujący przesyłanie komunikatów, składający się z wzajemnie na siebie oddziałujących procesów współbieżnych, może wykorzystać

przetwarzanie rozproszone, tzn. równolegle wykonywać procesy symulacyjne w wielu komputerach systemu rozproszonego.

Oddziaływanie procesów nazywane jest zdarzeniem (event), a chwile czasowe, w których występują zdarzenia, nazywane są chwilami zdarzeń (event times). Każdy fizyczny proces występujący w systemie symulowany jest przez proces logiczny.

Jako przykład można rozważyć symulację gabinetu lekarskiego w celu określenia podziału czasu oczekiwania przez pacjentów na przyjęcie. W rzeczywistym systemie pacjenci wchodzą do poczekalni i oczekują w kolejce do rejestracji. Rejestratorka odnotowuje przybycie pacjenta, który następnie przyjmowany jest przez lekarza. W problemie symulacyjnym lekarz, recepcjonistka i pacjenci są symulowani przez procesy logiczne. Przykładami zdarzeń w systemie są: wejście pacjenta do poczekalni i odnotowanie przez rejestratorkę oraz zakończenie wizyty u lekarza. W dalszych rozważaniach termin pacjent będzie oznaczał proces logiczny symulujący fizyczny *proces pacjenta*.

W symulacji opartej na przesyłaniu komunikatów zdarzenie jest reprezentowane przez przesłanie komunikatu. W rozproszonym systemie symulacyjnym musi zatem istnieć możliwość przesyłania komunikatów między procesami znajdującymi się w różnych węzłach systemu. Szczególnie istotne jest ponadto, aby mechanizm komunikacji lokalnej i zdalnej był identyczny, co pozwoli na swobodne przemieszczanie procesów w systemie i odciąży projektanta eksperymentu symulacyjnego od konieczności uwzględniania struktury sprzętowej systemu rozproszonego (ilość i numery węzłów). Jest to bardzo ważne, gdyż pozwala prowadzić raz zaprojektowany eksperyment symulacyjny w systemach rozproszonych o zmieniającej się strukturze (np. wskutek awarii węzła). System operacyjny QNX został wybrany jako platforma systemowa dla podsystemu symulacji rozproszonej właśnie ze względu (między innymi) na elastyczność i przezroczystość (w sensie przekraczania granic między niezależnymi węzłami) zaimplementowanej w nim komunikacji międzyprocesowej.

3. Mechanizmy wykorzystywane w symulacji rozproszonej dostarczane przez system QNX

3.1. Komunikacja międzyprocesowa

Jądro systemu QNX[3] dostarcza trzy mechanizmy komunikacji pomiędzy procesami:

- *komunikaty* (messages) - podstawowa forma komunikacji między procesami w systemie QNX, umożliwiająca synchroniczną komunikację między procesami współbieżnymi (proces wysyłający komunikat blokuje się w oczekiwaniu na odpowiedź),
- *skrzynki kontaktowe* (proxies) - specjalna forma komunikacji nieblokującej (proces inicjujący komunikat natychmiast wznawia realizację),
- *sygnały* (signals) - tradycyjna forma komunikacji między procesami, wykorzystywana do komunikacji asynchronicznej.

Komunikat w systemie QNX jest pakietem bitów przesyłanym od jednego procesu do drugiego, przy czym z punktu widzenia systemu zawartość komunikatu nie ma specjalnego znaczenia. Dane przesyłane w komunikacie mają znaczenie tylko dla nadawcy i odbiorcy komunikatu.

Do komunikacji między dwoma współdziałającymi programami wykorzystuje się następujące funkcje dostępne w języku C:

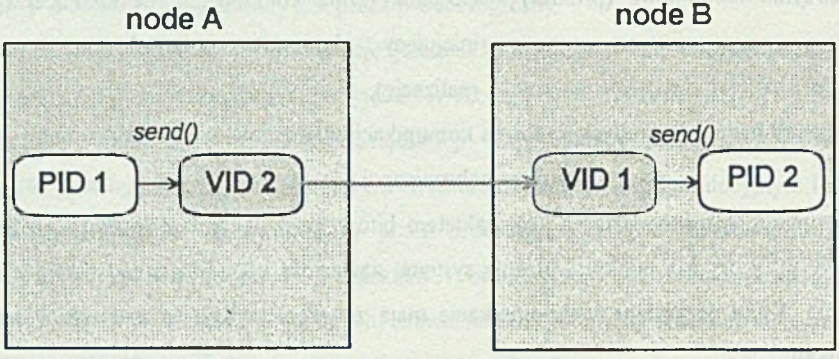
- *Send()* - w celu wysłania komunikatu;
- *Receive()* - w celu odbioru komunikatu;
- *Reply()* - w celu odpowiedzi procesowi, który wysłał wiadomość.

Funkcje te mogą być używane zarówno do komunikacji lokalnej, jak i zdalnej, co oznacza, że procesy komunikują się zawsze w taki sam sposób bez względu na to, gdzie się znajdują: w jednym węźle czy w dwóch niezależnych węzłach odległych fizycznie od siebie.

3.2. Połączenia wirtualne między procesami

Proces w systemie QNX może porozumiewać się z procesem uruchomionym na innym komputerze w sposób taki, jakby proces ten znajdował się na tym samym komputerze. Ten stopień przezroczystości osiągnięty został dzięki tzw. połączeniom wirtualnym (virtual circuits) dostarczanym przez system QNX. Proces nadawcy odpowiada za utworzenie połączenia wirtualnego między nim a procesem, z którym zamierza się komunikować. Połączenie wirtualne tworzone jest przez proces nadawcy za pomocą funkcji *qnx_vc_attach()*. Na każdym końcu połączenia wirtualnego tworzony jest proces wirtualny, który jest reprezentantem procesu rzeczywistego znajdującego się na drugim końcu połączenia. Procesy rzeczywiste komunikują się ze sobą za pośrednictwem procesów wirtualnych. Za odpowiednie

przesyłanie komunikatów między procesami wirtualnymi odpowiedzialny jest system operacyjny QNX. Na rys. 1 przedstawiony jest przykład komunikacji między dwoma procesami PID1 i PID2 znajdującymi się w dwóch odległych węzłach. Pośrednikami w przesyłaniu komunikatów są procesy wirtualne o identyfikatorach: VID1 i VID2.



Rys.1. Połączenie wirtualne w systemie operacyjnym QNX
Fig.1. Virtual circuit in QNX operating system

Procesy nie muszą korzystać z funkcji *qnx_vc_attach()*, która w sposób jawny tworzy połączenie wirtualne. W systemie QNX dostępne są funkcje (np. lokalizacja zarejestrowanego procesu *qnx_name_locate()*), w wyniku których system automatycznie tworzy połączenie wirtualne, jeśli stwierdzi, że poszukiwany obiekt (drugi proces) znajduje się w innym węźle. Taki mechanizm pozwala zapomnieć programiście o fizycznej strukturze systemu rozproszonego (sieci lokalnej), co okazało się niezwykle istotne w zaimplementowanym podsystemie symulacji rozproszonej.

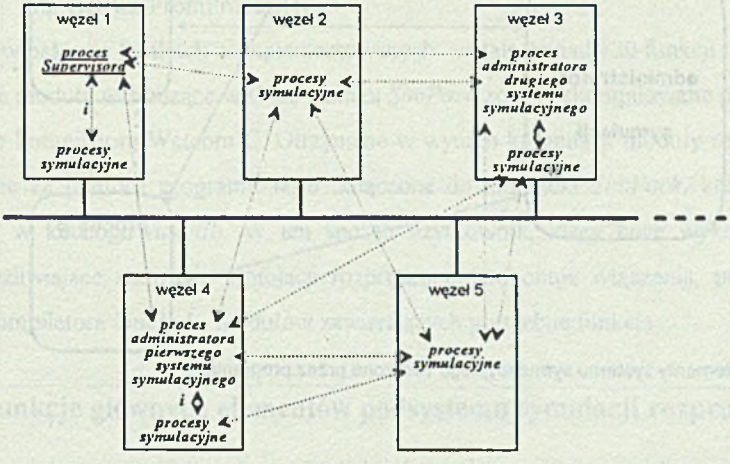
4. Organizacja symulacji rozproszonej w systemie QNX - pakiet *SimPack*

Podsystem umożliwiający realizację symulacji rozproszonej w systemie QNX zaimplementowany został w języku C w postaci pakietu procedur nazwanego *SimPack*, stanowiącego rozszerzenie standardowych bibliotek języka. Procedury dostarczane w pakiecie

SimPack umożliwiają zbudowanie modelu symulacyjnego opartego na modelu klient-usługodawca, a także operowanie na jego procesach logicznych.

Tworząc model symulacyjny systemu fizycznego, należy wydzielić procesy logiczne, odpowiadające procesom fizycznym, a także zależności występujące między tymi procesami oraz zdarzenia, będące skutkiem oddziaływania na siebie procesów.

W systemie QNX komunikacja między procesami odbywa się za pomocą przesyłania komunikatów. Każdemu zdarzeniu w systemie symulacyjnym musi odpowiadać wysłanie komunikatu określającego w sposób jednoznaczny rodzaj zdarzenia. Poszczególne procesy biorące udział w symulacji mogą być rozproszone w sieci lokalnej w celu lepszego wykorzystania możliwości przetwarzania komputerów sieci. Rozproszenie to realizowane jest przez proces *Supervisora* (główny proces sterujący eksperymentu symulacyjnego) na etapie tworzenia elementów składowych eksperymentu symulacyjnego. Tworzenie procesów symulacyjnych jest oparte na mechanizmach tworzenia procesów w systemie QNX. Na obecnym etapie rozwoju *pakietu SimPack* procesy rozpraszane mogą być w fizycznych węzłach sieci metodą cykliczną, tzn. każdy kolejny proces może być uruchomiony w kolejnym aktywnym węźle sieci. Docelowo wykorzystany zostanie system monitorowania obciążenia poszczególnych węzłów sieci. Na rys. 2 przedstawiony jest przykład rozproszenia procesów składających się na dwa eksperymenty symulacyjne realizowane jednocześnie w sieci QNX.

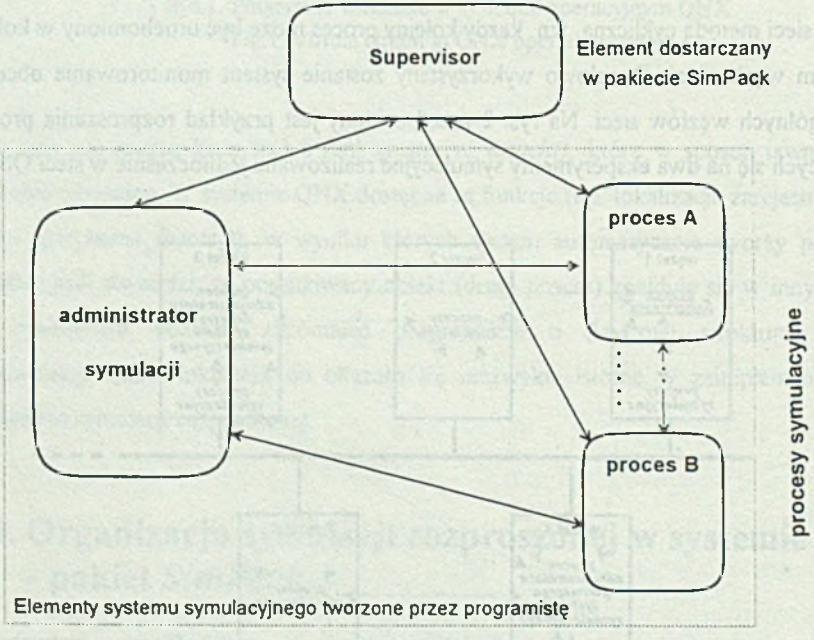


Rys. 2. Przykład fizycznego rozproszenia procesów w eksperymencie symulacyjnym

Fig.2. Example of phisical distribution of processes in simulation experiment

4.1. Struktura podsystemu symulacji

W systemie symulacyjnym wykorzystywany jest model *klient-usługodawca* (client-server). Działanie systemu symulacyjnego inicjowane jest przez uruchomienie procesu *Supervisor*. Proces ten spełnia rolę *usługodawcy* dla wszystkich klientów symulacji, którymi są *proces administratora symulacji* i *procesy symulacyjne*. Programista tworzący model symulacyjny systemu fizycznego dokonuje implementacji procesów w nim występujących (w postaci oddzielnych modułów dla każdego typu procesu), po czym tworzy moduł *administratora symulacji*. Tworzony przez programistę system symulacyjny składał się więc będzie z opisu *administratora symulacji* i opisów procesów symulacyjnych. Implementacje procesu administratora symulacji i procesów symulacyjnych będą zawierały wywołania procedur dostarczanych w modułach bibliotecznych pakietu. Schemat wymiany komunikatów między elementami systemu symulacyjnego tworzonymi przez programistę a *Supervisorem*, dostarczanym w pakiecie *SimPack*, przedstawiony jest na rys.3.



Rys. 3. Powiązania między elementami podsystemu symulacyjnego

Fig. 3. Interconnections of simulation subsystem elements

W skład pakietu realizującego podsystem symulacji wchodzą następujące moduły:

- **supervisor** - jest to moduł tworzący usługodawcę dla systemu symulacyjnego, zawierający funkcje obsługujące komunikaty odbierane od procesu administratora symulacji (np: BuildProcess(), InsertProcessToQueue(), HoldForTime());
- **curprochandler** - jest to moduł zawierający procedury umożliwiające operacje na aktywnym procesie symulacyjnym (np. ChangePriority(), TimeOfWait());
- **processh** - jest to moduł zawierający procedury umożliwiające operacje na pozostałych procesach symulacyjnych, znajdujących się w systemie (np. CreateProcess(), ActivateProcess());
- **queuchandler** - jest to moduł zawierający procedury umożliwiające operacje na kolejkach procesów symulacyjnych (np. CreateQueue(), ProcessFirstInQueue());
- **random** - jest to moduł zawierający procedury generujące liczby losowe o różnych rozkładach prawdopodobieństwa (np. RndNorm(), RndPois());
- **simpleservice** - jest to moduł zawierający procedury umożliwiające operacje w gniazdach obsługi (np. CreateServiceNest(), GetToService());
- **userinfo** - jest to moduł zawierający procedury umożliwiające operacje na zmiennych występujących w procesach symulacyjnych (np.GetRealFromProcess())

W sumie w pakiecie SimPack zaimplementowanych zostało ponad 120 funkcji.

Wszystkie moduły wchodzące w skład pakietu *SimPack* zostały skompilowane przy użyciu 32-bitowego kompilatora Watcom C. Otrzymane w wyniku kompilacji moduły relokowalne były następnie za pomocą programu *Wlib* dołączone do biblioteki *SimPack*, która została umieszczona w katalogu */usr/lib*. W ten sposób użytkownik, który chce wykorzystywać funkcje umożliwiające realizację symulacji rozproszonej, dokonuje włączenia, przy użyciu dyrektywy kompilatora *#include*, modułów zawierających potrzebne funkcje.

4.2. Funkcje głównych elementów podsystemu symulacji rozproszonej

Proces *Supervisora* zostaje uruchomiony (jako drugoplanowy) na początku pracy systemu symulacyjnego. Jego zadaniem jest obsługa komunikatów odbieranych od procesu administratora eksperymentu symulacyjnego. Administrator może żądać:

- utworzenia procesów symulacyjnych,
- utworzenia kolejki procesów symulacyjnych;
- wstawienia procesu na listę procesów aktywnych;
- usunięcia procesu z listy procesów aktywnych;
- zmiany priorytetu procesu symulacyjnego;
- zmiany czasu aktywacji procesu;
- informowania o powstaniu błędu symulacji;
- usunięcia procesu z systemu symulacyjnego;
- usunięcia kolejki procesów symulacyjnych;
- zakończenia pracy systemu symulacyjnego.

Moduł Supervisora zawiera procedury, które realizują wyżej wymienione podstawowe operacje. Zwalnia to programistę od każdorazowego implementowania w programach tych operacji oraz wpływa na zwiększenie niezawodności.

Administrator symulacji (klient) tworzony jest przez programistę. Przy wykorzystaniu procesu *Supervisora* i procedur zawartych w modułach pakietu *SimPack* spełnia następujące funkcje:

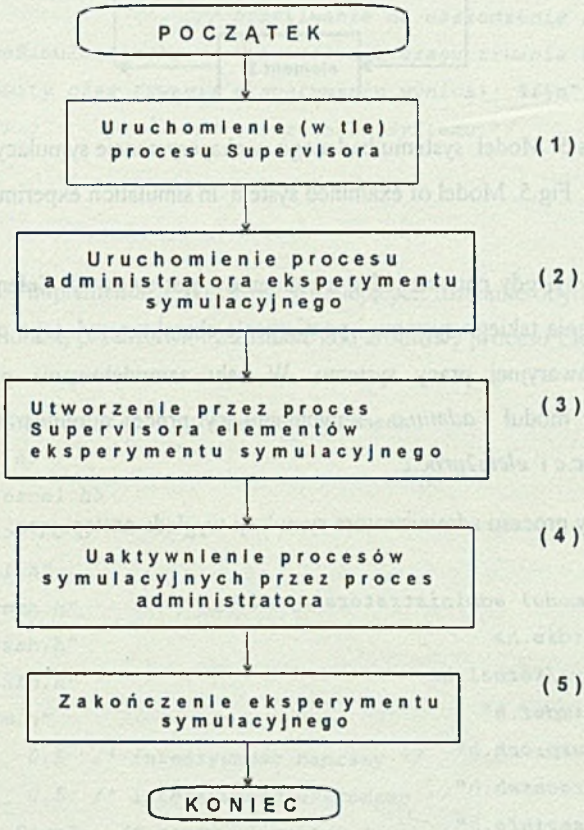
- definiuje zmienne globalne i stałe używane w systemie symulacyjnym;
- wywołuje funkcje (umieszczone w modułach bibliotecznych pakietu *SimPack*) tworzące kolejki zadań i procesy symulacyjne;
- kontroluje przebieg symulacji;
- sporządza raport z przebiegu symulacji, kończy działanie systemu symulacyjnego.

4.3. Wykorzystanie biblioteki funkcji umożliwiających symulację rozproszoną w systemie QNX

Przebieg eksperymentu symulacyjnego opartego na wykorzystaniu pakietu *SimPack* przedstawia schemat blokowy przestawiony na rys.4.

Po zakończeniu eksperymentu możliwe jest jego ponowne rozpoczęcie, gdyż proces *Supervisora* jest w dalszym ciągu w systemie i oczekuje na komunikaty od procesów biorących udział w symulacji, a funkcja *DeleteSystem()* dokonuje usunięcia z systemu tylko procesów symulacyjnych oraz ustawia czas symulacji na 0:0. Rozproszenie procesów symulacyjnych realizowane jest w trakcie ich tworzenia za pomocą funkcji *BuildProcess*. Na

obecnym etapie rozwoju pakietu *SimPack* procesy symulacyjne umieszczane mogą być w kolejnych aktywnych węzłach sieci. Fizyczne położenie procesów symulacyjnych jest nieistotne z punktu widzenia użytkownika systemu symulacyjnego. Wszelkie operacje związane z rozproszeniem systemu wykonywane są przez proces *Supervisora* z wykorzystaniem mechanizmu przesyłania komunikatów i połączeń wirtualnych między procesami znajdującymi się w różnych fizycznych węzłach sieci. Można w ten sposób wykorzystać wszystkie dostępne w systemie rozproszonym (sieci lokalnej) procesory i zasoby pamięci, co jest bardzo istotne w przypadku badania bardzo dużych modeli symulacyjnych.

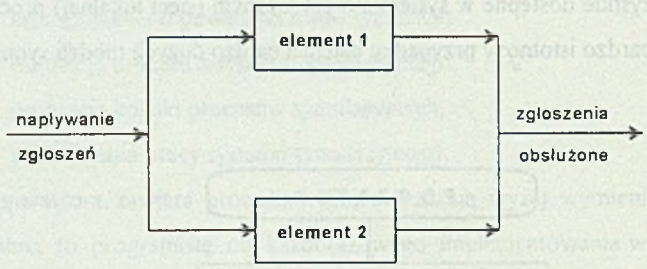


Rys. 4. Realizacja eksperymentu symulacyjnego w środowisku rozproszonego systemu QNX

Fig. 4. Realization of simulation experiment in QNX distributed environment

5. Przykład eksperymentu symulacyjnego z wykorzystaniem procedur pakietu *SimPack*

Celem zilustrowania możliwości pakietu *SimPack* zaprezentowany zostanie prosty eksperyment symulacyjny. Eksperyment polega na zasymulowaniu funkcjonowania systemu składającego się z dwóch elementów wykonawczych, które ulegają uszkodzeniom (rys.5).



Rys.5. Model systemu badanego w eksperymencie symulacyjnym

Fig.5. Model of examined system in simulation experiment

System jest sprawny, gdy chociaż jeden z elementów jest sprawny. Celem eksperymentu jest zbadanie zachowania takiego systemu i pomierzenie charakterystyk jego pracy, np. ile wynosi średni czas bezawaryjnej pracy systemu. W celu zamodelowania pracy tego systemu stworzony został moduł *admin.c* implementujący proces administratora symulacji oraz moduły *elem1proc.c* i *elem2proc.c*.

Kod źródłowy procesu administratora symulacji wygląda następująco:

```
// Admin - modul administratora symulacji
#include <stdio.h>
#include <sys/kernel.h>
#include "simdef.h"
#include "curproch.h"
#include "processh.h"
#include "userinfo.h"
#include "random.h"
Process  Elem1, Elem2;  // nazwy elementow
unsigned i;
double simtime; // calkowity czas trwania eksperymentu
```

```
main()
{
  for (i = 1; i<=5 ;i++) {  // 5-krotne powtorzenie eksperymentu
  CreateMainProcess();
  CreateProcess(&Elem1, "//1/home/sym/elem1proc") ;
  CreateProcess(&Elem2, "//1/home/sym/elem2proc") ;// utworzenie systemu

  SetProcessInProcess(1, Elem1, Elem2); // ustawienie powiazan miedzy
  SetProcessInProcess(1, Elem2, Elem1); // procesami symulacyjnymi
  SetCardInProcess(1, Elem1, 1);   // ustawienie sprawnosci elementow
  SetCardInProcess(1, Elem2, 1);
  ReactivateProcess(Elem1) ;      // wlasciwe uruchomienie systemu
  Passivate() ;                  // oczekiwanie na uszkodzenie systemu
  simtime = TimeSimulation(); // wyswietlenie czasu trwania eksperymentu
  printf("Calkowity czas trwania eksperymentu wyniosl: %f\n",simtime) ;
  DeleteSystem() ;              /* zniszczenie systemu */
  }
}
```

Moduły zawierające implementacje procesów symulujących działanie obydwu elementów są prawie identyczne. Poniżej przedstawiony zostanie kod źródłowy procesu Elem1Proc:

```
// Elem1Proc - implementacja pierwszego elementu
#include <stdio.h>
#include <sys/kernel.h>
#include <i86.h>
#include "simdef.h"
#include "curproch.h"
#include "processh.h"
#include "userinfo.h"
#include "random.h"
#define MI        0.5  /* intensywnosc naprawy */
#define LAMBDA    0.5  /* intensywnosc uszkodzen */
Process  Elem1, Elem2;  /* nazwy elementow */
unsigned i;
float zmgen1 ,zmgen2;
```

```
main()
{
  InitProcess();
  Elem1 = CurrentProcess();
  Elem2 = GetProcessFromProcess(1, Elem1);
  for(;;){
    zmgen1 = RndWyk(LAMBDA); // losowanie czasu pracy Elementu
    HoldForTime(zmgen1) ;    // wstrzymanie przez czas pracy
    SetCardInProcess(1, Elem1, 0) ; // ustawienie uszkodzenia elementu
    if ( GetCardFromProcess(1, Elem2)){ // sprawdzenie drugiego elementu
      ReactivateProcess(Elem2) ;      // uruchomienie drugiego
      zmgen2 = RndWyk(MI);          // losowanie czasu naprawy pierwszego
      HoldForTime(zmgen2) ;         // wstrzymanie przez czas naprawy
      SetCardInProcess(1, Elem1, 1) ; // ustawienie sprawnosci elementu
      Passivate() ; }               // zawieszenie w oczekiwaniu na
                                    //awarie drugiego elementu
    else {
      ReactivateProcess(MainProcess()) ;
      Passivate() ;         // koniec eksperymentu - system uszkodzony
    }
  }
}
```

Elementy, których zachowanie jest badane w eksperymencie symulacyjnym, działają w sposób cykliczny:

- pracują do momentu uszkodzenia przez czas wylosowany za pomocą generatora liczb losowych o rozkładzie wykładniczym z parametrem LAMBDA,
- są naprawiane przez czas wylosowany za pomocą generatora liczb losowych o rozkładzie wykładniczym z parametrem MI.

System jest sprawny, dopóki co najmniej jeden element jest sprawny.

6. Podsumowanie

W artykule zaprezentowany został podsystem umożliwiający realizację eksperymentów symulacyjnych w rozproszonym środowisku systemu operacyjnego QNX. Elementy

wykorzystywane do budowy eksperymentów symulacyjnych (procesy, kolejki, zarządca procesów) zaimplementowane zostały w języku C w postaci pakietu *SimPack* będącego rozszerzeniem standardowych bibliotek języka. W pakiecie dostępnych jest ponad 120 funkcji umożliwiających budowanie złożonych eksperymentów symulacyjnych. W skład pakietu wchodzi również moduł *Supervisor* spełniający funkcję procesu zarządzającego eksperymentem symulacyjnym. Proces ten umożliwia wykorzystanie przetwarzania rozproszonego w sieci lokalnej pracującej pod kontrolą systemu operacyjnego QNX. Aktualnie procesy symulacyjne umieszczane mogą być w kolejnych aktywnych węzłach sieci. Wszelkie operacje związane z rozproszeniem systemu wykonywane są przez proces *Supervisor* z wykorzystaniem mechanizmu przesyłania komunikatów i połączeń wirtualnych między procesami znajdującymi się w różnych fizycznych węzłach sieci. Fizyczne położenie procesów symulacyjnych jest nieistotne z punktu widzenia użytkownika systemu symulacyjnego. Można w ten sposób wykorzystać wszystkie dostępne w systemie rozproszonym (sieci lokalnej) procesory i zasoby pamięci, co jest bardzo istotne w przypadku badania bardzo dużych modeli symulacyjnych. Istotna jest również łatwość budowania i badania modeli symulacyjnych systemów fizycznych, które z natury są rozproszone, np. globalne systemy obsługi klienta, rozproszone systemy zarządzania i dowodzenia. Istnieje również możliwość symulowania systemów, w których wielu użytkowników operuje na terminalach prezentujących symulowaną sytuację, współpracując interaktywnie z systemem. Pozwala to na przyspieszenie procesu symulacji oraz na maksymalne przybliżenie zachowania systemu rzeczywistego, w którym istotne są naturalne zachowania i reakcje ludzi.

Rozwój pakietu *SimPack* może zmierzać w kierunku zapewnienia możliwości realizacji wielu eksperymentów symulacyjnych jednocześnie, przy wykorzystaniu jednego procesu usługodawcy symulacji - *Supervisora*. Realizacja wielu eksperymentów jednocześnie pozwoli na modelowanie złożonych systemów przy pełniejszym wykorzystaniu systemu komputerowego i zmniejszeniu czasu potrzebnego na przeprowadzenie badań.

Rozbudowa systemu może zmierzać także w kierunku umożliwienia otrzymywania innych charakterystyk potrzebnych do badania określonych klas rzeczywistych systemów oraz graficznej prezentacji otrzymywanych wyników.

Procesy symulacyjne rozpraszane są obecnie na zlokalizowane uprzednio aktywne węzły sieci, przy czym nie jest uwzględniane ich aktualne obciążenie. Może to prowadzić niekiedy do nieoptymalnych przydziałów procesów do komputerów. Kolejnym więc krokiem powinno

być zastosowanie mechanizmów monitorowania stanu węzłów (mocy procesora i obciążenia zadaniami) w celu wyboru do zdalnej realizacji węzłów najlepszych.

Podsumowując, można stwierdzić, że rozbudowa języka C o możliwości realizacji eksperymentów symulacyjnych oraz wykorzystanie systemu QNX pozwolą na efektywne prowadzenie badań bez potrzeby wykorzystywania w tym celu drogich i złożonych systemów komputerowych.

LITERATURA

[1] Bagrodia R., Chandy M., Misra J.: A Message-Based Approach to Discrete-Event Simulation, IEEE Transactions of Software Engineering, vol. 13, NO. 6, June 1987.

[2] Kolnick F.: The QNX Operating System - Programming with Messages in a Distributed Environment, Basis Computer Systems Inc., April 1989.

[3] QNX 4 Operating System - System Architecture, QNX Software Systems Ltd., July 1993.

[4] Tyszer J.: Symulacja cyfrowa, WNT, Warszawa 1990.

Recenzent: Dr inż. Ryszard Winiarczyk

Wpłynęło do Redakcji 17 listopada 1994 r.

Abstract

The aim of the work discribed in this paper is to develope subsystem which allows to realize discrete event simulation experiments in distributed environment. A special software packet called *Simpack* containing elements to build simulation experiments (processes, queues, random generators, etc.) is implemented in C language as an extension to standard library. Proposed subsystem can be used in QNX operating system environment. QNX has built in extensions which allow to treat it as distributed processing system. The principal mechanism of QNX i.e. *message passing* is the basis of internal structure of developed simulation subsystem. This subsystem consists of many processes which exchange messages between themselves. Simulation events are implemented as messages. Message passing in QNX is flexible and

transparent across the whole network. Processes can freely communicate across the LAN (Fig.1) and the mechanism of remote communication is still the same as for local. This feature allows that simulation processes can be distributed in the network. The main process of simulation subsystem called *Supervisor* can send processes to remote execution to any active node in distributed system (Fig. 2). Apart from Supervisor (which is constant process of subsystem) developer of simulation experiment have to create its main administrative process and proper number of simulation processes which will simulate behavior of physical processes of real system (Fig.3). Building the simulation experiment is quite simple when programmer can use *SimPack* functions (over 120). He should apply to rules of developing and running simulation experiments as outlined in Fig.4. There is an example of simple two-element working system (Fig.5) in chapter 5 explaining using of *SimPack* functions.

At the present the Supervisor can send processes to any active node in the network in circular fashion. This simple algorithm is not good as optimal decisions are concerned. The future work should focus on dynamic monitoring of computers and load balancing in the network. Other efforts should give solution to realize multiple different experiments at the same time. It helps to shorten simulation time and to better use of distributed system resources.

As one can see adaptation of regular C language to discrete event simulation plus efficient operating system with possibilities of distributed processing like QNX can help runninig even large experiments without expensive computer systems.