STUDIA INFORMATICA 2000
Volume 21 Number 3 (41)

Henryk ZYGMUNT, Grzegorz WRÓBEL, Paweł KWASNOWSKI,
Grzegorz HAYDUK, Marcin JACHIMSKI
Akademia Górniczo-Hutnicza, Katedra Automatyki Napędu i Urządzeń Przemysłowych

PROBLEMY REZERWOWANIA DANYCH W ROZPROSZONYM SYSTEMIE STEROWANIA ZBUDOWANYM NA BAZIE SYSTEMU OPERACYJNEGO QNX

Streszczenie. Przedmiotem niniejszego referatu jest analiza możliwych do występowania przerw i zakłóceń w ciągłości rejestracji gromadzonych danych oraz wskazanie sposobu na zabezpieczenie przed utratą całości lub części zbieranych informacji w rozproszonym systemie sterowania, wykorzystującym system operacyjny czasu rzeczywistego QNX. Metodą taką okazało się zastosowanie dynamicznej rozproszonej bazy danych czasu rzeczywistego oraz równoległych baz danych. Problematyka ta została zilustrowana przykładami zaczerpniętymi z doświadczeń we wdrażaniu systemów sterowania procesami technologicznymi na różnych obiektach przemysłowych, a w szczególności na wielkich piecach nr 5 i nr 3 w Hucie im. T. Sendzimira SA w Krakowie.

PROBLEMS OF THE DATA PROTECTION IN THE DISTRIBUTED CONTROL SYSTEM WORKING UNDER QNX REAL TIME OPERATING SYSTEM

Summary. The paper presents analysis of the possible interruptions and disturbances of the acquired data continuity and a method of protection against a loss, in whole or in part, of information acquired in the distributed control system, employing the real-time operating system QNX. The method consists in application of a dynamic, distributed, real-time data-base and parallel data-bases. The problems of the issue have been illustrated with examples from the authors' experience gained during implementation, start-up and commissioning of the technological process control systems in several industrial plants, in particular the blast furnaces No 5 and No 3 at the T. Sendzimir Steelworks SA in Kraków.

1. Wstęp

W licznych aplikacjach systemów sterowania procesami technologicznymi opracowanych w KANiUP AGH w Krakowie, a wykonywanych przez ZDANiA Sp. z o.o., oprócz sterowania logicznego i budowy cyfrowych regulatorów wielkości technologicznych, takich jak: temperatury, ciśnienia oraz przepływy gazów i cieczy, istnieje potrzeba rejestrowania i gromadzenia szeregu danych o przebiegu procesów technologicznych w tzw. systemie informacyjnym. System ten stanowi relacyjną (a niekiedy rozproszoną) bazę danych, z której można następnie czerpać potrzebne informacje w wybranych grupach oraz przedziałach czasowych celem ich wizualizowania lub sporządzania odpowiednich raportów.

Przypadki przerw w pracy bazy danych, uszkodzenia zbiorów bazowych (utrata spójności bazy) oraz ewentualne uszkodzenia komputera lub dysku twardego, na którym dane te są gromadzone i przechowywane, skłoniły autorów do poszukiwania prostej metody zapewnienia ciągłości w rejestrowaniu i przechowywaniu danych o procesach technologicznych w dużych obiektach przemysłowych, m. in. takich jak: wielkie piece, piece obrotowe do wypalania klinkieru w cementowniach, czy piece szybowe do wypalania wapna w zakładach przemysłu wapienniczego. Metodą taką okazało się zastosowanie dynamicznej, rozproszonej bazy danych pracującej w układzie równoległym w wielozadaniowym systemie operacyjnym czasu rzeczywistego QNX.

Za najdogodniejsze rozwiązanie uznano wytworzenie odpowiedniego oprogramowania w modelu klient - serwer, zawierającego lokalne bazy danych (tzw. **serwery podrzędne**) oraz proces nadzorujący (tzw. **serwer nadrzędny**).

2. Architektura systemu operacyjnego czasu rzeczywistego QNX

Uzasadnienie wyboru systemu QNX jako systemu operacyjnego czasu rzeczywistego do sterowania procesami technologicznymi oraz jego zalety zostały wyczerpująco opisane w pracy [1]. Cechą charakterystyczną wyróżniającą omawiany system operacyjny od innych systemów z rodziny UNIX jest możliwość zdeterminowania czasu reakcji na zdarzenia występujące w systemie, określająca go jako restrykcyjny system czasu rzeczywistego (hard real time system).

2.1. Sposoby komunikacji między procesami w systemie QNX

System QNX udostępnia trzy sposoby komunikacji między procesami: sygnały informujące o zdarzeniach (**signal**), synchroniczny przekaz wiadomości (**message**) oraz

asynchroniczny przekaz depozytów (**proxy**), zwanych też posłańcami. Mechanizmy przekazywania wiadomości i depozytów mogą być wykorzystane do komunikacji procesów wykonywanych w różnych węzłach sieci.

Sygnały - mechanizm sygnałów stanowi podstawowy środek informowania procesów o błędach i nienormalnych stanach powstających podczas działania programów. Przypominają one przerwania programowe, generowane przez programy jądra systemu, procesy systemowe lub procesy aplikacji w chwili wystąpienia sytuacji nienormalnej.

Wiadomości - jest to podstawowy sposób komunikacji między procesami, który opiera się na wymianie pary informacji: wiadomość - odpowiedź. Poszczególne etapy wymiany wiadomości są realizowane przez następujące funkcje systemowe:

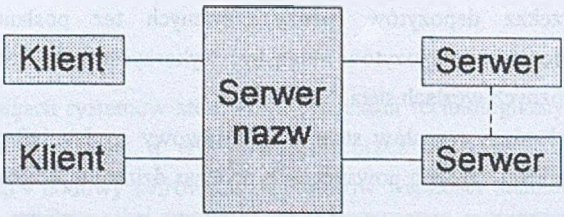
- *Send(T2,M,Mr)* - umożliwia wysłanie wiadomości M do procesu T2, powodując zawieszenie procesu nadającego (Send_Blocked) do czasu otrzymania odpowiedzi Mr.
- *Receive(T1,M)* - umożliwia odbiór wiadomości M od procesu T1, powodując zawieszenie procesu odbierającego (stan Receive_Blocked) do chwili nadejścia wiadomości.
- *Reply(T1,Mr)* - powoduje wznowienie procesu T1 i przekazanie mu odpowiedzi Mr.
- **Depozyty** - jest to podstawowe narzędzie komunikacji nieblokującej w systemie QNX. Depozyt (**proxy**) jest to pewien proces przypisany do procesu właściciela i przechowujący jedną ustaloną wiadomość. Jego zadaniem jest wysłanie przechowywanej wiadomości do procesu właściciela poprzez wywołanie funkcji pobudzenia *Trigger*. Właściciel depozytu może odebrać jedną wiadomość, wywołując w dowolnej chwili funkcję *Receive* adresowaną do swojego depozytu. Bardzo istotną cechą depozytów jest to, że proces pobudzający nigdy nie przechodzi w stan zablokowania

3. Komunikacja i architektura typu klient-serwer

Określenie klient-serwer jest związane z relacją między systemami lub między procesami. W relacji tej *klient* jest systemem, który zleca wykonanie pewnej usługi systemowi zwanemu *serwerem*.

3.1. Ulepszona technologia klient-serwer

Ulepszona technologia klient-serwer polega na tym, że pomiędzy klientami i serwerami występuje dodatkowy ***serwer nazw*** (rys. 1). Dla technologii tej spotyka się też nazwę angielską: **Client-Dispatcher-Server** [3].



Rys. 1. Przykład ulepszonej technologii klient-serwer
Fig. 1. Example of a Client-Dispatcher-Server technology

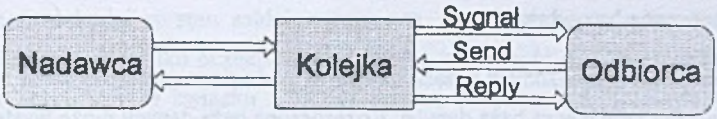
W ulepszonej architekturze klient-serwer klienci zgłaszają żądania usługi, podając jej **nazwę**. System przesyła te żądania do wykonania. Po wykonaniu wyniki są przesyłane do klienta. Ani klient, ani serwer nie muszą znać swojej lokalizacji. Ulepszona technologia klient-serwer jest dostępna za pośrednictwem specjalnego oprogramowania serwerów baz danych i serwerów plików.

3.2. Sposoby komunikacji w modelu klient-serwer

Wyróżniamy dwa sposoby komunikacji w modelu klient-serwer: 1) działanie na zasadzie blokowania któregoś z procesów (tzw. *synchroniczna transmisja danych*) oraz 2) działanie bez blokowania (tzw. *asynchroniczna transmisja danych*).

Cechą synchronicznej transmisji danych jest zablokowanie aplikacji klienta w momencie wydania komendy *Send* oraz zablokowanie aplikacji serwera w przypadku wydania komendy *Receive*. Wadą tego rodzaju transmisji jest możliwość zablokowania nadawcy na zawsze, w przypadku gdy serwer nie wyśle odpowiedzi. Wymaga to wprowadzenia limitu czasowego na oczekiwanie (timeout), który powoduje zakończenie operacji *Send* i sygnalizację błędu.

W przypadku działania bez blokowania, zwanego *asynchroniczną transmisją danych*, klient wysyłający wiadomości może wykonywać kolejne działania, przewidziane programem, równolegle z operacją transmisji komunikatu - realizowaną przez jądro. W transmisji tej wykorzystywany jest mechanizm zwany *kolejką* (rys. 2). Jest to usługa, która tymczasowo przechowuje dane przeznaczone dla procesu odbiorcy aż do chwili, gdy będzie on w stanie je przyjąć. Ważną cechą tego sposobu transmisji jest to, że *kolejka* nigdy nie zostanie zablokowana w trakcie przekazywania wiadomości ani podczas oczekiwania na odpowiedź od odbiorcy - dzięki temu, że *kolejka* nigdy nie inicjuje transmisji danych. *Kolejka* poprzez wysłanie sygnału do odbiorcy informuje go, że jest dla niego dostępna wiadomość. Odbiorca po otrzymaniu sygnału wysyła do *kolejki* polecenie przekazania mu oczekującej wiadomości. Po otrzymaniu polecenia *kolejka* odsyła w odpowiedzi do odbiorcy żądaną wiadomość.



Rys. 2. Asynchroniczna transmisja danych
Fig. 2. Asynchronous data transmision

Podstawowe wady asynchronicznej transmisji danych przy użyciu kolejki to:

- Nadawca i odbiorca muszą uzgodnić nazwę kolejki, której będą używać do przekazywania wiadomości.
- Istnieje możliwość narastania liczby wiadomości przechowywanych w kolejce (w przypadku gdy odbiorca nie jest w stanie odbierać danych z szybkością ich napływania do kolejki), co może doprowadzić do przepełnienia bufora wiadomości. W takim przypadku nadawca oznacza te dane jako nie dostarczone i podejmuje próbę ich transmisji po zwolnieniu miejsca w buforze kolejki. Gdy wcześniej nastąpi zmiana nie dostarczonych danych, to są one zastępowane nowymi danymi i po zwolnieniu bufora kolejki zostaje wysłana aktualna wartość danych.

4. Bazy danych

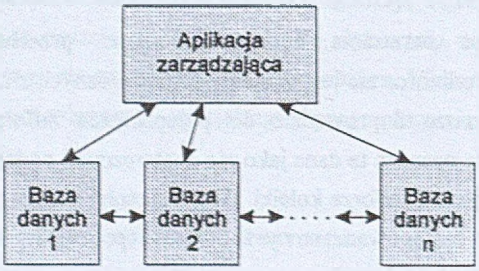
Baza Danych (**Data Base**) - jest to zbiór danych o określonej strukturze oraz narzędzia i funkcje do przetwarzania tego zbioru. Zbiór jest zapisany na zewnętrznym nośniku pamięciowym komputera. Ma on na celu zaspokojenie potrzeb wielu użytkowników, korzystających z niego w sposób selektywny, w dogodnym dla siebie czasie. W bazie danych zapisywane są fakty i niekiedy zdarzenia, zachodzące w pewnym wycinku rzeczywistości po to, aby możliwe było ich odtworzenie i analiza w dowolnej chwili. Oprogramowanie zarządzające bazą danych przekształca logiczny obraz danych, jakim posługuje się użytkownik w fizyczną reprezentację tych danych.

Jednym z możliwych podziałów baz danych ze względu na spełniane funkcje jest podział na bazy: historyczne, relacyjne oraz czasu rzeczywistego. Bazy czasu rzeczywistego wykorzystywane są głównie w systemach sterowania i innych aplikacjach czasu rzeczywistego. Różni się ona dość znacznie od tradycyjnych baz danych: relacyjnej i historycznej.

4.1. Rozproszone bazy danych

Dane zawarte w rozproszonych bazach danych są podzielone pomiędzy kilka baz, ale dla aplikacji są logiczną pojedynczą bazą danych. Rozproszona baza danych może występować na jednej platformie serwera albo (częściej) na wielu.

Definiuje się, że dane zapamiętywane w więcej niż jednej fizycznej (lub logicznej) pamięci określa się jako *dane rozproszone*. Natomiast oprogramowanie używane w celu uzyskania dostępu do danych rozproszonych zwane jest: *rozproszoną bazą danych*, co ilustruje rys. 3.



Rys. 3. Rozproszona baza danych
Fig. 3. Distributed data base

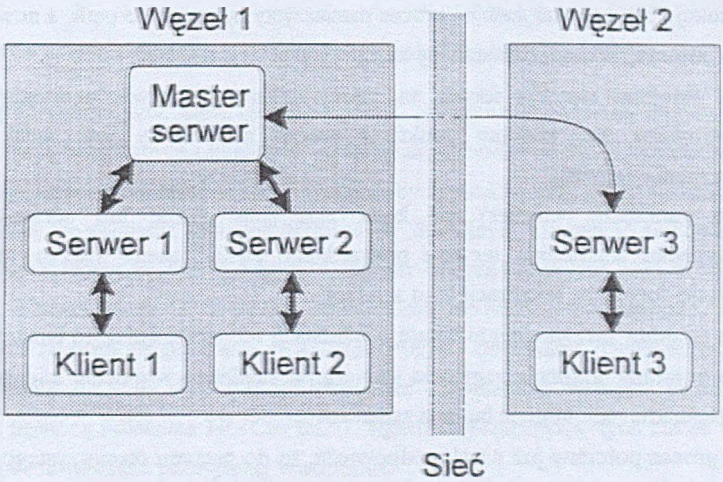
4.2. Dynamiczna rozproszona baza danych w systemie czasu rzeczywistego

Utworzenie dynamicznej, rozproszonej bazy danych w systemie czasu rzeczywistego wymaga napisania grupy aplikacji współpracujących ze sobą za pomocą określonego systemu komunikacji poprzez sieć. Do rozwiązania tego problemu wykorzystano architekturę klient-serwer oraz zalety systemu operacyjnego QNX, takie jak: komunikacja międzyprocesowa oparta na przesyłaniu wiadomości oraz wielozadaniowość, łatwość implementacji, prostota budowy i przejrzystość aplikacji.

Zbudowany system składa się z zespołu współpracujących ze sobą procesów, w których można wyróżnić: klientów, serwer nadrzędny zwany *master-server* oraz serwery podrzędne zwane *slave-server*. Podstawowym zadaniem serwera nadrzędnego jest przechowywanie informacji dotyczącej liczby działających w danej chwili serwerów podrzędnych. Do zadań serwerów podrzędnych należy przechowywanie informacji otrzymanych od klientów i w razie potrzeby rozsyłanie ich do innych klientów zainteresowanych tymi informacjami. Natomiast rolą klientów jest rejestrowanie i odczyt informacji, które gromadzi serwer podrzędny. Komunikacja pomiędzy poszczególnymi aplikacjami jest zorganizowana z wykorzystaniem mechanizmu funkcji *Send/Receive/Reply* systemu QNX oraz depozytów. Uwarunkowane jest to tym, że podczas komunikacji z klientem - serwer nigdy nie wywołuje funkcji *Send()*, dzięki

czemu nie przechodzi w stan zablokowania, tzn. nie oczekuje na odpowiedź od innego procesu. Serwer może tylko oczekiwać na dowolną wiadomość od klienta, jednak sam nigdy nie inicjuje połączenia. W związku z tym, że proces, który wyzwala depozyt, nie przechodzi w stan zablokowania, serwer może posłużyć się depozytem do powiadomienia klienta o danym zdarzeniu. Tak poinformowany klient może wydać odpowiednie polecenie serwerowi funkcją *Send()*. Do momentu uzyskania odpowiedzi klient jest w stanie zablokowania, zawieszając tym samym swoje działanie.

Rysunek 4 przedstawia uproszczony sposób komunikacji pomiędzy poszczególnymi aplikacjami. Klienci mogą odczytywać dane zarówno ze swojego lokalnego serwera (bazy danych), jak również z innej, aktualnie działającej, bazy danych. Serwery mogą działać na jednym węźle, jak również na kilku węzłach w sieci. Nad poprawnością pracy rozproszonej bazy danych czuwa proces nadrzędny *master-server*.



Rys. 4. Schemat komunikacji w rozproszonej bazie danych
Fig. 4. Diagram of the communication in a distributed data base

4.3. Struktura przesyłanych wiadomości

Podstawową daną, przesyłaną pomiędzy poszczególnymi aplikacjami, jest *punkt*. Punkt jest strukturą, składającą się z następujących pól:

- *name (nazwa punktu)* - pozwala ona klientom na jednoznaczne odwołanie się do określonego punktu.
- *id (identyfikator punktu)* - ustalany jest przez serwer i przekazywany klientowi, jest równocześnie dla serwera indeksem w jego wewnętrznej tablicy punktów.

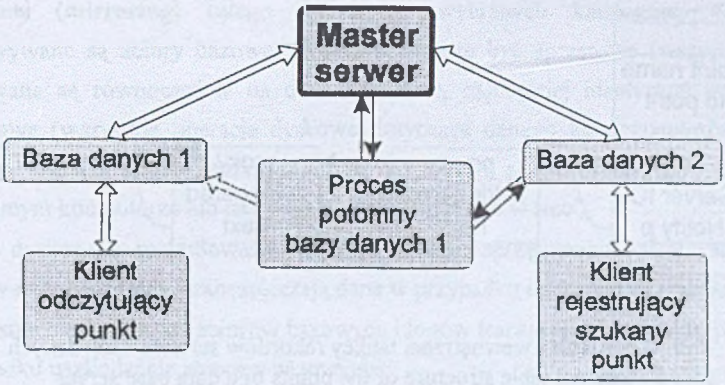
- *value (wartość punktu)* - jest to liczba, która może określać stan danego procesu technologicznego.
- *time (czas nadania wartości punktowi)* - czas ten związany jest z wartością punktu. Umożliwia on określenie momentu zajścia danego zdarzenia.
- *serverID (identyfikator serwera)* - jest to liczba wykorzystywana przez serwer do rozpoznawania sytuacji awaryjnej.

4.4. Zasada działania serwera nadrzędnego (master-serwer)

Serwer nadrzędny, tzw. master-serwer, przechowuje listę aktualnie działających serwerów podrzędnych. W przypadku gdy dany punkt nie istnieje w serwerze podrzędnym, któremu klient zleca jego odczytanie, wykonywane są następujące czynności:

- Serwer podrzędny rozwidla się (w celu uniknięcia zakleszczenia), wykorzystując do tego celu funkcję biblioteczną *fork()* - proces macierzysty powraca do pętli, a nowo utworzony proces potomny jest odpowiedzialny za dalszy przebieg zdarzeń.
- Proces potomny zapytuje serwer nadrzędny (master) o serwer podrzędny, w którym zarejestrowany jest szukany punkt; a proces macierzysty jest nadal gotowy do przyjmowania zapytań.
- Serwer nadrzędny (master) po kolei przepytuje znane mu serwery podrzędne i w przypadku znalezienia serwera podrzędnego posiadającego szukany punkt zwraca informację dotyczącą lokalizacji tego serwera.
- Proces potomny otrzymuje informację o lokalizacji serwera z żądanym punktem.
- Proces potomny dokonuje zapytania serwera podrzędnego w imieniu klienta, który żądał danego punktu (tzn. kopiuje bufor z zadaniem).
- Kiedy proces potomny już uzyska odpowiedź, to do procesu macierzystego odsyła kopię odpowiedzi z prośbą o zwrócenie jej do oczekującego klienta i następuje zakończenie działania tego procesu.
- Proces macierzysty, wykorzystując funkcję biblioteczną *Reply()*, odpowiada oczekującemu klientowi i go odblokowuje.

Na rys. 5 został zilustrowany opisany wyżej tok postępowania, w przypadku gdy klient, dla którego lokalną bazą jest baza danych 1, poszukuje punktu zlokalizowanego w bazie danych 2.



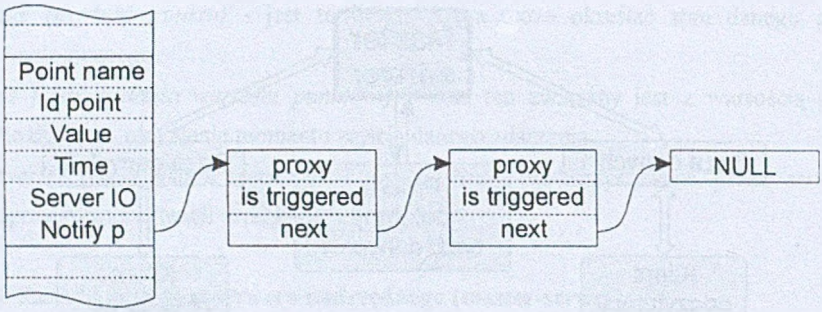
Rys. 5. Zasada działania rozproszonej bazy danych czasu rzeczywistego
Fig. 5. Working principle of the real time distributed data base

4.5. Zasada działania serwera podrzędnego (lokalnej bazy danych)

Serwer podrzędny, tzw. slave-server, zajmuje się głównie wymianą danych pomiędzy swoją wewnętrzną bazą danych i klientami, od których pochodzą odpowiednie zlecenia.

Serwer ten wytwarza tablicę rekordów opisujących punkty, przy czym po starcie tablica ta jest pusta. Jej wypełnianie jest wykonywane w dowolnym momencie podczas działania serwera na odpowiednie polecenia klientów. Serwer podrzędny można nazwać dynamiczną bazą danych. Klienci mogą w dowolnej chwili zlecić serwerowi podrzędnemu rejestrację nowego punktu za pomocą polecenia *REGISTERPOINT*, odczytanie istniejącego punktu za pomocą polecenia *READPOINT* lub informowanie o zmianach wartości danego punktu (śledzenie) za pomocą polecenia *TRACEPOINT*. Istotnym parametrem tych zleceń jest nazwa punktu, która stanowi część przesyłanej wiadomości.

Strukturę tablicy rekordów przedstawiono na rys. 6. W rekordach, oprócz przechowywanej struktury punktu, występuje dodatkowe pole oznaczone jako *notify_p*. Pole to jest wskaźnikiem listy depozytów (**proxy**) dla poszczególnych klientów, które są wyzwalane przez serwer podrzędny, w przypadku gdy nastąpi zmiana wartości punktu zarejestrowanego wcześniej do śledzenia. W polu *is_triggered* ustawiana jest zmienna logiczna tzw. flaga, za pomocą której serwer podrzędny rozpoznaje, czy dany depozyt został już wyzwolony – tzn. czy odpowiadający mu klient został poinformowany o zmianie. Pole *next* wskazuje na następny element listy (depozyt następnego klienta).



Rys. 6. Struktura wewnętrznej tablicy rekordów serwera bazy danych
Fig. 6. Internal table structure of the points of a data base server

Taki sposób komunikacji pomiędzy serwerem podrzędnym i klientami sprawia, że klient, który nie będzie mógł nadążyć z odbiorem danego punktu, otrzyma od serwera nadrzędnego zawsze najnowszą wartość danego punktu wtedy, kiedy będzie mógł ją odebrać.

4.6. Zasada działania klientów

Klientem może być dowolny proces, który potrafi komunikować się z serwerem podrzędnym za pomocą ustalonych poleceń. Klient, który chce się poprawnie komunikować z serwerem podrzędnym, powinien przestrzegać następujących reguł:

- odnaleźć symboliczną nazwę serwera podrzędnego w przestrzeni nazw, nawiązując tym samym połączenie z serwerem podrzędnym;
- zarejestrować - za pomocą polecenia: *REGISTERPOINT* - punkty, których wartości będą mogły być później zmieniane przez niego lub innego klienta;
- zlecić serwerowi podrzędnemu odesłanie żądanego punktu - za pomocą polecenia: *READPOINT*;
- zlecić zmianę wartości określonego punktu za pomocą polecenia: *POINTCHANGED*;
- lub zlecić serwerowi podrzędnemu informowanie go o zmianach danego punktu za pomocą polecenia: *TRACEPOINT*.

5. Równoległe bazy danych

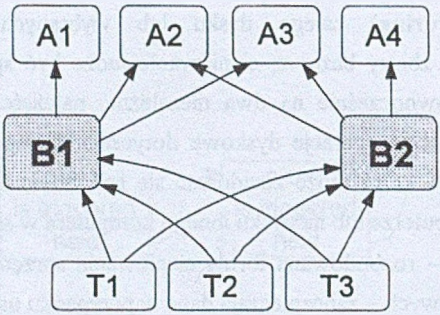
Najczęściej występującymi przyczynami utraty części informacji gromadzonych w bazie danych są: przerwy w działaniu bazy danych, uszkodzenie zbiorów bazowych oraz ewentualne uszkodzenie komputera lub dysku twardego, na którym dane te są gromadzone i przechowywane. Powszechnie stosowane metody zabezpieczenia przed utratą danych to:

- cieniowanie (**mirroring**) całego dysku lub wybranych katalogów, w których przechowywane są zbiory bazowe; cieniowanie może być sprzętowe (wszystkie zapisy wykonywane są równocześnie na dwa niezależne, najczęściej identyczne nośniki) lub programowe (wszystkie operacje dyskowe dotyczące danego katalogu są duplikowane w katalogu "cieniu", który może znajdować się na tym samym dysku, na innym dysku w tym samym komputerze lub na dysku innego komputera w sieci);
- macierze dyskowe – rozbudowana forma cieniowania sprzętowego wykorzystująca ideę nośników nadmiarowych – zabezpieczają dane w przypadku uszkodzenia nośnika;
- wykorzystanie zapasowych zbiorów bazowych i logów transakcji bazy – zabezpiecza dane w przypadku uszkodzenia zbiorów bazowych.

Metody te sprawdzają się głównie w przypadkach uszkodzeń nośnika lub zbiorów bazowych, nie zabezpieczają natomiast przed przerwą w pracy bazy danych spowodowaną na przykład awarią komputera. Jednocześnie każda z nich posiada pewne wady: w przypadku cieniowania występuje problem arbitrażu przy odczycie danych oraz przy cieniowaniu programowym znaczne niekiedy spowolnienie dostępu do bazy (zwłaszcza w przypadku dysku "cienia" pracującego na innym komputerze), w przypadku macierzy dyskowych stosunkowo wysoki koszt sprzętu, w przypadku zapasowych zbiorów bazowych i logów transakcji możliwa utrata pewnej ilości danych.

W celu wyeliminowania powyższych mankamentów autorzy zaproponowali zastosowanie równoległych baz danych, obsługujących te same procesy technologiczne. Rozwiązanie takie stwarza możliwość pracy bazy bez utraty danych zarówno w przypadku uszkodzenia nośnika, jak i przerwy w pracy samej bazy (np. awaria komputera). Ideę działania równoległych baz danych pokazuje rys. 7 na przykładzie skupionej bazy danych. Do obsługi N procesów technologicznych T1 . . . TN tworzy się na dwóch niezależnych serwerach dwie bazy danych: B1 i B2, które gromadzą i przechowują informacje z tych samych procesów. Rozwiązanie takie jest możliwe oczywiście tylko w przypadku, gdy dane dla każdego z procesów technologicznych T1 . . . TN mogą być zduplikowane na dwa strumienie, każdy skierowany do jednej bazy.

W procesie generowania tych baz danych tworzone są odpowiednie tablice punktów, w których odbywa się rejestracja i archiwizacja danych na dwóch płaszczyznach: czasowej (np.: dane: 2 sek., 30 sek., 5 min., 1 godz., 8 godz., 24 godz.) i technologicznej, np.: dane dotyczące składników załadunku (czas archiwizacji: 1 miesiąc) i dane dotyczące spustu (czas archiwizacji: 12 miesięcy). W miarę upływu czasu wpisywane są w tych tablicach odpowiednie wartości tych punktów, które następnie mogą być wywoływane z dowolnie wybranego przedziału czasu oraz grupy wielkości, w celu ich wizualizacji, analizowania lub sporządzania odpowiednich raportów, np.: godzinowych, zmianowych, dobowych czy miesięcznych.



Rys. 7. Równoległe bazy danych
Fig. 7. Parallel data bases

Programy aplikacyjne, które korzystają z bazy danych, mogą się odwoływać tylko do jednej, jak również do obu równoległych baz danych równocześnie.

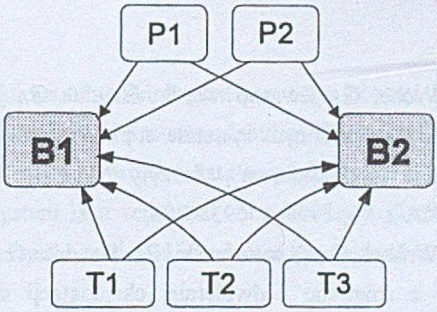
5.1. Synchronizacja baz danych

W przypadku przerwy (z dowolnego powodu) w działaniu jednej z równolegle działających baz danych (B1 lub B2), w momencie jej restartu generowany jest proces synchronizujący (P1 lub P2 na rys. 8), którego zadaniem jest:

- sprawdzenie, czy w tablicach punktów wypełnione są wszystkie rekordy czasowe do chwili obecnej,
- w przypadku braku rekordów, które powinny być wypełnione do chwili obecnej, ściągnięcie odpowiadających im rekordów z równoległej bazy danych (B2 lub B1).

W przypadku uszkodzenia zbiorów bazowych lub nośnika w jednym z serwerów bazowych proces synchronizujący po restarcie bazy danych (i ewentualnej wymianie nośnika na nowy) odtwarza wszystkie brakujące zbiory bazowe (w szczególnym przypadku całą bazę) i uzupełnia niewypełnione rekordy czasowe.

Dzięki takiemu rozwiązaniu zostają prawie idealnie odtworzone brakujące wartości punktów, a dane przechowywane w obu bazach danych powinny być spójne.



Rys. 8. Synchronizacja baz danych
Fig. 8. Data bases synchronizing

6. Podsumowanie

Metoda równoległych baz danych została z pozytywnym skutkiem zastosowana do realizacji systemu informacyjnego Wielkiego Pieca nr 3 w Hucie im. T. Sendzimira SA w Krakowie, który w ramach remontu kapitalnego został wyposażony w **nowoczesny, kompleksowy system sterowania wielkim piecem hutniczym**, opracowany w Katedrze Automatyki Napędu i Urządzeń Przemysłowych AGH, a następnie wdrożony do produkcji w Zakładzie Doświadczalnym Aparatury Naukowej i Automatyki d. ZDAN AGH Sp. z o.o. System ten został uruchomiony w grudniu 1999 r.

Obejmuje on funkcje załadunku wielkiego pieca (około 1500 sygnałów dwustanowych) oraz funkcje systemu informacyjnego obsługującego około 300 wielkości technologicznych. Zrealizowany został za pomocą 3 sterowników PLC 2000 oraz 3 komputerów klasy PC, stanowiących węzły sieci QNXnet, na nośniku typu ETHERNET. Sterowniki PLC 2000 zbudowane są na bazie przemysłowych komputerów w standardzie PC/104 (embeded system), z wykorzystaniem własnego oprogramowania narzędziowego CONTIX, służącego zarówno do realizacji tzw. systemu "Soft PLC", jak i tworzenia cyfrowych regulatorów PID dla regulacji wielkości analogowych.

LITERATURA

1. Zygmunt H., Wróbel G., Kwasnowski P., Hayduk G., Jachimski M.: Zastosowanie sieci QNXnet w rozproszonym systemie sterowania wielkim piecem. (Materiały VI Konferencji "Sieci komputerowe", Zeszyty Naukowe Politechniki Śląskiej, seria: INFORMATYKA z. 36, 1999, s. 693 - 708).
2. Zygmunt H., Wróbel G., Kwasnowski P., Hayduk G., Jachimski M., Mikoś Z.: Doświadczenia z rozruchu i dwuletniej eksploatacji systemu sterowania wielkim piecem nr 5 w Hucie im. T. Sendzimira SA w Krakowie. Zeszyty Naukowe AGH, seria: ELEKTROTECHNIKA i ELEKTRONIKA, z. 3, 1999.
3. Buschman F. et al.: Patern - oriented software architecture. John Wiley & Sons, 1996.
4. Garlan D., Schaw M.: Software Architecture Perspectives on an Emerging Discipline. Prentice Hall, New Jersey, 1996.
5. Tanenbaum Andrew S.: Rozproszone systemy operacyjne. PWN, Warszawa 1997.
6. Stevens W. R.: Programowanie zastosowań sieciowych w systemie UNIX. WNT, Warszawa 1996.
7. Lerch W.: System operacyjny QNX. Wersja 4.2. Wyd. Quantum Corp., Wrocław 1995.
8. Carl L., Hall: Techniczne podstawy systemów klient-serwer. WNT, Warszawa 1996.
9. QNX operating system architecture. QNX Software Systems Ltd, Ontario 1996.
10. Materiały w postaci elektronicznej o systemie QNX - udostępniane wraz z systemem QNX.
11. Materiały w postaci elektronicznej dotyczące bazy danych czasu rzeczywistego - udostępnione przez firmę: Cogent Real Time Systems.

Recenzent: Dr inż. Ryszard Winiarczyk

Wpłynęło do Redakcji 26 czerwca 2000 r.

Abstract

In the numerous applications of technological processes control systems, employing the logical control and digital controllers of process quantities like temperature, pressure, and gas or liquid flow, there is a necessity of process data acquisition and recording in the so-called information system. That system is a relational, or sometimes distributed, data-base from which necessary information can be retrieved in selected groups and time intervals, for further visualisation or reports.

Cases of failures of data-base, computer or hard disk, on which the data are acquired and stored, induced the authors to search for a simple method of securing continuity of the process data recording and storage in large industrial plants such as blast furnaces, rotary kilns in cement plants, or lime shaft furnaces. The method under the question involves application of the dynamic, distributed data-bases in the multitasking, real-time operating system QNX.

Developing a software in a client-server model comprising, besides the master server, also so-called slave servers where real-time data-bases have been implemented, and use of parallel data-bases with processes which ensure the data actualisation has been considered the best solution.

The parallel data-bases method has been successfully implemented in the information system of the blast furnace No 3 at the T. Sendzimir Steelworks in Kraków. This blast furnace has been supplied with an innovative, complex control system developed in the Institute of Electrical Drives and Industrial Equipment Control, University of Mining and Metallurgy, Kraków and manufactured by the Experimental Department of Scientific Equipment and Automation Ltd.

The system has been started-up and commissioned in December 1999.