CD-ROM: Serwer aplikacji BEA Weblogic

10/99

magazyn

Linux & UNIX

PHP - dynamiczne generowanie stron WWW

E-commerce w świecie UNIXA

Programowanie w języku PostScript

Serwer aplikacji Progress Open AppServer

Muzyczne biurko w Linuxie

AlphaServer 1000A

Linux w sieci Microsoft Networks

CD ROM

NR 10(14)/99 PAŹDZIERNIK CENA 17 ZŁ ISSN 1506-1299

SPIS TREŚCI

Adres kontaktowy: magazyn@tao.com.pl

E-commerce w świecie Unixa na przykładzie InterMarket 3D 6

Programowanie w języku PostScript 10
Adam Dawidziuk

nroff, troff - formatowanie dokumentacji w systemach uniksowych 14
Piotr Bolek

Produkty

Progress Open AppServer. Tworzenie aplikacji wielowarstwowych 13
Piotr Michałkiewicz

Mikroprojektor MP1600 19

Serwer aplikacji BEA WebLogic 20
Wojciech Ozimek

Uśpiona moc 24
Dariusz Piotr Pawlak

DIGI Media Conversion System 25

Programowanie

PHP - dynamiczne generowanie stron WWW 26
Piotr Bolek

Muzyczne biurko 30
Marek Sawerwain

Technologie sieciowe

TCP/IP od podstaw 33
Wojciech Bojdoł

PVM - wirtualna maszyna równoległa 34
Piotr Koch

Integracja

Linux w Sieci Microsoft Networks 40
Tomasz Gliczyński

UNIX

SCO OpenServer 5.0.5 48
Opracował Janusz Wirtz

Emulator systemu Linux dla platformy SCO 50

Wdrożenia

ASTER - Terminal Służb Ruchu Lotniczego 52
Dariusz Piotr Pawlak

PVM - wirtualna maszyna równoległa

Piotr Koch

System PVM (*Parallel Virtual Machine*) jest jednym z pierwszych pakietów programowych pozwalającym maszynom o odmiennych architekturach (sekwencyjnych, równoległych i wektorowych) oraz o różnych reprezentacjach danych (np. liczb zmiennoprzecinkowych), na wspólną pracę nad jednym zadaniem jako jeden komputer wirtualny (patrz Rysunek 1).

Komputery te połączone siecią, lokalną lub rozległą, tworzą jeden wirtualny komputer równoległy, na którym można uruchamiać program złożony z wielu wzajemnie komunikujących się procesów, PVM zapewnia automatyczną konwersję między różnymi reprezentacjami danych. Osiągnięte to zostało dzięki odpowiedniej budowie systemu, która uwzględnia różne najbardziej znane, popularne typy platform, od procesora Intel 386 poczynając, przez praktycznie wszystkie typy stacji roboczych, na architekturze CRAY kończąc. Po instalacji systemu otrzymujemy jego żądaną wersję dostosowaną do typu posiadanego sprzętu. Kod źródłowy podstawowej wersji pakietu PVM nie jest duży i zawiera nieco powyżej 1MB kodu napisanego w języku C (przy czym pakiet posiada dwa interfejsy, zarówno dla języka C jak i dla języka Fortran).

Dodatkową zaletą jest to, że system ten może być zainstalowany i uruchomiony przez użytkownika nie posiadającego specjalnych ani rozszerzonych praw w systemie bazowym, a jedynie konta na maszynach pracujących pod systemem operacyjnym UNIX mających dostęp do sieci komputerowej (np. Internet).

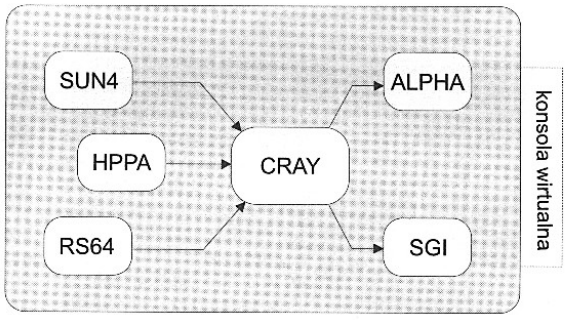
System PVM posiada ok. 50 podstawowych tzw. prymitywów w postaci funkcji włączanych w ciało programu docelowego, które w pełni obsługują wszystkie zdarzenia związane z przetwarzaniem równoległym w komputerze wirtualnym.

Pakiet PVM właściwie składa się z dwóch części. Pierwsza z nich jest procesem rezydentnym w systemie tzw. demonem (w skrócie nazywanym `pvmd`). Poprzez demony są uruchamiane i kontrolowane procesy użytkownika. Demony pośredniczą w komunikacji między procesami wchodzącymi w skład programu rozproszonego, a także zapewniają konwersję danych między różnymi architekturami. Możliwa jest również komunikacja z pominięciem demonów, co w określonych sytuacjach może spowodować jej przyspieszenie.

Aplikacja działająca z wykorzystaniem systemu PVM może zostać uruchomiona z dowolnego komputera wchodzącego w skład maszyny wirtualnej (jak również konsolę służącą do kontroli stanu systemu). Druga część to biblioteka `libpvm.a`, zawierająca szereg funkcji i procedur, będących interfejsem pomiędzy aplikacją a niższymi warstwami systemu (należy ją skonsolidować wraz z obiektami tworzonej aplikacji). PVM jest oparty na modelu obliczeń z przesyłaniem komunikatów (ang. *message passing*). Głównym jego zadaniem jest dostarczenie funkcji do komunikacji między procesami. Oprócz funkcji komunikacyjnych PVM dostarcza wielu funkcji pomocniczych, np. do zarządzania procesami, ich synchronizacją itp. Na uwagę zasługują funkcje pozwalające testować i rekonfigurować maszynę wirtualną z wnętrza programu. Umożliwia to tworzenie programów reagujących na zmiany w konfiguracji sieci, np. wyłączenie niektórych stacji roboczych biorących udział w obliczeniach rozproszonych nie musi spowodować przerwania obliczeń.

Dostarczane przez PVM'a funkcje można podzielić na kilka grup, które wraz z przykładami przedstawia Tabela 1.

Wymiana informacji pomiędzy rozproszonymi fragmentami projektu oparta jest na klasycznej technice przesyłania asynchronicznych komunikatów (MPP - *Message Pass – Passing*), która bazuje na technice tzw. aktywnych buforów. Jednocześnie w systemie może być wiele takich buforów, a ich działania mogą być zorganizowane na wiele sposobów.



Rysunek 1. Komputer wirtualny

Pracę takiego bufora można podzielić na dwa etapy:

- ładowanie/rozładowanie danych do/z bufora z ewentualnym kodowaniem/dekodowaniem,
- wysłanie/obiór danych.

Dodatkowo w systemie *PVM* można używać kodowania danych *XDR* (ang. *External Data Representation*) aby uniknąć niejednoznaczności reprezentacji danych podczas przetwarzania w sieciach heterogenicznych.

Typowy program rozproszony składa się z programu głównego, którego zadaniem jest powołanie procesów zdalnych i przydzielanie im pracy, oraz samych procesów zdalnych, które są uruchamiane na połączonych z sobą komputerach, realizując przetwarzanie w sposób rozproszony.

Wybór, na którym komputerze ma być uruchomiony proces może odbywać się w trzech trybach (patrz Tabela 2).

Tak elastyczny sposób wykonywania różnych zadań pozwala danej aplikacji zaprojektowanej przez użytkownika wykorzystać maksymalnie własności zesta-

Tabela 1. Podział funkcji zaimplementowanych w systemie PVM (z przykładami).

Kontrola procesów	
pvm_mytid	rejestracja procesu w maszynie wirtualnej
pvm_exit	wyrejestrowanie procesu z maszyny wirtualnej
pvm_spawn	uruchomienie jednego lub więcej procesów
Informacyjne	
pvm_parent	zwraca numer identyfikacyjny procesu rodzica
pvm_stat	zwraca status procesu
pvm_mstat	zwraca status hosta
pvm_config	pozwala odczytać konfigurację maszyny wirtualnej
pvm_tasks	pozwala odczytać informacje o procesach PVM
Dynamiczna konfiguracja	
pvm_addhost	dodanie hosta do maszyny wirtualnej
pvm_delhost	usunięcie hosta z maszyny wirtualnej
Sygnały	
pvm_sendsig	wysłanie sygnału do procesu
pvm_notify	zawiadomienie procesów o zajściu zdarzenia
Przesyłanie komunikatów	
pvm_send	wysłanie komunikatu
pvm_mcast	wysłanie komunikatu do pewnego zbioru procesów
pvm_recv	odebranie komunikatu
pvm_nrecv	nie blokujący odbiór komunikatu
Pakowanie i rozpakowywanie danych	
pvm_pkint	zapakowanie danych typu integer do przesłania
pvm_upkint	wypakowanie danych typu integer po przesłaniu
Obsługa buforów komunikacyjnych	
pvm_mkbuf	utworzenie nowego bufora do komunikacji
pvm_freebuf	zwolnienie bufora
Pvm_getsbuf	pobranie aktualnego bufora dla wysyłania
pvm_getrbutf	pobranie aktualnego bufora dla odbierania
pvm_setsbuf	wybranie bufora dla nadawania
pvm_setrbuf	wybranie bufora dla odbierania
Dynamiczne grupy procesów	
pvm_joingroup	podłączenie procesu do grupy
pvm_lvgroup	odłączenie procesu od grupy
pvm_barrier	synchronizacja wszystkich członków grupy
pvm_bcast	wystanie komunikatu do członków grupy

wionego komputera wirtualnego.

Niewątpliwą zaletą systemu PVM jest istnienie całego szeregu nakładek oraz rozszerzeń, które zwiększających możliwości tego systemu (patrz Tabela 3).

System *HeNCE* oraz technologia *PVM* rozwija się bardzo gwałtownie, co też świadczy o ogromnej ich przydatności, łatwości użycia oraz znakomicie opracowanym interfejsie użytkownika. Sprawiło to, że jest on jednym z najpopularniejszych systemów równoległego przetwarzania danych oraz zrównoleglania aplikacji. System *PVM* pozwala zrównoleglać aplikacje, jak wiadomo debugowanie takich programów nie jest proste, mimo to zapewnia on dość dogodną formę analizy zdarzeń i występujących sytuacji awaryjnych (dobre rozwiązanie tego zagadnienia stawia dodatkowo ten system w dobrym świetle).

System ten został z powodzeniem zastosowany m.in. w następujących dziedzinach:

- ☑ Technologia materiałowa
- ☑ Modelowanie klimatu i pogody
- ☑ Prognozowanie pogody
- ☑ Studia atmosfery, oceanów, przestrzeni kosmicznej
- ☑ Trójwymiarowe modelowanie zasobów wód gruntowych
- ☑ Badanie nadprzewodnictwa oraz dynamiki molekularnej
- ☑ Metody obliczeniowe *Monte Carlo*
- ☑ Modelowanie 2D i 3D sejsmiki Ziemi
- ☑ Symulacje i obliczenia numeryczne
- ☑ Przetwarzanie obrazu, dźwięku
- ☑ Obróbka, generacja oraz przetwarzanie fraktali i animacji

Jak widać są to zadania, których realizacja na pojedynczych jednostkach komputerowych jest trudna lub wręcz czasami niemożliwa z uwagi na wysoki nakład kosztów obliczeniowych oraz czasochłonność.

Kontrola procesów zdalnych

Procedury do kontroli procesów obejmują funkcje do tworzenia (powoływania) procesów, rejestrowania ich w wirtualnej maszynie PVM, przechwytywania wyjść powołanych procesów w celu kontrolowania ich pracy i wyświetlania komunikatów o błędach oraz realizują wyjście z systemu PVM i likwidowanie procesów, które nie zakończyły pracy w normalny sposób.

Aby program rozproszony funkcjonował na wirtualnej maszynie PVM, musi być uruchomiona główna aplikacja, która następnie uruchomi zadania zdalne, realizujące rozproszone obliczenia zlecane przez zadanie koordynujące. W systemie PVM funkcja taka ma następującą postać:

```
int numt = pvm_spawn( char *task, char **argv,
→ int flag, char *where, int ntask, int *tids)
```

`pvm_spawn()` uruchamia określoną przez zmienną `ntask` liczbę kopii wyznaczonego przez zmienną znakową `task` programu wykonywalnego na wirtualnych maszynach. Wykonywalny plik musi znajdować się na każdym z komputerów, na którym ma być uruchomio-

Tabela 2. Sposób wyboru komputera, na którym ma być uruchomiony proces zdalny.

Tryb	Znaczenie
Przeźroczystości	Podzadanie zlecane jest do wykonania najbardziej odpowiedniemu komputerowi, który jest wybierany poprzez funkcje systemowe w sposób automatyczny.
Zależna od architektury	Użytkownik zaznacza specyficzny rodzaj architektury, na której szczególne podzadanie będzie wykonane.
Z wskazaniem maszyny	Użytkownik zaznacza specyficzną maszynę, na której dane podzadanie ma

ny proces zdalny. `argv` jest wskaźnikiem na tablicę z parametrami dla uruchamianego programu określonego przez zmienną `task`, w której koniec tablicy jest wyspecyfikowany przez znak `NULL`. Jeśli zadanie nie potrzebuje żadnych parametrów, wartość argumentu `argv` wynosi `NULL`. Opcje ustawiane przy użyciu parametru *flag* specyfikuje Tabela 4.

Nazwy parametrów zostały w systemie PVM zdefiniowane w pliku `pvm3/include/pvm3.h`.

PvmTaskTrace jest nową własnością PVM, która pojawiła się od wersji 3.3, sprawia ona, że uruchamiane zadanie generuje zdarzenia śledzące (ang. *task events*). PvmTaskTrace jest wykorzystywane w XPVM. W innym wypadku trzeba określić gdzie mają być wysyłane zdarzenia wynikające ze śledzenia wykonywania programu funkcją `pvm_setopt()`.

Parametr `where` pozwala wyspecyfikować gdzie mają być uruchamiane procesy PVM. W zależności od wartości `flag`, `where` może być nazwą hosta (na przykład "ibm1.epm.ornl.gov") lub architektury (przykładowo "SUN4"). Jeśli flaga ma wartość 0, parametr `where` jest ignorowany, a system PVM wybiera sam najbardziej odpowiedni komputer.

Zwracana przez funkcję wartość `numt` jest ustawiana zależnie od ilości zadań jakie się udało prawidłowo powołać lub przybiera wartość kodu błędu jeśli zadań nie udało się uruchomić. Jeśli zadania zostały uruchomione, `pvm_spawn()` zwraca wektor uruchomionych zadań `tids`, jeśli jakieś zadania nie mogły być uruchomione, odpowiednie kody błędów zostają umieszczone w ostatnich `ntask - numt` pozycjach wektora `tids`.

Gdy `flag` ma wartość 0 i `where` jest ustawione na `NULL`, to zadania są uruchamiane na maszynie wirtualnej w sposób heurystyczny. Na początku brane są kolejne maszyny z list hostów, a później o wyborze najbardziej odpowiedniego komputera decyduje jego obciążenie i wydajność.

W szczególnym przypadku parametr `where` może wskazywać komputer wieloprocesorowy i w takim wypadku `pvm_spawn()` uruchamia wszystkie `ntask` zadań na jednej maszynie.

Każdy proces PVM powinien rejestrować się w systemie poprzez wywołanie procedury:

```
int tid = pvm_mytid( void )
```

Procedura `pvm_mytid()` zwraca TID - identyfikator procesu, który ją wywołał, procedura może być wywoływana wielokrotnie. TID jest przydzielany procesowi podczas tworzenia go instrukcją `pvm_spawn()`, a jeśli proces był stworzony w jakiś inny sposób jest on generowany w wyniku wywołania funkcji `pvm_mytid()`. TID jest 32-bitową dodatnią liczbą całkowitą, generowaną przez lokalnego demona `pvmd`. Na ten 32-bitowy identyfikator procesu składa się kilka pól, zawierających informacje o procesie, takie jak: lokacja procesu w wirtualnej maszynie PVM (na przykład adres lokalnego demona `pvmd`), numer procesora, w wypadku gdy proces jest uruchomiony na komputerze wieloprocesorowym, oraz ID procesu. Informacje te są używane przez system PVM, ale nie jest przewidziane wykorzystywanie ich przez użytkownika. Pierwsze wywołanie tej funkcji, powoduje zarejestrowanie zadania w systemie PVM. Wywołanie dowolnej funkcji PVM powoduje zarejestrowanie zadania w systemie, jeśli nie został on zarejestrowany w wcześniej, ale powszechnie zakłada się, że funkcję tę powinno się wywołać jako pierwszą w celu zarejestrowania. Gdy aplikacja zostanie uruchomiona zanim zostanie wystartowany PVM, wartość zwracana przez funkcję `pvm_mytid()` jest mniejsza od zera.

W celu ułatwienia śledzenia i szukania błędów występujących w procesach zdalnych PVM udostępnia funkcje:

```
int info = pvm_catchout( FILE *ff)
```

Domyślnie PVM zapisuje strumienie `stderr` i `stdout` tworzonych procesów do pliku `/tmp/pvml.<uid>`. Procedura `pvm_catchout()` powoduje, że proces wywołujący (rodzic) przechwytuje strumień wyjściowy procesów tworzonych po wywołaniu tej procedury. Znaki wysyłane do standardowych strumieni wyjściowych (`stdout`, `stderr`) przez procesy potomne są zbierane przez demona `pvmds` i wysyłane kontrolnymi komunikatami do procesu macierzystego (rodzica), który etykietuje każdą linię i dodaje do specjalnego pliku. Każda linia w pliku wyjściowym ma następującą postać:

```
[txxxxx] BEGIN
[txxxxx] (teks od procesu potomnego)
[txxxxx] END
```

Gdzie [txxxxx] identyfikuje proces, BEGIN i END oznaczają początek i koniec linii, a miedzy nimi znajdują się znaki, wysłane przez proces potomny.

Przed zakończeniem pracy zadania powinna być wywołana funkcja odwrotna do `pvm_mytid()`, informująca system rozproszony, że proces go opuszcza:

```
int info = pvm_exit( void )
```

Wywołanie tej procedury mówi lokalnemu demonowi PVM (`pvmd`), że ten proces opuszcza PVM. Procedura ta nie zabija procesu, zadanie może się dalej wykonywać jako zwyczajny proces systemu (np. systemu

Tabela 3. Zestawienie rozszerzeń systemu PVM.

Nazwa pakietu	Opis
DoPVM	Rozszerzenie pozwalające tworzyć aplikację w C++.
Xab	Graficzna nakładka na system PVM.
XPVM	Jest wersją pakietu PVM wzbogaconą o nowy kompletny zintegrowany interfejs graficzny zrealizowany m.in. w języku Tcl/Tk, system ten posiada wiele nowych możliwości.
HeNCE	*HeNCE (Heterogenous Network Computer Environment)* jest najnowszą propozycją wywodzącą się po części z technologii PVM, lecz posiada znacznie bardziej rozbudowane algorytmy obsługujące i tworzące pracę równoległą, doskonały interfejs graficzny oraz, co jest nieosiągalne w innych systemach tego typu, pozwala na graficzna edycję programu, który ma być zrównoleglony lub udoskonalony.

Unix). `Pvm_exit()` powinny wywoływać wszystkie procesy kończące swoje działanie, a w szczególności te, które nie buły tworzone poleceniem `pvm_spawn()`.

Wszystkie "niepokorne" procesy, które nie zakończyły swojego działania, bądź z winy systemu lub ewentualnego błędu programisty mogą być usunięte bezwzględnie następującą funkcją:

```
int info = pvm_kill( int tid )
```

Procedura `pvm_kill()` likwiduje proces PVM identyfikowany przez `tid`, wysyłając sygnał przerywający (ang. *terminate signal*, SIGTERM). W przypadku komputerów wieloprocesorowych sygnał ten jest zamieniany na odpowiednią metodę służącą do likwidowania procesów, zależną od rodzaju maszyny.

Procedura ta nie służy do normalnego kończenia pracy wcześniej powołanych procesów, ale do likwidowania tych, których nie udało się usunąć mimo wywołania `pvm_exit()`, po którym nastąpiło `exit()`.

Konfiguracja, Funkcje informacyjne, Zdarzenia

Funkcje informacyjne

Nie jest możliwe istnienie sprawnego systemu bez licznych funkcji udostępniających informacje o jego konfiguracji i zadaniach na nim uruchamianych.

Jedną z bardzo ważnych informacji dla każdego procesu zdalnego jest identyfikator procesu, który go powołał, informacja ta może być bardzo pożyteczna w przypadku gdy proces zdalny oczekuje komunikatów od procesu-rodzica, zawierających polecenia lub dane do obliczeń. Taką funkcje pełni procedura:

```
int tid = pvm_parent( void )
```

Funkcja `pvm_parent()` zwraca TID procesu, który powołał zadanie, w którym miało miejsce wywołanie tej procedury. Jeśli zadanie nie było powołane przez `pvm_spawn()` zwracana jest wartość *PvmNoParent*.

Niekiedy ważną informacją może być to na jakim hoście został uruchomiony proces. Możemy to uzyskać przez wywołanie następującej funkcji.

```
int dtid = pvm_tidtohost( int
tid )
```

Wywołując tą procedurę można poznać TID `dtid` demona, na którym uruchomiony jest proces. Procedura ta może być bardzo pomocna w ustalaniu, na których hostach są uruchamiane poszczególne zadania.

W celu zoptymalizowania pracy systemu może być przydatna znajomość konfiguracji całego systemu, może to pomóc właściwie rozmieścić procesy zdalne, tak aby wykorzystać w pełni moc obliczeniową aktualnej konfiguracji. Może być w tym bardzo pomocna poniższa procedura:

```
int info =  pvm_config( int *nhost, int *narch,
            struct pvmhostinfo **hostp )
```

Procedura `pvm_config()` udostępnia wiele informacji związanych z konfiguracją wirtualnej maszyny. Wywołując tę funkcję możemy dowiedzieć się ile hostów tworzy maszynę wirtualną, `nhost`, i iloma różnymi formatami danych się one posługują. `hostp` jest wskaźnikiem do zdefiniowanej przez użytkownika tablicy struktur `pvmhostinfo`. Rozmiar tablicy powinien być przynajmniej równy ilości hostów. Struktura `pvmhostinfo` zawiera TID'a `pvmd`, nazwę hosta, nazwę architek-

Tabela 4. Możliwe wartości parametru flag.

Wartość	Nazwa	Opis
0	PvmTaskDefault	PVM sam wybierze gdzie zostanie powołany proces
1	PvmTaskHost	Argument *where* wyszczególni hosta, na którym będzie powołany proces
2	PvmTaskArch	Argument *where* będzie typu PVM_ARCH i określi architekturę, na której ma być powołany proces
4	PvmTaskDebug	Uruchamia zadania pod debuggerem
8	PvmTaskTrace	Generowane są dane do śledzenia
16	PvmMppFront	Uruchamia zadanie na komputerze MPP
32	PvmHostCompl	Uzupełnia zbiór hostów, parametrem *where*

Tabela 14. Zdarzenia sygnalizowane przez pvm_notify().

Wartość	Znaczenie
PvmTaskExit	Powiadom jeśli nastąpiło wyjście jakiegoś zadania, przesyłany jest komunikat do każdego procesu z listy *tids* zawierający TID wychodzącego zadania i oznaczony zgodnie z parametrem *msgtag* w wcześniejszym wywołaniu pvm_notify()
PvmHostDelete	Powiadom jeśli host został usunięty, przesyłany jest komunikat do każdego procesu z listy *tids* zawierający TID demona pvmd, który właśnie opuścił PVM. Komunikat jest oznaczony zgodnie z parametrem *msgtag* w wcześniejszym wywołaniu pvm_notify()
PvmHostAdd	Wysyła do *cnt* komunikatów. Komunikaty zawierają nową liczbę demonów pvmd.

tury i relatywną względem innych hostów w konfiguracji prędkość CPU.

Bywa, że proces potrzebuje znać informacje o innych zadaniach uruchomionych z nim współbieżnie, może to uzyskać wykonując operacje:

```
int info = pvm_task( int which, int *ntask,
        → struct pvmtaskinfo**taskp )
```

Procedura `pvm_task()` zwraca informacje o zadaniach uruchomionych na wirtualnej maszynie PVM. Jeśli podamy jako parametr `which` wartość 0, to chodzi nam o wszystkie uruchomione zadania, jeśli podamy TID demona `pvmd`, będzie to rozumiane w ten sposób, że interesują nas wszystkie zadania na danym hoście, gdy wartością `which` będzie TID procesu, to uzyskamy informację wyłącznie o nim. Ilość zadań jest zwracana przez zmienną `ntask`. `taskp` jest wskaźnikiem na tablicę struktur `pvmtaskinfo`. Każda struktura `pvmtaskinfo` zawiera TID procesu, TID demona `pvmd`, TID procesu macierzystego (rodzica), flagi aktualnego stanu zadania, nazwę pliku, który został użyty przez instrukcję `pvm_spawn()` w celu powołania procesu. Gdy proces został powołany ręcznie, z konsoli PVM, to ostanie pole jest puste, gdyż w takim wypadku system nie zapamiętuje jego nazwy.

Dynamiczna konfiguracja

Jedną z wielu cech, które decydują o tym że system PVM jest jednym z najlepszych narzędzi w przetwarzaniu rozproszonym decyduje to, że istnieje możliwość dynamicznej konfiguracji systemu. Istnieją funkcje dodające i usuwające maszyny tworzące PVM'a, które mogą być wywoływane z poziomu aplikacji. Funkcje te w połączeniu z funkcjami informacyjnymi tworzą PVM'a systemem odpornym na błędy, pozwalają bowiem wykrywać niepoprawnie pracujące komputery i w sposób dynamiczny, na poziomie aplikacji usuwać je z systemu. Na dynamiczną konfigurację systemu składają się następujące funkcje:

```
pvm_addhost( char **hosts, int
nhost, int *infos )
pvm_delhost( char **hosts, int
nhost, int *infos )
```

Procedury te dodają lub usuwają zbiór hostów (*hosts*) z wirtualnej maszyny. Zwracana wartość `info` określa ile hostów udało się dodać. Argument `infos` jest tablicą długości `nhosts`, która zawiera kody statusu dla każdego dodawanego lub usuwanego hosta. Pozwala to sprawdzić użytkownikowi czy tylko jeden ze zbioru hostów jest źródłem problemów, bez zbędnego ponownego dodawania lub usuwania całego zbioru.

Funkcje te mogą służyć do konstruowania wirtualnej maszyny, ale częściej jednak są wykorzystywane do zwiększania elastyczności i tolerancji błędów dużych aplikacji. Procedury te pozwalają aplikacjom zwiększyć ogólną dostępną moc obliczeniową poprzez dynamiczne zwiększanie liczby hostów w miarę możliwości (na przykład wykorzystanie dodatkowych bezczynnych w weekendy komputerów) lub w miarę potrzeb, gdy rozwiązywany problem wydaje się być na tyle skomplikowany, że dotychczasowa moc obliczeniowa wydaje się nie być wystarczająca.

Używając tych funkcji w połączeniu `pvm_mstat()` i `pvm_config()` możemy tworzyć programy bardziej odporne na błędy, aplikacja może wykrywać niepoprawnie pracującego hosta, usunąć go lub ewentualnie zastąpić innym.

Sygnały

PVM zapewnia dwie funkcje związane z sygnalizacją zdarzeń w systemie.

```
int info = pvm_sendsig( int tid, int signum )
```

Procedura `pvm_sendsig()` wysyła sygnał o numerze `signum` do procesu identyfikowanego przez PVM z użyciem jego identyfikatora `tid`. Jeśli występuje jakiś błąd, wartość `info` jest mniejsza od zera. Funkcja ta powinna być wykorzystywana tylko przez programistów mających doświadczenie w obsługiwaniu sygnałów, gdyż nieprawidłowe posługiwanie się nimi może prowadzić do blokady, niedeterministycznego działania programu, a nawet do zawieszenia całego systemu.

```
int info = pvm_notify( int what, int msgtag,
        → int cnt, int tids )
```

`pvm_notify()` jest pewnym rodzajem żądania, zobowiązującego PVM do powiadamiania procesów `tids` o pewnych zdarzeniach. W odpowiedzi na takie żądanie PVM pewna ilość komunikatów jest wysyłana z powrotem do wywołujących zadań. Komunikaty są znaczone odpowiednimi kodami `msgtag` określonymi podczas wywołania `pvm_notify()`. Tablica `tids` spe-

cyfikuje kto chce monitorować zdarzenia TaskExit lub HostDelete, nie powinna zawierać nic jeśli używamy HostAdd. `what` specyfikuje o jakim zdarzeniu ma powiadamiać system PVM (patrz Tabela 14).

Za odebranie i obsługę komunikatu są opowiedziane procesy, które żądały powiadomienia i do nich też należy podjęcie odpowiedniej akcji. Jak widzimy funkcja `pvm_notify()` pozwala tworzyć aplikacje, które dynamicznie reagują na zmiany konfiguracji.

Pobieranie i ustawianie opcji

Konfigurowanie i zmiana opcji sytemu może się również odbywać z poziomu programu. Daje to możliwość eksperymentowania i optymalizowania aplikacji. Służą do tego dwie opcje:

```
int oldval = pvm_setopt( int what, int val )
int val = pvm_getopt( int oldval )
```

`pvm_setopt()` i `pvm_getopt()` są uniwersalnymi funkcjami, które pozwalają użytkownikowi czytać i ustawiać opcje systemu PVM. Obie funkcje zwracają stan opcji, `pvm_getopt()` aktualny, a `pvm_setopt()` przed modyfikacją. Parametry te mogą dotyczyć różnych ustawień, włącznie z automatycznym wyświetlaniem komunikatów błędów, poziomem debugowania i wyborem metod routingu komunikacyjnego. Wartość `what` definiuje co będzie modyfikowane. Z użyciem tych funkcji jest możliwe ustalenie strategii routingu. Istnieją trzy metody, zalecana *PvmRouteDirect*, pozwalająca na ustalenie bezpośredniego połączenia zadania z zadaniem z użyciem protokołu TCP. Takie połączenie raz ustawione pozostaje do końca pracy aplikacji, użycie takiej metody może nawet dwukrotnie przyspieszyć komunikację między procesami. Jeśli połączenie takie nie może być ustawione ponieważ jedno z dwóch zadań zażądało *PvmDontRoute* lub nie ma więcej wolnych zasobów systemowych, wtedy jest używane domyślne routowanie demonów PVM. Bywa, że opcja ta jest ignorowana, tak jak na przykład w przypadku multiprocesora Intel Paragon, ponieważ do komunikacji między zadaniami jest używany jego własny protokół. `pvm_setopt()` może być wywoływne wielokrotnie do selektywnego ustawiania połączeń, ale jej typowe użycie ma miejsce na początku każdego zadania. Standardowym routowaniem PVM jest *PvmAllowDirect*. Ustawienie takie w przypadku zadania A pozwala się innym łączyć bezpośrednio i raz ustawione pozwala obu zadaniom na wysyłanie komunikatów. ■

Dokończenie artykułu ze strony 33

Dzięki takiemu podziałowi dane o dużych rozmiarach mogą być swobodnie przesyłane na duże odległości, a w wypadku błędu w którymkolwiek pakiecie - ponawia się przesyłanie tylko tego pakietu.

IP **Internet Protocol**
Protokół internetowy - w wersji 4 i 6, związany jest z adresowaniem oraz dostarczaniem pakietów

ICMP **Internet Control Message Protocol**
Protokół komunikatów kontrolnych - istnieje w wersji 4 i 6 - pozwala na śledzenie działania sieci, lokalizację problemów oraz testowanie połączeń

IGMP **Internet Group Management Protocol**
Protokół zarządzania grupami - używany przez routery i bramy

GGP **Gateway-Gateway Protocol**
Protokół komunikacji między bramami

TCP **Transmission Control Protocol**
Protokół kontroli transmisji - odpowiedzialny za kontrolę poprawności przesyłanych pakietów

UDP **User Datagram Protocol**
Protokół datagramów użytkownika - protokół pozwalający wysyłać i odbierać datagramy nie kontrolując poprawności

EGP **Exterior Gateway Protocol**
Protokół zewnętrznej bramki - używany przez routery (bramy)

HMP **Host Monitoring Protocol**
Protokół monitoringu maszyn

RDP **Reliable Data Protocol**
Protokół niezawnodnego przesyłania danych

Przykłady zastosowania w praktyce

Istnieje już wiele popularnych programów pozwalających korzystać nam z usług Internetu. Do jednego z nich należy Netscape Navigator. Jest to przegladarka internetowa, której działanie opiera się właśnie na protokołach TCP/IP. Użytkownik wpisuje adres serwera, którego zawartość chciałby zobaczyć oraz adres konkretnego pliku na tym serwerze, po czym program stara się nawiązać połączenie i pobrać strone z odległego serwera. W przypadku jakichkolwiek błędów przy przesyłaniu danych protokoły postarają się przetransmitować te dane automatycznie jeszcze raz tak, że jeśli to możliwe, to użytkownik o niczym nie będzie wiedział.

Na cały scenariusz można też popatrzeć od strony serwera - serwer wykrywa rządanie nawiązania połączenia, po czym stara się jak najszybciej je obsłużyć. Odbiera od programu żądanie i stara się odpowiedzieć na nie - jeśli zapytanie było prawidłowe, to zwróci prawidłową odpowiedz na nie, w przeciwnym przypadku poinformuje program kliencki o powstałym błędzie.

W kolejnych częściach tego artykułu postaram się opisać podstawy tych protokołów, jakie są między nimi zależności oraz gdzie znalazły one zastosowanie. ■