Lab 13 – Adrian Wanczewski

PVM (Parallel Virtual Machine) - zestaw narzędzi do tworzenia oprogramowania dla sieci równolegle połączonych komputerów. Został zaprojektowany i stworzony by umożliwić łączenie komputerów o różnych konfiguracjach sprzętowych w jeden równolegle działający komputer.
Jest narzędziem służącym głównie do pośrednictwa w wymianie informacji pomiędzy procesami uruchomionymi na oddzielnych maszynach oraz do zarządzania nimi za pośrednictwem konsoli pvm.

W dużym uproszczeniu PVM pozwala na "połączenie" większej ilości komputerów w jeden. Odbywa się to na zasadzie dołączania kolejnych hostów (komputerów), na których został zainstalowany i skonfigurowany pvm. Podłączanie hosta jest w uproszczeniu procedurą połączenia przez rsh, uruchomieniu demona pvm i dostarczenia mu informacji o tym, kto jest jego "rodzicem" oraz o parametrach istniejącej sieci.

W momencie, kiedy cała "maszyna wirtualna" już jest skonfigurowana, rola pvm sprowadza się do stanowienia pomostu wymiany informacji pomiędzy procesami, które się w pvm "zarejestrują", oraz umożliwienia administrowania stanem zarejestrowanych w pvm procesów.

Identyfikacja w PVM odbywa się przez oddzielne numery identyfikacyjne przydzielane procesom w momencie rejestrowania się w PVM. Numery te stanowią zupełną abstrakcję od systemu operacyjnego i sprzętu, na którym uruchomiony jest dany proces. Numer identyfikacyjny jest bardzo istotnym elementem PVM, ponieważ stanowi on niejako adres danego procesu i jest on niepowtarzalny w zakresie pojedynczej maszyny PVM. Adres ten jest wykorzystywany przy wszystkich procedurach dotyczących innych procesów - można przesłać informacje pod dany adres, można "zabić" zdalny adres i wykonać wiele innych czynności.

Można dostrzec więc pewne podobieństwa do MPI, ale pierwsza wersja PVM została napisana w ORNL w 1989 roku. Natomiast MPI zostało stworzone dopiero w maju 1994 r. więc całkiem możliwe ze PVM było w jakiejś części wzorem dla MPI.

Dzisiaj PVM jest nadal aktywnym projektem i nadal stosuje się go w programowaniu równoległym.

Poniżej przykłady użycia PVM (tj. Komunikacja punkt-punkt, oraz obliczanie liczby PI)

Komunikacja punkt-punkt

pvm_send

```
int retval = pvm_send(int tid ,int msgtag); //
```

Parametry

- **tid** – identyfikator procesu, do którego ma zostać wysłany komunikat.
- **msgtag** – liczba całkowita definiowana przez użytkownika jako etykieta komunikatu (powinna być >=0).
- **retval** – kod statusu zwracany przez funkcję (retval < 0 oznacza błąd podczas wykonania operacji).

pvm_recv

```
int retval = pvm_recv(int tid ,int msgtag); //
```

Parametry

- **tid** – identyfikator procesu, od którego ma zostać odebrany komunikat.
- **msgtag** – liczba całkowita definiowana przez użytkownika jako etykieta komunikatu (powinna być >=0).
- **retval** – kod statusu zwracany przez funkcję (retval < 0 oznacza błąd podczas wykonania operacji).

Obliczanie liczby PI metodą "Monte Carlo"

```
///// mc-master.c /////

#include <stdio.h>
#include <pvm3.h>
#include <time.h>
#include <stdlib.h>

#define ileprocow 6
#define mcastmsg 10
#define bcastmsg 11
#define reducemsg 12
#define ilelosowan 1400000000

/*W przypadku tego programu należy pamiętać, iż
nazwa programu "niewolnika" musi być mc-slave
(bądź dowolna inna, w przypadku gdy zmienimy
wartość w pvm_spawn)
*/

int main(int argc, char *argv[]) {

int nproc = ileprocow;
long i, ile = ilelosowan, seed = time(NULL), ilewkole = 0;
char *nazwaGrupy;
nazwaGrupy = "mc-row";
double interwal, x, y;
long double pi;
time_t poczatek, koniec;
int myId = pvm_mytid(), taskNo, tIds[nproc], inum, info;

poczatek = time(0);
taskNo = pvm_spawn("mc-slave", (char**) NULL, PvmTaskDefault, "", ( nproc -1 ), tIds); //

pvm_initsend( PvmDataDefault ); //
pvm_pkint(&nproc, 1, 0); //
pvm_pkstr(nazwaGrupy); //
pvm_mcast(tIds, taskNo, mcastmsg); //

inum = pvm_joingroup( nazwaGrupy ); //
info = pvm_barrier( nazwaGrupy , nproc ); //

ile = ile / nproc;
pvm_initsend(PvmDataDefault); //
pvm_pklong(&ile , 1 , 0); //
pvm_bcast( nazwaGrupy, bcastmsg ); //

srand( seed * myId  - myId );
for(ile; ile > 0 ; ile--)  {
   x = ((double)rand() / RAND_MAX)*2 - 1;
   y = ((double)rand() / RAND_MAX)*2 - 1;
   if(!(((x*x) + (y*y))>1)) ilewkole++ ;
};

int myGid = pvm_getinst(nazwaGrupy, myId); //
info = pvm_barrier( nazwaGrupy , nproc ); //
pvm_reduce( PvmSum, &ilewkole, 1, PVM_LONG, reducemsg, nazwaGrupy, myGid); //
info = pvm_barrier( nazwaGrupy , nproc ); //

pvm_exit(); //

pi = ilewkole ;
pi = pi * 4;
pi = pi / ilelosowan;

printf("punktow w kole = %d sposrod = %d\n", ilewkole, ilelosowan);
printf("liczba pi wynosi: %.64Le\n", pi);

koniec = time(0);
interwal = koniec - poczatek;
printf("\nIlosc procesow obliczeniowych: %d \nCzas trwania obliczen: %.0lf sekund\n", nproc, interwal);
return 0;
}
```

```
///// mc-slave.c /////

#include <stdio.h>
#include <pvm3.h>
#include <time.h>
#include <stdlib.h>
#define mcastmsg 10
#define bcastmsg 11
#define reducemsg 12

int main(int argc, char *argv[]){
char *nazwaGrupy;
int nproc;
int parentId = pvm_parent(), myId = pvm_mytid(), info, inum;
long ile, ilewkole = 0, seed = time(NULL);
double x, y;

pvm_recv(parentId, mcastmsg); //
pvm_upkint(&nproc, 1, 0); //
pvm_upkstr(nazwaGrupy); //

inum = pvm_joingroup(nazwaGrupy); //
info = pvm_barrier(nazwaGrupy, nproc); //
pvm_recv(parentId, bcastmsg); //
pvm_upklong(&ile, 1, 0); //

srand(seed * myId - myId);
for (ile; ile > 0; ile--) {
    x = ((double)rand() / RAND_MAX)*2 - 1;
    y = ((double)rand() / RAND_MAX)*2 - 1;
    if(!(((x*x) + (y*y))>1)) ilewkole++;
};

int pGid = pvm_getinst(nazwaGrupy, parentId); //
info = pvm_barrier(nazwaGrupy, nproc); //
pvm_reduce( PvmSum, &ilewkole, 1, PVM_LONG, reducemsg, nazwaGrupy, pGid); //
info = pvm_barrier(nazwaGrupy, nproc); //

pvm_exit(); //
return 0;
}
```