PVMsnap User Manual

Application Program Organization

The computation must be started by a unique process, called starter. The starter only can spawn other processes and call a cut operation. No process can be created after the fist cut request.

```
Scheme of the starter:

   initialization
   process spawn
   loop
     compute
     cut
   end loop
```

Each regular message received by a user process has an additional parameter indicating whether the message is in transit through a cut or not. Two new (control) messages can be received. They are not sent by a user process and carry no data: they are generated by the PVM daemons to notify the user process of the beginning and of the termination of the cut. The first message (begin cut) is the local cut pseudo event of the consistent cut, the second message (end cut) informs the user process that it has received all its in-transit messages. The starter receives the cut termination message only when all the in-transit messages have been delivered to their receiver process, that is, when the cut algorithm is terminated. After that, it can call for a new cut.

Since the selective receive is allowed, the termination condition could be never verified, then a cut could never terminate if a user process does not require to receive all its in-intransit messages.

Both control messages are delivered to the user process, independently of its possible conditional receive, at first it is possible. That is, the begin cut message is delivered to user processes when received from the PVM local daemon, before any other buffered message. The termination message is delivered to a user process after its receipt from the daemon and after the delivery of every in-transit buffered message.

Cut Request

The cut is performed by the function pvm_startcut().

Synopsis: int info = pvm_startcut()

Description: It can be called only by the starter in order to start a cut operation. The starter also participates to the cut. A negative value is returned if the function does not succeed. The function is not blocking. The starter is notified of the global termination of the cut by the receipt of a control message with tag *PvmTermCut*.

EXAMPLE

```
info = pvm_startcut();
while ((bufid = pvm_recv(tid, msgtag, &flag)) != PvmTermCut)
{
  ...  /* compute */
}
/* end cut */
info = pvm_startcut()
```

Message receipt

The original pvm_recv() function has been modified as follows.

SYNOPSIS: int bufinfo = pvm_recv(int tid, int msgtag, int *transmsg)

PARAMETERS:

tid, msgtag - as in the original PVM,

transmsg - = 1 if the received message is in transit, = 0 otherwise,

bufid - a positive value is the identifier of the active input buffer which contains the message. Negative values are:
PvmCutEvent: control message which notifies the beginning of the cut operation;
PvmTermCut: control message which notifies the termination of the cut operation;
PvmBadParam, PvmSysErr: error messages, as in the original PVM.

EXAMPLE

```
bufid = pvm_recv( tid, msgtag, &transmsg)
if (bufid >= 0)
{
   if (transmsg)
   {
      /* regular in-transit message */
   }
   else
   {
      /* regular non in-transit message */
   }
}
else
  switch(bufid)
  {
    case PvmCutEvent :
      ...     /* begin cut */
```

```
        break;
      case PvmTermCut :
        ...      /* end cut */
        break;
      default :
        ...      /* error */
    }
```

How to use PVMsnap

The file PVMsnap.uue contains a uuencoded, compressed tar file with the directories NEWLIB and MY_EXAMPLE. The first one contains the executable files for the PVM-snap console and daemon and a library for compiling user processes, for SUN4 with SUN OS. The second one contains the code of a starter process (test00.c) and a slave process (test00slave.c) which can be compiled using the script file npvmcc (npvmcc test00 and npvmcc test00slave commands produce the executable files test00 and test00slave in my_example/SUN4). The two environment variables PVM_ROOT and PVM_ARCH must be set to the parent directory and to SUN4 respectively.

The test program test00

The example in directory $PVM_ROOT/my_example is a test to verify the correctness of the cut protocol.

A. The starter process spawn n processes. Initially each process sends a message to each other. Then, it enters a loop in which a message is read (from ANY) and a message carrying an integer value is sent to a process randomly choosen. Moreover, each process also maintains a table, with two integer variables for each process of the program (initially =0). The first variable is updated with the last value received from the related process. The second variable is incremented each time a message is sent to the related process. Such a value is the data sent within the message itself.

B. After m messages the starter calls for a cut. Eventually, each process receives a begin cut control message. After the receipt, the content of the table is written on a log file (out.<process pid>) and its variables are then set to 0. Each process continues to read the input messages. Messages marked as in-transit are written in the log file until the cut termination message is received.

Section **B** is repeated for k times. The values n, m and k are set by the user, starting the program.

Syntax

```
test00 [n1 [n2 [n3]]]
```

where:
n1 = number of slave processes (test00slave),
n2 = number of messages sent by the starter before the cut request,
n3 = number of cuts.

default values:
n1 = 10
n2 = 10
n3 = 5

Warnings

PVM console

The console is considered by PVM like any other process. Since in the available version no changes have been performed in order to exclude the console from the cut evaluation, the TEST00 program must be executed without the console. That is, the operation are the followings:

- Start PVM by the console or by the daemon,
- Configure the virtual machine,
- quit the console,
- start the test TEST00.

The console must not be activated during the computation. Morever, PVM must be stopped at the end of the computation, before any other execution.

Available pvm calls

The only I/O modified PVM procedures are pvm_send() and pvm_recv(). Multiple send and non blocking receive cannot be used.

Process spawn

Only the starter can spawn PVM processes.