PTW-006: How Do I Kill Thee?

Paul Koufalis
President
Progresswiz Consulting

PROGRESS SOFTWARE

Progresswiz Consulting

- **Based in Montréal, Québec, Canada**
- **Providing technical consulting in Progress, Oracle, UNIX, Windows, MFG/PRO, Caméléon and more**
- **Specialized in performance tuning, system availability and business continuity planning**

Agenda

- **Introduction**
- **Goals**
- **Background**
- **Myths and Misconceptions**
- **Enough already…let's kill something!**
- **Questions**

First things first...

- **Most of this work is based on Progress 9.1D on AIX**
- **I do not work for Progress, nor do I have access to any insider information, so:**
  - **Your Mileage May Vary**
  - **Use the information in this presentation at your own risk and peril!**

Introduction

- **There are still a lot of ChUI, UNIX-based applications out there**
  - **Clients are connecting to databases via shared-memory**
  - **No matter how hard you try, a user will still figure out a way to hang a session**
  - **Killing users will remain a common task (but hopefully not too common)**

Introduction (cont...)

- **The need was there, so I went about trying to figure out the best method**
  - **Searched the peg**
  - **Searched the Progress KB**
- **Finally, I found Tom Bascom's article "Traps and Kills"**
  - **Great start, but not enough to satisfy the techy geek in me**

Introduction (cont...)

- **And so a project was born:**
  - **Figure out the best, safest way to kill a Progress session**
  - **Write a script that will automate the task as much as possible**
- **The result: killprosession.sh!**
  - **(All you fancy Powerpoint people would have inserted some sound-effect here, right?)**

Goals

- **The goals of this script, simply stated:**

  “*Provide the Progress community with a simple method to safely kill shared-memory Progress processes*.”

Background

- **Just to make sure everyone is one the same page, we need to quickly cover some background information**

Background (cont...)

- **Self-service or shared memory clients:**
  - **"Self-service clients access the database directly through shared memory and not through servers, because server code is part of the self-service client process"**
- **Network or client/server clients:**
  - **"The network client accesses the database through a server process that the broker starts over a network connection."**

Background (cont...)

- **Kill**
  - **Unfortunate and misleading name**
  - **Man page says it all: "Sends a signal to running processes."**
    - **That's it: Hello Mr. PID #453982, here's a 2.**
  - **Use "kill –l" to see a list of signals**

Background (cont...)

- **VST's: Virtual System Tables**
  - **Available since 8.3x**
    - **Enable with proutil in V8**
    - **Automatically enabled in V9+**

Background (cont...)

- **Traps**
  - **"Runs the specified command when the shell receives the specified signal or signals"**
  - **Often found in /etc/profile $HOME/.profile or similar**
  - **Type "trap" at the command line to see if your shell has any traps set**
  - **If you trap at the shell level, Progress will never get the signal (so that's why my kills never work!!)**
  - **"Just say no" in the Progress world**

Myths and Misconceptions

- **Myth #1: Killing a Progress process will bring down the database**
  - **Not necessarily**
    - **Probably not a good idea to kill _mprosrv**

Myths and Misconceptions

- **Myth #2: One kill, two kill, three kill, it's all the same**
  - **Uhh…no**
  - **If a process is not responding to a signal it's either**
    - **Hung**
    - **Busy (and hence ignoring you)**
  - **Repeatedly killing a busy process is a no-no**

Myths and Misconceptions

- **Myth #3: Kill -8 is a safe, final and effective way to kill a Progress process**
  - **Final: sometimes**
  - **Effective: sometimes**
  - **Safe: no**
  - **To quote Mr. Bascom:**

    "Kill -8 (SIGFPE) varies in behavior depending on the release of Progress. In some releases it just acts like an unhandled signal…which is essentially the same as kill -9"

Myths and Misconceptions

- **Myth #4: Kill -9 is a safe, final and effective way to kill a Progress process**
  - **Final: yes**
  - **Effective: yes**
  - **Safe: no**
  - **It gets the job done, but it's going to leave one heck of a mess**
  - **Make sure your *curriculum vitae* is up to date**

Don't Forget

- **Our main goal is to kill a Progress session *safely***
- **If the process refuses to die, live with that**
  - **Find another solution**
  - **Do NOT risk bringing down production**
    - **You think this would be obvious, wouldn't you?**

Let's Get Started

- **Outline:**
  1. **Validate PID and confirm destruction**
  2. **List the DB's to which the process is connected**
  3. **Send a few signals**
  4. **Monitor the process' reaction**
  5. **Disconnect the process from any remaining connected databases**
  6. **Wait and monitor some more**

Validate PID and Confirm

- Make sure the PID exists
- Make sure you didn't accidentally pass 1 as the PID
  - 1 is the "init" process
  - And you thought bringing the DB down would be bad for your career!
- Make sure the PID has no children
  - This is often overlooked

Validate PID and Confirm (cont...)

```
# killprosession.sh 59272
2005/06/19-22:49:59 Child process exists:
     UID    PID   PPID   C    STIME    TTY  TIME CMD
    koup 375748  59272   0 22:49:49  pts/0  0:00 /usr/bin/ksh
2005/06/19-22:49:59 Please kill all child PIDs first
2005/06/19-22:49:59 ************** KILL IGNORED FOR 59272
```

Validate PID and Confirm (cont...)

- **Show the process and arguments to the user for confirmation**

```
# killprosession.sh 59272

koup 59272 pts/0 _progres -pf /usr/local/etc/params.m.pf

Please confirm the kill of this session (y/n):
```

Get List of Connected DB's

- **Many ways to do this**
  - **Variation of "ps | grep" should do the job**

```
# for lines like: _mprosrv -db /db/test/test –pf /db/test/test.pf
ps -ef -o args | grep _mprosrv | grep -v m1 | grep -v m3 | \
awk '{print $3}' | while read PathDb
do
  ProUser=$(proshut $PathDb -C list|grep $username|grep $tty|awk '{print $1}')
  ProUser=$(_progres -b $PathDb -p $RSYS/killprosession.p \
              -param  "CONNECTED $PID")
  if [[ $ProUser != "NOT_CONNECTED" ]]
   then
      DBList="${DBList}$ProUser $PathDb\n"
  fi
done
```

Now We Kill (finally!)

- **Signal 7 (EMT) is just a “wakeup” that Progress uses**
  - **You’ve probably seen it polluting your log files in 9.1D09**
- **Signal 2 is an INT. This should raise a STOP in the Progress code**
- **Signal 1 is HUP (hangup). Generated when a user closes their terminal session**

More killing...

```
kill -7 $PID; sleep 5
kill -2 $PID; sleep 5
kill -1 $PID; sleep 10
if ( ! PIDExist $PID )
then
   Note "The process no longer exists"
   exit 0
fi
```

- **Most sessions will be gone by this point**

Now we wait

- **An impatient system administrator will only wait a couple of minutes before issuing another kill**
  - **No no no!**
  - **Let's go look in the database VST's and see why if we can figure out why the process is still alive**

Monitoring VST's

- **Let's check each DB to which the process is connected for action and wait for it to finish:**

```
# The WaitProsession function returns:
# 0 : User no longer connected
# 1 : Process not using any CPU or IO
# 2 : User still active.  May still be rolling back tx.
WaitProSession $ThisDb $ThisUser
if [[ $? = 2 ]]
then
Note "User is active - USE $0 again - DO NOT MANUALLY KILL!"
   exit 0
fi
```

WaitProsession

- **Uses killprosession.p to monitor VST's**

```
_progres -b $1 -param "MONITOR $UsrNum" -p $RSYS/killprosession.p | \
  while read Line
  do
    if [[ $Line = "NOT_CONNECTED" ]]
    then return 0
    elif [[ $Line = "NO_CPU_OR_DBIO" ]]
    then Note "Process not using and CPU or doing any DB IO“; return 1
    elif [[ $Line = "STABLE_PROCESS" ]]
    then Note "Process not in transaction nor rolling back“;  return 1
    elif [[ $Line = "TIMEOUT" ]]
    then Note "DO NOT TRY AND KILL THIS PROCESS!“; return 2
    elif [[ $Line = "TIMEOUT_RESYNC" ]]
    then
       Note "PROCESS STILL ROLLING BACK A TRANSACTION"
       Note "DO NOT TRY AND KILL THIS PROCESS!"
       return 2
    else echo "$Line"
    fi
  done
```

WaitProsession (cont...)

- **Killprosession.p is the heart of the waiting mechanism**
  - **We'll get back to this in detail later**
- **We want to make sure we leave the process alone if it's rolling back a transaction**
- **A non-responsive process that is not doing any I/O is probably hung for real**

Disconnect

- At this point we're finished waiting
- The process is either not in any kind of transaction/rollback or is totally frozen
- We try to disconnect the user from the databases

Disconnect (cont...)

```
DBList=$(ConnectedDB ${User[$i]} ${Tty[$i]} ${PidArray[$i]})
echo "$DBList" | while read ThisUser ThisDb
do
  DisconnectUser $ThisDb $ThisUser
done
sleep 10
DBList=$(ConnectedDB ${User[$i]} ${Tty[$i]} ${PidArray[$i]})
DisconnectedFlag=0
echo "$DBList" | while read ThisUser ThisDb
do
  Note "The process was not disconnect from $ThisDb"
  DisconnectedFlag=1
done
if [[ $DisconnectedFlag = 0 ]]
  then Note "Process no longer connected to any databases"
fi
```

Still not dead...

- **Much to the dismay of many a system administrator, the safest thing to do is to wait some more**
  - **The process may still be rolling back**
  - **The process may still be doing some I/O**
  - **Or maybe it's just really, really frozen**
- **In all cases WaitProsession will tell you**

I give up (well, almost)

- If really, really nothing is working, we look at what resources the process is using
  - lsof: List Open Files
  - svmon: Snapshot Virtual memory Monitor
- Does the process have any database files open?
- Is the process attached to any database shared memory segments?

LSOF

- **Requires root access**

```
# lsof -p 59272
COMMAND    PID USER   FD   TYPE     DEVICE   SIZE/OFF    NODE NAME
_progres 59272 koup  cwd   VDIR       61,5       1536 2130019 /home (/dev/lvhome
)
_progres 59272 koup    0u  VCHR       30,0    0t38852   41094 /dev/pts/0
_progres 59272 koup    1u  VCHR       30,0    0t38852   41094 /dev/pts/0
_progres 59272 koup    2u  VCHR       30,0    0t38852   41094 /dev/pts/0
_progres 59272 koup    3r  VREG       61,2    1134729  149092 /opt/rona (/dev/lv
optrona)
_progres 59272 koup    4r  VREG       61,2     865553  149101 /opt/rona (/dev/lv
optrona)
_progres 59272 koup    5u  VREG       61,3          0   49300 /usr/local (/dev/l
vlocal)
_progres 59272 koup    6w  VREG       64,2       6771  901150 /dbdvp (/dev/lvdbw
rk)
```

- **STOP: A file is open /dbdvp – a db FS**

SVMON

- **Also requires root**

```
# svmon -P 59272 | grep "shared memory segment"
   16244  3 work shared memory segment         -  25394     0    0 25394
   6d12b  6 work shared memory segment         -  21306     0    0 21306
   6ffef  4 work shared memory segment         -   5978     0    0  5978
```

- **If one of those shared memory segments belongs to a database then Hand's Off!**
- **Can check with svmon of server processes**
  - **Of course it's automated in the killprosession script**

So?

- **If the process is not using any database files or shared memory segment, it is potentially safe to kill it more violently without affecting the database server**
- **!! Proceed with Caution !!**

```
kill -8 $PID
```

But...

- If it still has db files open, or more importantly, db shared memory segments attached, you're better off letting it be
  - I can't remember the last time with V9 that I was stuck with a process that would not respond to anything
    - Of course, if there are traps in place, all bets are off

KillProsession.p

- **-param parameters:**
  - **CONNECTED <PID>**
    - **Simply returns the user # for the PID**
    - **Returns error message if PID not connected**
  - **MONITOR <UserNum>**
    - **Monitors process activity for a set number of minutes or until the process exits**

KillProsession.p (cont...)

```
DO WHILE Cnt < TotalSec:
  FIND DICTDB._Connect WHERE DICTDB._Connect._Connect-Usr = pUserNum
       NO-LOCK NO-ERROR.

  IF NOT AVAIL DICTDB._Connect THEN
  DO:
     MESSAGE "NOT_CONNECTED".
     QUIT.
  END.
  IF DICTDB._Connect._Connect-Disconnect <> 0 THEN
  DO:
     ASSIGN Disconct = YES.
     MESSAGE "INFO    : User disconnect already initiated".
  END.
  ELSE
     MESSAGE "WARNING : No user disconnect seems to have been
    initiated".
```

KillProsession.p (cont...)

```
IF DICTDB._Connect._Connect-TransID <> 0 THEN
 DO:
    Assign InTrans = YES.
    MESSAGE "INFO    : Process in transaction "
            DICTDB._Connect._Connect-TransID.
 END.
 ELSE
    MESSAGE "INFO    : Process NOT in transaction ".

 IF DICTDB._Connect._Connect-Resync = 1 THEN
 DO:
    ASSIGN InRollBk = YES.
    MESSAGE "CRITICAL: Process rolling back transaction - do not
  kill!" .
 END.
 ELSE
    MESSAGE "INFO    : Process NOT attempting to roll back a
  transaction".
```

KillProsession.p (cont...)

```
IF DICTDB._Connect._Connect-Interrupt <> 0 THEN
    MESSAGE "INFO    : Process interrupted by signal".
 IF TRIM(DICTDB._Connect._Connect-Wait) <> "--" AND
    DICTDB._Connect._Connect-Wait1 <> 0 THEN
 DO:
    MESSAGE "CRITICAL: Process waiting for resource :"
    DICTDB._Connect._Connect-Wait.
 END.
 /** Check if the user may be holding latches. WARNING: _Latch shows
  the LAST user to hold the latch.  THIS DOES NOT NECESSARILY MEAN
  THE USER IS HOLDING THE LATCH NOW!!!! **/
 ASSIGN OldLatchList = LatchList.
           LatchList = "".
 FOR EACH DICTDB._Latch WHERE DICTDB._Latch._Latch-hold = pUserNum
     NO-LOCK:
   Assign LatchList = LatchList + STRING(DICTDB._Latch._Latch-ID) +
                      " ".
 END.
 MESSAGE "INFO    : Old Latch List: " OldLatchList.
 MESSAGE "INFO    : New Latch List: " LatchList.
```

KillProsession.p (cont...)

```
/** Check if the process is actually reading/writing to the db **/
 FIND DICTDB._UserIO WHERE DICTDB._UserIO._UserIO-Usr = pUserNum
      NO-LOCK NO-ERROR.
 IF AVAILABLE DICTDB._UserIO THEN
 DO:
    Assign DBIO2 = DICTDB._UserIO._UserIO-DbAccess +
                   DICTDB._UserIO._UserIO-BiRead   +
                   DICTDB._UserIO._UserIO-BiWrite.
    MESSAGE "INFO    : Database I/O in last" PauseSec "seconds :"
            (DBIO2 - DBIO1).
 END.
 IF DBIO2 > DBIO1 THEN
    ASSIGN DBIO1  = DBIO2
           DeadIO = 0.
 ELSE
    Assign DeadIO = DeadIO + 1.
```

KillProsession.p (cont...)

```
/** Check if the process is consuming CPU time **/
 INPUT THROUGH "ps -o ~"time=~" -p" VALUE(DICTDB._Connect._Connect-
  Pid)
       NO-ECHO.
 SET PSLine.
 INPUT CLOSE.
 Time2 = (INT(ENTRY(1,PSLine,":")) * 3600) +
         (INT(ENTRY(2,PSLine,":")) * 60) +
          INT(ENTRY(3,PSLine,":")).

 MESSAGE "INFO    : CPU Time in last" PauseSec "seconds :"
         (Time2 - Time1) "secs".
 IF Time2 > Time1 THEN
    ASSIGN Time1   = Time2
           DeadCnt = 0
           .
 ELSE
    DeadCnt = DeadCnt + 1.
```

KillProsession.p (cont...)

```
  IF DeadCnt >= 5  AND DeadIO >= 5 THEN
  DO:
    MESSAGE "INFO    : No change in CPU use or DB I/O".
    MESSAGE  "NO_CPU_OR_DBIO".
    QUIT.
  END.
  IF Disconct = NO AND InTrans = NO AND InRollBk = NO THEN
  DO:
    MESSAGE "INFO    : Process seems stable".
    MESSAGE  "STABLE_PROCESS".
    QUIT.
  END.
  /** Wait and loop **/
  MESSAGE "".
  MESSAGE "************** SLEEP" PauseSec "seconds **************".
  MESSAGE "".
  PAUSE PauseSec.
  Cnt = Cnt + PauseSec.
END.
```

KillProsession.p (cont...)

```
/** TotalSec seconds are up and the process is still alive.  We quit
    with the appropriate msg based on whether or not the process is
    still rolling back. **/
FIND DICTDB._Connect WHERE DICTDB._Connect._Connect-Usr = pUserNum
     NO-LOCK NO-ERROR.
IF NOT AVAIL DICTDB._Connect THEN
DO:
   MESSAGE "NOT_CONNECTED".
   QUIT.
END.
IF DICTDB._Connect._Connect-Resync = 1 THEN
DO:
   MESSAGE "CRITICAL: Process still rolling back actively".
   MESSAGE "TIMEOUT_RESYNC".
END.
ELSE
DO:
   MESSAGE "INFO    : Still waiting - process active.".
   MESSAGE "TIMEOUT".
END.
QUIT.
```

Example Log

```
lavf 490384 pts/110 /opt/rona/dlc/bin/_progres -pf /usr/local/etc/sidma.m.pf

Please confirm the kill of this session (y/n):
2005/01/28-10:07:36 Kill -7 executed.  Wait 5 sec

2005/01/28-10:07:41 Kill -2 executed.  Wait 5 sec
2005/01/28-10:07:46 Kill -1 executed.  Wait 10 sec
2005/01/28-10:07:56 Waiting for end of session for DB /database/db01/lansing/ron
a-lnsng
WARNING : No user disconnect seems to have been initiated
INFO    : Process NOT in transaction
INFO    : Process NOT attempting to roll back a transaction
INFO    : Old Latch List:
INFO    : New Latch List:
INFO    : Database I/O in last 10 seconds : 499
INFO    : CPU Time in last 10 seconds : 4 secs
INFO    : Process seems stable
2005/01/28-10:07:56 Process not in transaction nor rolling back
```

Example Log

```
2005/01/28-10:07:56 Waiting for end of session for DB /database/db01/revy/rona-r
evy
WARNING : No user disconnect seems to have been initiated
INFO    : Process in transaction  854
INFO    : Process NOT attempting to roll back a transaction
INFO    : Old Latch List:
INFO    : New Latch List:  2 9
INFO    : Database I/O in last 10 seconds : 366
INFO    : CPU Time in last 10 seconds : 4 secs
********************* SLEEP 10 seconds *****************
<Parts of log snipped>
INFO    : CPU Time in last 10 seconds : 0 secs
INFO    : No change in CPU use or DB I/O
2005/01/28-10:08:47 Process not using and CPU or doing any DB IO
2005/01/28-10:08:49 Disconnecting user 54 from /database/db01/lansing/rona-lnsng
2005/01/28-10:08:49 Disconnecting user 592 from /database/db01/gl/rona-gl
2005/01/28-10:08:50 Disconnecting user 54 from /database/db01/revy/rona-revy
2005/01/28-10:08:50 All db disconnects attempted
2005/01/28-10:08:50 Pausing 10 seconds

2005/01/28-10:09:03 Process no longer connected to any databases
2005/01/28-10:09:06 Unable to kill process
```

Example Log

```
2005/01/28-10:09:06 Checking open files (lsof)
2005/01/28-10:09:06 Look for open files in the database lv in the lsof listing
lsof: WARNING: compiled for AIX version 4.3.2.0; this is 5.2.0.0.
COMMAND     PID USER   FD   TYPE     DEVICE SIZE/OFF   NODE NAME
_progres 490384 lavf  cwd   VDIR       89,2      512 219200 /home (/dev/lvshome)
_progres 490384 lavf    0u  VCHR     22,110 0t361144   8412 /dev/pts/110
_progres 490384 lavf    1u  VCHR     22,110 0t361144   8412 /dev/pts/110
_progres 490384 lavf    2u  VCHR     22,110 0t361144   8412 /dev/pts/110
_progres 490384 lavf    3r  VREG       89,2  1134729 268431 /home (/dev/lvshome)
_progres 490384 lavf    4r  VREG       89,2   865553 268398 /home (/dev/lvshome)
_progres 490384 lavf    5u  VREG       89,2  1121142 219205 /home (/dev/lvshome)
_progres 490384 lavf   78u  IPv4 0x82cfc9f0    0t128    TCP 0.0.255.255:sqlco->0
.0.255.255:sqlbc (ESTABLISHED)
_progres 490384 lavf   79u  IPv4 0x755961f0 0t580494    TCP 0.0.255.255:sqlco->0
.0.255.255:sqlco (CLOSE_WAIT)
_progres 490384 lavf  152u  VREG       89,2  2016984 219206 /home (/dev/lvshome)
_progres 490384 lavf  179u  IPv4 0x81dfc1f0  0t68826    TCP 0.0.255.255:sqlco->0
.0.255.255:sqlco (CLOSE_WAIT)

<Parts of log snipped>
```

Example Log

```
2005/01/28-10:09:07 Checking attached shared memory segments
2005/01/28-10:09:07 There should be no attached shared memory segments
2005/01/28-10:09:07 Attached shared memory segments:

2005/01/28-10:09:07 Listing all matching database shared memory segments:


If there were no open database files NOR attached database shared
memory segments then the process can be killed with a kill -8
Otherwise call Progress technical support.
```

- **I would kill -8 this process**
  - **Remember – YMMV!**

Conclusion

- **You can safely kill a Progress shared-memory client**
  - **Just be patient**
- **Use a script to eliminate dangerous improvisations**

Questions?

Progresswiz Consulting

- **Want the full script or have questions or comments? Send me an email**

  pk@progresswiz.com
- **And feel free to visit my website for this and other interesting information:**

  www.progresswiz.com

Thank you for your time!

PROGRESS

SOFTWARE

Simplify your business

Progress Technology World 2005