Solving the Locked Record 'Whodunit'

The Situation

By Susan Darling

Colonel Mustard calls the operations room in a panic. He must access a critical intelligence master record on the computer database immediately but cannot; the record is 'busy.' Quite obviously, the record is locked but who "dunit" and how do we "un-dunit"?

In the past, identifying the locked-record perpetrator was difficult. Today's tools will ID which user needs to exit the record, but what if that user's office is locked? What if it is at a remote location somewhere else in the world?

Earlier that day, Mr. White made several mission critical updates to this same intelligence master record. Without exiting that record, he left for his usual lunch. Normally, resolution would be just an hour away, but this day an offsite emergency has tied Mr. White up the entire afternoon. Furthermore, Mr. White has been accessing the system from a remote location several time zones away.

The Solution

By Jim Douglas

While national security may not really be threatened by the locked record illustrated in this tongue-in-cheek introduction, a locked customer record could result in an equally vulnerable and costly situation. In PRO/5® on UNIX, the only option has been something like this:

```
$ /sbin/fuser INVENTORY
INVENTORY:              10033
$ ps -p 10033 -f
UID      PID  PPID  C STIME TTY       TIME CMD
jsmith 10033 15419  0 15:08 pts/1 00:00:00 /usr/local/bin/pro5/pro5s -m1024 -cconfig.bbx
$ kill -9 10033
```

A more direct approach is the **`fuser -k`** option that kills all processes that have opened the specified file:

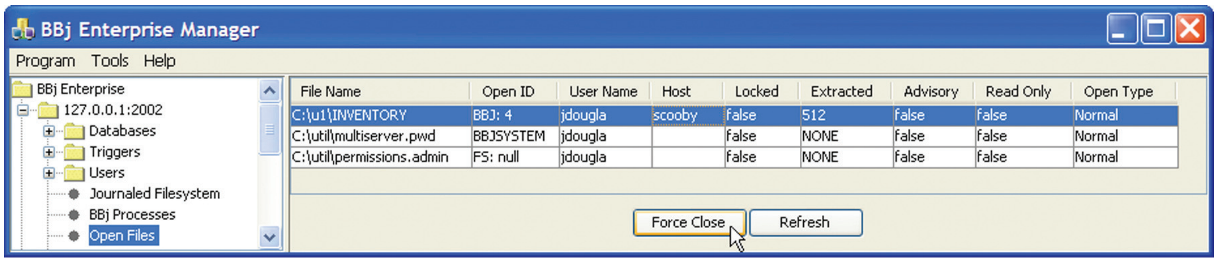
```
$ /sbin/fuser -k INVENTORY
INVENTORY:              10033
```

While this approach can work, it has some significant problems.

- It does not work with BBj®
- It only works on UNIX (and UNIX-like) systems and it requires the fuser utility
- It identifies all users who have the file open, but not the user who is locking a particular record, which is a problem if multiple users have the same file open (the typical case)
- Killing the process could cause collateral damage to other tasks the locked user was performing so the ability to force-close the channel that has the record extracted would be less drastic

The BBj Enterprise Manager includes a much more flexible way to view open files and to optionally force a file closed. To use this feature, open BBj Enterprise Manager and click on Open Files as shown in **Figure 1**.

continued...



File Name	Open ID	User Name	Host	Locked	Extracted	Advisory	Read Only	Open Type
C:\u1\INVENTORY	BBJ: 4	jdougla	scooby	false	512	false	false	Normal
C:\util\multiserver.pwd	BBJSYSTEM	jdougla		false	NONE	false	false	Normal
C:\util\permissions.admin	FS: null	jdougla		false	NONE	false	false	Normal

Figure 1. Open Files dialog box

Jim Douglas
Software Engineer
Contractor

Enterprise Manager is an improvement over the UNIX fuser approach in a few ways.

- It works on all BBj platforms, not just UNIX, and it does not depend on an external utility program
- It shows which file references are currently extracting a record (as opposed to merely having the file open)

But this approach still has some problems.

- The user must have BBj administrator privileges to log in to Enterprise Manager
- The user must navigate through a very technical screen and identify the correct file channel from the list, which can be very difficult, especially if there are dozens or hundreds of files opened across the system
- If multiple channels are currently extracting a record, there is no way to tell which one is holding the particular record of interest

BBj 6.0 improves on this functionality in the following ways.

- The new BBjOpenFileInfo API exposes information about open files in a format that can be processed so end users do not need to go into Enterprise Manager
- The TCB(10) function returns the lockbyte within the file when reporting !ERROR=0, enabling developers to locate the specific file channel that holds the lock to the record they are currently attempting to read

First, run **program1.bbj** in BBj (**Figure 2**).

```
0010 filename$="sample.dat"
0020 erase filename$,err=*next
0030 mkeyed filename$,1,0,16
0040 open (1)filename$
0050 write (1,key="X")"X"
0060 extract (1,key="X")
0070 escape
0080 print fid(1)(9)
```

Figure 2. Code sample **program1.bbj**

Next, with that program sitting at the escape, run **program2.bbj** in a different BBj 6.0 (**Figure 3**) interpreter session.

```
0010 filename$="sample.dat"
0020 open (1)filename$
0030 extract (1,key="X",err=errtrap)
0040 print "Record successfully extracted!"
0050 stop
0060 errtrap:
0070 if err<>0 then escape
0080 channel=tcb(12)
0090 lockbyte=tcb(10)
0100 fs!=bbjapi().getFileSystem()
0110 fileinfo!=fs!.getFileInfo(channel)
0120 filename$=fileinfo!.getFilename()
0130 filename$=fnslash$(filename$)
0140 BBjAdmin!=bbjapi().getAdmin("admin","admin123")
0150 BBjVector!=BBjAdmin!.getOpenFileInfos()
0160 for i=0 to BBjVector!.size()-1
0170    BBjOpenFileInfo!=BBjVector!.get(i)
0180    openfile$=BBjOpenFileInfo!.getFilename()
0190    openfile$=fnslash$(openfile$)
0200    found=openfile$=filename$ and BBjOpenFileInfo!.getExtracted()=lockbyte
0210    if found then break
0220 next i
0230 if found then print openfile$," closed"; BBjOpenFileInfo!.forceClose()
0240 retry
0250 def fnslash$(x$)
0260    x=pos("\"=x$)
0270    while x
0280       x$(x,1)="/",x=pos("\"=x$)
0290    wend
0300    return x$
0310 fnend
```

Figure 3. Code sample **program2.bbj** for finding who locked the record

continued...

Now type RUN at the first interpreter session. Attempting to access the file channel now reports !ERROR=13 because the file has been forced closed.

```
READY
>run
0070 escape

READY
>run
!ERROR=13  (File is closed.)
0080 print fid(1)(9)

READY
>
```

The previous sample contains a lot of activity. **Figure 4** shows a line-by-line breakdown of the second program.

Lines	Notes
0010..0050	Open a file, attempt to extract a record.
0060..0240	Error handler – on !ERROR=0, force close the channel that holds the lock.
0070	Only proceed if this was !ERROR=0 (locked record).
0080	Retrieve the last channel referenced (the channel with the error).
0090	Retrieve the reported lockbyte in the file.
0100..0130	Get the name of the file opened on this channel.
0140	Get the BBjAdmin object that is needed for any tasks that require BBj Administrator privileges.
0150	Get the list of all files currently opened within this copy of BBjServices.
0160..0220	Loop through the list of opened files and find the one that holds the lock that just reported !ERROR=0. The channel can be identified by matching on both filename and lockbyte.
0230	If the lock-holder was successfully identified, print it and force-close the channel.
0240	Retry the extract. With the lock-holder closed, it should succeed this time.
0250..0310	Utility function to convert any back-slashes in a filename to forward slashes (used for comparing filenames).

Figure 4. A line-by-line breakdown of the activity in the second program

Summary

Today, BBjOpenFileInfo solves the longstanding locked-record mystery for interpreters running in the same virtual machine as the DBMS and provides an easy solution to who "dunit" and how to "un-dunit" scenario. Locked customer records will no longer be a difficult issue to deal with in day-to-day operations.

Download the sample programs referenced in this article from www.basis.com/advantage/mag_v10n1/lockedrecord.zip

For a more complete and polished sample, download a mockup of a customer master file maintenance program from www.basis.com/advantage/mag-v10n1/customer.zip