Shiloh's Raid: 1541 Relative File Bug Spray

David Shiloh
Eugene, Oregon

First we squashed the SAVE@ bug with Phillip Slaymaker's article. . . now David Shiloh kills the dastardly relative file bug -- right at its roots!

It appears that there has not previously appeared in print a dissection of the huge relative file bug in the DOS, although the save "@0:bug" was a major controversy for years: the reason of this escapes me somehow, since relative files seem more major in relation to practical uses of the 1541. . .how have the gurus been distracted from such a serious problem with the DOS?

Dr. Gerald Neufeld, whose Inside Commodore DOS has proved to be indispensable, mentions the bug in his 1541 User's Guide, correctly locating it in the "position" command and offering an effective fix that exacts a 30%–40% access–time penalty. While his fix reaches two of the specific DOS failures that are involved, his discussion does not define the conditions under which problems occur, and his test program yields results that establish the existence of the bug but are otherwise almost completely misleading. Until now, this has been the most comprehensive mention of this bug.

The Position Command

The actual write to a relative file uses the same PRINT# command as any other write operation. With relative files, however, the write goes to a specific record within the file: DOS has to be positioned to the record you want to write to, and to the spot within that record where you want to begin writing. This is done with the "position" command, sent on the DOS command channel; the actual information to be written to that record is sent to the relative file following the position command. The position command is sent with the syntax:

```
print#FN, " p " chr$(96 + SA)chr$(lo)chr$(hi)chr$(po);
```

where "p" is the actual "position" instruction, followed by three parameters and a final semicolon (";") to suppress the sending of a carriage return after the command string.

The chr$(96 + SA) sends DOS the secondary address (SA) of the relative file OPEN command, which is used by DOS to assign internal channels and buffers for the relative file operations: this value is OR'd with 96 ($60) to form the byte sent to the DOS.

The chr$(lo) and chr$(hi) are one parameter, the record number (nu): lo is the low byte of the record number in low–byte/high–byte format, "hi" is the high byte, taken by

```
hi = int(nu/256):lo = nu–hi*256
```

The chr$(po) is the exact position within the relative record where the write is to begin, and is an optional parameter. However, unless you suppress the carriage return that follows the command string, this parameter chr$(po) must be included: otherwise, DOS will read the chr$(13) carriage return as the parameter and point there.

When the position command is sent, DOS retrieves the record sector you have addressed into its RAM buffers and sets the relative file channel to the selected position in the record. The same "position" command is used to position the relative file channel for reading from the file.

The Bug

Theoretically, the "position" command will allow you to position to any character in any record. In fact, this is true only for reading the file: for writing, it is 100% reliable except under certain conditions in which it is 100% unreliable.

When DOS receives a position command, it checks to see whether the desired bytes are already in one of the two buffers allocated for records. If the necessary sector is not in the "active" buffer, but the immediately preceding file sector is, then DOS simply "toggles" the buffers and makes the one containing the necessary sector active: unless it just toggled during the last access for that reason. This convenience also sets up the bug: the fatal sequence is as follows:

1. A write is performed that runs from one sector (A, in buffer a) to the next (B, in buffer b). During the write, DOS toggles from buffer a to buffer b and makes a note of the toggle.
2. A second write is performed to a record that is entirely contained on sector B in the now–active buffer b. This write does NOT toggle, and DOS makes a note of the no–toggle. Now the bug is waiting.

3. A third write is directed to the sector following B; and instead of fetching sector C, DOS toggles from buffer b to buffer a since no toggle was performed during the last access.

Unfortunately, sector A is still in buffer a and this third write goes to exactly the same place on sector A that it should have gone to on sector C -- and often overwrites two records, the last characters of one and the first characters of the next. Thus three records are in jeopardy: these two and the one that did not get written to sector C.

The program listing below demonstrates the bug, then sprays it with Shiloh's Raid.

The program creates a relative file of 100 records for each record size from 42 through 88, spending about 10 minutes with each (6 minutes compiled). Since the entire program runs over 8 hours, I set it up to rotate among my three 1541 drives, which are hardware set to device numbers 8, 9 and 10. The program will rotate among any number of drives by changing the 'nd = 1' in line 1140; the lowest drive number used can be changed from 8 by modifying the 'sd = 8' in line 1130. If you are using just one drive, you may want to use a cooling fan, or run the test for fewer trials (reduce the value of 'el' in line 1120). If you are using the program with a non-Commodore printer, check the control codes in lines 1660, 1830 and 2010 (control-j, chr$(10) for a line-feed) for compatibility with your interface.

Also, in line 1090-1120, "nr=100" determines the size of the relative file (number of records); "nt=15" is the number of test strings written to the file (it must be a multiple of 15); sl=41 is the record length of the first test file; and el=88 is the record length of the last test file (the entire test is performed using files with record lengths from 'sl' to 'el')

Lines 1880-2050 reset the drive, short new the disk, open a relative file, force creation of 'nr' empty records, and then write a unique identifying string to each 8-character field of every record, in the format

nnnn/ff*

where nnnn is a four-digit record number (with any leading zeroes) and ff is a two-digit field number (with any leading zero). Thus every record looks like this:

0123/01*0123/02*0123/03*0123/04*0123/05*012

(this is 43-character record #123), with a longer final field if the record length is not a multiple of 8.

Then the fun begins. . .three passes are made through the file.

Pass 1 selects a random field of a random record and tests to insure that the write (which goes to the end of the record) spans two sectors, then constructs a string to overwrite the selected record fields with the identifying string already there. (In literally over a million trials, we found that the initial write to the records always works. If you're skeptical, put a 'GOSUB 1600' in line 2060 to verify the contents of all records.) This pass then calls the position routine at line 1420, and the write is sent to the disk. A second write is sent to the next record, which lies entirely in the sector where the first write ended; and a third to a record lying entirely in the next sector in the file.

Pass 1 will produce an error on every third write, corrupting one or two records and leaving the "updated" record untouched. It may write the same series of three more than once during the pass: a detailed report is sent to the printer for study.

The first (identifying) field of each re-write, the number of the sector (in file sequence) and the initial byte (2-255) of the write, are stored in an array in the order written. On completion of nt/3 sets, the entire file is read by the subroutine at line 1610; and on detection of a variance, this array is sent to the printer from line 1510 followed by a report on the corrupted record (its number within the file, the starting sector and byte) and the actual contents from the disk. Subsequent variances are also printed with their identifying data: this information enables you to see exactly what was overwritten, by which write in which set of three; as well as what might have been restored by a later write and any duplicated sets (duplication confuses the error count). The printer output is formatted to produce a one-page report on each record size (two if needed).

Shiloh's Raid

We have been able to develop a short subroutine to anticipate the bug and apply a fix only when it is needed — less than 1% of the time — and otherwise use the position command as already described, without the 30%-40% time penalty. This subroutine is situated in lines 1380 through 1470 and includes the usual position routine and a variation on Dr. Neufeld's "point twice and wait" fix, which it selectively incorporates.

Line 1380 is the write entry point: if the immediately previous call to the position routine spanned two sectors, then it identifies the second and jeopardized sectors arising from that call and sets a counter to be active during the next two accesses. Line 1390 (the read entry point since reads do not need protection but do need to set a flag) calculates the end position of the current record within the record sector and, if a split record, the start position; and flags a split-write condition when the current access spans two sectors. This is the flag detected during the next position call in Line 1380. Line 1420 (the "index search" entry point, when a single character is to be retrieved for a search comparison, since a single-character retrieval cannot span two sectors) calculates the high and low bytes of the record number; and if a jeopardy flag has been set up by one of the two previous calls to the position routine, checks the sector of the current access against the sectors identified in line 1380; pointing once and setting up the wait

flag when an endangered sector is being accessed. Line 1450 sends the position command and, if the wait flag is set, waits 30 jiffies before returning from Line 1470.

Pass 2 performs exactly like Pass 1 except that it calls Shiloh's Raid at line 1380 and produces no errors.

Pass 3 makes 20*nt random selections, not writing a sequence of records unless they occur as a result of the random selection, and counts the number of times (1) that a flagged condition arises and (2) that a full fix is required. Although actual relative file use is not usually as random as this, the 1–2–3 sequence of passes 1 and 2 is just as untypical in the opposite direction. Pass 3 does, however, give some idea of how often Shiloh's Raid calls the delay fix, sending the count to the printer at the end of the pass. Our results depended on the size of the file: fewer waits with larger files, 0.08% in half a million accesses of disk–sized (664–block) files.

The time involved in the flagging algorithm also varied with the size of the file. Calls to Shiloh's Raid cost from 0.039 seconds per call for larger files to 0.048 seconds per call for smaller files: smaller files more often randomly encountered the flag conditions. Enlarge the file and change the subroutine call for Pass 3 in line 2210, and you will get an idea of how often C–64/1541 users encounter this bug: since it bites on 100% of these occasions, the two–jiffy price of reliability is low.

Dr. Neufeld's fix — point a second time and wait half a second — forces DOS to look at the active buffer, where it finds the wrong sector, writes that (previously changed) sector back to the disk, and then fetches the correct sector. The wait is necessary because without it, an immediately following PRINT# command causes an ATN interrupt that is waiting (with a higher IRQ priority than the fetch job) to take over when the DOS comes back from writing the old sector, before the fetch job is put in either the job queue or the buffer's track and sector pointers. The write is performed to the buffer, the buffer dirty flag is set, the poisoned sector is written over the last write–to–disk with the mis–directed information, and then the correct sector is fetched from the disk into the buffer. . . but too late.

Although the position command is entirely reliable for reading from the file, the bug may bite on a write that follows a read access, making the detection algorithm necessary on read accesses since it flags a condition about to arise. Shiloh's Raid still allows retrieval to the screen of an 85–character record in an average 1.17 seconds from a disk–sized file.

With Shiloh's Raid in place, the position command is 100% reliable. Now, perhaps CBM will consider an upgrade chip, since the 1541 outsold their wildest expectations and is still selling: I'd prefer that to a shiny new plastic face. I need three. . . just send them to me at PO Box 10976, Eugene OR 97440, and I'll express my complete surprise and profound astonishment in an appropriate fashion. . .

Shiloh's Raid: The Program

```
CN 1000 rem*******************************
JN 1010 rem*          " Shiloh's Raid "      *
DH 1020 rem*      this program demontrates   *
MJ 1030 rem*      the 1541 relative file bug,*
II 1040 rem*      and gives an efficient way *
GE 1050 rem*          to work around it.     *
GH 1060 rem*        (c) 1986 david shiloh    *
IB 1070 rem*******************************
MK 1080 :
NM 1090 nr  = 100:rem* number of records
NA 1100 nt  = 15 :rem* number of writes
DK 1110 sl  = 41 :rem* start record length
BI 1120 el  = 88 :rem* end record length
LL 1130 sd  = 8  :rem* first drive number
JH 1140 nd  = 1  :rem* number of drives
KN 1150 ed=sd+nd-1
MP 1160 :
PD 1170 gosub 1710: rem* initial prompts
CH 1180 goto 1810: rem* continue main routine
PM 1190 rem* subroutines follow
EC 1200 :
AP 1210 rem** create formatted output **
LP 1220 r$(ct)=left$(r$,7)+";"
IH 1230 r$(ct)=r$(ct)+right$("  "
        +str$(q%+1+(l>q)),3)+":"
EL 1240 r$(ct)=r$(ct)+left$(mid$(str$(q-l+p
        +1-(q-l+p<1)*254),2)+" [3 spcs]",4)
OP 1250 return
AG 1260 :
FI 1270 rem** create record contents **
JJ 1280 r$="": n$=right$(z$+mid$(str$(n),2),4)
JL 1290 for fs=f to nf+1
EE 1300 fs$=z$+mid$(str$(fs),2)
BF 1310 r$=r$+n$+"/"+right$(fs$,2)+"*"
MC 1320 next
FM 1330 r$=left$(r$,l-8*(f-1))
IF 1340 return
KL 1350 :
DA 1360 rem** shiloh's raid subroutine **
GD 1370 rem  (write relative record)
PM 1380 if sr then r1=sr+1: r2=sr+2: r=2
LO 1390 q=n*l: q%=q/254: q=q-q%*254
        : sr=q%*-(l>q)
AH 1400 if sr then sr=q%*-(q-l+p<1)
JC 1410 rem* entry point for no-fix write
PL 1420 h%=n/pg: lo=n-h%*pg
FH 1430 rem point twice & wait if needed
IC 1440 if r then r=r-1: rs=rs+r: if q%=r1 or q%=r2
        then gosub 1450: w=162
CP 1450 print#1,"pB" chr$(lo)chr$(h%)chr$(p);
GH 1460 if w then poke w,2: wait w,32: w=0: c=c+1
KN 1470 return
MD 1480 :
```

```
NG 1490 rem** print bad record message **
MN 1500 if e goto 1540
GG 1510 print#7,r$(0)
NA 1520 for t=1 to nt+1: print#7,r$(t);: next
EA 1530 print#7: x=x+nt/5+3
LK 1540 e=e+1: q=(n-1)*l+1: q%=q/254
        : q=q-q%*254
BP 1550 if n<>sn then print#7," record " n " sector "
        q%+1 " byte " q+1: te=te+1: x=x+1
OJ 1560 sn=n+1: print#7,ck$: x=x+1-(l>80)
BA 1570 if ps<3 then gosub 1420: print#2,r$;: n=n-1
IE 1580 return
KK 1590 :
OJ 1600 rem** read and check all records
DG 1610 print: p=1: f=1: e=0: te=0
AA 1620 for n=1 to nr: print"     reading";n
MD 1630 gosub 1280: gosub 1420
IL 1640 input#2,ck$: if ck$<>r$ then gosub 1500
GH 1650 next
MK 1660 print#7," j" r$(0)te " errors in " e " records, "
        rs " calls, " c " to wait routine"
AG 1670 print" qpass";ps;":";te;" bad to";e;
        " records";rs;" calls";c
MK 1680 return
OA 1690 :
EJ 1700 rem** print initial prompts   **
HO 1710 print" qnh Output to (S)creen or (P)rinter ?"
KH 1720 get a$: if a$<>"p" and a$<>"s" goto 1720
NF 1730 sp=3: if a$="p" then sp=4
KG 1740 print" Insert a scratch disk and
        press RETURN."
ME 1750 get a$: if a$<>chr$(13) goto 1750
MP 1760 return
OF 1770 :
IG 1780 rem************************
CN 1790 rem** mainline follows: ***
MH 1800 rem************************
HH 1810 pg=256: l$=chr$(157): s=rnd(-ti): d=sd
IH 1820 open 7,sp,7: rem printout file
GB 1830 z$="000": dim r$(nt+1)
        : r$(nt+1)=" jerrors:"
EK 1840 :
HC 1850 rem- do for all record lengths -
JC 1860 for l=sl to el
OM 1870 kn=254/l
JA 1880 rem- reset drive -
AB 1890 close1: open1,d,15,"ui": for t=1 to 500: next t
ID 1900 b=int(nr*l/254)+1: n=nr: nf=int(l/8): f=1: p=1
KO 1910 :
DB 1920 rem- new disk & open rel file -
CO 1930 x$="0:test"+str$(l): print#1,"n" x$
LH 1940 close2: open 2,d,2,x$+",l,"+chr$(l): ps=0
        : x=0
GO 1950 print" Sqqq Shiloh's Raid: Relative File
        Bug Spray"
GF 1960 print"          (c) 1986 by David Shiloh"
DJ 1970 print" qq test" l;l$" x" mid$(str$(nf),2)nr;b
        " sectors" nt " test sq "
AD 1980 :
OO 1990 rem- initialize all records -
CJ 2000 for t=0 to nt: r$(t)="": next
PB 2010 print#7," jtest" l;nr " records" nf " fields" b
        " sectors" nt " re-writes j "
CI 2020 print" setting up the file. . .": gosub 1420
        : print#2
EJ 2030 for n=1 to nr: gosub 1280
AF 2040 print"     writing " left$(r$,20)" . . .Q"
        : gosub 1420
NC 2050 print#2,r$;: next
OD 2060 print
AG 2070 rem- write random records -
AN 2080 for ps=1 to 3: rem three passes
CJ 2090 r$(0) = " qpass" + str$(ps) + " re-writes:"
HI 2100 ne=0: c=0: rs=0: sr=0: print r$(0)
HM 2110 rem- write nt records -
AF 2120 for ct=1 to nt-(ps=3)*19*nt
EH 2130 if ne then n=n+1-(ne=2)*int(kn): goto 2180
LC 2140 n=int(rnd(1)*(nr-kn)+1): f=int(rnd(1)*nf+1)
        : p=8*f-7
GG 2150 if ps=3 goto 2190
JK 2160 gosub 1390: if sr=0 goto 2140
OA 2170 sr=0
KB 2180 ne=ne+1: if ne>2 then ne=0
HH 2190 gosub 1280: print"     writing " left$(r$,7);ct
II 2200 rem* write rec with or w/o "raid"
HJ 2210 on ps gosub 1420, 1380, 1380: print#2,r$;
LC 2220 if ps<3 then gosub 1220
DN 2230 next ct
IN 2240 gosub 1610:rem verify written records
KA 2250 next ps
IE 2260 :
CP 2270 r$="full wait in" + str$(int(50*c/nt)/10)
ID 2280 r$=r$+"%"+str$(nt*20)+" pass 3
        accesses"
DA 2290 print r$: print#7,r$
MO 2300 rem -page printer & do next file-
MB 2310 for t=x to 55-66*(x>54): print#7: next t
OG 2320 d=d+1: if d>ed then d=sd: rem for
        multiple drives
CA 2330 next l
NO 2340 close 1: close 7
OC 2350 end
```