

[illegible]

```
CCCCCCCC HH HH AAAAAA RRRRRRRR GGGGGGGG EEEEEEEEE QQQQQQ
CCCCCCCC HH HH AAAAAA RRRRRRRR GGGGGGGG EEEEEEEEE QQQQQQ
CC        HH HH AA AA RR RR GG GG EEEEEEEEE QQ QQ
CC        HH HH AA AA RR RR GG GG EEEEEEEEE QQ QQ
CC        HH HH AA AA RR RR GG GG EEEEEEEEE QQ QQ
CC        HH HH AA AA RR RR GG GG EEEEEEEEE QQ QQ
CC        HHHHHHHHHH AA AA RRRRRRRR GG GG EEEEEEEEE QQ QQ
CC        HHHHHHHHHH AA AA RRRRRRRR GG GG EEEEEEEEE QQ QQ
CC        HH HH AAAAAAAAAA RR RR GGGGGG GG GGGGGG EE EEEEEEEEE QQ QQ
CC        HH HH AAAAAAAAAA RR RR GGGGGG GG GGGGGG EE EEEEEEEEE QQ QQ
CC        HH HH AA AA RR RR GG GG GGGGGG GG GGGGGG EE EEEEEEEEE QQ QQ
CC        HH HH AA AA RR RR GG GG GGGGGG GG GGGGGG EE EEEEEEEEE QQ QQ
CCCCCCCC HH HH AA AA RR RR GGGGGG GGGGGG EEEEEEEEE QQQQ QQ
CCCCCCCC HH HH AA AA RR RR GGGGGG GGGGGG EEEEEEEEE QQQQ QQ
.....
.....
.....
.....
```

```
LL        IIIIII SSSSSSSS
LL        IIIIII SSSSSSSS
LL        II     SS
LL        II     SS
LL        II     SS
LL        II     SS
LL        II     SSSSSS
LL        II     SSSSSS
LL        II     SS
LL        II     SS
LL        II     SS
LL        II     SS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS
```


K 4
15-Sep-1984 23:56:13
14-Sep-1984 12:30:09VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11X.SRC]CHARGEQ.B32;1 Page 1 (1)

```
1 0001 0 MODULE CHARGEQ (  
2 0002 0 LANGUAGE (BLISS32),  
3 0003 0 IDENT = 'V04-000'  
4 0004 0 ) =  
5 0005 1 BEGIN  
6 0006 1  
7 0007 1  
8 0008 1 *****  
9 0009 1 *  
10 0010 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *  
11 0011 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *  
12 0012 1 * ALL RIGHTS RESERVED. *  
13 0013 1 *  
14 0014 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *  
15 0015 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *  
16 0016 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *  
17 0017 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *  
18 0018 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *  
19 0019 1 * TRANSFERRED. *  
20 0020 1 *  
21 0021 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *  
22 0022 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *  
23 0023 1 * CORPORATION. *  
24 0024 1 *  
25 0025 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *  
26 0026 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *  
27 0027 1 *  
28 0028 1 *  
29 0029 1 *****  
30 0030 1  
31 0031 1 ++  
32 0032 1  
33 0033 1 FACILITY: F11ACP Structure Level 2  
34 0034 1  
35 0035 1 ABSTRACT:  
36 0036 1  
37 0037 1 This module contains the routines for charging disk blocks  
38 0038 1 against a particular quota file entry.  
39 0039 1  
40 0040 1 ENVIRONMENT:  
41 0041 1  
42 0042 1 STARLET operating system, including privileged system services  
43 0043 1 and internal exec routines.  
44 0044 1  
45 0045 1 --  
46 0046 1  
47 0047 1  
48 0048 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 22-May-1979 20:51  
49 0049 1  
50 0050 1 MODIFIED BY:  
51 0051 1  
52 0052 1 V03-012 CDS0004 Christian D. Saether 29-Aug-1984  
53 0053 1 Be prepared to find multiple headers when rebuilding  
54 0054 1 the quota file fcb. Reread header for PRIMARY_FCB  
55 0055 1 when rebuilding the quota fcb if it is not the quota  
56 0056 1 file fcb.  
57 0057 1
```

```

58 0058 1 V03-011 CDS0003 Christian D. Saether 23-Aug-1984
59 0059 1 Check quota fcb for staleness and rebuild if necessary.
60 0060 1
61 0061 1 V03-010 ACG0443 Andrew C. Goldstein, 21-Aug-1984 19:51
62 0062 1 Fix setup of REAL_Q_REC in file search so removal works
63 0063 1 on a cache miss.
64 0064 1
65 0065 1 V03-009 ACG0438 Andrew C. Goldstein, 19-Jul-1984 16:45
66 0066 1 Implement write access cache interlock
67 0067 1
68 0068 1 V03-008 ACG0430 Andrew C. Goldstein, 31-May-1984 15:07
69 0069 1 Fix reference to quota cache value block in REL_QUOTA_LOCK
70 0070 1
71 0071 1 V03-007 ACG0429 Andrew C. Goldstein, 21-May-1984 12:00
72 0072 1 Fix flow bug in ACG0428
73 0073 1
74 0074 1 V03-006 ACG0428 Andrew C. Goldstein, 18-May-1984 14:29
75 0075 1 Re-read quota record if value block not valid
76 0076 1
77 0077 1 V03-005 ACG0408 Andrew C. Goldstein, 23-Mar-1984 14:40
78 0078 1 Add AST parameter so that impure storage is fully based
79 0079 1
80 0080 1 V03-004 ACG0400 Andrew C. Goldstein, 1-Mar-1984 21:09
81 0081 1 Implement cluster-wide quota cacheing
82 0082 1
83 0083 1 V03-003 CDS0002 Christian D. Saether 29-Dec-1983
84 0084 1 Use L_NORM linkage and BIND_COMMON macro.
85 0085 1
86 0086 1 V03-002 CDS0001 Christian D. Saether 6-Dec-1983
87 0087 1 Serialize quota checking operations using allocation lock.
88 0088 1
89 0089 1 V03-001 ACG0337 Andrew C. Goldstein, 16-May-1983 16:04
90 0090 1 Fix handling of quota cache counters
91 0091 1
92 0092 1 V02-006 ACG0229 Andrew C. Goldstein, 23-Dec-1981 21:45
93 0093 1 Add counters for quota cache hits and misses
94 0094 1
95 0095 1 V02-005 ACG0167 Andrew C. Goldstein, 16-Apr-1980 19:25
96 0096 1 Previous revision history moved to F11B.REV
97 0097 1 **
98 0098 1
99 0099 1
100 0100 1 LIBRARY 'SYS$LIBRARY:LIB.L32';
101 0101 1 REQUIRE 'SRC$FCPDEF.B32';
102 1092 1
103 1093 1 FORWARD ROUTINE
104 1094 1 CHARGE_QUOTA : L_NORM NOVALUE, : check and/or charge disk blocks
105 1095 1 SEARCH_QUOTA : L_NORM, : search for a quota file record
106 1096 1 WRITE_QUOTA : L_NORM NOVALUE, : write back a quota record
107 1097 1 SCAN_QUO_CACHE : L_NORM, : search the quota cache
108 1098 1 GET_QUOTA_LOCK : L_NORM NOVALUE, : acquire lock on quota file entry
109 1099 1 REL_QUOTA_LOCK : L_NORM NOVALUE, : release lock on quota file entry
110 1100 1 CLEAN_QUO_CACHE : L_NORM NOVALUE, : write modified cache entry
111 1101 1 ENTER_QUO_CACHE : L_NORM NOVALUE, : make a new cache entry
```



```
113 1102 1 GLOBAL ROUTINE CHARGE_QUOTA (UIC, BLOCK COUNT, FLAGS) : L_NORM NOVALUE =
114 1103 1
115 1104 1 !++
116 1105 1
117 1106 1 FUNCTIONAL DESCRIPTION:
118 1107 1
119 1108 1 This routine locates the quota file entry identified by the UIC
120 1109 1 given and checks and/or charges the indicated number of blocks,
121 1110 1 as specified by the flags.
122 1111 1
123 1112 1 CALLING SEQUENCE:
124 1113 1 CHARGE_QUOTA (ARG1, ARG2, ARG3)
125 1114 1
126 1115 1 INPUT PARAMETERS:
127 1116 1 ARG1: UIC of entry to charge
128 1117 1 ARG2: number of blocks to charge (negative to credit)
129 1118 1 ARG3: bit encoded flags
130 1119 1 bit 0 set to check if quota will be exceeded
131 1120 1 bit 1 set to actually charge blocks to the quota entry
132 1121 1
133 1122 1 IMPLICIT INPUTS:
134 1123 1 IO_PACKET: user's I/O packet
135 1124 1 CURRENT_RVN: RVN of volume
136 1125 1
137 1126 1 OUTPUT PARAMETERS:
138 1127 1 NONE
139 1128 1
140 1129 1 IMPLICIT OUTPUTS:
141 1130 1 NONE
142 1131 1
143 1132 1 ROUTINE VALUE:
144 1133 1 NONE
145 1134 1
146 1135 1 SIDE EFFECTS:
147 1136 1 quota file modified
148 1137 1
149 1138 1 --
150 1139 1
151 1140 1
152 1141 2 BEGIN
153 1142 2
154 1143 2 MAP
155 1144 2 FLAGS : BITVECTOR; ! flags argument
156 1145 2
157 1146 2 LABEL
158 1147 2 CHECK_QUOTA; ! block of code to check quota
159 1148 2
160 1149 2 LOCAL
161 1150 2 SAVE_RVN, ! place to save current RVN
162 1151 2 Q_RECORD : REF BBLOCK; ! address of quota file record
163 1152 2
164 1153 2 BIND_COMMON;
165 1154 2
166 1155 2 EXTERNAL ROUTINE
167 1156 2 SWITCH_VOLUME : L_NORM; ! switch to desired RVN
168 1157 2
169 1158 2
```

```
170 1159 2 ! Save the current RVN and then switch context to the root RVN.
171 1160 2 ! First locate the quota file record. If there is no quota file enabled, this
172 1161 2 ! routine is a NOP.
173 1162 2 !
174 1163 2 !
175 1164 2 SAVE_RVN = .CURRENT_RVN;
176 1165 2 SWITCH_VOLUME (1);
177 1166 2 !
178 1167 2 CHECK_QUOTA: BEGIN
179 1168 2 !
180 1169 2 Q_RECORD = SEARCH_QUOTA (.UIC, 0, 0, 1);
181 1170 2 IF .Q_RECORD EQL =1 THEN LEAVE CHECK_QUOTA;
182 1171 2 !
183 1172 2 ! Check for quota exceeded if requested and the user does not have EXQUOTA
184 1173 2 ! privilege. If we are to check, lack of a quota record is an error; if
185 1174 2 ! we do not check, this routine is a NOP.
186 1175 2 !
187 1176 2 !
188 1177 2 IF .FLAGS[QUOTA_CHECK]
189 1178 2 AND NOT .BBLOCK[BBLOCK [.IO_PACKET[IRP$L_ARB], ARB$Q_PRIV], PRV$V_EXQUOTA]
190 1179 2 THEN
191 1180 2 BEGIN
192 1181 2 IF .Q_RECORD EQL 0
193 1182 2 THEN ERR_EXIT (SS$_EXDISKQUOTA);
194 1183 2 IF .Q_RECORD[DQF$L_USAGE] + .BLOCK_COUNT GTRU .Q_RECORD[DQF$L_PERMQUOTA]
195 1184 2 THEN
196 1185 2 BEGIN
197 1186 2 IF .CURRENT_WINDOW NEQ 0
198 1187 2 THEN
199 1188 2 BEGIN
200 1189 2 IF .CURRENT_WINDOW[WCB$V_OVERDRAWN]
201 1190 2 THEN
202 1191 2 BEGIN
203 1192 2 IF .Q_RECORD[DQF$L_USAGE] + .BLOCK_COUNT GTRU
204 1193 2 .Q_RECORD[DQF$L_PERMQUOTA] + .Q_RECORD[DQF$L_OVERDRAFT]
205 1194 2 THEN ERR_EXIT (SS$_EXDISKQUOTA)
206 1195 2 ELSE ERR_STATUS (SS$_OVRDSKQUOTA);
207 1196 2 END
208 1197 2 ELSE
209 1198 2 BEGIN
210 1199 2 CURRENT_WINDOW[WCB$V_OVERDRAWN] = 1;
211 1200 2 ERR_EXIT (SS$_EXDISKQUOTA);
212 1201 2 END;
213 1202 2 END
214 1203 2 ELSE
215 1204 2 ERR_EXIT (SS$_EXDISKQUOTA);
216 1205 2 END;
217 1206 2 END
218 1207 2 !
219 1208 2 ELSE
220 1209 2 IF .Q_RECORD EQL 0 THEN LEAVE CHECK_QUOTA;
221 1210 2 !
222 1211 2 ! If the record is to be charged, do so. Check the result to see if it
223 1212 2 ! is negative; if so, zero it to prevent absurd results.
224 1213 2 !
225 1214 2 !
226 1215 2 IF .FLAGS[QUOTA_CHARGE]
```



```

: 227      1216 3 THEN
: 228      1217 4 BEGIN
: 229      1218 4 Q_RECORD[DQF$L_USAGE] = Q_RECORD[DQF$L_USAGE] + .BLOCK_COUNT;
: 230      1219 4 IF Q_RECORD[DQF$L_USAGE] [SS 0
: 231      1220 4 THEN Q_RECORD[DQF$L_USAGE] = 0;
: 232      1221 4 WRITE_QUOTA (.Q_RECORD);
: 233      1222 3 END;
: 234      1223 3
: 235      1224 2 END;
: 236      1225 2
: 237      1226 2 SWITCH_VOLUME (.SAVE_RVN);
: 238      1227 2
: 239      1228 1 END;

```

! end of block CHECK_QUOTA

! end of routine CHARGE_QUOTA

				.TITLE	CHARGEQ	
				.IDENT	\V04-000\	
				.EXTRN	SWITCH_VOLUME	
				.PSECT	\$CODE\$,NOWRT,2	
				.ENTRY	CHARGE_QUOTA, Save R2,R3	1102
	53	A0	AA D0 00002	MOVL	-96(BASE), SAVE_RVN	1164
			01 DD 00006	PUSHL	#1	1165
0000G	CF		01 FB 00008	CALLS	#1, SWITCH_VOLUME	
			01 DD 0000D	PUSHL	#1	1169
			7E 7C 0000F	CLRQ	-(SP)	
		04	AC DD 00011	PUSHL	UIC	
	0000V	CF	04 FB 00014	CALLS	#4, SEARCH_QUOTA	
FFFFFFFF	8F		50 D1 00019	CMPL	Q_RECORD, #-1	1170
			62 13 00020	BEQL	6\$	
	44	0C	AC E9 00022	BLBC	FLAGS, 3\$	1177
	51	90	AA D0 00026	MOVL	-112(BASE), R1	1178
3B	58	B1	13 E0 0002A	BBS	#19, @88(R1), 3\$	
			50 D5 0002F	TSTL	Q_RECORD	1181
			32 13 00031	BEQL	2\$	
52	08	A0	AC C1 00033	ADDL3	BLOCK_COUNT, 8(Q_RECORD), R2	1183
	0C	A0	52 D1 00039	CMPL	R2, 12(Q_RECORD)	
			2F 1B 0003D	BLEQU	4\$	
	51	0C	AA D0 0003F	MOVL	12(BASE), R1	1186
			20 13 00043	BEQL	2\$	
17	0B	A1	04 E1 00045	BBC	#4, 11(R1), 1\$	1189
51	0C	A0	A0 C1 0004A	ADDL3	16(Q_RECORD), 12(Q_RECORD), R1	1193
		51	52 D1 00050	CMPL	R2, R1	
			10 1A 00053	BGTRU	2\$	
	15	80	AA E9 00055	BLBC	-123(BASE), 4\$	1195
	80	AA	0669 8F B0 00059	MOVW	#1641, -128(BASE)	
			0D 11 0005F	BRB	4\$	1189
	0B	A1	10 88 00061 1\$:	BISB2	#16, 11(R1)	1199
		03EC	8F BF 00065 2\$:	CHMU	#1004	1204
			04 00069	RET		
			50 D5 0006A 3\$:	TSTL	Q_RECORD	1209
			16 13 0006C	BEQL	6\$	
11	0C	AC	01 E1 0006E 4\$:	BBC	#1, FLAGS, 6\$	1215
	0B	A0	AC C0 00073	ADDL2	BLOCK_COUNT, 8(Q_RECORD)	1218
			03 18 00078	BGEQ	5\$	1219

CHARGEQ
V04-000

C 5
15-Sep-1984 23:56:13
14-Sep-1984 12:30:09

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11X.SRC]CHARGEQ.B32;1 Page 6
(2)

		08	A0	D4	0007A	CLRL	8(Q RECORD)	:	1220
			50	DD	0007D	PUSHL	Q RECORD	:	1221
0000V	CF		01	FB	0007F	CALLS	#T, WRITE_QUOTA	:	
			53	DD	00084	PUSHL	SAVE RVN	:	1226
0000G	CF		01	FB	00086	CALLS	#1, SWITCH_VOLUME	:	
			04		0008B	RET		:	1228

; Routine Size: 140 bytes, Routine Base: \$CODE\$ + 0000

CH
VO


```
1229 1 GLOBAL ROUTINE SEARCH_QUOTA (UIC, FLAGS, START_REC, USE_CACHE) : L_NORM =
1230 1
1231 1 ++
1232 1
1233 1 FUNCTIONAL DESCRIPTION:
1234 1
1235 1     This routine searches the quota file for the specified UIC under
1236 1     control of the match flags.
1237 1
1238 1 CALLING SEQUENCE:
1239 1     SEARCH_QUOTA (ARG1, ARG2, ARG3, ARG4)
1240 1
1241 1 INPUT PARAMETERS:
1242 1     ARG1: UIC to search for
1243 1     ARG2: match control flags from FIB
1244 1     ARG3: record number at which to start
1245 1     ARG4: 1 to find record in the cache
1246 1           0 to unconditionally go to the quota file
1247 1
1248 1 IMPLICIT INPUTS:
1249 1     CURRENT_VCB: address of volume's VCB
1250 1     context set to RVN 1 of volume set
1251 1
1252 1 OUTPUT PARAMETERS:
1253 1     NONE
1254 1
1255 1 IMPLICIT OUTPUTS:
1256 1     QUOTA_RECORD: record number of found record
1257 1     FREE_QUOTA: record number of first free quota file entry
1258 1     QUOTA_INDEX: cache index of cache entry if found
1259 1     DUMMY_REC: filled in with cache contents if found
1260 1
1261 1 ROUTINE VALUE:
1262 1     address of quota file record found, 0 if none, -1 if not quota file
1263 1
1264 1 SIDE EFFECTS:
1265 1     quota file read, contents of buffer cache altered
1266 1
1267 1 --
1268 1
1269 2 BEGIN
1270 2
1271 2 MAP
1272 2     FLAGS          : BITVECTOR;      ! match control flags
1273 2
1274 2 LITERAL
1275 2     ALL_GROUP      = $BITPOSITION (FIB$V_ALL_GRP),
1276 2     ALL_MEMBER     = $BITPOSITION (FIB$V_ALL_MEM),
1277 2     RECS_PER_BLOCK = 512 / DQF$C_LENGTH;
1278 2
1279 2 LABEL
1280 2     QUOTA_SCAN;          ! search quota file
1281 2
1282 2 LOCAL
1283 2     FCB             : REF BBLOCK,    ! address of quota file FCB
1284 2     QUOTA_CACHE     : REF BBLOCK,    ! address of quota cache
1285 2     QUOTA_LIST      : REF BBLOCKVECTOR [,VCA$C_QUOLENGTH],
```

```
.. 298      1286      2      J      ! address of quota cache entries
.. 299      1287      2      REC_NUM, ! index into quota cache
.. 300      1288      2      FIRST_REC, ! quota file record to read
.. 301      1289      2      VBN,      ! first record in block to use
.. 302      1290      2      Q_RECORD   : REF BBLOCK; ! block number of quota file
.. 303      1291      2      ! address of quota file record
.. 304      1292      2
.. 305      1293      2      BIND_COMMON;
.. 306      1294      2
.. 307      1295      2      EXTERNAL ROUTINE
.. 308      1296      2      ALLOCATION_LOCK : L_NORM,      ! allocation lock serialization
.. 309      1297      2      BUILD_EXT_FCBS : L_NORM NOVALUE, ! build extension fcb chain.
.. 310      1298      2      SERIAL_FILE   : L_NORM,      ! get serialization lock
.. 311      1299      2      REBLD PRIM_FCB : L_NORM NOVALUE, ! rebuild fcb from header
.. 312      1300      2      READ_HEADER    : L_NORM,      ! read file header
.. 313      1301      2      RELEASE_SERIAL_LOCK : L_NORM NOVALUE, ! release serialization lock
.. 314      1302      2      READ_BLOCK     : L_NORM;      ! read a disk block
.. 315      1303      2
.. 316      1304      2
.. 317      1305      2      ! Get the FCB address for the quota file. If none, take an error exit.
.. 318      1306      2      !
.. 319      1307      2
.. 320      1308      2      FCB = .CURRENT_VCB[VCB$QUOTAFCB];
.. 321      1309      2      IF .FCB EQL 0 THEN RETURN -1;
.. 322      1310      2
.. 323      1311      2      ! Serialize quota search using allocation lock.
.. 324      1312      2      !
.. 325      1313      2
.. 326      1314      2      ALLOCATION_LOCK ();
.. 327      1315      2
.. 328      1316      2      ! Check to see if the quota fcb is stale, that is, it has been modified
.. 329      1317      2      ! on another node. If so, serialize on the quota file itself, read
.. 330      1318      2      ! the header and rebuild the fcb.
.. 331      1319      2      !
.. 332      1320      2
.. 333      1321      2      IF .FCB [FCB$V_STALE]
.. 334      1322      2      THEN
.. 335      1323      2          BEGIN
.. 336      1324      2              LOCAL
.. 337      1325      2                  HEADER,
.. 338      1326      2                  SAV_CURRLCKINDX;
.. 339      1327      2
.. 340      1328      2              SAV_CURRLCKINDX = .CURR_LCKINDX;
.. 341      1329      2
.. 342      1330      2              SERIAL_FILE (FCB [FCB$W_FID]);
.. 343      1331      2
.. 344      1332      2      ! Setting this flag prevents READ_HEADER from modifying FILE HEADER.
.. 345      1333      2      ! It is also set when the BUILD_EXT_FCBS routine calls READ_HEADER
.. 346      1334      2      ! if BUILD_EXT_FCBS is called with the optional fcb argument.
.. 347      1335      2      !
.. 348      1336      2
.. 349      1337      2      STSFLGS [STS_LEAVE_FILEHDR] = 1;
.. 350      1338      2
.. 351      1339      2      HEADER = READ_HEADER (0, .FCB);
.. 352      1340      2
.. 353      1341      2      REBLD_PRIM_FCB (.FCB, .HEADER);
.. 354      1342      2
```



```
355 1343 3 BUILD_EXT_FCBS (.HEADER, .FCB);
356 1344 3
357 1345 3 IF .SAV_CURRLCKINDX NEQ .CURR_LCKINDX
358 1346 3 THEN
359 1347 4 BEGIN
360 1348 4 RELEASE_SERIAL_LOCK (.CURR_LCKINDX);
361 1349 4 CURR_LCKINDX = .SAV_CURRLCKINDX;
362 1350 4 END;
363 1351 3
364 1352 3 ! If PRIMARY_FCB is nonzero and not the quota file fcb, and also if
365 1353 3 ! it is the same lockbasis as the current lock index, then reread
366 1354 3 ! the header for it to re-establish FILE_HEADER.
367 1355 3
368 1356 3
369 1357 3 IF .PRIMARY_FCB NEQ 0
370 1358 3 THEN
371 1359 3 IF .PRIMARY_FCB NEQ .FCB
372 1360 3 AND .PRIMARY_FCB [FCB$LOCKBASIS] EQL .LB_BASIS [.CURR_LCKINDX]
373 1361 3 THEN
374 1362 3 READ_HEADER (0, .PRIMARY_FCB);
375 1363 3
376 1364 2 END;
377 1365 2
378 1366 2 ! If there are no wild cards in the search, scan the quota cache first.
379 1367 2 ! If the value block was lost, the cache entry comes back not valid,
380 1368 2 ! but with contents. In this case, and if this is a write-through operation,
381 1369 2 ! use the record number in the cache to read the record to save the search.
382 1370 2 ! As long as the record is read, also update it if the cache entry is dirty.
383 1371 2
384 1372 2
385 1373 2 REAL Q_REC = 0;
386 1374 2 QUOTA_CACHE = .CURRENT_VCB[VCB$QUOCACHE];
387 1375 2 QUOTA_LIST = QUOTA_CACHE[VCA$QUOOLIST];
388 1376 2 IF NOT .FLAGS[ALL_MEMBER] AND NOT .FLAGS[ALL_GROUP]
389 1377 2 THEN
390 1378 3 BEGIN
391 1379 3 J = SCAN_QUO_CACHE (.UIC, .USE_CACHE);
392 1380 3 IF .QUOTA_LIST[J-1, VCA$QUORECNUM] NEQ 0
393 1381 3 THEN
394 1382 4 BEGIN
395 1383 4 IF NOT .USE_CACHE
396 1384 4 OR NOT .QUOTA_LIST[J-1, VCA$QUOVALID]
397 1385 4 OR NOT .QUOTA_CACHE[VCA$QUOCACHEVALID]
398 1386 4 THEN
399 1387 5 BEGIN
400 1388 5 QUOTA_RECORD = .QUOTA_LIST[J-1, VCA$QUORECNUM];
401 1389 5 REC_NUM = .QUOTA_LIST[J-1, VCA$QUORECNUM] - 1;
402 1390 5 REAL_Q_REC = READ_BLOCK (.REC_NUM / RECS_PER_BLOCK
403 1391 5 + .FCB[FCB$ST[BN]], 1, QUOTA_TYPE)
404 1392 5 + (.REC_NUM MOD RECS_PER_BLOCK) * DQFSC_LENGTH;
405 1393 5 IF .QUOTA_LIST[J-1, VCA$QUOVALID]
406 1394 5 THEN
407 1395 5 CLEAN_QUO_CACHE (.J, .REAL_Q_REC)
408 1396 5 ELSE
409 1397 6 BEGIN
410 1398 6 ENTER_QUO_CACHE (.J, .REAL_Q_REC, 0, .USE_CACHE);
411 1399 6 CHSMOVE (DQFSC_LENGTH, .REAL_Q_REC, DUMMY_REC);
```

```

412 1400 5      END;
413 1401 4      END;
414 1402 4      RETURN DUMMY_REC;
415 1403 3      END;
416 1404 2      END;
417 1405 2
418 1406 2 ! We couldn't find a valid cache entry (either because it's not there
419 1407 2 ! or the operation won't allow it). Scan the blocks of the quota file,
420 1408 2 ! looking for a matching record.
421 1409 2
422 1410 2
423 1411 3 QUOTA_SCAN: BEGIN
424 1412 3
425 1413 3 FIRST_REC = .START_REC MOD RECS_PER_BLOCK;
426 1414 3 INCR VBN FROM .START_REC/RECS_PER_BLOCK TO .FCB[FCBSL_EFBLK] - 1
427 1415 3 DO
428 1416 4     BEGIN
429 1417 4     Q_RECORD = READ_BLOCK (.VBN + .FCB[FCBSL_STLBN],
430 1418 4     .FCB[FCBSL_EFBLK] - .VBN,
431 1419 4     QUOTA_TYPE)
432 1420 4     + .FIRST_REC * DQFSC_LENGTH;
433 1421 4
434 1422 4     INCR J FROM .FIRST_REC TO RECS_PER_BLOCK - 1
435 1423 4     DO
436 1424 5     BEGIN
437 1425 5     QUOTA_RECORD = .VBN * RECS_PER_BLOCK + .J + 1;
438 1426 5
439 1427 5     IF .Q_RECORD[DQF$V_ACTIVE]
440 1428 5     THEN
441 1429 6     BEGIN
442 1430 7     IF (.FLAGS[ALL_MEMBER] OR .UIC<00,16> EQL .(Q_RECORD[DQF$UIC])<00,16>)
443 1431 7     AND (.FLAGS[ALL_GROUP] OR .UIC<16,16> EQL .(Q_RECORD[DQF$UIC])<16,16>)
444 1432 6     THEN LEAVE QUOTA_SCAN;
445 1433 6     END
446 1434 6
447 1435 5     ELSE
448 1436 6     BEGIN
449 1437 6     IF .FREE_QUOTA EQL 0
450 1438 6     THEN FREE_QUOTA = .QUOTA_RECORD;
451 1439 5     END;
452 1440 5
453 1441 5     Q_RECORD = .Q_RECORD + DQFSC_LENGTH;
454 1442 4     END; ! end of inner loop
455 1443 4
456 1444 4     FIRST_REC = 0;
457 1445 3     END; ! end of block scan loop
458 1446 3
459 1447 3 IF NOT .FLAGS[ALL_MEMBER] AND NOT .FLAGS[ALL_GROUP]
460 1448 3 THEN REL_QUOTA_LOCK (.J);
461 1449 3 RETURN 0; ! return 0 if not found
462 1450 2 END; ! end of block QUOTA_SCAN
463 1451 2
464 1452 2 ! We have found a record in the quota file. If there were wild cards, now
465 1453 2 ! scan the quota cache to see if an entry is present. With wild cards, the
466 1454 2 ! file must be scanned first to be able to return the entries in a coherent
467 1455 2 ! order; yet we must look at the cache in case a modified entry is present.
468 1456 2
```



```

469      1457 2
470      1458 2 REAL Q_REC = .Q_RECORD;
471      1459 2 IF .FLAGS[ALL_MEMBER] OR .FLAGS[ALL_GROUP]
472      1460 2 THEN
473      1461 2 BEGIN
474      1462 3 J = SCAN_QUO_CACHE (.Q_RECORD[DQF$SL_UIC], 0);
475      1463 3 IF .QUOTA_LIST[J-1, V$ASV_QUOVALID]
476      1464 3 THEN
477      1465 4 BEGIN
478      1466 4 CLEAN_QUO_CACHE (.J, .Q_RECORD);
479      1467 4 RETURN DUMMY_REC;
480      1468 3 END;
481      1469 2 END;
482      1470 2
483      1471 2 ! Finally enter the new record into the quota cache.
484      1472 2 !
485      1473 2
486      1474 2 ENTER_QUO_CACHE (.J, .Q_RECORD, 0, .USE_CACHE);
487      1475 2 CHSMOVE (DQF$C_LENGTH, .Q_RECORD, DUMMY_REC);
488      1476 2 RETURN DUMMY_REC;
489      1477 2
490      1478 1 END;

```

```
! end of routine SEARCH_QUOTA
```

Address	Op Code	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418
---------	---------	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

				14	OC 13 00059	BEQL 2\$		
					AA DD 0005B	PUSHL 20(BASE)		1348
	0000G	CF			01 FB 0005E	CALLS #1, RELEASE_SERIAL_LOCK		
	14	AA			52 D0 00063	MOVL SAV CURRLCKINDX, 20(BASE)		1349
		51		08	AA D0 00067 2\$:	MOVL 8(BASE), R1		1357
					1B 13 0006B	BEQL 3\$		
		57			51 D1 0006D	CMLPL R1, FCB		1359
					16 13 00070	BEQL 3\$		
		50		14	AA D0 00072	MOVL 20(BASE), R0		1360
	0080	CA40		4C	A1 D1 00076	CMLPL 76(R1), 128(BASE)[R0]		
					09 12 0007D	BNEQ 3\$		
					51 DD 0007F	PUSHL R1		1362
					7E D4 00081	CLRL -(SP)		
	0000G	CF			02 FB 00083	CALLS #2, READ_HEADER		
					69 D4 00088 3\$:	CLRL (R9)		1373
		50		98	AA D0 0008A	MOVL -104(BASE), R0		1374
		52		5C	A0 D0 0008E	MOVL 92(R0), QUOTA_CACHE		
		56		44	A2 9E 00092	MOVAB 68(R2), QUOTA_LIST		1375
		03		08	AC E9 00096	BLBC FLAGS, 5\$		1376
					008F 31 0009A 4\$:	BRW 10\$		
					01 E0 0009D 5\$:	BBS #1, FLAGS, 4\$		
	F8	08	AC		AC DD 000A2	PUSHL USE_CACHE		1379
				10	AC DD 000A5	PUSHL UIC		
				04	02 FB 000A8	CALLS #2, SCAN_QUO_CACHE		
		58			50 D0 000AD	MOVL R0, J		
		58			1C C5 000B0	MULL3 #28, J, R0		1380
		50			56 C0 000B4	ADDL2 QUOTA_LIST, R0		
00		EC	A0		00 ED 000B7	CMPZV #0, #24, -20(R0), #0		
					6D 13 000BD	BEQL 10\$		
		08		10	AC E9 000BF	BLBC USE_CACHE, 6\$		1383
		04		EF	A0 E9 000C3	BLBC -17(R0), 6\$		1384
		5E		0B	A2 E8 000C7	BLBS 11(QUOTA_CACHE), 9\$		1385
00	BE	EC	A0		00 EF 000CB 6\$:	EXTZV #0, #24, -20(R0), @0(SP)		1388
		50			1C C5 000D2	MULL3 #28, J, R0		1389
52		EC	A046		00 EF 000D6	EXTZV #0, #24, -20(R0)[QUOTA_LIST], REC_NUM		
					52 D7 000DD	DECL REC_NUM		
					05 DD 000DF	PUSHL #5		1390
					01 DD 000E1	PUSHL #1		
		50			10 C7 000E3	DIVL3 #16, REC_NUM, R0		
		52		30	B740 9F 000E7	PUSHAB @48(FCB)[R0]		1391
					03 FB 000EB	CALLS #3, READ_BLOCK		
					01 7A 000F0	EMUL #1, REC_NUM, #0, -(SP)		1392
					10 7B 000F5	EDIV #16, (SP)+, R1, R1		
					20 C4 000FA	MULL2 #32, R1		
					51 C1 000FD	ADDL3 R1, R0, (R9)		
					1C C5 00101	MULL3 #28, J, R0		1393
					00 E1 00105	BBC #0, -17(R0)[QUOTA_LIST], 8\$		
					69 DD 0010B	PUSHL (R9)		1395
					58 DD 0010D 7\$:	PUSHL J		
					02 FB 0010F	CALLS #2, CLEAN_QUO_CACHE		
					13 11 00114	BRB 9\$		
				10	AC DD 00116 8\$:	PUSHL USE_CACHE		1398
					7E D4 00119	CLRL -(SP)		
					69 DD 0011B	PUSHL (R9)		
					58 DD 0011D	PUSHL J		
					04 FB 0011F	CALLS #4, ENTER_QUO_CACHE		
6B					20 28 00124	MOVCL #32, @0(R9), TR11)		1399

7E	00	0C	AC	00C1	31	00129	9\$:	BRW	21\$		1402
53	53		8E	01	7A	0012C	10\$:	EMUL	#1, START_REC, #0, -(SP)		1413
52		0C	AC	10	7B	00132		EDIV	#16, (SP)7, FIRST_REC, FIRST_REC		
			55	10	C7	00137		DIVL3	#16, START_REC, R2		1414
				3C	A7	D0	0013C	MOVL	60(FCB), R5		
					52	D7	00140	DECL	VBN		
					59	11	00142	BRB	17\$		
					05	DD	00144	11\$:	PUSHL	#5	1417
7E	3C	A7			52	C3	00146	SUBL3	VBN, 60(FCB), -(SP)		1418
				30	B742	9F	0014B	PUSHAB	@48(FCB)[VBN]		1417
	0000G	CF			03	FB	0014F	CALLS	#3, READ_BLOCK		
51		53			05	78	00154	ASHL	#5, FIRST_REC, R1		1420
54		50			51	C1	00158	ADDL3	R1, R0, Q_RECORD		
		50		FF	A3	9E	0015C	MOVAB	-1(R3), J		1422
					35	11	00160	BRB	16\$		
51		52			04	78	00162	12\$:	ASHL	#4, VBN, R1	1425
	00	BE		01	A041	9E	00166	MOVAB	1(J)[R1], @0(SP)		
		19			64	E9	0016C	BLBC	(Q_RECORD), 14\$		1427
		07		08	AC	E8	0016F	BLBS	FLAGS, 13\$		1430
	04	A4		04	AC	B1	00173	CMPW	UIC, 4(Q_RECORD)		
					1A	12	00178	BNEQ	15\$		
34	08	AC			01	E0	0017A	13\$:	BBS	#1, FLAGS, 18\$	1431
	06	A4		06	AC	B1	0017F	CMPW	UIC+2, 6(Q_RECORD)		
					0E	12	00184	BNEQ	15\$		
				02B8	2B	11	00186	BRB	18\$		1432
					CA	D5	00188	14\$:	TSTL	696(BASE)	1437
	02B8	CA		00	06	12	0018C	BNEQ	15\$		
		54			BE	D0	0018E	MOVL	@0(SP), 696(BASE)		1438
C7		50			20	C0	00194	15\$:	ADDL2	#32, Q_RECORD	1441
					0F	F3	00197	16\$:	AOBLEQ	#15, J, 12\$	1422
A3		52			53	D4	0019B	CLRL	FIRST_REC		1444
		4C			55	F2	0019D	17\$:	AOBLSS	R5, VBN, 11\$	1414
47	08	AC		08	AC	E8	001A1	BLBS	FLAGS, 22\$		1447
					01	E0	001A5	BBS	#1, FLAGS, 22\$		
					58	DD	001AA	PUSHL	J		1448
	0000V	CF			01	FB	001AC	CALLS	#1, REL_QUOTA_LOCK		
		69			3E	11	001B1	BRB	22\$		1449
		05			54	D0	001B3	18\$:	MOVL	Q_RECORD, (R9)	1458
1C	08	AC		08	AC	E8	001B6	BLBS	FLAGS, 19\$		1459
					01	E1	001BA	BBC	#1, FLAGS, 20\$		
					7E	D4	001BF	19\$:	CLRL	-(SP)	1462
	0000V	CF		04	A4	DD	001C1	PUSHL	4(Q_RECORD)		
		58			02	FB	001C4	CALLS	#2, SCAN_QUO_CACHE		
50		58			50	D0	001C9	MOVL	R0, J		
05	EF	A046			1C	C5	001CC	MULL3	#28, J, R0		1463
					00	E1	001D0	BBC	#0, -17(R0)[QUOTA_LIST], 20\$		
					54	DD	001D6	PUSHL	Q_RECORD		1466
				10	FF32	31	001D8	BRW	7\$		
					AC	DD	001DB	20\$:	PUSHL	USE_CACHE	1474
					7E	D4	001DE	CLRL	-(SP)		
					54	DD	001E0	PUSHL	Q_RECORD		
					58	DD	001E2	PUSHL	J		
6B	0000V	CF			04	FB	001E4	CALLS	#4, ENTER_QUO_CACHE		
		64			20	28	001E9	MOVCL3	#32, (Q_RECORD), (R11)		1475
		50			5B	D0	001ED	21\$:	MOVL	R11, R0	1476
					04	001F0	RET				
					50	D4	001F1	22\$:	CLRL	R0	1478

CHARGEQ
V04-000

K 5
15-Sep-1984 23:56:13
14-Sep-1984 12:30:09

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11X.SRC]CHARGEQ.B32;1
Page 14
(3)

04 001F3

RET

;

; Routine Size: 500 bytes, Routine Base: \$CODE\$ + 008C


```
492 1479 1 GLOBAL ROUTINE WRITE_QUOTA (Q_RECORD) : L_NORM NOVALUE =
493 1480 1
494 1481 1 ++
495 1482 1
496 1483 1 FUNCTIONAL DESCRIPTION:
497 1484 1
498 1485 1 This routine writes the indicated quota record. If a cache entry
499 1486 1 exists for the record being processed (indicated by the record
500 1487 1 being the dummy record), we update the cache entry. If we also
501 1488 1 have the real quota record in memory, then mark it for write-back.
502 1489 1
503 1490 1
504 1491 1 CALLING SEQUENCE:
505 1492 1 WRITE_QUOTA (ARG1)
506 1493 1
507 1494 1 INPUT PARAMETERS:
508 1495 1 ARG1: address of quota record
509 1496 1
510 1497 1 IMPLICIT INPUTS:
511 1498 1 REAL_Q_REC: buffer of real quota record if exists
512 1499 1 QUOTA_INDEX: cache index of cache entry
513 1500 1
514 1501 1 OUTPUT PARAMETERS:
515 1502 1 NONE
516 1503 1
517 1504 1 IMPLICIT OUTPUTS:
518 1505 1 NONE
519 1506 1
520 1507 1 ROUTINE VALUE:
521 1508 1 NONE
522 1509 1
523 1510 1 SIDE EFFECTS:
524 1511 1 quota cache modified, quota record marked for write-back
525 1512 1
526 1513 1 --
527 1514 1
528 1515 2 BEGIN
529 1516 2
530 1517 2 MAP
531 1518 2 Q_RECORD : REF BBLOCK; ! address of quota record
532 1519 2
533 1520 2 BIND_COMMON;
534 1521 2
535 1522 2 EXTERNAL ROUTINE
536 1523 2 MARK_DIRTY : L_NORM; ! mark buffer for write back
537 1524 2
538 1525 2
539 1526 2 ! If the specified record is the dummy record, there is a cache entry.
540 1527 2 ! Therefore, update it. Also update the associated real record if there
541 1528 2 ! is one.
542 1529 2
543 1530 2
544 1531 2 IF .Q_RECORD EQL DUMMY_REC
545 1532 2 THEN
546 1533 3 BEGIN
547 1534 3 ENTER_QUO_CACHE (.QUOTA_INDEX, .Q_RECORD, .REAL_Q_REC EQL 0, 2);
548 1535 3 IF .REAL_Q_REC NEQ 0
```

```

: 549      1536 3      THEN
: 550      1537 4      BEGIN
: 551      1538 4      CH$MOVE (DQF$C_LENGTH, .Q_RECORD, .REAL_Q_REC);
: 552      1539 4      MARK_DIRTY (.REAL_Q_REC);
: 553      1540 3      END;
: 554      1541 3      END
: 555      1542 3      ! Otherwise, if there is no cache entry, we just have to mark the
: 556      1543 3      ! buffer dirty.
: 557      1544 3      !
: 558      1545 3      !
: 559      1546 3      !
: 560      1547 2      ELSE
: 561      1548 2      MARK_DIRTY (.Q_RECORD);
: 562      1549 2
: 563      1550 1      END;

```

! end of routine WRITE_QUOTA

				.EXTRN MARK_DIRTY		
				.ENTRY	WRITE_QUOTA, Save R2,R3,R4,R5,R6	: 1479
56	02BC	CA	9E 00002	MOVAB	700(BASE), R6	: 1518
50	02C4	CA	9E 00007	MOVAB	708(BASE), R0	: 1531
50	04	AC	D1 0000C	CMPL	Q_RECORD, R0	
		24	12 00010	BNEQ	2\$	
		02	DD 00012	PUSHL	#2	: 1534
		7E	D4 00014	CLRL	-(SP)	
		66	D5 00016	TSTL	(R6)	
		02	12 00018	BNEQ	1\$	
		6E	D6 0001A	INCL	(SP)	
	04	AC	DD 0001C 1\$:	PUSHL	Q_RECORD	
	02C0	CA	DD 0001F	PUSHL	704(BASE)	
0000V	CF	04	FB 00023	CALLS	#4, ENTER_QUO_CACHE	
		66	D5 00028	TSTL	(R6)	: 1535
		12	13 0002A	BEQL	4\$	
00	B6	04	BC	MOVC3	#32, @Q_RECORD, @0(R6)	: 1538
		66	DD 00032	PUSHL	(R6)	: 1539
		03	11 00034	BRB	3\$	
		04	AC DD 00036 2\$:	PUSHL	Q_RECORD	: 1548
0000G	CF	01	FB 00039 3\$:	CALLS	#T, MARK_DIRTY	
		04	0003E 4\$:	RET		: 1550

; Routine Size: 63 bytes, Routine Base: \$CODE\$ + 0280


```
565 1551 1 ROUTINE SCAN_QUO_CACHE (UIC, MARK_USE) : L_NORM =
566 1552 1
567 1553 1 ++
568 1554 1
569 1555 1 FUNCTIONAL DESCRIPTION:
570 1556 1
571 1557 1 This routine scans the quota cache for the indicated UIC. If found,
572 1558 1 it returns the contents, and marks the entry used if requested.
573 1559 1
574 1560 1
575 1561 1 CALLING SEQUENCE:
576 1562 1 SCAN_QUO_CACHE (ARG1, ARG2)
577 1563 1
578 1564 1 INPUT PARAMETERS:
579 1565 1 ARG1: UIC to search for
580 1566 1 ARG2: 1 to record new use
581 1567 1 0 to not
582 1568 1
583 1569 1 IMPLICIT INPUTS:
584 1570 1 CURRENT_VCB: VCB of volume
585 1571 1
586 1572 1 OUTPUT PARAMETERS:
587 1573 1 NONE
588 1574 1
589 1575 1 IMPLICIT OUTPUTS:
590 1576 1 DUMMY_REC: receives contents of cache entry if found
591 1577 1 QUOTA_INDEX: receives index of cache entry found
592 1578 1 QUOTA_RECORD: quota file record number of found entry
593 1579 1
594 1580 1 ROUTINE VALUE:
595 1581 1 index of entry found
596 1582 1
597 1583 1 SIDE EFFECTS:
598 1584 1 quota cache entry modified
599 1585 1
600 1586 1 --
601 1587 1
602 1588 2 BEGIN
603 1589 2
604 1590 2 LITERAL
605 1591 2 RECS_PER_BLOCK = 512 / DQF$C_LENGTH;
606 1592 2
607 1593 2 LABEL
608 1594 2 QUOTA_SEARCH; ! body of search code
609 1595 2
610 1596 2 LOCAL
611 1597 2 QUOTA_CACHE : REF BBLOCK, ! address of quota cache
612 1598 2 QUOTA_LIST : REF BBLOCKVECTOR [, VCA$C_QUOLENGTH],
613 1599 2 ! address of quota cache entries
614 1600 2 J, ! index into quota cache
615 1601 2 LOWEST_LRU, ! oldest quota LRU index
616 1602 2 LOWEST_J, ! oldest quota cache entry index
617 1603 2 LRU_DELTA, ! LRU index of current entry
618 1604 2 OLD_RECORD : REF BBLOCK, ! address of old quota record
619 1605 2 REC_NUM, ! quota file record to read
620 1606 2 FCB : REF BBLOCK; ! address of quota file FCB
621 1607 2
```

```
1608 2 EXTERNAL
1609      PMSSGL_QUOHIT : ADDRESSING_MODE (GENERAL),
1610      count of quota cache hits
1611      PMSSGL_QUOMISS : ADDRESSING_MODE (GENERAL);
1612      count of quota cache misses
1613
1614 2 EXTERNAL ROUTINE
1615      CACHE_LOCK : L_NORM, ! acquire cache lock
1616      READ_BLOCK : L_NORM; ! read a disk block
1617
1618 2
1619 2 BIND_COMMON;
1620
1621 2
1622 2 ! If the cache is not currently marked valid, do so if possible.
1623 2 ! This involves taking out the cache lock if the volume is cluster
1624 2 ! accessible, and checking for quota file writers and a non-null
1625 2 ! cache size.
1626 2
1627 2
1628 2 QUOTA_CACHE = .CURRENT_VCB[VCB$L_QUOCACHE];
1629 2 FCB = .CURRENT_VCB[VCB$L_QUOTAFCB];
1630
1631 2 IF NOT .QUOTA_CACHE[VCASV_CACHEVALID]
1632 2 AND NOT .QUOTA_CACHE[VCASV_CACHEFLUSH]
1633 2 THEN
1634 2 BEGIN
1635 2 IF .QUOTA_CACHE[VCASW_QUOSIZE] GTRU 1
1636 2 AND NOT .BBLOCK [CURRENT_UCB[UCB$L_DEVCHAR], DEV$V_DMT]
1637 2 AND .FCB[FCB$W_WCNT] LEQ 1
1638 2 AND
1639 2 BEGIN
1640 2 IF .BBLOCK [CURRENT_UCB[UCB$L_DEVCHAR2], DEV$V_CLU]
1641 2 THEN CACHE_LOCK (.FCB[FCB$L_LOCKBASIS], QUOTA_CACHE[VCASL_QUOCLKID], 0)
1642 2 ELSE 1
1643 2 END
1644 2 THEN
1645 2 QUOTA_CACHE[VCASV_CACHEVALID] = 1
1646 2 ELSE
1647 2 QUOTA_CACHE[VCASV_CACHEFLUSH] = 1;
1648 2 END;
1649
1650 2 ! Search the quota cache for an active entry with a matching UIC.
1651 2
1652 2
1653 2 QUOTA_SEARCH: BEGIN
1654 2
1655 2 QUOTA_LIST = QUOTA_CACHE[VCASL_QUOLIST];
1656 2 INCR R FROM 1 TO .QUOTA_CACHE[VCASW_QUOSIZE]
1657 2 DO
1658 2 BEGIN
1659 2 IF .QUOTA_LIST[R-1, VCASL_QUORECNUM] NEQ 0
1660 2 AND .QUOTA_LIST[R-1, VCASL_QUOUIC] EQL .UIC
1661 2 THEN
1662 2 BEGIN
1663 2 IF .MARK_USE
1664 2 THEN
```



```

: 679      1665 6      BEGIN
: 680      1666 6      QUOTA_LIST[K-1, VCASW_QUOLRUX] = .QUOTA_CACHE[VCASW_QUOLRU];
: 681      1667 6      QUOTA_CACHE[VCASW_QUOLRU] = .QUOTA_CACHE[VCASW_QUOLRU] + 1;
: 682      1668 5      END;
: 683      1669 5      PMSSGL_QUOHIT = .PMSSGL_QUOHIT + 1;
: 684      1670 5      J = .K;
: 685      1671 5      LEAVE QUOTA_SEARCH;
: 686      1672 4      END;
: 687      1673 3      END;
: 688      1674 3
: 689      1675 3      ! We failed to find a match in the quota cache. Search the cache for a free
: 690      1676 3      ! entry, or, failing that, the entry with the oldest LRU index.
: 691      1677 3
: 692      1678 3
: 693      1679 3      PMSSGL_QUOMISS = .PMSSGL_QUOMISS + 1;
: 694      1680 3      LOWEST_LRU = 0;
: 695      1681 3      LOWEST_J = 1;
: 696      1682 3      INCR J FROM 1 TO .QUOTA_CACHE[VCASW_QUOSIZE]
: 697      1683 3      DO
: 698      1684 4      BEGIN
: 699      1685 4      IF .QUOTA_LIST[J-1, VCASL_QUORECNUM] EQL 0
: 700      1686 4      THEN
: 701      1687 5      BEGIN
: 702      1688 5      LOWEST_J = .J;
: 703      1689 5      EXITLOOP;
: 704      1690 4      END;
: 705      1691 4      LRU_DELTA = .QUOTA_CACHE[VCASW_QUOLRU] - .QUOTA_LIST[J-1, VCASW_QUOLRUX];
: 706      1692 4      IF .LRU_DELTA GTRU .LOWEST_LRU
: 707      1693 4      THEN
: 708      1694 5      BEGIN
: 709      1695 5      LOWEST_LRU = .LRU_DELTA;
: 710      1696 5      LOWEST_J = .J;
: 711      1697 4      END;
: 712      1698 3      END;
: 713      1699 3
: 714      1700 3      ! If the cache entry we are about to use contains a modified entry, we must
: 715      1701 3      ! read the corresponding record, update it, and write it. If it represents a
: 716      1702 3      ! held lock, we must release it.
: 717      1703 3
: 718      1704 3
: 719      1705 3      J = .LOWEST_J;
: 720      1706 3      IF .QUOTA_LIST[J-1, VCASV_QUOVALID]
: 721      1707 3      AND .QUOTA_LIST[J-1, VCASV_QUODIRTY]
: 722      1708 3      THEN
: 723      1709 4      BEGIN
: 724      1710 4      REC_NUM = .QUOTA_LIST[J-1, VCASL_QUORECNUM] - 1;
: 725      1711 4      OLD_RECORD = READ_BLOCK (.REC_NUM / RECS_PER_BLOCK
: 726      1712 4      + .FCB[FCBSL_ST[BN], 1, QUOTA_TYPE)
: 727      1713 4      + (.REC_NUM MOD RECS_PER_BLOCK) * DQFSC_LENGTH;
: 728      1714 4      CLEAN_QUO_CACHE (.J, .OLD_RECORD);
: 729      1715 3      END;
: 730      1716 3      REL_QUOTA_LOCK (.J);
: 731      1717 2      END;
: 732      1718 2
: 733      1719 2      ! end of block QUOTA_SEARCH
: 734      1720 2      ! If the quota cache entry is not marked valid, take out the lock on it.
: 735      1721 2      ! If thereafter it is valid, fill in the dummy record with its contents.
```



```
1722 2 IF NOT .QUOTA_LIST[J-1, VCASV_QUOVALID]
1723 2 THEN
1724 2 BEGIN
1725 2 QUOTA_LIST[J-1, VCASL_QUOUIC] = .UIC;
1726 2 GET_QUOTA_LOCK (.J, LCR$K_PWMODE);
1727 2 END;
1728 2 IF .QUOTA_LIST[J-1, VCASV_QUOVALID]
1729 2 THEN
1730 2 BEGIN
1731 2 DUMMY_REC[DQF$S_FLAGS] = DQF$M_ACTIVE;
1732 2 QUOTA_RECORD = .QUOTA_LIST[J-T, VCASL_QUORECNUM];
1733 2 DUMMY_REC[DQF$S_UIC] = .QUOTA_LIST[J-T, VCASL_QUOUIC];
1734 2 CH$MOVE (12, QUOTA_LIST[J-1, VCASL_USAGE], DUMMY_REC[DQF$S_USAGE]);
1735 2 END;
1736 2 QUOTA_INDEX = .J;
1737 2 .J
1738 2 .J
1739 2 .J
1740 1 END;
```

! end of routine SCAN_QUO_CACHE

```
.EXTRN PM$SGL_QUOHIT, PM$SGL_QUOMISS
.EXTRN CACHE_LOCK
```

OBFC 00000 SCAN_QUO_CACHE:

	SE	04	C2	00002	WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R11	1551
	5B	02C4	CA	9E 00005	SUBL2	#4, SP	
	50	98	AA	D0 0000A	MOVAB	708(BASE), R11	1616
	53	5C	A0	D0 0000E	MOVL	-104(BASE), R0	1628
	50	98	AA	D0 00012	MOVL	92(R0), QUOTA_CACHE	
	54	54	A0	D0 00016	MOVL	-104(BASE), R0	1629
	3B	0B	A3	E8 0001A	MOVL	84(R0), FCB	
36	OB		A3	01 E0 0001E	BLBS	11(QUOTA_CACHE), 3\$	1631
	01		63	B1 00023	BBS	#1, 11(QUOTA_CACHE), 3\$	1632
			2D	1B 00026	CMPW	(QUOTA_CACHE), #1	1635
	50	94	AA	D0 00028	BLEQU	2\$	
24	3A		05	E0 0002C	MOVL	-108(BASE), R0	1636
	01	1C	A4	B1 00031	BBS	#5, 58(R0), 2\$	
			1E	1A 00035	CMPW	28(FCB), #1	1637
	50	94	AA	D0 00037	BGTRU	2\$	
	10	3C	A0	E9 0003B	MOVL	-108(BASE), R0	1640
			7E	D4 0003F	BLBC	60(R0), 1\$	
		04	A3	9F 00041	CLRL	-(SP)	1641
		4C	A4	DD 00044	PUSHAB	4(QUOTA_CACHE)	
0000G	CF		03	FB 0C047	PUSHL	76(FCB)	
	06		50	E9 0004C	CALLS	#3, CACHE_LOCK	
OB	A3		01	88 0004F 1\$:	BLBC	R0, 2\$	
			04	11 00053	BISB2	#1, 11(QUOTA_CACHE)	1645
OB	A3		02	88 00055 2\$:	BRB	3\$	
	52	44	A3	9E 00059 3\$:	BISB2	#2, 11(QUOTA_CACHE)	1647
	55		63	3C 0005D	MOVAB	68(R3), QUOTA_LIST	1655
			50	D4 00060	MOVZWL	(QUOTA_CACHE), R5	1656
			2E	11 00062	CLRL	K	
51	50		1C	C5 00064 4\$:	BRB	6\$	
	51		52	C0 00068	MULL3	#28, K, R1	1659
					ADDL2	QUOTA_LIST, R1	

00	EC	A1	18	00	ED	0006B	CMPZV	#0, #24, -20(R1), #0	:	
				1F	13	00071	BEQL	6\$	:	
		04	AC	FC	A1	D1	00073	CMPL	-4(R1), UIC	1660
				18	12	00078	BNEQ	6\$	:	
		08	08	AC	E9	0007A	BLBC	MARK USE, 5\$	:	1663
		E6	A1	02	A3	B0	0007E	MOVW	2(QUOTA_CACHE), -26(R1)	1666
				02	A3	B6	00083	INCW	2(QUOTA_CACHE)	1667
			00000000G	00	D6	00086	INCL	PMSSGL_QUOHIT	:	1669
		56		50	D0	0008C	MOVL	K, J	:	1670
				0090	31	0008F	BRW	12\$	:	1671
		CE	50	55	F3	00092	AOBLEQ	R5, K, 4\$	:	1656
			00000000G	00	D6	00096	INCL	PMSSGL_QUOMISS	:	1679
				58	D4	0009C	CLRL	LOWEST_LRU	:	1680
				55	01	D0	0009E	MOVL	#1, LOWEST_J	1681
				59	63	3C	000A1	MOVZWL	(QUOTA_CACHE), R9	1682
					50	D4	000A4	CLRL	J	1691
					2A	11	000A6	BRB	9\$	
		51	50	1C	C5	000A8	MULL3	#28, J, R1	:	1685
			51	52	C0	000AC	ADDL2	QUOTA_LIST, R1	:	
00	EC	A1	18	00	ED	000AF	CMPZV	#0, #24, -20(R1), #0	:	
				05	12	000B5	BNEQ	8\$	:	
			55	50	D0	000B7	MOVL	J, LOWEST_J	:	1688
				1A	11	000BA	BRB	10\$	:	1687
			57	A3	3C	000BC	MOVZWL	2(QUOTA_CACHE), LRU_DELTA	:	1691
			6E	A1	3C	000C0	MOVZWL	-26(R1), (SP)	:	
			57	6E	C2	000C4	SUBL2	(SP), LRU_DELTA	:	
			58	57	D1	000C7	CMPL	LRU_DELTA, LOWEST_LRU	:	1692
				06	1B	000CA	BLEQU	9\$	:	
			58	57	D0	000CC	MOVL	LRU_DELTA, LOWEST_LRU	:	1695
			55	50	D0	000CF	MOVL	J, LOWEST_J	:	1696
		D2	50	59	F3	000D2	AOBLEQ	R9, J, 7\$	:	1682
			56	55	D0	000D6	MOVL	LOWEST_J, J	:	1705
		50	56	1C	C5	000D9	MULL3	#28, J, R0	:	1706
			50	52	C0	000DD	ADDL2	QUOTA_LIST, R0	:	
			37	A0	E9	000E0	BLBC	-17(R0), 11\$	:	
		32	EF	A0	01	E1	000E4	BBC	#1, -17(R0), 11\$	1707
55	EC	A0	18	00	EF	000E9	EXTZV	#0, #24, -20(R0), REC_NUM	:	1710
				55	D7	000EF	DECL	REC_NUM	:	
				05	DD	000F1	PUSHL	#5	:	1711
				01	DD	000F3	PUSHL	#1	:	
		50	55	10	C7	000F5	DIVL3	#16, REC_NUM, R0	:	
				30	B440	9F	000F9	PUSHAB	248(FCB)[R0]	1712
			0000G	03	FB	000FD	CALLS	#3, READ_BLOCK	:	
			55	01	7A	00102	EMUL	#1, REC_NUM, #0, -(SP)	:	1713
7E		00	55	10	7B	00107	EDIV	#16, (SP)+, R1, R1	:	
51		51	8E	20	C4	0010C	MULL2	#32, R1	:	
			51	51	C0	0010F	ADDL2	R1, OLD_RECORD	:	
			50	50	DD	00112	PUSHL	OLD_RECORD	:	1714
				56	DD	00114	PUSHL	J	:	
		0000V	CF	02	FB	00116	CALLS	#2, CLEAN_QUO_CACHE	:	
				56	DD	0011B	PUSHL	J	:	1716
		0000V	CF	01	FB	0011D	CALLS	#1, REL_QUOTA_LOCK	:	
		50	56	1C	C5	00122	MULL3	#28, J, R0	:	1723
			50	52	C0	00126	ADDL2	QUOTA_LIST, R0	:	
			0E	A0	E8	00129	BLBS	-17(R0), 13\$	:	
		FC	A0	04	AC	D0	0012D	MOVL	UIC, -4(R0)	1726
				04	DD	00132	PUSHL	#4	:	1727

CHARGEQ
V04-000

F 6
15-Sep-1984 23:56:13
14-Sep-1984 12:30:09

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11X.SRC]CHARGEQ.B32;1
Page 22
(5)

02B4	CA	50	0000V	CF	56	DD	00134	PUSHL	J	:		
		27	EF	A042	56	02	FB	00136	CALLS	#2, GET_QUOTA_LOCK	:	
				6B	1C	C5	0013B	13\$:	MULL3	#28, J, R0	1729	
		50		56	00	E1	0013F	BBC	#0, -17(R0)[QUOTA_LIST], 14\$	:		
		50		18	01	D0	00145	MOVL	#1, (R11)	:	1732	
		50		56	1C	C5	00148	MULL3	#28, J, R0	:	1733	
				56	00	EF	0014C	EXTZV	#0, #24, -20(R0)[QUOTA_LIST], 692(BASE)	:		
					1C	C5	00155	MULL3	#28, J, R0	:	1734	
					FC	A042	9F	00159	PUSHAB	-4(R0)[QUOTA_LIST]	:	
			04	AB	9E	D0	0015D	MOVL	@(SP)+, 4(R11)	:		
		50		56	1C	C5	00161	MULL3	#28, J, R0	:	1735	
		08	AB	56	0C	28	00165	MOVC3	#12, -16(R0)[QUOTA_LIST], 8(R11)	:		
				56	56	D0	0016C	14\$:	MOVL	J, 704(BASE)	:	1737
				56	56	D0	00171	MOVL	J, R0	:	1740	
						04	00174	RET		:		

; Routine Size: 373 bytes, Routine Base: \$CODE\$ + 02BF


```

756 1741 1 GLOBAL ROUTINE GET_QUOTA_LOCK (J, MODE) : L_NORM NOVALUE =
757 1742 1
758 1743 1 ++
759 1744 1
760 1745 1 FUNCTIONAL DESCRIPTION:
761 1746 1
762 1747 1 This routine acquires the lock associated with a quota cache
763 1748 1 entry. The lock is raised to PW, and the value block is stored
764 1749 1 in the quota cache entry.
765 1750 1
766 1751 1 CALLING SEQUENCE:
767 1752 1 GET_QUOTA_LOCK (J, MODE)
768 1753 1
769 1754 1 INPUT PARAMETERS:
770 1755 1 J: index of quota cache entry
771 1756 1 MODE: lock mode to use
772 1757 1
773 1758 1 IMPLICIT INPUTS:
774 1759 1 CURRENT_VCB: VCB of volume
775 1760 1 CURRENT_RVT: RVT of volume set
776 1761 1
777 1762 1 OUTPUT PARAMETERS:
778 1763 1 NONE
779 1764 1
780 1765 1 IMPLICIT OUTPUTS:
781 1766 1 NONE
782 1767 1
783 1768 1 ROUTINE VALUE:
784 1769 1 NONE
785 1770 1
786 1771 1 SIDE EFFECTS:
787 1772 1 Lock taken out; value block written into cache entry.
788 1773 1
789 1774 1 --
790 1775 1
791 1776 2 BEGIN
792 1777 2
793 1778 2 LOCAL
794 1779 2 CACHE_ENTRY : REF BBLOCK, ! quota cache entry pointer
795 1780 2 STATUS, ! general status value
796 1781 2 LOCK_FLAGS : BBLOCK [4], ! flags to $ENQ call
797 1782 2 SAVE_LRU, ! save cache entry LRU index
798 1783 2 RESNAM : VECTOR [22, BYTE], ! resource name buffer
799 1784 2 RESNAM_D : VECTOR [2] INITIAL (22, RESNAM);
800 1785 2
801 1786 2 EXTERNAL ROUTINE
802 1787 2 WAIT_FOR_AST : L_NORM, ! wait for completion AST
803 1788 2 CONTINUE_THREAD : L_NORM, ! continue execution thread
804 1789 2 XQPSREL_QUOTA : ADDRESSING_MODE (GENERAL);
805 1790 2 ! unlock cache entry on blocking AST
806 1791 2
807 1792 2 BIND_COMMON;
808 1793 2
809 1794 2 ! If the volume is not cluster accessible, we don't have to bother with
810 1795 2 ! locks.
811 1796 2
812 1797 2
```

```

813 1798 2 IF .BBLOCK [CURRENT_UCB[UCBSL_DEVCHAR], DEV$V_ALL]
814 1799 2 OR NOT .BBLOCK [CURRENT_UCB[UCBSL_DEVCHAR2], DEV$V_CLU]
815 1800 2 THEN RETURN;
816 1801 2
817 1802 2
818 1803 2 ! See if we have a lock ID for the cache entry. If so, this is just a
819 1804 2 ! conversion. Otherwise, generate the resource name, using the facility
820 1805 2 ! prefix and the volume or volume set name.
821 1806 2 !
822 1807 2
823 1808 2 CACHE_ENTRY = BBLOCKVECTOR [BBLOCK [.CURRENT_VCB[VCBSL_QUOCACHE],
824 1809 2 VCSL_QUOLIST], .J-1, VCASR_QUOLOCK, VCASC_QUOLENGTH];
825 1810 2
826 1811 2 LOCK_FLAGS = LCK$M_SYSTEM + LCK$M_VALBLK + LCK$M_NOQUOTA;
827 1812 2
828 1813 2 IF .CACHE_ENTRY[VCASL_QUOLKID] NEQ 0
829 1814 2 THEN LOCK_FLAGS[LCK$V_CONVERT] = 1
830 1815 2
831 1816 2 ELSE
832 1817 2 BEGIN
833 1818 2 CH$MOVE (6, UPLIT BYTE ('F11B$q'), RESNAM[0]);
834 1819 2 CH$MOVE (12,
835 1820 2 IF .CURRENT_VCB[VCBSW_RVN] EQL 0
836 1821 2 THEN CURRENT_VCB[VCBSL_VOLCKNAM]
837 1822 2 ELSE CURRENT_RVT[RVT$T_VLSLCKNAM],
838 1823 2 RESNAM[6]);
839 1824 2 (RESNAM[18]) = .CACHE_ENTRY[VCASL_QUOUIC];
840 1825 2 END;
841 1826 2
842 1827 2 ! Acquire the lock.
843 1828 2 !
844 1829 2
845 1830 2 SAVE_LRU = .CACHE_ENTRY[VCASW_QUOLRUX];
846 1831 2 STATUS = $ENQ (EFN = EFN,
P 1832 2 LKMODE = .MODE,
P 1833 2 FLAGS = .LOCK_FLAGS,
P 1834 2 LKSB = .CACHE_ENTRY[VCASR_QUOLOCK],
P 1835 2 ASTADR = CONTINUE_THREAD,
P 1836 2 ASTPRM = .BASE,
P 1837 2 RESNAM = RESNAM_D
838 1838 2 );
839 1839 2 IF NOT .STATUS
840 1840 2 THEN
841 1841 2 BEGIN
842 1842 2 CH$FILL (0, VCASC_QUOLENGTH, .CACHE_ENTRY);
843 1843 2 IF .LOCK_FLAGS[LCK$V_CONVERT]
844 1844 2 THEN BUG_CHECK (XQPERR, FATAL, 'Unexpected lock manager error')
845 1845 2 ELSE ERR_EXIT (.STATUS);
846 1846 2 END;
847 1847 2
848 1848 2 IF .STATUS EQL SSS_NORMAL
849 1849 2 THEN WAIT_FOR_AST (?);
850 1850 2
851 1851 2 ! Deal with lock completion and handle any errors. If the lock comes back
852 1852 2 ! with value not valid, turn off the valid bit but preserve the contents.
853 1853 2 ! We will still use the record number to avoid a complete search.
854 1854 2 !

```



```

870 1855 2 STATUS = .CACHE_ENTRY[VCASW_QUOSTATUS];
871 1856 2
872 1857 2
873 1858 2 IF NOT .STATUS
874 1859 2 THEN
875 1860 2 BEGIN
876 1861 2 IF .STATUS EQL SSS_VALNOTVALID
877 1862 2 THEN CACHE_ENTRY[VCASV_QUOVALID] = 0
878 1863 2 ELSE
879 1864 2 BEGIN
880 1865 2 CH$FILL (0, VCASC_QUOLENGTH, .CACHE_ENTRY);
881 1866 2 IF .LOCK_FLAGS[LCK$V_CONVERT]
882 1867 2 THEN BUG_CHECK (XQPERR, FATAL, 'Unexpected lock manager error')
883 1868 2 ELSE ERR_EXIT (.STATUS);
884 1869 2 END;
885 1870 2 END;
886 1871 2
887 1872 2 ! Having acquired the lock, convert it to system owned.
888 1873 2
889 1874 2
890 P 1875 2 STATUS = $ENQ (EFN = EFN,
891 P 1876 2 LKMODE = .MODE,
892 P 1877 2 FLAGS = LCK$M_NOQUEUE OR LCK$M_SYNCSTS OR LCK$M_CVTSYS OR LCK$M_CONVERT,
893 P 1878 2 LKSB = CACHE_ENTRY[VCASR_QUOLOCK],
894 P 1879 2 BLKAST = XQP$REL_QUOTA,
895 P 1880 2 ASTPRM = .CACHE_ENTRY
896 1881 2 );
897 1882 2 IF .STATUS
898 1883 2 THEN STATUS = .CACHE_ENTRY[VCASW_QUOSTATUS];
899 1884 2 IF NOT .STATUS
900 1885 2 THEN BUG_CHECK (XQPERR, FATAL, 'Unexpected lock manager error');
901 1886 2
902 1887 2 CACHE_ENTRY[VCASW_QUOINDEX] = .J;
903 1888 2 CACHE_ENTRY[VCASW_QUOLRUX] = .SAVE_LRU;
904 1889 2
905 1890 1 END;
! End of routine GET_QUOTA_LOCK
```

71 24 42 31 31 46 00434 P.AAA: .ASCII \F11B\$q\ ;

```
.EXTRN WAIT_FOR_AST, CONTINUE_THREAD
.EXTRN XQP$REL_QUOTA, SY$ENQ
.EXTRN BUG$XQPERR
```

				OBFC 00000	.ENTRY	GET_QUOTA_LOCK, Save R2,R3,R4,R5,R6,R7,R8,-	1741
						R9,R11	
	5B	00000000G	00	9E 00002	MOVAB	SY\$ENQ, R11	
	5E		1C	C2 00009	SUBL2	#28, SP	
			16	DD 0000C	PUSHL	#22	1776
04	AE	08	AE	9E 0000E	MOVAB	RESNAM, RESNAM_D+4	
	50	94	AA	D0 00013	MOVL	-108(BASE), R0	1798
		3A	A0	95 00017	TSTB	58(R0)	
			01	18 0001A	BGEQ	1\$	
				04 0001C	RET		
	01	3C	A0	E8 0001D 1\$:	BLBS	60(R0), 2\$	1799
				04 00021	RET		

56	04	50	98	AA	D0	00022	2\$:	MOVL	-104(BASE), R0	1808
		AC		1C	C5	00026		MULL3	#28, J, R6	1809
		56	5C	A0	C0	0002B		ADDL2	92(R0), R6	
		56		28	C0	0002F		ADDL2	#40, CACHE_ENTRY	
		58		31	D0	00032		MOVL	#49, LOCK_FLAGS	1811
			04	A6	D5	00035		TSTL	4(CACHE_ENTRY)	1813
				05	13	00038		BEQL	3\$	
		58		02	88	0003A		BISB2	#2, LOCK_FLAGS	1814
08	AE	B7	AF	25	11	0003D		BRB	6\$	
		50		06	28	0003F	3\$:	MOVC3	#6, P.AAA, RESNAM	1818
			98	AA	D0	00045		MOVL	-104(BASE), R0	1820
			0E	A0	B5	00049		TSTW	14(R0)	
		50	0080	07	12	0004C		BNEQ	4\$	
				C0	9E	0004E		MOVAB	128(R0), R0	1821
				05	11	00053		BRB	5\$	1822
		9C	AA	18	C1	00055	4\$:	ADDL3	#24, -100(BASE), R0	
0E	AE	1A	60	0C	28	0005A	5\$:	MOVC3	#12, (R0), RESNAM+6	1823
			AE	A6	D0	0005F		MOVL	24(CACHE_ENTRY), RESNAM+18	1824
			59	02	A6	3C	6\$:	MOVZWL	2(CACHE_ENTRY), SAVE_LRU	1830
					7E	7C		CLRQ	-(SP)	1838
					7E	D4		CLRL	-(SP)	
					5A	DD		PUSHL	BASE	
			0000G	CF	9F	0006E		PUSHAB	CONTINUE_THREAD	
				7E	D4	00072		CLRL	-(SP)	
			18	AE	9F	00074		PUSHAB	RESNAM_D	
			0140	8F	BB	00077		PUSHR	#*M<R6,R8>	
			08	AC	DD	0007B		PUSHL	MODE	
				1E	DD	0007E		PUSHL	#30	
		6B		0B	FB	00080		CALLS	#11, SYS\$ENQ	
		57		50	D0	00083		MOVL	R0, STATUS	
		0E		57	E8	00086		BLBS	STATUS, 7\$	1839
1C	00	6E		00	2C	00089		MOVC5	#0, (SP), #0, #28, (CACHE_ENTRY)	1842
				66		0008E				
	33	58		01	E1	0008F		BBC	#1, LOCK_FLAGS, 10\$	1843
					FEFF	00093		BUGW		1844
					0000*	00095		.WORD	<BUG\$ XQPERR!4>	
		01		57	D1	00097	7\$:	CMPL	STATUS, #1	1848
				05	12	0009A		BNEQ	8\$	
		0000G	CF	00	FB	0009C		CALLS	#0, WAIT FOR AST	1849
			57	66	3C	000A1	8\$:	MOVZWL	(CACHE_ENTRY), STATUS	1856
			22	57	E8	000A4		BLBS	STATUS, 11\$	1858
		000009F0	8F	57	D1	000A7		CMPL	STATUS, #2544	1861
				06	12	000AE		BNEQ	9\$	
		0B	A6	01	8A	000B0		BICB2	#1, 11(CACHE_ENTRY)	1862
				13	11	000B4		BRB	11\$	
1C	00	6E		00	2C	000B6	9\$:	MOVC5	#0, (SP), #0, #28, (CACHE_ENTRY)	1865
				66		000BB				
	06	58		01	E1	000BC		BBC	#1, LOCK_FLAGS, 10\$	1866
					FEFF	000C0		BUGW		1867
					0000*	000C2		.WORD	<BUG\$ XQPERR!4>	
				03	11	000C4		BRB	11\$	
				57	BF	000C6	10\$:	CHMU	STATUS	1868
					04	000C8		RET		
				7E	7C	000C9	11\$:	CLRQ	-(SP)	1881
		00000000G		00	9F	000CB		PUSHAB	XQP\$REL QUOTA	
				56	DD	000D1		PUSHL	CACHE_ENTRY	
				7E	7C	000D3		CLRQ	-(SP)	

CHARGEQ
V04-000

K 6
15-Sep-1984 23:56:13
14-Sep-1984 12:30:09

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11X.SRC]CHARGEQ.B32;1 Page 27
(6)

7E	4E	7E	D4	000D5	CLRL	-(SP)	:
		8F	9A	000D7	MOVZBL	#78, -(SP)	:
		56	DD	000DB	PUSHL	CACHE_ENTRY	:
	08	AC	DD	000DD	PUSHL	MODE	:
		1E	DD	000E0	PUSHL	#30	:
6B		0B	FB	000E2	CALLS	#11, SYSENQ	:
57		50	D0	000E5	MOVL	R0, STATUS	:
06		57	E9	000E8	BLBC	STATUS, 12\$	: 1882
57		66	3C	000EB	MOVZWL	(CACHE_ENTRY), STATUS	: 1883
04		57	E8	000EE	BLBS	STATUS, 13\$	: 1884
			FEFF	000F1	BUGW		: 1885
			0000*	000F3	.WORD	<BUG\$ XQPERR!4>	:
02	66	04	AC	B0	MOVW	J, (CACHE_ENTRY)	: 1887
A6			59	B0	MOVW	SAVE_LRU, 2(CACHE_ENTRY)	: 1888
			04	000FD	RET		: 1890

; Routine Size: 254 bytes, Routine Base: \$CODE\$ + 043A

```
.. 907 1891 1 GLOBAL ROUTINE REL_QUOTA_LOCK (J) : L_NORM NOVALUE =
.. 908 1892 1
.. 909 1893 1 ++
.. 910 1894 1
.. 911 1895 1 FUNCTIONAL DESCRIPTION:
.. 912 1896 1
.. 913 1897 1 This routine releases the lock associated with a quota cache
.. 914 1898 1 entry. The value block held in the cache entry is written to
.. 915 1899 1 the lock.
.. 916 1900 1
.. 917 1901 1 CALLING SEQUENCE:
.. 918 1902 1 REL_QUOTA_LOCK (J)
.. 919 1903 1
.. 920 1904 1 INPUT PARAMETERS:
.. 921 1905 1 J: index of quota cache entry
.. 922 1906 1
.. 923 1907 1 IMPLICIT INPUTS:
.. 924 1908 1 NONE
.. 925 1909 1
.. 926 1910 1 OUTPUT PARAMETERS:
.. 927 1911 1 NONE
.. 928 1912 1
.. 929 1913 1 IMPLICIT OUTPUTS:
.. 930 1914 1 NONE
.. 931 1915 1
.. 932 1916 1 ROUTINE VALUE:
.. 933 1917 1 NONE
.. 934 1918 1
.. 935 1919 1 SIDE EFFECTS:
.. 936 1920 1 Lock released, value block written, cache entry marked non valid.
.. 937 1921 1
.. 938 1922 1 --
.. 939 1923 1
.. 940 1924 2 BEGIN
.. 941 1925 2
.. 942 1926 2 LOCAL
.. 943 1927 2 CACHE_ENTRY : REF BBLOCK; ! quota cache entry pointer
.. 944 1928 2
.. 945 1929 2 BIND_COMMON;
.. 946 1930 2
.. 947 1931 2
.. 948 1932 2 ! Release the lock.
.. 949 1933 2 !
.. 950 1934 2
.. 951 1935 2 CACHE_ENTRY = BBLOCKVECTOR [BBLOCK [.CURRENT_VCB[VCBSL_QUOCACHE],
.. 952 1936 2 VCASL_QUOLIST], .J-1, VCASL_QUOLOCK; .VCASL_QUOLENGTH];
.. 953 1937 2
.. 954 1938 2 IF .CACHE_ENTRY[VCASL_QUOLKID] NEQ 0
.. 955 1939 2 THEN
.. 956 1940 3 BEGIN
.. 957 1941 3
.. 958 1942 3 IF NOT $DEQ (LKID = .CACHE_ENTRY[VCASL_QUOLKID],
.. 959 1943 3 VALBLK = CACHE_ENTRY[VCASL_QUORECNUM]
.. 960 1944 3 )
.. 961 1945 3 THEN BUG_CHECK (XQPERR, FATAL, 'Unexpected lock manager error');
.. 962 1946 2 END;
.. 963 1947 2
```



```

.: 964      1948  2  ! Mark the cache entry no longer valid
.: 965      1949  2  !
.: 966      1950  2
.: 967      1951  2  CH$FILL (0, VCASC_QUOLENGTH, .CACHE_ENTRY);
.: 968      1952  2
.: 969      1953  1  END;                                     ! E

```

```
! End of routine REL_QUOTA_LOCK
```

```

      .EXTRN  SYS$DEQ

```

					003C	00000	
		50		98	AA	D0	00002
	52	04	AC		1C	C5	00006
			52	5C	A0	C0	0000B
			52		28	C0	0000F
				04	A2	D5	00012
					16	13	00015
					7E	7C	00017
				08	A2	9F	00019
				04	A2	DD	0001C
		00000000G	00		04	FB	0001F
			04		50	E8	00026
						FEFF	00029
						0000*	0002B
1C	00		6E		00	2C	0002D 1\$:
					62		00032
						04	00033

```

.ENTRY    REL QUOTA LOCK, Save R2,R3,R4,R5
MOVL     -104(BASE), R0
MULL3    #28, J, R2
ADDL2    92(R0), R2
ADDL2    #40, CACHE_ENTRY
TSTL     4(CACHE_ENTRY)
BEQL     1$
CLRQ     -(SP)
PUSHAB   8(CACHE_ENTRY)
PUSHL    4(CACHE_ENTRY)
CALLS    #4, SYS$DEQ
BLBS     R0, 1$
BUGW
.WORD    <BUG$_XQPERR!4>
MOVCS    #0, (SP), #0, #28, (CACHE_ENTRY)

RET

```

; Routine Size: 52 bytes, Routine Base: \$CODE\$ + 0538

```

: 971 1954 1 GLOBAL ROUTINE CLEAN_QUO_CACHE (J, Q_RECORD) : L_NORM NOVALUE =
: 972 1955 1
: 973 1956 1 !++
: 974 1957 1
: 975 1958 1 FUNCTIONAL DESCRIPTION:
: 976 1959 1
: 977 1960 1 This routine updates the indicated quota record buffer from the
: 978 1961 1 indicated cache entry, and marks the record dirty and marks the
: 979 1962 1 cache entry clean if necessary.
: 980 1963 1
: 981 1964 1
: 982 1965 1 CALLING SEQUENCE:
: 983 1966 1 CLEAN_QUO_CACHE (ARG1, ARG2)
: 984 1967 1
: 985 1968 1 INPUT PARAMETERS:
: 986 1969 1 ARG1: index in quota cache
: 987 1970 1 0 to not
: 988 1971 1
: 989 1972 1 IMPLICIT INPUTS:
: 990 1973 1 CURRENT_VCB: VCB of volume
: 991 1974 1
: 992 1975 1 OUTPUT PARAMETERS:
: 993 1976 1 ARG2: address of record buffer
: 994 1977 1
: 995 1978 1 IMPLICIT OUTPUTS:
: 996 1979 1 NONE
: 997 1980 1
: 998 1981 1 ROUTINE VALUE:
: 999 1982 1 1
: 1000 1983 1
: 1001 1984 1 SIDE EFFECTS:
: 1002 1985 1 quota cache entry modified, buffer marked dirty
: 1003 1986 1
: 1004 1987 1 --
: 1005 1988 1
: 1006 1989 2 BEGIN
: 1007 1990 2
: 1008 1991 2 MAP
: 1009 1992 2 Q_RECORD : REF BBLOCK; ! address of quota record
: 1010 1993 2
: 1011 1994 2 LOCAL
: 1012 1995 2 CACHE_ENTRY : REF BBLOCK; ! quota cache entry pointer
: 1013 1996 2
: 1014 1997 2 BIND_COMMON;
: 1015 1998 2
: 1016 1999 2 EXTERNAL ROUTINE
: 1017 2000 2 MARK_DIRTY : L_NORM; ! mark buffer for write back
: 1018 2001 2
: 1019 2002 2
: 1020 2003 2 ! Copy the cache entry to the record buffer. If the cache entry is marked
: 1021 2004 2 ! dirty, mark it clean and mark the record dirty.
: 1022 2005 2
: 1023 2006 2
: 1024 2007 2 CACHE_ENTRY = BBLOCKVECTOR [BBLOCK [.CURRENT_VCB[VCB$L_QUOCACHE],
: 1025 2008 2 VCAS$L_QUOLIST], .J-1, VCAS$R_QUOLOCK; ,VCAS$C_QUOLENGTH];
: 1026 2009 2
: 1027 2010 2 Q_RECORD[DQF$L_UIC] = .CACHE_ENTRY[VCAS$L_QUOUIC];
```



```
; 1028      2011 2 CH$MOVE (12, CACHE_ENTRY[VCA$L_USAGE], Q_RECORD[DQF$L_USAGE]);
; 1029      2012 2 IF .CACHE_ENTRY[VCA$V_QUODIRTY]
; 1030      2013 2 THEN
; 1031      2014 2 BEGIN
; 1032      2015 2     CACHE_ENTRY[VCA$V_QUODIRTY] = 0;
; 1033      2016 2     MARK_DIRTY (.Q_RECORD);
; 1034      2017 2     END;
; 1035      2018 2
; 1036      2019 1 END;                                ! end of routine CLEAN_QUO_CACHE
```

				007C 00000	.ENTRY CLEAN QUO CACHE, Save R2,R3,R4,R5,R6	: 1954
		50	98	AA D0 00002	MOVL -104(BASE), R0	: 2007
56	04	AC		1C C5 00006	MULL3 #28, J, R6	: 2008
		56	5C	A0 C0 0000B	ADDL2 92(R0), R6	
		56		28 C0 0000F	ADDL2 #40, CACHE_ENTRY	
		50	08	AC D0 00012	MOVL Q_RECORD, R0	: 2010
	04	A0	18	A6 D0 00016	MOVL 24(CACHE_ENTRY), 4(R0)	
		50	08	AC D0 0001B	MOVL Q_RECORD, R0	: 2011
08	A0	0C		0C 28 0001F	MOVC3 #T2, 12(CACHE_ENTRY), 8(R0)	
	0C	0B		01 E1 00025	BBC #1, 11(CACHE_ENTRY), 1\$	: 2012
		0B		02 8A 0002A	BICB2 #2, 11(CACHE_ENTRY)	: 2015
			08	AC DD 0002E	PUSHL Q_RECORD	: 2016
	0000G	CF		01 FB 00031	CALLS #T, MARK_DIRTY	
				04 00036 1\$:	RET	: 2019

; Routine Size: 55 bytes, Routine Base: \$CODE\$ + 056C


```
1038 2020 1 ROUTINE ENTER_QUO_CACHE (J, Q_RECORD, MARK_DIRTY, MARK_USE) : L_NORM NOVALUE =
1039 2021 1
1040 2022 1 !++
1041 2023 1
1042 2024 1 FUNCTIONAL DESCRIPTION:
1043 2025 1
1044 2026 1 This routine enters the given quota record into the cache at the
1045 2027 1 indicated cache index. If requested, the cache entry is marked dirty.
1046 2028 1
1047 2029 1
1048 2030 1 CALLING SEQUENCE:
1049 2031 1 ENTER_QUO_CACHE (ARG1, ARG2, ARG3, ARG4)
1050 2032 1
1051 2033 1 INPUT PARAMETERS:
1052 2034 1 ARG1: index in quota cache
1053 2035 1 ARG2: address of record buffer
1054 2036 1 ARG3: 1 to mark record dirty
1055 2037 1 0 to not
1056 2038 1 ARG4: 0 to set lowest possible LRU
1057 2039 1 1 to set current LRU
1058 2040 1 2 to leave LRU alone
1059 2041 1
1060 2042 1 IMPLICIT INPUTS:
1061 2043 1 CURRENT_VCB: VCB of volume
1062 2044 1 QUOTA_RECORD: record number of quota record
1063 2045 1
1064 2046 1 OUTPUT PARAMETERS:
1065 2047 1 NONE
1066 2048 1
1067 2049 1 IMPLICIT OUTPUTS:
1068 2050 1 NONE
1069 2051 1
1070 2052 1 ROUTINE VALUE:
1071 2053 1 1
1072 2054 1
1073 2055 1 SIDE EFFECTS:
1074 2056 1 quota cache entry modified
1075 2057 1
1076 2058 1 !--
1077 2059 1
1078 2060 2 BEGIN
1079 2061 2
1080 2062 2 MAP
1081 2063 2 Q_RECORD : REF BBLOCK; ! address of quota record
1082 2064 2
1083 2065 2 LOCAL
1084 2066 2 QUOTA_CACHE : REF BBLOCK; ! address of quota cache
1085 2067 2 CACHE_ENTRY : REF BBLOCK; ! quota cache entry pointer
1086 2068 2
1087 2069 2 BIND_COMMON;
1088 2070 2
1089 2071 2 ! Copy the record data to the cache entry. If requested, mark the cache
1090 2072 2 entry dirty.
1091 2073 2 !
1092 2074 2
1093 2075 2 QUOTA_CACHE = .CURRENT_VCB[VCB$$_QUOCACHE];
1094 2076 2 CACHE_ENTRY = BBLOCKVECTOR [BBLOCK [.QUOTA_CACHE, VCB$$_QUOLIST],
```



```

: 1095      2077      .J-1, VCASR_QUOLOCK; ,VCASC_QUOLENGTH];
: 1096      2078
: 1097      2079
: 1098      2080      CACHE_ENTRY[VCASL_QUOUIC] = .Q RECORD[DQFSL UIC];
: 1099      2081      CH$MOVE (12, Q RECORD[DQFSL_USAGE], CACHE_ENTRY[VCASL_USAGE]);
: 1100      2082      CACHE_ENTRY[VCASB_QUOFLAGS] = VCASB_QUOVAID;
: 1101      2083      CACHE_ENTRY[VCASW_QUOINDEX] = .J;
: 1102      2084      CACHE_ENTRY[VCASL_QUORECNUM] = (IF .Q RECORD[DQF$V_ACTIVE]
: 1103      2085      THEN .QUOTA_RECORD
: 1104      2086      ELSE 0);
: 1105      2087      IF .MARK_USE
: 1106      2088      THEN
: 1107      2089      BEGIN
: 1108      2090      CACHE_ENTRY[VCASW_QUOLRUX] = .QUOTA_CACHE[VCASW_QUOLRU];
: 1109      2091      QUOTA_CACHE[VCASW_QUOLRU] = .QUOTA_CACHE[VCASW_QUOLRU] + 1;
: 1110      2092      END
: 1111      2093      ELSE IF .MARK_USE EQL 0
: 1112      2094      THEN
: 1113      2095      BEGIN
: 1114      2096      CACHE_ENTRY[VCASW_QUOLRUX] = .QUOTA_CACHE[VCASW_QUOLRU] - 1^15;
: 1115      2097      END;
: 1116      2098
: 1117      2099      IF .MARK_DIRTY
: 1118      2100      THEN CACHE_ENTRY[VCASV_QUODIRTY] = 1;
: 1119      2101
: 1120      2102      1 END;
                                ! end of routine ENTER_QUO_CACHE

```

```

                                00FC 00000 ENTER_QUO_CACHE:
                                .WORD      Save R2,R3,R4,R5,R6,R7
                                50          98      AA      D0      00002      MOVL      -104(BASE), R0
                                57          5C      A0      D0      00006      MOVL      92(R0), QUOTA_CACHE
                                50          1C      C5      0000A      MULL3     #28, J, R0
                                56          28      A047     9E      0000F      MOVAB     40(R0)[QUOTA_CACHE], CACHE_ENTRY
                                50          08      AC      D0      00014      MOVL      Q_RECORD, R0
                                18          04      A0      D0      00018      MOVL      4(R0), 24(CACHE_ENTRY)
                                50          08      AC      D0      0001D      MOVL      Q_RECORD, R0
                                0C      A6      08      A0      0C      28      00021      MOV C3    #12, 8(R0), 12(CACHE_ENTRY)
                                08          0B      A6      01      90      00027      MOV B     #1, 11(CACHE_ENTRY)
                                66          04      AC      B0      0002B      MOVW     J, (CACHE_ENTRY)
                                07          08      BC      E9      0002F      BLBC     @Q_RECORD, 1$
                                50          02B4     CA      D0      00033      MOVL      692(BASE), R0
                                02          11      00038      BRB      2$
                                50          D4      0003A      1$:      CLRL      R0
                                08      A6      18          50      F0      0003C      2$:      INSV     R0, #0, #24, 8(CACHE_ENTRY)
                                0A          10      AC      E9      00042      BLBC     MARK_USE, 3$
                                02          A6      02      A7      B0      00046      MOVW     2(QUOTA_CACHE), 2(CACHE_ENTRY)
                                02          02      A7      B6      0004B      INCW     2(QUOTA_CACHE)
                                0D          11      0004E      BRB      4$
                                10          AC      D5      00050      3$:      TSTL     MARK_USE
                                08          12      00053      BNEQ     4$
                                02      A6      02      A7      8000     8F      A1      00055      ADDW3     #-32768, 2(QUOTA_CACHE), 2(CACHE_ENTRY)
                                04          0C      AC      E9      0005D      4$:      BLBC     MARK_DIRTY, 5$
                                0B          A6      02      88      00061      BISB2     #2, T1(CACHE_ENTRY)

```

CHARGEQ
V04-000

E 7
15-Sep-1984 23:56:13
14-Sep-1984 12:30:09

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[F11X.SRC]CHARGEQ.B32;1
Page 34
(9)

04 00065 5\$: RET

; 2102

; Routine Size: 102 bytes, Routine Base: \$CODE\$ + 05A3

: 1121 2103 1
: 1122 2104 1 END
: 1123 2105 0 ELUDOM

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	1545	NOVEC,NOWRT, RD , EXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	82	0	1000	00:02.0

COMMAND QUALIFIERS

; BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LISS:CHARGEQ/OBJ=OBJ\$:CHARGEQ MSRC\$:CHARGEQ/UPDATE=(ENHS:CHARGEQ)

; Size: 1539 code + 6 data bytes
; Run Time: 01:30.3
; Elapsed Time: 03:00.2
; Lines/CPU Min: 1398
; Lexemes/CPU-Min: 61595
; Memory Used: 310 pages
; Compilation Complete

0168

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY