

new/usr/src/lib/brand/solaris10/s10_brand/common/s10_brand.c

1

```
*****
37582 Thu Jun  4 11:45:16 2009
new/usr/src/lib/brand/solaris10/s10_brand/common/s10_brand.c
sysinfo emulation
*****
1 /*
2  * CDDL HEADER START
3  *
4  * The contents of this file are subject to the terms of the
5  * Common Development and Distribution License (the "License").
6  * You may not use this file except in compliance with the License.
7  *
8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
9  * or http://www.opensolaris.org/os/licensing.
10 * See the License for the specific language governing permissions
11 * and limitations under the License.
12 *
13 * When distributing Covered Code, include this CDDL HEADER in each
14 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15 * If applicable, add the following below this CDDL HEADER, with the
16 * fields enclosed by brackets "[]" replaced with your own identifying
17 * information: Portions Copyright [yyyy] [name of copyright owner]
18 *
19 * CDDL HEADER END
20 */
21 /*
22 * Copyright 2009 Sun Microsystems, Inc. All rights reserved.
23 * Use is subject to license terms.
24 */

26 #include <errno.h>
27 #include <fcntl.h>
28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <strings.h>
31 #include <unistd.h>
32 #include <sys/auxv.h>
33 #include <sys/bitmap.h>
34 #include <sys/brand.h>
35 #include <sys/inttypes.h>
36 #include <sys/lwp.h>
37 #include <sys/syscall.h>
38 #include <sys/system.h>
39 #include <sys/utsname.h>
40 #include <sys/systeminfo.h>
41 #include <sys/zone.h>
42 #include <sys/stat.h>
43 #include <sys/mntent.h>
44 #include <sys/ctfs.h>
45 #include <sys/priv.h>
46 #include <sys/acctctl.h>
47 #include <libgen.h>

49 #include <s10_brand.h>
50 #include <s10_misc.h>

52 /*
53 * Principles of emulation 101.
54 *
55 *
56 * *** Setting errno
57 *
58 * Just don't do it. This emulation library is loaded onto a
59 * separate link map from the application who's address space we're
60 * running in. We have our own private copy of libc, so there for,
61 * the errno value accessible from here is is also private and changing
```

new/usr/src/lib/brand/solaris10/s10_brand/common/s10_brand.c

2

```
62 * it will not affect any errno value that the processes who's address
63 * space we are running in will see. To return an error condition we
64 * should return the negated errno value we'd like the system to return.
65 * For more information about this see the comment in s10_handler().
66 * Basically, when we return to the caller that initiated the system
67 * call it's their responsibility to set errno.
68 *
69 *
70 * *** Recursion Considerations
71 *
72 * When emulating system calls we need to be very careful about what
73 * library calls we invoke. Library calls should be kept to a minimum.
74 * One issue is that library calls can invoke system calls, so if we're
75 * emulating a system call and we invoke a library call that depends on
76 * that system call we will probably enter a recursive loop, which would
77 * be bad.
78 *
79 *
80 * *** Return Values.
81 *
82 * When declaring new syscall emulation functions, it is very important
83 * to set the proper RV_* flags in the s10_sysent_table. Upon failure,
84 * syscall emulation functions should return an errno value. Upon success
85 * syscall emulation functions should return 0 and set the sysret_t return
86 * value parameters accordingly.
87 *
88 *
89 * *** Agent lwp considerations
90 *
91 * It is currently impossible to do any emulation for these system call
92 * when they are being invoked on behalf of an agent lwp. To understand why
93 * it's impossible you have to understand how agent lwp syscalls work.
94 *
95 * The agent lwp syscall process works as follows:
96 * 1 The controlling process stops the target.
97 * 2 The controlling process injects an agent lwp which is also stopped.
98 * This agent lwp assumes the userland stack and register values
99 * of another stopped lwp in the current process.
100 * 3 The controlling process configures the agent lwp to start
101 * executing the requested system call.
102 * 4 The controlling process configure /proc to stop the agent lwp when
103 * it enters the requested system call.
104 * 5 The controlling processes allows the agent lwp to start executing.
105 * 6 The agent lwp traps into the kernel to perform the requested system
106 * call and immediately stop.
107 * 7 The controlling process copies all the arguments for the requested
108 * system call onto the agent lwp's stack.
109 * 8 The controlling process configures /proc to stop the agent lwp
110 * when it completes the requested system call.
111 * 9 The controlling processes allows the agent lwp to start executing.
112 * 10 The agent lwp executes the system call and then stop before returning
113 * to userland.
114 * 11 The controlling process copies the return value and return arguments
115 * back from the agent lwps stack.
116 * 12 The controlling process destroys the agent lwp and restarts
117 * the target process.
118 *
119 * The fundamental problem is that when the agent executes the request
120 * system call in step 5, if we're emulating that system call then the
121 * lwp is redirected back to our emulation layer without blocking
122 * in the kernel. But our emulation layer can't access the arguments
123 * for the system call because they haven't been copied to the stack
124 * yet and they still only exist in the controlling processes address
125 * space. This prevents us from being able to do any emulation of
126 * agent lwp system calls. Hence, currently our brand trap interposition
127 * callback (s10_brand_syscall_callback_common) will detect if a system
```

```

128 * call is being made by an agent lwp, and if this is the case it will
129 * never redirect the system call to this emulation library.
130 *
131 * In the future, if this proves to be a problem the the easiest solution
132 * would probably be to replace the branded versions of these application
133 * with their native counterparts. Ie, truss, plimit, and pfiles could be
134 * replace with wrapper scripts that execute the native versions of these
135 * applications. In the case of plimit and pfiles this should be pretty
136 * strait forward. Truss would probably be more tricky since it can
137 * execute applications which would be branded applications, so in that
138 * case it might be necessary to create a loadable library which could
139 * be LD_PRELOADED into truss and this library would interpose on the
140 * exec() system call to allow truss to correctly execute branded
141 * processes. It should be pointed out that this solution could work
142 * because "native agent lwps" (ie, agent lwps created by native
143 * processes) can be treated differently from "branded aged lwps" (ie,
144 * agent lwps created by branded processes), since native agent lwps
145 * would presumably be making native system calls and hence not need
146 * any interposition.
147 *
148 *
149 * *** s10 brand emulation scope considerations
150 *
151 * One of the differences between the lx brand and the s8 and s9
152 * brands, is that the s8 and s9 brands only interpose on syscalls
153 * that need some kind of emulation, where as the lx brand interposes
154 * on _all_ system calls. Lx branded system calls that don't need
155 * any emulation are then redirected back to the kernel from the
156 * userland library via the IN_KERNEL_SYSCALL macro. The lx-syscall
157 * dtrace provider depends on this behavior.
158 *
159 */

161 static zoneid_t zoneid;
162 static boolean_t ipshared;
163 static boolean_t emul_global_zone = B_FALSE;
164 static int emul_vers;
165 pid_t zone_init_pid;

167 #define EMULATE(cb, args)    { (sysent_cb_t)(cb), (args) }
168 #define NOSYS                EMULATE(s10_unimpl, (0 | RV_DEFAULT))

170 typedef long (*sysent_cb_t)();
171 typedef struct s10_sysent_table {
172     sysent_cb_t    st_callc;
173     uintptr_t     st_args;
174 } s10_sysent_table_t;
175
176 unchanged_portion_omitted

649 int
650 s10_sysinfo(sysret_t *rv, int command, char *buf, long count)
651 {
652     char *value;
653     int err, len;

655     /*
656      * We must interpose on the sysinfo(2) commands SI_RELEASE and
657      * SI_VERSION; all others get passed to the native sysinfo(2)
658      * command.
659      */
660     switch (command) {
661         case SI_RELEASE:
662             value = s10_UTS_RELEASE;
663             break;

665         case SI_VERSION:

```

```

666         value = s10_UTS_VERSION;
667         break;

669     default:
670         /*
671          * The default action is to pass the command to the
672          * native sysinfo(2) syscall.
673          */
674         if ((err = __systemcall(rv, SYS_systeminfo + 1024,
675             command, buf, count)) != 0)
676             return (err);

678         return (0);
679     }

681     len = strlen(value) + 1;
682     if (count > 0) {
683         if (s10_ucopystr(value, buf, count) != 0)
684             return (EFAULT);

686         /* Assure NULL termination of buf as s10_ucopystr() doesn't. */
687         if (len > count && s10_ucopy("\0", buf + (count - 1), 1) != 0)
688             return (EFAULT);
689     }

691     /*
692      * On success, sysinfo(2) returns the size of buffer required to hold
693      * the complete value plus its terminating NULL byte.
694      */
695     rv->sys_rval1 = len;
696     rv->sys_rval2 = 0;
697     S10_TRUSS_POINT_3(rv, SYS_systeminfo, 0, command, buf, count);
698     return (0);
699 }

701 /*
702 * If the emul_global_zone flag is set then emulate some aspects of the
703 * zone system call. In particular, emulate the global zone ID on the
704 * ZONE_LOOKUP subcommand and emulate some of the global zone attributes
705 * on the ZONE_GETATTR subcommand. If the flag is not set or we're performing
706 * some other operation, simply pass the calls through.
707 */
708 int
709 s10_zone(sysret_t *rval, int cmd, void *arg1, void *arg2, void *arg3,
710     void *arg4)
711 {
712     char          *aval;
713     int           len;
714     zoneid_t     zid;
715     int           attr;
716     char          *buf;
717     size_t       bufsize;

719     /*
720      * We only emulate the zone syscall for a subset of specific commands,
721      * otherwise we just pass the call through.
722      */
723     if (!emul_global_zone)
724         return (__systemcall(rval, SYS_zone + 1024, cmd, arg1, arg2,
725             arg3, arg4));

727     switch (cmd) {
728     case ZONE_LOOKUP:
729         (void) S10_TRUSS_POINT_1(rval, SYS_zone, 0, cmd);
730         rval->sys_rval1 = GLOBAL_ZONEID;
731         rval->sys_rval2 = 0;

```

```

732         return (0);
733
734     case ZONE_GETATTR:
735         zid = (zoneid_t)(uintptr_t)arg1;
736         attr = (int)(uintptr_t)arg2;
737         buf = (char *)arg3;
738         bufsize = (size_t)arg4;
739
740         /*
741          * If the request is for the global zone then we're emulating
742          * that, otherwise pass this thru.
743          */
744         if (zid != GLOBAL_ZONEID)
745             goto passthru;
746
747         (void) S10_TRUSS_POINT_3(rval, SYS_zone, 0, cmd, zid, attr);
748
749         switch (attr) {
750         case ZONE_ATTR_NAME:
751             aval = GLOBAL_ZONENAME;
752             break;
753
754         case ZONE_ATTR_BRAND:
755             aval = NATIVE_BRAND_NAME;
756             break;
757         default:
758             /*
759              * We only emulate a subset of the attrs, use the
760              * real zone id to pass thru the rest.
761              */
762             arg1 = (void *) (uintptr_t) zid;
763             goto passthru;
764         }
765
766         len = strlen(aval) + 1;
767         if (len > bufsize)
768             return (ENAMETOOLONG);
769
770         if (buf != NULL) {
771             if (len == 1) {
772                 if (s10_uucopy("\0", buf, 1) != 0)
773                     return (EFAULT);
774             } else {
775                 if (s10_uucopystr(aval, buf, len) != 0)
776                     return (EFAULT);
777
778                 /*
779                  * Assure NULL termination of "buf" as
780                  * s10_uucopystr() does NOT.
781                  */
782                 if (s10_uucopy("\0", buf + (len - 1), 1) != 0)
783                     return (EFAULT);
784             }
785         }
786
787         rval->sys_rval1 = len;
788         rval->sys_rval2 = 0;
789         return (0);
790
791     default:
792         break;
793 }
794
795 passthru:
796     return (__systemcall(rval, SYS_zone + 1024, cmd, arg1, arg2, arg3,
797                         arg4));

```

```

798 }
799     unchanged_portion_omitted
800
801 1034 /*
802 1035 * This table must have at least NSYSCALL entries in it.
803 1036 *
804 1037 * The second parameter of each entry in the s10_sysent_table
805 1038 * contains the number of parameters and flags that describe the
806 1039 * syscall return value encoding. See the block comments at the
807 1040 * top of this file for more information about the syscall return
808 1041 * value flags and when they should be used.
809 1042 */
810 1043 s10_sysent_table_t s10_sysent_table[] = {
811 1044 #if defined(__sparc) && !defined(__sparcv9)
812 1045     EMULATE(s10_indir, 9 | RV_64RVAL), /* 0 */
813 1046 #else /* !__sparc || __sparcv9 */
814 1047     NOSYS, /* 0 */
815 1048 #endif /* !__sparc || __sparcv9 */
816 1049     NOSYS, /* 1 */
817 1050     NOSYS, /* 2 */
818 1051     NOSYS, /* 3 */
819 1052     NOSYS, /* 4 */
820 1053     NOSYS, /* 5 */
821 1054     NOSYS, /* 6 */
822 1055     NOSYS, /* 7 */
823 1056     NOSYS, /* 8 */
824 1057     NOSYS, /* 9 */
825 1058     NOSYS, /* 10 */
826 1059     EMULATE(s10_exec, 2 | RV_DEFAULT), /* 11 */
827 1060     NOSYS, /* 12 */
828 1061     NOSYS, /* 13 */
829 1062     NOSYS, /* 14 */
830 1063     NOSYS, /* 15 */
831 1064     NOSYS, /* 16 */
832 1065     NOSYS, /* 17 */
833 1066     NOSYS, /* 18 */
834 1067     NOSYS, /* 19 */
835 1068     NOSYS, /* 20 */
836 1069     NOSYS, /* 21 */
837 1070     NOSYS, /* 22 */
838 1071     NOSYS, /* 23 */
839 1072     NOSYS, /* 24 */
840 1073     NOSYS, /* 25 */
841 1074     NOSYS, /* 26 */
842 1075     NOSYS, /* 27 */
843 1076     NOSYS, /* 28 */
844 1077     NOSYS, /* 29 */
845 1078     NOSYS, /* 30 */
846 1079     NOSYS, /* 31 */
847 1080     NOSYS, /* 32 */
848 1081     NOSYS, /* 33 */
849 1082     NOSYS, /* 34 */
850 1083     NOSYS, /* 35 */
851 1084     NOSYS, /* 36 */
852 1085     NOSYS, /* 37 */
853 1086     NOSYS, /* 38 */
854 1087     NOSYS, /* 39 */
855 1088     NOSYS, /* 40 */
856 1089     NOSYS, /* 41 */
857 1090     NOSYS, /* 42 */
858 1091     NOSYS, /* 43 */
859 1092     NOSYS, /* 44 */
860 1093     NOSYS, /* 45 */
861 1094     NOSYS, /* 46 */
862 1095     NOSYS, /* 47 */
863 1096     NOSYS, /* 48 */

```

```

1097     NOSYS,                /* 49 */
1098     NOSYS,                /* 50 */
1099     NOSYS,                /* 51 */
1100     NOSYS,                /* 52 */
1101     NOSYS,                /* 53 */
1102     EMULATE(s10_ioctl, 3 | RV_DEFAULT), /* 54 */
1103     NOSYS,                /* 55 */
1104     NOSYS,                /* 56 */
1105     NOSYS,                /* 57 */
1106     NOSYS,                /* 58 */
1107     EMULATE(s10_execve, 3 | RV_DEFAULT), /* 59 */
1108     NOSYS,                /* 60 */
1109     NOSYS,                /* 61 */
1110     NOSYS,                /* 62 */
1111     NOSYS,                /* 63 */
1112     NOSYS,                /* 64 */
1113     NOSYS,                /* 65 */
1114     NOSYS,                /* 66 */
1115     NOSYS,                /* 67 */
1116     NOSYS,                /* 68 */
1117     NOSYS,                /* 69 */
1118     NOSYS,                /* 70 */
1119     EMULATE(s10_acctctl, 3 | RV_DEFAULT), /* 71 */
1120     NOSYS,                /* 72 */
1121     EMULATE(s10_getpagesizes, 2 | RV_DEFAULT), /* 73 */
1122     NOSYS,                /* 74 */
1123     EMULATE(s10_issetugid, 0 | RV_DEFAULT), /* 75 */
1124     NOSYS,                /* 76 */
1125     NOSYS,                /* 77 */
1126     NOSYS,                /* 78 */
1127     NOSYS,                /* 79 */
1128     NOSYS,                /* 80 */
1129     NOSYS,                /* 81 */
1130     NOSYS,                /* 82 */
1131     NOSYS,                /* 83 */
1132     NOSYS,                /* 84 */
1133     NOSYS,                /* 85 */
1134     NOSYS,                /* 86 */
1135     NOSYS,                /* 87 */
1136     NOSYS,                /* 88 */
1137     NOSYS,                /* 89 */
1138     NOSYS,                /* 90 */
1139     NOSYS,                /* 91 */
1140     NOSYS,                /* 92 */
1141     NOSYS,                /* 93 */
1142     NOSYS,                /* 94 */
1143     NOSYS,                /* 95 */
1144     NOSYS,                /* 96 */
1145     NOSYS,                /* 97 */
1146     NOSYS,                /* 98 */
1147     NOSYS,                /* 99 */
1148     NOSYS,                /* 100 */
1149     NOSYS,                /* 101 */
1150     NOSYS,                /* 102 */
1151     NOSYS,                /* 103 */
1152     NOSYS,                /* 104 */
1153     NOSYS,                /* 105 */
1154     NOSYS,                /* 106 */
1155     NOSYS,                /* 107 */
1156     NOSYS,                /* 108 */
1157     NOSYS,                /* 109 */
1158     NOSYS,                /* 110 */
1159     NOSYS,                /* 111 */
1160     NOSYS,                /* 112 */
1161     NOSYS,                /* 113 */
1162     NOSYS,                /* 114 */

```

```

1163     NOSYS,                /* 115 */
1164     NOSYS,                /* 116 */
1165     NOSYS,                /* 117 */
1166     NOSYS,                /* 118 */
1167     NOSYS,                /* 119 */
1168     NOSYS,                /* 120 */
1169     NOSYS,                /* 121 */
1170     NOSYS,                /* 122 */
1171     NOSYS,                /* 123 */
1172     NOSYS,                /* 124 */
1173     NOSYS,                /* 125 */
1174     NOSYS,                /* 126 */
1175     NOSYS,                /* 127 */
1176     NOSYS,                /* 128 */
1177     NOSYS,                /* 129 */
1178     NOSYS,                /* 130 */
1179     NOSYS,                /* 131 */
1180     NOSYS,                /* 132 */
1181     NOSYS,                /* 133 */
1182     NOSYS,                /* 134 */
1183     EMULATE(s10_uname, 1 | RV_DEFAULT), /* 135 */
1184     NOSYS,                /* 136 */
1185     NOSYS,                /* 137 */
1186     NOSYS,                /* 138 */
1187     EMULATE(s10_sysinfo, 3 | RV_DEFAULT), /* 139 */
1188     NOSYS,                /* 140 */
1189     NOSYS,                /* 141 */
1190     NOSYS,                /* 142 */
1191     NOSYS,                /* 143 */
1192     NOSYS,                /* 144 */
1193     NOSYS,                /* 145 */
1194     NOSYS,                /* 146 */
1195     NOSYS,                /* 147 */
1196     NOSYS,                /* 148 */
1197     NOSYS,                /* 149 */
1198     NOSYS,                /* 150 */
1199     NOSYS,                /* 151 */
1200     NOSYS,                /* 152 */
1201     NOSYS,                /* 153 */
1202     NOSYS,                /* 154 */
1203     NOSYS,                /* 155 */
1204     NOSYS,                /* 156 */
1205     NOSYS,                /* 157 */
1206     NOSYS,                /* 158 */
1207     NOSYS,                /* 159 */
1208     NOSYS,                /* 160 */
1209     NOSYS,                /* 161 */
1210     NOSYS,                /* 162 */
1211     NOSYS,                /* 163 */
1212     NOSYS,                /* 164 */
1213     NOSYS,                /* 165 */
1214     NOSYS,                /* 166 */
1215     NOSYS,                /* 167 */
1216     NOSYS,                /* 168 */
1217     NOSYS,                /* 169 */
1218     NOSYS,                /* 170 */
1219     NOSYS,                /* 171 */
1220     NOSYS,                /* 172 */
1221     NOSYS,                /* 173 */
1222     EMULATE(s10_pwrite, 4 | RV_DEFAULT), /* 174 */
1223     NOSYS,                /* 175 */
1224     NOSYS,                /* 176 */
1225     NOSYS,                /* 177 */
1226     NOSYS,                /* 178 */
1227     NOSYS,                /* 179 */

```

```

1228     NOSYS, /* 180 */
1229     NOSYS, /* 181 */
1230     NOSYS, /* 182 */
1231     NOSYS, /* 183 */
1232     NOSYS, /* 184 */
1233     NOSYS, /* 185 */
1234     NOSYS, /* 186 */
1235     NOSYS, /* 187 */
1236     NOSYS, /* 188 */
1237     NOSYS, /* 189 */
1238     EMULATE(s10_sigqueue, 4 | RV_DEFAULT), /* 190 */
1239     NOSYS, /* 191 */
1240     NOSYS, /* 192 */
1241     NOSYS, /* 193 */
1242     NOSYS, /* 194 */
1243     NOSYS, /* 195 */
1244     NOSYS, /* 196 */
1245     NOSYS, /* 197 */
1246     NOSYS, /* 198 */
1247     NOSYS, /* 199 */
1248     NOSYS, /* 200 */
1249     NOSYS, /* 201 */
1250     NOSYS, /* 202 */
1251     NOSYS, /* 203 */
1252     NOSYS, /* 204 */
1253     NOSYS, /* 205 */
1254     NOSYS, /* 206 */
1255     NOSYS, /* 207 */
1256     NOSYS, /* 208 */
1257     NOSYS, /* 209 */
1258     NOSYS, /* 210 */
1259     NOSYS, /* 211 */
1260     NOSYS, /* 212 */
1261     NOSYS, /* 213 */
1262     NOSYS, /* 214 */
1263     NOSYS, /* 215 */
1264     NOSYS, /* 216 */
1265     NOSYS, /* 217 */
1266     NOSYS, /* 218 */
1267     NOSYS, /* 219 */
1268     NOSYS, /* 220 */
1269     NOSYS, /* 221 */
1270     NOSYS, /* 222 */
1271 #ifdef  _LP64
1272     NOSYS, /* 223 */
1273 #else  /* !_LP64 */
1274     EMULATE(s10_pwrite64, 5 | RV_DEFAULT), /* 223 */
1275 #endif /* !_LP64 */
1276     NOSYS, /* 224 */
1277     NOSYS, /* 225 */
1278     NOSYS, /* 226 */
1279     EMULATE(s10_zone, 5 | RV_DEFAULT), /* 227 */
1280     NOSYS, /* 228 */
1281     NOSYS, /* 229 */
1282     NOSYS, /* 230 */
1283     NOSYS, /* 231 */
1284     NOSYS, /* 232 */
1285     NOSYS, /* 233 */
1286     NOSYS, /* 234 */
1287     NOSYS, /* 235 */
1288     NOSYS, /* 236 */
1289     NOSYS, /* 237 */
1290     NOSYS, /* 238 */
1291     NOSYS, /* 239 */
1292     NOSYS, /* 240 */
1293     NOSYS, /* 241 */

```

```

1294     NOSYS, /* 242 */
1295     NOSYS, /* 243 */
1296     NOSYS, /* 244 */
1297     NOSYS, /* 245 */
1298     NOSYS, /* 246 */
1299     NOSYS, /* 247 */
1300     NOSYS, /* 248 */
1301     NOSYS, /* 249 */
1302     NOSYS, /* 250 */
1303     NOSYS, /* 251 */
1304     NOSYS, /* 252 */
1305     NOSYS, /* 253 */
1306     NOSYS, /* 254 */
1307     NOSYS /* 255 */
1308 };
_____unchanged_portion_omitted_

```

24368 Thu Jun 4 11:45:21 2009

new/usr/src/uts/common/brand/solaris10/s10_brand.c

sysinfo emulation

unchanged portion omitted

```
892 int
893 _init(void)
894 {
895     int err;

897     /*
898      * Set up the table indicating which system calls we want to
899      * interpose on. We should probably build this automatically from
900      * a list of system calls that is shared with the user-space
901      * library.
902      */
903     s10_emulation_table = kmem_zalloc(NSYSCALL, KM_SLEEP);
904     s10_emulation_table[SYS_exec] = 1; /* 11 */
905     s10_emulation_table[SYS_ioctl] = 1; /* 54 */
906     s10_emulation_table[SYS_execve] = 1; /* 59 */
907     s10_emulation_table[SYS_acctctl] = 1; /* 71 */
908     s10_emulation_table[SYS_getpagesizes] = 1; /* 73 */
909     s10_emulation_table[S10_SYS_issetugid] = 1; /* 75 */
910     s10_emulation_table[SYS_uname] = 1; /* 135 */
911     s10_emulation_table[SYS_systeminfo] = 1; /* 139 */
912     s10_emulation_table[SYS_pwrite] = 1; /* 174 */
913     s10_emulation_table[SYS_sigqueue] = 1; /* 190 */
914     s10_emulation_table[SYS_pwrite64] = 1; /* 223 */
915     s10_emulation_table[SYS_zone] = 1; /* 227 */

917     err = mod_install(&modlinkage);
918     if (err) {
919         cmn_err(CE_WARN, "Couldn't install brand module");
920         kmem_free(s10_emulation_table, NSYSCALL);
921     }

923     return (err);
924 }
```

unchanged portion omitted