```
**********************************************************
   18058 Sat Aug 17 17:56:51 2013
new/usr/src/cmd/dis/dis_main.c
4054 dis sometimes decides random symbols are functions
**********************************************************
______unchanged_portion_omitted_
```

```
 99 /*
100  * The main disassembly routine.  Given a fixed-sized buffer and starting
101  * address, disassemble the data using the supplied target and libdisasm handle.
102  */
103 void
104 dis_data(dis_tgt_t *tgt, dis_handle_t *dhp, uint64_t addr, void *data,
105     size_t datalen)
106 {
107         dis_buffer_t db = { 0 };
108         char buf[BUFSIZE];
109         char symbuf[BUFSIZE];
110         const char *symbol;
111         const char *last_symbol;
112         off_t symoffset;
113         int i;
114         int bytesperline;
115         size_t symsize;
116         int isfunc = 0;
116         int isfunc;
117         size_t symwidth = 0;

119         db.db_tgt = tgt;
120         db.db_data = data;
121         db.db_addr = addr;
122         db.db_size = datalen;

124         dis_set_data(dhp, &db);

126         if ((bytesperline = dis_max_instrlen(dhp)) > 6)
127                 bytesperline = 6;

129         symbol = NULL;

131         while (addr < db.db_addr + db.db_size) {

133                 if (dis_disassemble(dhp, addr, buf, BUFSIZE) != 0) {
134 #if defined(__sparc)
135                         /*
136                          * Since sparc instructions are fixed size, we
137                          * always know the address of the next instruction
138                          */
139                         (void) snprintf(buf, sizeof (buf),
140                             "*** invalid opcode ***");
141                         db.db_nextaddr = addr + 4;

143 #else
144                         off_t next;

146                         (void) snprintf(buf, sizeof (buf),
147                             "*** invalid opcode ***");

149                         /*
150                          * On architectures with variable sized instructions
151                          * we have no way to figure out where the next
152                          * instruction starts if we encounter an invalid
153                          * instruction.  Instead we print the rest of the
154                          * instruction stream as hex until we reach the
155                          * next valid symbol in the section.
156                          */
```



```
157                         if ((next = dis_tgt_next_symbol(tgt, addr)) == 0) {
158                                 db.db_nextaddr = db.db_addr + db.db_size;
159                         } else {
160                                 if (next > db.db_size)
161                                         db.db_nextaddr = db.db_addr +
162                                             db.db_size;
163                                 else
164                                         db.db_nextaddr = addr + next;
165                         }
166 #endif
167                 }

169                 /*
170                  * Print out the line as:
171                  *
172                  *      address:        bytes   text
173                  *
174                  * If there are more than 6 bytes in any given instruction,
175                  * spread the bytes across two lines.  We try to get symbolic
176                  * information for the address, but if that fails we print out
177                  * the numeric address instead.
178                  *
179                  * We try to keep the address portion of the text aligned at
180                  * MINSYMWIDTH characters.  If we are disassembling a function
181                  * with a long name, this can be annoying.  So we pick a width
182                  * based on the maximum width that the current symbol can be.
183                  * This at least produces text aligned within each function.
184                  */
185                 last_symbol = symbol;
186                 symbol = dis_tgt_lookup(tgt, addr, &symoffset, 1, &symsize,
187                     &isfunc);
188                 if (symbol == NULL) {
189                         symbol = dis_find_section(tgt, addr, &symoffset);
190                         symsize = symoffset;
191                 }

193                 if (symbol != last_symbol)
194                         getsymname(addr, symbol, symsize, symbuf,
195                             sizeof (symbuf));

197                 symwidth = MAX(symwidth, strlen(symbuf));
198                 getsymname(addr, symbol, symoffset, symbuf, sizeof (symbuf));

200                 /*
201                  * If we've crossed a new function boundary, print out the
202                  * function name on a blank line.
203                  */
204                 if (!g_quiet && symoffset == 0 && symbol != NULL && isfunc)
205                         (void) printf("%s()\n", symbol);

207                 (void) printf("    %s:%*s ", symbuf,
208                     symwidth - strlen(symbuf), "");

210                 /* print bytes */
211                 for (i = 0; i < MIN(bytesperline, (db.db_nextaddr - addr));
212                     i++) {
213                         int byte = *((uchar_t *)data + (addr - db.db_addr) + i);
214                         if (g_flags & DIS_OCTAL)
215                                 (void) printf("%03o ", byte);
216                         else
217                                 (void) printf("%02x ", byte);
218                 }

220                 /* trailing spaces for missing bytes */
221                 for (; i < bytesperline; i++) {
222                         if (g_flags & DIS_OCTAL)
```

```
223                                 (void) printf("    ");
224                         else
225                                 (void) printf("   ");
226                 }

228                 /* contents of disassembly */
229                 (void) printf(" %s", buf);

231                 /* excess bytes that spill over onto subsequent lines */
232                 for (; i < db.db_nextaddr - addr; i++) {
233                         int byte = *((uchar_t *)data + (addr - db.db_addr) + i);
234                         if (i % bytesperline == 0)
235                                 (void) printf("\n    %*s  ", symwidth, "");
236                         if (g_flags & DIS_OCTAL)
237                                 (void) printf("%03o ", byte);
238                         else
239                                 (void) printf("%02x ", byte);
240                 }

242                 (void) printf("\n");

244                 addr = db.db_nextaddr;
245         }
246 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   23135 Sat Aug 17 17:56:51 2013
new/usr/src/cmd/dis/dis_target.c
4054 dis sometimes decides random symbols are functions
**********************************************************
______unchanged_portion_omitted_

 624 /*
 625  * Given an address, returns the name of the corresponding symbol, as well as
 626  * the offset within that symbol.  If no matching symbol is found, then NULL is
 627  * returned.
 628  *
 629  * If 'cache_result' is specified, then we keep track of the resulting symbol.
 630  * This cached result is consulted first on subsequent lookups in order to avoid
 631  * unecessary lookups.  This flag should be used for resolving the current PC,
 632  * as the majority of addresses stay within the current function.
 633  */
 634 const char *
 635 dis_tgt_lookup(dis_tgt_t *tgt, uint64_t addr, off_t *offset, int cache_result,
 636     size_t *size, int *isfunc)
 637 {
 638         int lo, hi, mid;
 639         sym_entry_t *sym, *osym, *match;
 640         int found;

 642         if (tgt->dt_symcache != NULL &&
 643             addr >= tgt->dt_symcache->se_sym.st_value &&
 644             addr < tgt->dt_symcache->se_sym.st_value +
 645             tgt->dt_symcache->se_sym.st_size) {
 646                 sym = tgt->dt_symcache;
 647                 *offset = addr - sym->se_sym.st_value;
 648                 *size = sym->se_sym.st_size;
 649                 if (isfunc != NULL)
 650                         *isfunc = (GELF_ST_TYPE(sym->se_sym.st_info) ==
 651                             STT_FUNC);
 652                 return (sym->se_name);
 646                 *offset = addr - tgt->dt_symcache->se_sym.st_value;
 647                 *size = tgt->dt_symcache->se_sym.st_size;
 648                 return (tgt->dt_symcache->se_name);
 653         }

 655         lo = 0;
 656         hi = (tgt->dt_symcount - 1);
 657         found = 0;
 658         match = osym = NULL;
 659         while (lo <= hi) {
 660                 mid = (lo + hi) / 2;

 662                 sym = &tgt->dt_symtab[mid];

 664                 if (addr >= sym->se_sym.st_value &&
 665                     addr < sym->se_sym.st_value + sym->se_sym.st_size &&
 666                     (!found || sym->se_sym.st_value > osym->se_sym.st_value)) {
 667                         osym = sym;
 668                         found = 1;
 669                 } else if (addr == sym->se_sym.st_value) {
 670                         /*
 671                          * Particularly for .plt objects, it's possible to have
 672                          * a zero sized object.  We want to return this, but we
 673                          * want it to be a last resort.
 674                          */
 675                         match = sym;
 676                 }

 678                 if (addr < sym->se_sym.st_value)
 679                         hi = mid - 1;
```



```
 680                 else
 681                         lo = mid + 1;
 682         }

 684         if (!found) {
 685                 if (match)
 686                         osym = match;
 687                 else
 688                         return (NULL);
 689         }

 691         /*
 692          * Walk backwards to find the best match.
 693          */
 694         do {
 695                 sym = osym;

 697                 if (osym == tgt->dt_symtab)
 698                         break;

 700                 osym = osym - 1;
 701         } while ((sym->se_sym.st_value == osym->se_sym.st_value) &&
 702             (addr >= osym->se_sym.st_value) &&
 703             (addr < osym->se_sym.st_value + osym->se_sym.st_size));

 705         if (cache_result)
 706                 tgt->dt_symcache = sym;

 708         *offset = addr - sym->se_sym.st_value;
 709         *size = sym->se_sym.st_size;
 710         if (isfunc)
 711                 *isfunc = (GELF_ST_TYPE(sym->se_sym.st_info) == STT_FUNC);

 713         return (sym->se_name);
 714 }
______unchanged_portion_omitted_
```