```
**********************************************************
  145005 Fri Jan  4 21:33:34 2013
new/usr/src/cmd/sgs/elfdump/common/elfdump.c
3450 elfdump crashes on non-Solaris/Linux objects
**********************************************************
______unchanged_portion_omitted_

2208 /*
2209  * Display version section information if the flags require it.
2210  * Return version information needed by other output.
2211  *
2212  * entry:
2213  *      cache - Cache of all section headers
2214  *      shnum - # of sections in cache
2215  *      file - Name of file
2216  *      flags - Command line option flags
2217  *      versym - VERSYM_STATE block to be filled in.
2218  */
2219 static void
2220 versions(Cache *cache, Word shnum, const char *file, uint_t flags,
2221     VERSYM_STATE *versym)
2222 {
2223         GElf_Word       cnt;
2224         Cache           *verdef_cache = NULL, *verneed_cache = NULL;


2227         /* Gather information about the version sections */
2228         bzero(versym, sizeof (*versym));
2228         versym->max_verndx = 1;
2229         for (cnt = 1; cnt < shnum; cnt++) {
2230                 Cache           *_cache = &cache[cnt];
2231                 Shdr            *shdr = _cache->c_shdr;
2232                 Dyn             *dyn;
2233                 ulong_t         numdyn;

2235                 switch (shdr->sh_type) {
2236                 case SHT_DYNAMIC:
2237                         /*
2238                          * The GNU ld puts a DT_VERSYM entry in the dynamic
2239                          * section so that the runtime linker can use it to
2240                          * implement their versioning rules. They allow multiple
2241                          * incompatible functions with the same name to exist
2242                          * in different versions. The Solaris ld does not
2243                          * support this mechanism, and as such, does not
2244                          * produce DT_VERSYM. We use this fact to determine
2245                          * which ld produced this object, and how to interpret
2246                          * the version values.
2247                          */
2248                         if ((shdr->sh_entsize == 0) || (shdr->sh_size == 0) ||
2249                             (_cache->c_data == NULL))
2250                                 continue;
2251                         numdyn = shdr->sh_size / shdr->sh_entsize;
2252                         dyn = (Dyn *)_cache->c_data->d_buf;
2253                         for (; numdyn-- > 0; dyn++)
2254                                 if (dyn->d_tag == DT_VERSYM) {
2255                                         versym->gnu_full =
2256                                             versym->gnu_needed = 1;
2257                                         break;
2258                                 }
2259                         break;

2261                 case SHT_SUNW_versym:
2262                         /* Record data address for later symbol processing */
2263                         if (_cache->c_data != NULL) {
2264                                 versym->cache = _cache;
2265                                 versym->data = _cache->c_data->d_buf;
```



```
2266                                 continue;
2267                         }
2268                         break;

2270                 case SHT_SUNW_verdef:
2271                 case SHT_SUNW_verneed:
2272                         /*
2273                          * Ensure the data is non-NULL and the number
2274                          * of items is non-zero. Otherwise, we don't
2275                          * understand the section, and will not use it.
2276                          */
2277                         if ((_cache->c_data == NULL) ||
2278                             (_cache->c_data->d_buf == NULL)) {
2279                                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSZ),
2280                                     file, _cache->c_name);
2281                                 continue;
2282                         }
2283                         if (shdr->sh_info == 0) {
2284                                 (void) fprintf(stderr,
2285                                     MSG_INTL(MSG_ERR_BADSHINFO),
2286                                     file, _cache->c_name,
2287                                     EC_WORD(shdr->sh_info));
2288                                 continue;
2289                         }

2291                         /* Make sure the string table index is in range */
2292                         if ((shdr->sh_link == 0) || (shdr->sh_link >= shnum)) {
2293                                 (void) fprintf(stderr,
2294                                     MSG_INTL(MSG_ERR_BADSHLINK), file,
2295                                     _cache->c_name, EC_WORD(shdr->sh_link));
2296                                 continue;
2297                         }

2299                         /*
2300                          * The section is usable. Save the cache entry.
2301                          */
2302                         if (shdr->sh_type == SHT_SUNW_verdef) {
2303                                 verdef_cache = _cache;
2304                                 /*
2305                                  * Under Solaris rules, if there is a verdef
2306                                  * section, the max versym index is number
2307                                  * of version definitions it supplies.
2308                                  */
2309                                 versym->max_verndx = shdr->sh_info;
2310                         } else {
2311                                 verneed_cache = _cache;
2312                         }
2313                         break;
2314                 }
2315         }

2317         /*
2318          * If there is a Verneed section, examine it for information
2319          * related to GNU style versioning.
2320          */
2321         if (verneed_cache != NULL)
2322                 update_gnu_verndx((Verneed *)verneed_cache->c_data->d_buf,
2323                     verneed_cache->c_shdr->sh_info, versym);

2325         /*
2326          * Now that all the information is available, display the
2327          * Verdef and Verneed section contents, if requested.
2328          */
2329         if ((flags & FLG_SHOW_VERSIONS) == 0)
2330                 return;
2331         if (verdef_cache != NULL) {
```

```
2332                 dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
2333                 dbg_print(0, MSG_INTL(MSG_ELF_SCN_VERDEF),
2334                     verdef_cache->c_name);
2335                 version_def((Verdef *)verdef_cache->c_data->d_buf,
2336                     verdef_cache->c_shdr->sh_info, verdef_cache,
2337                     &cache[verdef_cache->c_shdr->sh_link], file);
2338         }
2339         if (verneed_cache != NULL) {
2340                 dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
2341                 dbg_print(0, MSG_INTL(MSG_ELF_SCN_VERNEED),
2342                     verneed_cache->c_name);
2343                 /*
2344                  * If GNU versioning applies to this object, version_need()
2345                  * will update versym->max_verndx, and it is not
2346                  * necessary to call update_gnu_verndx().
2347                  */
2348                 version_need((Verneed *)verneed_cache->c_data->d_buf,
2349                     verneed_cache->c_shdr->sh_info, verneed_cache,
2350                     &cache[verneed_cache->c_shdr->sh_link], file, versym);
2351         }
2352 }
```

*unchanged_portion_omitted_*

```
4731 int
4732 regular(const char *file, int fd, Elf *elf, uint_t flags,
4733     const char *wname, int wfd, uchar_t osabi)
4734 {
4735         enum { CACHE_NEEDED, CACHE_OK, CACHE_FAIL} cache_state = CACHE_NEEDED;
4736         Elf_Scn         *scn;
4737         Ehdr            *ehdr;
4738         size_t          ndx, shstrndx, shnum, phnum;
4739         Shdr            *shdr;
4740         Cache           *cache;
4741         VERSYM_STATE    versym = { 0 };
4742         VERSYM_STATE    versym;
4742         int             ret = 0;
4743         int             addr_align;

4745         if ((ehdr = elf_getehdr(elf)) == NULL) {
4746                 failure(file, MSG_ORIG(MSG_ELF_GETEHDR));
4747                 return (ret);
4748         }

4750         if (elf_getshdrnum(elf, &shnum) == -1) {
4751                 failure(file, MSG_ORIG(MSG_ELF_GETSHDRNUM));
4752                 return (ret);
4753         }

4755         if (elf_getshdrstrndx(elf, &shstrndx) == -1) {
4756                 failure(file, MSG_ORIG(MSG_ELF_GETSHDRSTRNDX));
4757                 return (ret);
4758         }

4760         if (elf_getphdrnum(elf, &phnum) == -1) {
4761                 failure(file, MSG_ORIG(MSG_ELF_GETPHDRNUM));
4762                 return (ret);
4763         }
4764         /*
4765          * If the user requested section headers derived from the
4766          * program headers (-P option) and this file doesn't have
4767          * any program headers (i.e. ET_REL), then we can't do it.
4768          */
4769         if ((phnum == 0) && (flags & FLG_CTL_FAKESHDR)) {
4770                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_PNEEDSPH), file);
4771                 return (ret);
4772         }
```



```
4775         if ((scn = elf_getscn(elf, 0)) != NULL) {
4776                 if ((shdr = elf_getshdr(scn)) == NULL) {
4777                         failure(file, MSG_ORIG(MSG_ELF_GETSHDR));
4778                         (void) fprintf(stderr, MSG_INTL(MSG_ELF_ERR_SCN), 0);
4779                         return (ret);
4780                 }
4781         } else
4782                 shdr = NULL;

4784         /*
4785          * Print the elf header.
4786          */
4787         if (flags & FLG_SHOW_EHDR)
4788                 Elf_ehdr(0, ehdr, shdr);

4790         /*
4791          * If the section headers or program headers have inadequate
4792          * alignment for the class of object, print a warning. libelf
4793          * can handle such files, but programs that use them can crash
4794          * when they dereference unaligned items.
4795          *
4796          * Note that the AMD64 ABI, although it is a 64-bit architecture,
4797          * allows access to data types smaller than 128-bits to be on
4798          * word alignment.
4799          */
4800         if (ehdr->e_machine == EM_AMD64)
4801                 addr_align = sizeof (Word);
4802         else
4803                 addr_align = sizeof (Addr);

4805         if (ehdr->e_phoff & (addr_align - 1))
4806                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADPHDRALIGN), file);
4807         if (ehdr->e_shoff & (addr_align - 1))
4808                 (void) fprintf(stderr, MSG_INTL(MSG_ERR_BADSHDRALIGN), file);


4811         /*
4812          * Determine the Operating System ABI (osabi) we will use to
4813          * interpret the object.
4814          */
4815         if (flags & FLG_CTL_OSABI) {
4816                 /*
4817                  * If the user explicitly specifies '-O none', we need
4818                  * to display a completely generic view of the file.
4819                  * However, libconv is written to assume that ELFOSABI_NONE
4820                  * is equivalent to ELFOSABI_SOLARIS. To get the desired
4821                  * effect, we use an osabi that libconv has no knowledge of.
4822                  */
4823                 if (osabi == ELFOSABI_NONE)
4824                         osabi = ELFOSABI_UNKNOWN4;
4825         } else {
4826                 /* Determine osabi from file */
4827                 osabi = ehdr->e_ident[EI_OSABI];
4828                 if (osabi == ELFOSABI_NONE) {
4829                         /*
4830                          * Chicken/Egg scenario:
4831                          *
4832                          * Ideally, we wait to create the section header cache
4833                          * until after the program headers are printed. If we
4834                          * only output program headers, we can skip building
4835                          * the cache entirely.
4836                          *
4837                          * Proper interpretation of program headers requires
4838                          * the osabi, which is supposed to be in the ELF header.
```

```
4839                         * However, many systems (Solaris and Linux included)
4840                         * have a history of setting the osabi to the generic
4841                         * SysV ABI (ELFOSABI_NONE). We assume ELFOSABI_SOLARIS
4842                         * in such cases, but would like to check the object
4843                         * to see if it has a Linux .note.ABI-tag section,
4844                         * which implies ELFOSABI_LINUX. This requires a
4845                         * section header cache.
4846                         *
4847                         * To break the cycle, we create section headers now
4848                         * if osabi is ELFOSABI_NONE, and later otherwise.
4849                         * If it succeeds, we use them, if not, we defer
4850                         * exiting until after the program headers are out.
4851                         */
4852                        if (create_cache(file, fd, elf, ehdr, &cache,
4853                            shstrndx, &shnum, &flags) == 0) {
4854                                cache_state = CACHE_FAIL;
4855                        } else {
4856                                cache_state = CACHE_OK;
4857                                if (has_linux_abi_note(cache, shnum, file)) {
4858                                        Conv_inv_buf_t  ibuf1, ibuf2;

4860                                        (void) fprintf(stderr,
4861                                            MSG_INTL(MSG_INFO_LINUXOSABI), file,
4862                                            conv_ehdr_osabi(osabi, 0, &ibuf1),
4863                                            conv_ehdr_osabi(ELFOSABI_LINUX,
4864                                            0, &ibuf2));
4865                                        osabi = ELFOSABI_LINUX;
4866                                }
4867                        }
4868                }
4869                /*
4870                 * We treat ELFOSABI_NONE identically to ELFOSABI_SOLARIS.
4871                 * Mapping NONE to SOLARIS simplifies the required test.
4872                 */
4873                if (osabi == ELFOSABI_NONE)
4874                        osabi = ELFOSABI_SOLARIS;
4875        }

4877        /*
4878         * Print the program headers.
4879         */
4880        if ((flags & FLG_SHOW_PHDR) && (phnum != 0)) {
4881                Phdr    *phdr;

4883                if ((phdr = elf_getphdr(elf)) == NULL) {
4884                        failure(file, MSG_ORIG(MSG_ELF_GETPHDR));
4885                        return (ret);
4886                }

4888                for (ndx = 0; ndx < phnum; phdr++, ndx++) {
4889                        if (!match(MATCH_F_PHDR| MATCH_F_NDX | MATCH_F_TYPE,
4890                            NULL, ndx, phdr->p_type))
4891                                continue;

4893                        dbg_print(0, MSG_ORIG(MSG_STR_EMPTY));
4894                        dbg_print(0, MSG_INTL(MSG_ELF_PHDR), EC_WORD(ndx));
4895                        Elf_phdr(0, osabi, ehdr->e_machine, phdr);
4896                }
4897        }

4899        /*
4900         * If we have flag bits set that explicitly require a show or calc
4901         * operation, but none of them require the section headers, then
4902         * we are done and can return now.
4903         */
4904        if (((flags & (FLG_MASK_SHOW | FLG_MASK_CALC)) != 0) &&
```



```
4905            ((flags & (FLG_MASK_SHOW_SHDR | FLG_MASK_CALC_SHDR)) == 0))
4906                return (ret);

4908        /*
4909         * Everything from this point on requires section headers.
4910         * If we have no section headers, there is no reason to continue.
4911         *
4912         * If we tried above to create the section header cache and failed,
4913         * it is time to exit. Otherwise, create it if needed.
4914         */
4915        switch (cache_state) {
4916        case CACHE_NEEDED:
4917                if (create_cache(file, fd, elf, ehdr, &cache, shstrndx,
4918                    &shnum, &flags) == 0)
4919                        return (ret);
4920                break;
4921        case CACHE_FAIL:
4922                return (ret);
4923        }
4924        if (shnum <= 1)
4925                goto done;

4927        /*
4928         * If -w was specified, find and write out the section(s) data.
4929         */
4930        if (wfd) {
4931                for (ndx = 1; ndx < shnum; ndx++) {
4932                        Cache   *_cache = &cache[ndx];

4934                        if (match(MATCH_F_STRICT | MATCH_F_ALL, _cache->c_name,
4935                            ndx, _cache->c_shdr->sh_type) &&
4936                            _cache->c_data && _cache->c_data->d_buf) {
4937                                if (write(wfd, _cache->c_data->d_buf,
4938                                    _cache->c_data->d_size) !=
4939                                    _cache->c_data->d_size) {
4940                                        int err = errno;
4941                                        (void) fprintf(stderr,
4942                                            MSG_INTL(MSG_ERR_WRITE), wname,
4943                                            strerror(err));
4944                                        /*
4945                                         * Return an exit status of 1, because
4946                                         * the failure is not related to the
4947                                         * ELF file, but by system resources.
4948                                         */
4949                                        ret = 1;
4950                                        goto done;
4951                                }
4952                        }
4953                }
4954        }

4956        /*
4957         * If we have no flag bits set that explicitly require a show or calc
4958         * operation, but match options (-I, -N, -T) were used, then run
4959         * through the section headers and see if we can't deduce show flags
4960         * from the match options given.
4961         *
4962         * We don't do this if -w was specified, because (-I, -N, -T) used
4963         * with -w in lieu of some other option is supposed to be quiet.
4964         */
4965        if ((wfd == 0) && (flags & FLG_CTL_MATCH) &&
4966            ((flags & (FLG_MASK_SHOW | FLG_MASK_CALC)) == 0)) {
4967                for (ndx = 1; ndx < shnum; ndx++) {
4968                        Cache   *_cache = &cache[ndx];

4970                        if (!match(MATCH_F_STRICT | MATCH_F_ALL, _cache->c_name,
```

```
4971                            ndx, _cache->c_shdr->sh_type))
4972                                continue;
4973
4974                        switch (_cache->c_shdr->sh_type) {
4975                        case SHT_PROGBITS:
4976                                /*
4977                                 * Heuristic time: It is usually bad form
4978                                 * to assume the meaning/format of a PROGBITS
4979                                 * section based on its name. However, there
4980                                 * are ABI mandated exceptions. Check for
4981                                 * these special names.
4982                                 */
4983
4984                                /* The ELF ABI specifies .interp and .got */
4985                                if (strcmp(_cache->c_name,
4986                                    MSG_ORIG(MSG_ELF_INTERP)) == 0) {
4987                                        flags |= FLG_SHOW_INTERP;
4988                                        break;
4989                                }
4990                                if (strcmp(_cache->c_name,
4991                                    MSG_ORIG(MSG_ELF_GOT)) == 0) {
4992                                        flags |= FLG_SHOW_GOT;
4993                                        break;
4994                                }
4995                                /*
4996                                 * The GNU compilers, and amd64 ABI, define
4997                                 * .eh_frame and .eh_frame_hdr. The Sun
4998                                 * C++ ABI defines .exception_ranges.
4999                                 */
5000                                if ((strncmp(_cache->c_name,
5001                                    MSG_ORIG(MSG_SCN_FRM),
5002                                    MSG_SCN_FRM_SIZE) == 0) ||
5003                                    (strncmp(_cache->c_name,
5004                                    MSG_ORIG(MSG_SCN_EXRANGE),
5005                                    MSG_SCN_EXRANGE_SIZE) == 0)) {
5006                                        flags |= FLG_SHOW_UNWIND;
5007                                        break;
5008                                }
5009                                break;
5010
5011                        case SHT_SYMTAB:
5012                        case SHT_DYNSYM:
5013                        case SHT_SUNW_LDYNSYM:
5014                        case SHT_SUNW_versym:
5015                        case SHT_SYMTAB_SHNDX:
5016                                flags |= FLG_SHOW_SYMBOLS;
5017                                break;
5018
5019                        case SHT_RELA:
5020                        case SHT_REL:
5021                                flags |= FLG_SHOW_RELOC;
5022                                break;
5023
5024                        case SHT_HASH:
5025                                flags |= FLG_SHOW_HASH;
5026                                break;
5027
5028                        case SHT_DYNAMIC:
5029                                flags |= FLG_SHOW_DYNAMIC;
5030                                break;
5031
5032                        case SHT_NOTE:
5033                                flags |= FLG_SHOW_NOTE;
5034                                break;
5035
5036                        case SHT_GROUP:
```



```
5037                                flags |= FLG_SHOW_GROUP;
5038                                break;
5039
5040                        case SHT_SUNW_symsort:
5041                        case SHT_SUNW_tlssort:
5042                                flags |= FLG_SHOW_SORT;
5043                                break;
5044
5045                        case SHT_SUNW_cap:
5046                                flags |= FLG_SHOW_CAP;
5047                                break;
5048
5049                        case SHT_SUNW_move:
5050                                flags |= FLG_SHOW_MOVE;
5051                                break;
5052
5053                        case SHT_SUNW_syminfo:
5054                                flags |= FLG_SHOW_SYMINFO;
5055                                break;
5056
5057                        case SHT_SUNW_verdef:
5058                        case SHT_SUNW_verneed:
5059                                flags |= FLG_SHOW_VERSIONS;
5060                                break;
5061
5062                        case SHT_AMD64_UNWIND:
5063                                flags |= FLG_SHOW_UNWIND;
5064                                break;
5065                        }
5066                }
5067        }
5068
5069
5070        if (flags & FLG_SHOW_SHDR)
5071                sections(file, cache, shnum, ehdr, osabi);
5072
5073        if (flags & FLG_SHOW_INTERP)
5074                interp(file, cache, shnum, phnum, elf);
5075
5076        if ((osabi == ELFOSABI_SOLARIS) || (osabi == ELFOSABI_LINUX))
5077                versions(cache, shnum, file, flags, &versym);
5078
5079        if (flags & FLG_SHOW_SYMBOLS)
5080                symbols(cache, shnum, ehdr, osabi, &versym, file, flags);
5081
5082        if ((flags & FLG_SHOW_SORT) && (osabi == ELFOSABI_SOLARIS))
5083                sunw_sort(cache, shnum, ehdr, osabi, &versym, file, flags);
5084
5085        if (flags & FLG_SHOW_HASH)
5086                hash(cache, shnum, file, flags);
5087
5088        if (flags & FLG_SHOW_GOT)
5089                got(cache, shnum, ehdr, file);
5090
5091        if (flags & FLG_SHOW_GROUP)
5092                group(cache, shnum, file, flags);
5093
5094        if (flags & FLG_SHOW_SYMINFO)
5095                syminfo(cache, shnum, ehdr, osabi, file);
5096
5097        if (flags & FLG_SHOW_RELOC)
5098                reloc(cache, shnum, ehdr, file);
5099
5100        if (flags & FLG_SHOW_DYNAMIC)
5101                dynamic(cache, shnum, ehdr, osabi, file);
```

```
5103         if (flags & FLG_SHOW_NOTE) {
5104                 Word    note_cnt;
5105                 size_t  note_shnum;
5106                 Cache   *note_cache;

5108                 note_cnt = note(cache, shnum, ehdr, file);

5110                 /*
5111                  * Solaris core files have section headers, but these
5112                  * headers do not include SHT_NOTE sections that reference
5113                  * the core note sections. This means that note() won't
5114                  * find the core notes. Fake section headers (-P option)
5115                  * recover these sections, but it is inconvenient to require
5116                  * users to specify -P in this situation. If the following
5117                  * are all true:
5118                  *
5119                  *      - No note sections were found
5120                  *      - This is a core file
5121                  *      - We are not already using fake section headers
5122                  *
5123                  * then we will automatically generate fake section headers
5124                  * and then process them in a second call to note().
5125                  */
5126                 if ((note_cnt == 0) && (ehdr->e_type == ET_CORE) &&
5127                     !(flags & FLG_CTL_FAKESHDR) &&
5128                     (fake_shdr_cache(file, fd, elf, ehdr,
5129                     &note_cache, &note_shnum) != 0)) {
5130                         (void) note(note_cache, note_shnum, ehdr, file);
5131                         fake_shdr_cache_free(note_cache, note_shnum);
5132                 }
5133         }

5135         if ((flags & FLG_SHOW_MOVE) && (osabi == ELFOSABI_SOLARIS))
5136                 move(cache, shnum, file, flags);

5138         if (flags & FLG_CALC_CHECKSUM)
5139                 checksum(elf);

5141         if ((flags & FLG_SHOW_CAP) && (osabi == ELFOSABI_SOLARIS))
5142                 cap(file, cache, shnum, phnum, ehdr, osabi, elf, flags);

5144         if ((flags & FLG_SHOW_UNWIND) &&
5145             ((osabi == ELFOSABI_SOLARIS) || (osabi == ELFOSABI_LINUX)))
5146                 unwind(cache, shnum, phnum, ehdr, osabi, file, elf, flags);


5149         /* Release the memory used to cache section headers */
5150 done:
5151         if (flags & FLG_CTL_FAKESHDR)
5152                 fake_shdr_cache_free(cache, shnum);
5153         else
5154                 free(cache);

5156         return (ret);
5157 }
```

_____***unchanged_portion_omitted_***