```
**********************************************************
   22304 Wed Dec 10 14:40:47 2014
new/usr/src/cmd/sgs/libld/common/unwind.c
5425 ld_unwind_populate_hdr likely misaccounts for 'P'
**********************************************************
______unchanged_portion_omitted_

 482 uintptr_t
 483 ld_unwind_populate_hdr(Ofl_desc *ofl)
 484 {
 485         uchar_t         *hdrdata;
 486         uint_t          *binarytable;
 487         uint_t          hdroff;
 488         Aliste          idx;
 489         Addr            hdraddr;
 490         Os_desc         *hdrosp;
 491         Os_desc         *osp;
 492         Os_desc         *first_unwind;
 493         uint_t          fde_count;
 494         uint_t          *uint_ptr;
 495         int             bswap = (ofl->ofl_flags1 & FLG_OF1_ENCDIFF) != 0;

 497         /*
 498          * Are we building the unwind hdr?
 499          */
 500         if ((hdrosp = ofl->ofl_unwindhdr) == 0)
 501                 return (1);

 503         hdrdata = hdrosp->os_outdata->d_buf;
 504         hdraddr = hdrosp->os_shdr->sh_addr;
 505         hdroff = 0;

 507         /*
 508          * version == 1
 509          */
 510         hdrdata[hdroff++] = 1;
 511         /*
 512          * The encodings are:
 513          *
 514          *  eh_frameptr_enc     sdata4 | pcrel
 515          *  fde_count_enc       udata4
 516          *  table_enc           sdata4 | datarel
 517          */
 518         hdrdata[hdroff++] = DW_EH_PE_sdata4 | DW_EH_PE_pcrel;
 519         hdrdata[hdroff++] = DW_EH_PE_udata4;
 520         hdrdata[hdroff++] = DW_EH_PE_sdata4 | DW_EH_PE_datarel;

 522         /*
 523          *      Header Offsets
 524          *      ----------------------------------
 525          *      byte        version             +1
 526          *      byte        eh_frame_ptr_enc    +1
 527          *      byte        fde_count_enc       +1
 528          *      byte        table_enc           +1
 529          *      4 bytes     eh_frame_ptr        +4
 530          *      4 bytes     fde_count           +4
 531          */
 532         /* LINTED */
 533         binarytable =  (uint_t *)(hdrdata + 12);
 534         first_unwind = 0;
 535         fde_count = 0;

 537         for (APLIST_TRAVERSE(ofl->ofl_unwind, idx, osp)) {
 538                 uchar_t         *data;
 539                 size_t          size;
 540                 uint64_t        off = 0;
```



```
 541                 uint_t          cieRflag = 0, ciePflag = 0;
 542                 Shdr            *shdr;

 544                 /*
 545                  * remember first UNWIND section to
 546                  * point to in the frame_ptr entry.
 547                  */
 548                 if (first_unwind == 0)
 549                         first_unwind = osp;

 551                 data = osp->os_outdata->d_buf;
 552                 shdr = osp->os_shdr;
 553                 size = shdr->sh_size;

 555                 while (off < size) {
 556                         uint_t      length, id;
 557                         uint64_t    ndx = 0;

 559                         /*
 560                          * Extract length in lsb format.  A zero length
 561                          * indicates that this CIE is a terminator and that
 562                          * processing of unwind information is complete.
 563                          */
 564                         length = extract_uint(data + off, &ndx, bswap);
 565                         if (length == 0)
 566                                 goto done;

 568                         /*
 569                          * Extract CIE id in lsb format.
 570                          */
 571                         id = extract_uint(data + off, &ndx, bswap);

 573                         /*
 574                          * A CIE record has a id of '0'; otherwise
 575                          * this is a FDE entry and the 'id' is the
 576                          * CIE pointer.
 577                          */
 578                         if (id == 0) {
 579                                 char    *cieaugstr;
 580                                 uint_t  cieaugndx;
 581                                 uint_t  cieversion;

 583                                 ciePflag = 0;
 584                                 cieRflag = 0;
 585                                 /*
 586                                  * We need to drill through the CIE
 587                                  * to find the Rflag.  It's the Rflag
 588                                  * which describes how the FDE code-pointers
 589                                  * are encoded.
 590                                  */

 592                                 cieversion = data[off + ndx];
 593                                 ndx += 1;

 595                                 /*
 596                                  * augstr
 597                                  */
 598                                 cieaugstr = (char *)(&data[off + ndx]);
 599                                 ndx += strlen(cieaugstr) + 1;

 601                                 /*
 602                                  * calign & dalign
 603                                  */
 604                                 (void) uleb_extract(&data[off], &ndx);
 605                                 (void) sleb_extract(&data[off], &ndx);
```

```
607                             /*
608                              * retreg
609                              */
610                             if (cieversion == 1)
611                                     ndx++;
612                             else
613                                     (void) uleb_extract(&data[off], &ndx);
614                             /*
615                              * we walk through the augmentation
616                              * section now looking for the Rflag
617                              */
618                             for (cieaugndx = 0; cieaugstr[cieaugndx];
619                                 cieaugndx++) {
620                                 /* BEGIN CSTYLED */
621                                 switch (cieaugstr[cieaugndx]) {
622                                 case 'z':
623                                     /* size */
624                                     (void) uleb_extract(&data[off],
625                                         &ndx);
626                                     break;
627                                 case 'P':
628                                     /* personality */
629                                     ciePflag = data[off + ndx];
630                                     ndx++;
631                                         /*
632                                          * Just need to extract the
633                                          * value to move on to the next
634                                          * field.
635                                          */
636                                     (void) dwarf_ehe_extract(
637                                         &data[off],
638                                         &data[off + ndx],
639                                         &ndx, ciePflag,
640                                         ofl->ofl_dehdr->e_ident, B_FALSE
641                                         shdr->sh_addr, off + ndx, 0);
642                                     break;
643                                 case 'R':
644                                     /* code encoding */
645                                     cieRflag = data[off + ndx];
646                                     ndx++;
647                                     break;
648                                 case 'L':
649                                     /* lsda encoding */
650                                     ndx++;
651                                     break;
652                                 }
653                                 /* END CSTYLED */
654                             }
655                     } else {
656                             uint_t      bintabndx;
657                             uint64_t    initloc;
658                             uint64_t    fdeaddr;
659                             uint64_t    gotaddr = 0;
660
661                             if (ofl->ofl_osgot != NULL)
662                                     gotaddr =
663                                         ofl->ofl_osgot->os_shdr->sh_addr;
664
665                             initloc = dwarf_ehe_extract(&data[off],
666                                 &ndx, cieRflag, ofl->ofl_dehdr->e_ident,
667                                 B_FALSE,
668                                 shdr->sh_addr, off + ndx,
669                                 gotaddr);
670
671                             /*
```



```
672                              * initloc will not be 0 unless this FDE was
673                              * abandoned due to GNU linkonce processing.
674                              * The 0 value occurs because we don't resolve
675                              * sloppy relocations for unwind header target
676                              * sections.
677                              */
678                             if (initloc != 0) {
679                                     bintabndx = fde_count * 2;
680                                     fde_count++;
681
682                                     /*
683                                      * FDEaddr is adjusted
684                                      * to account for the length & id which
685                                      * have already been consumed.
686                                      */
687                                     fdeaddr = shdr->sh_addr + off;
688
689                                     binarytable[bintabndx] =
690                                         (uint_t)(initloc - hdraddr);
691                                     binarytable[bintabndx + 1] =
692                                         (uint_t)(fdeaddr - hdraddr);
693                             }
694                     }
695
696                     /*
697                      * the length does not include the length
698                      * itself - so account for that too.
699                      */
700                     off += length + 4;
701             }
702     }
703
704 done:
705     /*
706      * Do a quicksort on the binary table. If this is a cross
707      * link from a system with the opposite byte order, xlate
708      * the resulting values into LSB order.
709      */
710     framehdr_addr = hdraddr;
711     qsort((void *)binarytable, (size_t)fde_count,
712         (size_t)(sizeof (uint_t) * 2), bintabcompare);
713     if (bswap) {
714             uint_t  *btable = binarytable;
715             uint_t  cnt;
716
717             for (cnt = fde_count * 2; cnt-- > 0; btable++)
718                     *btable = ld_bswap_Word(*btable);
719     }
720
721     /*
722      * Fill in:
723      *      first_frame_ptr
724      *      fde_count
725      */
726     hdroff = 4;
727     /* LINTED */
728     uint_ptr = (uint_t *)(&hdrdata[hdroff]);
729     *uint_ptr = first_unwind->os_shdr->sh_addr -
730         (hdrosp->os_shdr->sh_addr + hdroff);
731     if (bswap)
732             *uint_ptr = ld_bswap_Word(*uint_ptr);
733
734     hdroff += 4;
735     /* LINTED */
736     uint_ptr = (uint_t *)&hdrdata[hdroff];
737     *uint_ptr = fde_count;
```

```
738         if (bswap)
739                 *uint_ptr = ld_bswap_Word(*uint_ptr);

741         /*
742          * If relaxed relocations are active, then there is a chance
743          * that we didn't use all the space reserved for this section.
744          * For details, see the note at head of ld_unwind_make_hdr() above.
745          *
746          * Find the PT_SUNW_UNWIND program header, and change the size values
747          * to the size of the subset of the section that was actually used.
748          */
749         if (ofl->ofl_flags1 & FLG_OF1_RLXREL) {
750                 Word    phnum = ofl->ofl_nehdr->e_phnum;
751                 Phdr    *phdr = ofl->ofl_phdr;

753                 for (; phnum-- > 0; phdr++) {
754                         if (phdr->p_type == PT_SUNW_UNWIND) {
755                                 phdr->p_memsz = 12 + (8 * fde_count);
756                                 phdr->p_filesz = phdr->p_memsz;
757                                 break;
758                         }
759                 }
760         }

762         return (1);
763 }
```

*unchanged_portion_omitted_*