```
**********************************************************
   54095 Wed Jun  5 03:13:14 2019
new/usr/src/tools/cpcgen/cpcgen.c
11200 cpcgen needs smatch fixes again
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2019, Joyent, Inc.
 * Copyright (c) 2019, Joyent, Inc.
 */

/*
 * This program transforms Intel perfmon and AMD PMC data files into C files and
 * manual pages.
 */

#include <stdio.h>
#include <stdarg.h>
#include <unistd.h>
#include <err.h>
#include <libgen.h>
#include <libnvpair.h>
#include <strings.h>
#include <errno.h>
#include <limits.h>
#include <sys/mman.h>
#include <sys/param.h>
#include <assert.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>

#include <json_nvlist.h>

#define EXIT_USAGE      2
#define CPROC_MAX_STEPPINGS     16

typedef enum {
        CPCGEN_MODE_UNKNOWN = 0,
        CPCGEN_MODE_INTEL,
        CPCGEN_MODE_AMD
} cpc_mode_t;
```

*unchanged_portion_omitted_*

```
/*
 * Parse a string of the form 'GenuineIntel-6-2E' and get out the family and
 * model.
 */
static void
cpcgen_parse_model(char *fsr, uint_t *family, uint_t *model, uint_t *nstepp,
    uint_t *steppings)
{
        const char *bstr = "GenuineIntel";
        const char *brand, *fam, *mod, *step;
```



```
        char *last;
        long l;
        uint_t nstep = 0;

        /*
         * Tokeninze the string. There may be an optional stepping portion,
         * which has a range of steppings enclosed by '[' and ']' characters.
         * While the other parts are required, the stepping may be missing.
         */
        if ((brand = strtok_r(fsr, "-", &last)) == NULL ||
            (fam = strtok_r(NULL, "-", &last)) == NULL ||
            (mod = strtok_r(NULL, "-", &last)) == NULL) {
                errx(EXIT_FAILURE, "failed to parse processor id \"%s\"", fsr);
        }
        step = strtok_r(NULL, "-", &last);

        if (strcmp(bstr, brand) != 0) {
                errx(EXIT_FAILURE, "brand string \"%s\" did not match \"%s\"",
                    brand, bstr);
        }

        errno = 0;
        l = strtol(fam, &last, 16);
        if (errno != 0 || l < 0 || l >= INT_MAX || *last != '\0') {
        if (errno != 0 || l < 0 || l > UINT_MAX || *last != '\0') {
                errx(EXIT_FAILURE, "failed to parse family \"%s\"", fam);
        }
        *family = (uint_t)l;

        l = strtol(mod, &last, 16);
        if (errno != 0 || l < 0 || l >= INT_MAX || *last != '\0') {
        if (errno != 0 || l < 0 || l > UINT_MAX || *last != '\0') {
                errx(EXIT_FAILURE, "failed to parse model \"%s\"", mod);
        }
        *model = (uint_t)l;

        if (step == NULL) {
                *nstepp = 0;
                return;
        }

        if (*step != '[' || ((last = strrchr(step, ']')) == NULL)) {
                errx(EXIT_FAILURE, "failed to parse stepping \"%s\": missing "
                    "stepping range brackets", step);
        }
        step++;
        *last = '\0';
        while (*step != '\0') {
                if (!isxdigit(*step)) {
                        errx(EXIT_FAILURE, "failed to parse stepping: invalid "
                            "stepping identifier '0x%x'", *step);
                }

                if (nstep >= CPROC_MAX_STEPPINGS) {
                        errx(EXIT_FAILURE, "failed to parse stepping: "
                            "encountered too many steppings");
                }

                switch (*step) {
                case '0':
                        steppings[nstep] = 0x0;
                        break;
                case '1':
                        steppings[nstep] = 0x1;
                        break;
                case '2':
```

```
463                         steppings[nstep] = 0x2;
464                         break;
465                 case '3':
466                         steppings[nstep] = 0x3;
467                         break;
468                 case '4':
469                         steppings[nstep] = 0x4;
470                         break;
471                 case '5':
472                         steppings[nstep] = 0x5;
473                         break;
474                 case '6':
475                         steppings[nstep] = 0x6;
476                         break;
477                 case '7':
478                         steppings[nstep] = 0x7;
479                         break;
480                 case '8':
481                         steppings[nstep] = 0x8;
482                         break;
483                 case '9':
484                         steppings[nstep] = 0x9;
485                         break;
486                 case 'a':
487                 case 'A':
488                         steppings[nstep] = 0xa;
489                         break;
490                 case 'b':
491                 case 'B':
492                         steppings[nstep] = 0xb;
493                         break;
494                 case 'c':
495                 case 'C':
496                         steppings[nstep] = 0xc;
497                         break;
498                 case 'd':
499                 case 'D':
500                         steppings[nstep] = 0xd;
501                         break;
502                 case 'e':
503                 case 'E':
504                         steppings[nstep] = 0xe;
505                         break;
506                 case 'f':
507                 case 'F':
508                         steppings[nstep] = 0xf;
509                         break;
510                 default:
511                         errx(EXIT_FAILURE, "encountered non-hex stepping "
512                             "character: '%c'", *step);
513                 }
514                 nstep++;
515                 step++;
516         }

518         *nstepp = nstep;
519 }
```

*unchanged_portion_omitted_*

```
839 static boolean_t
840 cpcgen_manual_intel_file_before(FILE *f, cpc_map_t *map)
841 {
842         size_t i;
843         char *upper;
844         cpc_proc_t *proc;
```



```
846         if ((upper = strdup(map->cmap_name)) == NULL) {
847                 warn("failed to duplicate manual name for %s", map->cmap_name);
848                 return (B_FALSE);
849         }

851         for (i = 0; upper[i] != '\0'; i++) {
852                 upper[i] = toupper(upper[i]);
853         }

855         if (fprintf(f, cpcgen_manual_intel_intel_header, map->cmap_path, upper,
856             map->cmap_name) == -1) {
857                 warn("failed to write out manual header for %s",
858                     map->cmap_name);
859                 free(upper);
860                 return (B_FALSE);
861         }
862         free(upper);

864         for (proc = map->cmap_procs; proc != NULL; proc = proc->cproc_next) {
865                 if (proc->cproc_nsteps > 0) {
866                         uint_t step;

868                         for (step = 0; step < proc->cproc_nsteps; step++) {
869                                 if (fprintf(f, ".It\n.Sy Family 0x%x, Model "
870                                     "0x%x, Stepping 0x%x\n",
871                                     proc->cproc_family, proc->cproc_model,
872                                     proc->cproc_steppings[step]) == -1) {
873                                         warn("failed to write out model "
874                                             "information for %s",
875                                             map->cmap_name);
876                                         return (B_FALSE);
877                                 }
878                         }
879                 } else {
880                         if (fprintf(f, ".It\n.Sy Family 0x%x, Model 0x%x\n",
881                             proc->cproc_family, proc->cproc_model) == -1) {
882                                 warn("failed to write out model information "
883                                     "for %s", map->cmap_name);
884                                 return (B_FALSE);
885                         }
886                 }
887         }

889         if (fprintf(f, cpcgen_manual_intel_data) == -1) {
890                 warn("failed to write out manual header for %s",
891                     map->cmap_name);
892                 return (B_FALSE);
893         }

895         free(upper);
895         return (B_TRUE);
896 }
```

*unchanged_portion_omitted_*