```
**********************************************************
   14783 Mon Jan 21 16:21:44 2019
new/usr/src/cmd/csplit/csplit.c
10128 csplit should use strlcpy
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */

/*
 * csplit - Context or line file splitter
 * Compile: cc -O -s -o csplit csplit.c
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>
#include <limits.h>
#include <regexpr.h>
#include <signal.h>
#include <locale.h>
#include <libintl.h>

#define LAST    0LL
#define ERR     -1
#define FALSE   0
#define TRUE    1
#define EXPMODE 2
#define LINMODE 3
#define LINSIZ  LINE_MAX        /* POSIX.2 - read lines LINE_MAX long */

        /* Globals */

char linbuf[LINSIZ];            /* Input line buffer */
char *expbuf;
```



```
char tmpbuf[BUFSIZ];            /* Temporary buffer for stdin */
char file[8192] = "xx";         /* File name buffer */
char *targ;                     /* Arg ptr for error messages */
char *sptr;
FILE *infile, *outfile;         /* I/O file streams */
int silent, keep, create;       /* Flags: -s(ilent), -k(eep), (create) */
int errflg;
int fiwidth = 2;                /* file index width (output file names) */
extern int optind;
extern char *optarg;
offset_t offset;                /* Regular expression offset value */
offset_t curline;               /* Current line in input file */

/*
 * These defines are needed for regexp handling(see regexp(7))
 */
#define PERROR(x)       fatal("%s: Illegal Regular Expression\n", targ);

static int asc_to_ll(char *, long long *);
static void closefile(void);
static void fatal(char *, char *);
static offset_t findline(char *, offset_t);
static void flush(void);
static FILE *getfile(void);
static char *getaline(int);
static void line_arg(char *);
static void num_arg(char *, int);
static void re_arg(char *);
static void sig(int);
static void to_line(offset_t);
static void usage(void);

int
main(int argc, char **argv)
{
        int ch, mode;
        char *ptr;

        (void) setlocale(LC_ALL, "");
#if !defined(TEXT_DOMAIN)               /* Should be defined by cc -D */
#define TEXT_DOMAIN     "SYS_TEST"      /* Use this only if it weren't */
#endif
        (void) textdomain(TEXT_DOMAIN);

        while ((ch = getopt(argc, argv, "skf:n:")) != EOF) {
                switch (ch) {
                        case 'f':
                                (void) strcpy(file, optarg);
                                if ((ptr = strrchr(optarg, '/')) == NULL)
                                        ptr = optarg;
                                else
                                        ptr++;

                                break;
                        case 'n':               /* POSIX.2 */
                                for (ptr = optarg; *ptr != NULL; ptr++)
                                        if (!isdigit((int)*ptr))
                                                fatal("-n num\n", NULL);
                                fiwidth = atoi(optarg);
                                break;
                        case 'k':
                                keep++;
                                break;
                        case 's':
                                silent++;
                                break;
```

```
128                         case '?':
129                                 errflg++;
130                 }
131         }

133         argv = &argv[optind];
134         argc -= optind;
135         if (argc <= 1 || errflg)
136                 usage();

138         if (strcmp(*argv, "-") == 0) {
139                 infile = tmpfile();

141                 while (fread(tmpbuf, 1, BUFSIZ, stdin) != 0) {
142                         if (fwrite(tmpbuf, 1, BUFSIZ, infile) == 0)
143                                 if (errno == ENOSPC) {
144                                         (void) fprintf(stderr, "csplit: ");
145                                         (void) fprintf(stderr, gettext(
146                                             "No space left on device\n"));
147                                         exit(1);
148                                 } else {
149                                         (void) fprintf(stderr, "csplit: ");
150                                         (void) fprintf(stderr, gettext(
151                                             "Bad write to temporary "
152                                             "file\n"));
153                                         exit(1);
154                                 }

156         /* clear the buffer to get correct size when writing buffer */

158                         (void) memset(tmpbuf, '\0', sizeof (tmpbuf));
159                 }
160                 rewind(infile);
161         } else if ((infile = fopen(*argv, "r")) == NULL)
162                 fatal("Cannot open %s\n", *argv);
163         ++argv;
164         curline = (offset_t)1;
165         (void) signal(SIGINT, sig);

167         /*
168          * The following for loop handles the different argument types.
169          * A switch is performed on the first character of the argument
170          * and each case calls the appropriate argument handling routine.
171          */

173         for (; *argv; ++argv) {
174                 targ = *argv;
175                 switch (**argv) {
176                 case '/':
177                         mode = EXPMODE;
178                         create = TRUE;
179                         re_arg(*argv);
180                         break;
181                 case '%':
182                         mode = EXPMODE;
183                         create = FALSE;
184                         re_arg(*argv);
185                         break;
186                 case '{':
187                         num_arg(*argv, mode);
188                         mode = FALSE;
189                         break;
190                 default:
191                         mode = LINMODE;
192                         create = TRUE;
193                         line_arg(*argv);
```



```
194                         break;
195                 }
196         }
197         create = TRUE;
198         to_line(LAST);
199         return (0);
200 }
_____unchanged_portion_omitted_

343 /*
344  * Getfile does nothing if the create flag is not set.  If the create
345  * flag is set, getfile positions the file pointer(fptr) at the end of
346  * the file name prefix on the first call(fptr=0).  The file counter is
347  * stored in the file name and incremented.  If the subsequent fopen
348  * fails, the file name is copied to tfile for the error message, the
349  * previous file name is restored for cleanup, and fatal is called.  If
350  * the fopen succeeds, the stream(opfil) is returned.
351  */

353 FILE *
354 getfile()
355 {
356         static char *fptr;
357         static int ctr;
358         FILE *opfil;
359         char tfile[15];
360         char *delim;
361         char savedelim;

363         if (create) {
364                 if (fptr == 0)
365                         for (fptr = file; *fptr != NULL; fptr++)
366                                 continue;
367                 (void) sprintf(fptr, "%.*d", fiwidth, ctr++);

369                 /* check for suffix length overflow */
370                 if (strlen(fptr) > fiwidth) {
371                         fatal("Suffix longer than %ld chars; increase -n\n",
372                             (char *)fiwidth);
373                 }

375                 /* check for filename length overflow */

377                 delim = strrchr(file, '/');
378                 if (delim == (char *)NULL) {
379                         if (strlen(file) > pathconf(".", _PC_NAME_MAX)) {
380                                 fatal("Name too long: %s\n", file);
381                         }
382                 } else {
383                         /* truncate file at pathname delim to do pathconf */
384                         savedelim = *delim;
385                         *delim = '\0';
386                         /*
387                          * file: ppppppppp\0fffff\0
388                          * ..... ^ file
389                          * ............. ^ delim
390                          */
391                         if (strlen(delim + 1) > pathconf(file, _PC_NAME_MAX)) {
392                                 fatal("Name too long: %s\n", delim + 1);
393                         }
394                         *delim = savedelim;
395                 }

397                 if ((opfil = fopen(file, "w")) == NULL) {
398                         (void) strlcpy(tfile, file, sizeof (tfile));
394                         (void) strcpy(tfile, file);
```

```
 399                        (void) sprintf(fptr, "%.*d", fiwidth, (ctr-2));
 400                        fatal("Cannot create %s\n", tfile);
 401                }
 402                return (opfil);
 403        }
 404        return (NULL);
 405 }
```

_____***unchanged_portion_omitted_***