Sample Application – Multithreaded grep

A

Description of tgrep

The tgrep sample program is a multithreaded version of find(1) combined with grep(1). tgrep supports all but the -w (word search) options of the normal grep, and a few exclusively available options.

By default, the tgrep searches are like the following command:

```
find . -exec grep [ options ] pattern {} \;
```

For large directory hierarchies, tgrep gets results more quickly than the find command, depending on the number of processors available. On uniprocessor machines it is about twice as fast, and on four processor machines it is about four times as fast.

The -e option changes the way tgrep interprets the pattern string. Ordinarily (without the -e option) tgrep uses a literal string match. With the -e option, tgrep uses an MT-Safe public domain version of a regular expression handler. The regular expression method is slower.

The -B option tells tgrep to use the value of the environment variable called TGLIMIT to limit the number of threads it will use during a search. This option has no affect if TGLIMIT is not set. Because tgrep can use a lot of system resources, this is a way to run it politely on a timesharing system.

A

Getting Online Source Code

Source for `tgrep` is included on the Catalyst Developer's CD. Contact your sales representative to find out how you can get a copy.

A copy might also be available on the World Wide Web (WWW) at the following URL:

http://www.sun.com/sunsoft/Products/Developer-products/sig/threads/

Only the multithreaded `main.c` module appears here. Other modules, including those for regular expression handling, plus documentation and Makefiles, might be available from the sources listed above.

Code Example A-1 Source Code for `tgrep` Program

```
/* Copyright (c) 1993, 1994  Ron Winacott                               */
/* This program may be used, copied, modified, and redistributed freely */
/* for ANY purpose, so long as this notice remains intact.              */

#define _REENTRANT

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <assert.h>
#include <errno.h>
#include <ctype.h>
#include <sys/types.h>
#include <time.h>
#include <sys/stat.h>
#include <dirent.h>

#include "version.h"

#include <fcntl.h>
#include <sys/uio.h>
#include <pthread.h>
#include <sched.h>

#ifdef MARK
#include <prof.h> /* to turn on MARK(), use -DMARK to compile (see man prof5)*/
#endif

#include "pmatch.h"
```

Code Example A-1 Source Code for `tgrep` Program

```
#define PATH_MAX                1024 /* max # of characters in a path name */
#define HOLD_FDS                6  /* stdin,out,err and a buffer */
#define UNLIMITED               99999 /* The default tglimit */
#define MAXREGEXP               10  /* max number of -e options */

#define FB_BLOCK                0x00001
#define FC_COUNT                0x00002
#define FH_HOLDNAME             0x00004
#define FI_IGNCASE              0x00008
#define FL_NAMEONLY             0x00010
#define FN_NUMBER               0x00020
#define FS_NOERROR              0x00040
#define FV_REVERSE              0x00080
#define FW_WORD                 0x00100
#define FR_RECUR                0x00200
#define FU_UNSORT               0x00400
#define FX_STDIN                0x00800
#define TG_BATCH                0x01000
#define TG_FILEPAT              0x02000
#define FE_REGEXP               0x04000
#define FS_STATS                0x08000
#define FC_LINE                 0x10000
#define TG_PROGRESS             0x20000

#define FILET                   1
#define DIRT                    2

typedef struct work_st {
    char                *path;
    int                 tp;
    struct work_st      *next;
} work_t;

typedef struct out_st {
    char                *line;
    int                 line_count;
    long                byte_count;
    struct out_st       *next;
} out_t;

#define ALPHASIZ        128
typedef struct bm_pattern {     /* Boyer - Moore pattern                */
        short            p_m;           /* length of pattern string     */
```

Code Example A-1 Source Code for `tgrep` Program

```
        short            p_r[ALPHASIZ]; /* "r" vector                    */
        short           *p_R;           /* "R" vector                    */
        char            *p_pat;         /* pattern string                */
} BM_PATTERN;

/* bmpmatch.c */
extern BM_PATTERN *bm_makepat(char *p);
extern char *bm_pmatch(BM_PATTERN *pat, register char *s);
extern void bm_freepat(BM_PATTERN *pattern);
BM_PATTERN      *bm_pat;  /* the global target read only after main */

/* pmatch.c */
extern char *pmatch(register PATTERN *pattern, register char *string, int *len);
extern PATTERN *makepat(char *string, char *metas);
extern void freepat(register PATTERN *pat);
extern void printpat(PATTERN *pat);
PATTERN         *pm_pat[MAXREGEXP];  /* global targets read only for pmatch */

#include "proto.h"  /* function prototypes of main.c */

/* local functions to POSIX only */
void pthread_setconcurrency_np(int con);
int pthread_getconcurrency_np(void);
void pthread_yield_np(void);

pthread_attr_t  detached_attr;
pthread_mutex_t output_print_lk;
pthread_mutex_t global_count_lk;

int             global_count = 0;

work_t          *work_q = NULL;
pthread_cond_t  work_q_cv;
pthread_mutex_t work_q_lk;
pthread_mutex_t debug_lock;

#include "debug.h"  /* must be included AFTER the
                        mutex_t debug_lock line */

work_t          *search_q = NULL;
pthread_mutex_t search_q_lk;
pthread_cond_t  search_q_cv;
int             search_pool_cnt = 0;    /* the count in the pool now */
int             search_thr_limit = 0;   /* the max in the pool */
```

Code Example A-1 Source Code for `tgrep` Program

```
work_t           *cascade_q = NULL;
pthread_mutex_t cascade_q_lk;
pthread_cond_t  cascade_q_cv;
int             cascade_pool_cnt = 0;
int             cascade_thr_limit = 0;

int             running = 0;
pthread_mutex_t running_lk;

pthread_mutex_t stat_lk;
time_t          st_start = 0;
int             st_dir_search = 0;
int             st_file_search = 0;
int             st_line_search = 0;
int             st_cascade = 0;
int             st_cascade_pool = 0;
int             st_cascade_destroy = 0;
int             st_search = 0;
int             st_pool = 0;
int             st_maxrun = 0;
int             st_worknull = 0;
int             st_workfds = 0;
int             st_worklimit = 0;
int             st_destroy = 0;

int             all_done = 0;
int             work_cnt = 0;
int             current_open_files = 0;
int             tglimit = UNLIMITED;    /* if -B limit the number of
                                   threads */
int             progress_offset = 1;
int             progress = 0;  /* protected by the print_lock ! */
unsigned int    flags = 0;
int             regexp_cnt = 0;
char            *string[MAXREGEXP];
int             debug = 0;
int             use_pmatch = 0;
char            file_pat[255];  /* file patten match */
PATTERN         *pm_file_pat; /* compiled file target string (pmatch()) */

/*
 * Main: This is where the fun starts
 */
```

Code Example A-1 Source Code for `tgrep` Program

```
int
main(int argc, char **argv)
{
    int         c,out_thr_flags;
    long        max_open_files = 0l, ncpus = 0l;
    extern int  optind;
    extern char *optarg;
    int         prio = 0;
    struct stat sbuf;
    pthread_t tid,dtid;
    void        *status;
    char        *e = NULL, *d = NULL; /* for debug flags */
    int         debug_file = 0;
    struct sigaction sigact;
    sigset_t    set,oset;
    int         err = 0, i = 0, pm_file_len = 0;
    work_t      *work;
    int         restart_cnt = 10;

    /* NO OTHER THREADS ARE RUNNING */
    flags = FR_RECUR;  /* the default */

    while ((c = getopt(argc, argv, "d:e:bchilnsvwruf:p:BCSZzHP:")) != EOF) {
        switch (c) {
#ifdef DEBUG
        case 'd':
            debug = atoi(optarg);
            if (debug == 0)
                debug_usage();

            d = optarg;
            fprintf(stderr,"tgrep: Debug on at level(s) ");
            while (*d) {
                for (i=0; i<9; i++)
                    if (debug_set[i].level == *d) {
                        debug_levels |= debug_set[i].flag;
                        fprintf(stderr,"%c ",debug_set[i].level);
                        break;
                    }
                d++;
            }
            fprintf(stderr,"\n");
            break;
        case 'f': debug_file = atoi(optarg); break;
```

Code Example A-1 Source Code for `tgrep` Program

```
#endif      /* DEBUG */

        case 'B':
            flags |= TG_BATCH;
#ifndef __lock_lint
        /* locklint complains here, but there are no other threads */
            if ((e = getenv("TGLIMIT"))) {
                tglimit = atoi(e);
            }
            else {
                if (!(flags & FS_NOERROR))  /* order dependent! */
                    fprintf(stderr,"env TGLIMIT not set, overriding -B\n");
                flags &= ~TG_BATCH;
            }
#endif
            break;
        case 'p':
            flags |= TG_FILEPAT;
            strcpy(file_pat,optarg);
            pm_file_pat = makepat(file_pat,NULL);
            break;
        case 'P':
            flags |= TG_PROGRESS;
            progress_offset = atoi(optarg);
            break;
        case 'S': flags |= FS_STATS;    break;
        case 'b': flags |= FB_BLOCK;    break;
        case 'c': flags |= FC_COUNT;    break;
        case 'h': flags |= FH_HOLDNAME; break;
        case 'i': flags |= FI_IGNCASE;  break;
        case 'l': flags |= FL_NAMEONLY; break;
        case 'n': flags |= FN_NUMBER;   break;
        case 's': flags |= FS_NOERROR;  break;
        case 'v': flags |= FV_REVERSE;  break;
        case 'w': flags |= FW_WORD;     break;
        case 'r': flags &= ~FR_RECUR;   break;
        case 'C': flags |= FC_LINE;     break;
        case 'e':
            if (regexp_cnt == MAXREGEXP) {
                fprintf(stderr,"Max number of regexp's (%d) exceeded!\n",
                        MAXREGEXP);
                exit(1);
            }
            flags |= FE_REGEXP;
```

Code Example A-1 Source Code for `tgrep` Program

```
            if ((string[regexp_cnt] =(char *)malloc(strlen(optarg)+1))==NULL){
                fprintf(stderr,"tgrep: No space for search string(s)\n");
                exit(1);
            }
            memset(string[regexp_cnt],0,strlen(optarg)+1);
            strcpy(string[regexp_cnt],optarg);
            regexp_cnt++;
            break;
        case 'z':
        case 'Z': regexp_usage();
            break;
        case 'H':
        case '?':
        default : usage();
        }
    }
    if (flags & FS_STATS)
        st_start = time(NULL);

    if (!(flags & FE_REGEXP)) {
       if (argc - optind < 1) {
            fprintf(stderr,"tgrep: Must supply a search string(s) "
                    "and file list or directory\n");
            usage();
        }
        if ((string[0]=(char *)malloc(strlen(argv[optind])+1))==NULL){
            fprintf(stderr,"tgrep: No space for search string(s)\n");
            exit(1);
        }
        memset(string[0],0,strlen(argv[optind])+1);
        strcpy(string[0],argv[optind]);
        regexp_cnt=1;
        optind++;
    }

    if (flags & FI_IGNCASE)
        for (i=0; i<regexp_cnt; i++)
            uncase(string[i]);

    if (flags & FE_REGEXP) {
        for (i=0; i<regexp_cnt; i++)
            pm_pat[i] = makepat(string[i],NULL);
        use_pmatch = 1;
    }
```

Code Example A-1 Source Code for `tgrep` Program

```
    else {
        bm_pat = bm_makepat(string[0]); /* only one allowed */
    }

    flags |= FX_STDIN;


    max_open_files = sysconf(_SC_OPEN_MAX);
    ncpus = sysconf(_SC_NPROCESSORS_ONLN);
    if ((max_open_files - HOLD_FDS - debug_file) < 1) {
        fprintf(stderr,"tgrep: You MUST have at least ONE fd "
                "that can be used, check limit (>10)\n");
        exit(1);
    }
    search_thr_limit = max_open_files - HOLD_FDS - debug_file;
    cascade_thr_limit = search_thr_limit / 2;
    /* the number of files that can be open */
    current_open_files = search_thr_limit;

    pthread_attr_init(&detached_attr);
    pthread_attr_setdetachstate(&detached_attr,
        PTHREAD_CREATE_DETACHED);

    pthread_mutex_init(&global_count_lk,NULL);
    pthread_mutex_init(&output_print_lk,NULL);
    pthread_mutex_init(&work_q_lk,NULL);
    pthread_mutex_init(&running_lk,NULL);
    pthread_cond_init(&work_q_cv,NULL);
    pthread_mutex_init(&search_q_lk,NULL);
    pthread_cond_init(&search_q_cv,NULL);
    pthread_mutex_init(&cascade_q_lk,NULL);
    pthread_cond_init(&cascade_q_cv,NULL);

    if ((argc == optind) && ((flags & TG_FILEPAT) || (flags & FR_RECUR))) {
        add_work(".",DIRT);
        flags = (flags & ~FX_STDIN);
    }
    for ( ; optind < argc; optind++) {
        restart_cnt = 10;
        flags = (flags & ~FX_STDIN);
      STAT_AGAIN:
        if (stat(argv[optind], &sbuf)) {
            if (errno == EINTR) { /* try again !, restart */
                if (--restart_cnt)
```

Code Example A-1 Source Code for `tgrep` Program

```
                    goto STAT_AGAIN;
            }
            if (!(flags & FS_NOERROR))
                fprintf(stderr,"tgrep: Can't stat file/dir %s, %s\n",
                        argv[optind], strerror(errno));
            continue;
        }
        switch (sbuf.st_mode & S_IFMT) {
        case S_IFREG :
            if (flags & TG_FILEPAT) {
                if (pmatch(pm_file_pat, argv[optind], &pm_file_len))
                    DP(DLEVEL1,("File pat match %s\n",argv[optind]));
                    add_work(argv[optind],FILET);
            }
            else {
                add_work(argv[optind],FILET);
            }
            break;
        case S_IFDIR :
            if (flags & FR_RECUR) {
                add_work(argv[optind],DIRT);
            }
            else {
                if (!(flags & FS_NOERROR))
                    fprintf(stderr,"tgrep: Can't search directory %s, "
                            "-r option is on. Directory ignored.\n",
                            argv[optind]);
            }
            break;
        }
    }

    pthread_setconcurrency_np(3);

    if (flags & FX_STDIN) {
        fprintf(stderr,"tgrep: stdin option is not coded at this time\n");
        exit(0);                        /* XXX Need to fix this SOON */
        search_thr(NULL);
        if (flags & FC_COUNT) {
            pthread_mutex_lock(&global_count_lk);
            printf("%d\n",global_count);
            pthread_mutex_unlock(&global_count_lk);
        }
        if (flags & FS_STATS)
```

Code Example A-1 Source Code for `tgrep` Program

```
            prnt_stats();
        exit(0);
    }

    pthread_mutex_lock(&work_q_lk);
    if (!work_q) {
        if (!(flags & FS_NOERROR))
            fprintf(stderr,"tgrep: No files to search.\n");
        exit(0);
    }
    pthread_mutex_unlock(&work_q_lk);

    DP(DLEVEL1,("Starting to loop through the work_q for work\n"));

    /* OTHER THREADS ARE RUNNING */
    while (1) {
        pthread_mutex_lock(&work_q_lk);
        while ((work_q == NULL || current_open_files == 0 || tglimit <= 0) &&
               all_done == 0) {
            if (flags & FS_STATS) {
                pthread_mutex_lock(&stat_lk);
                if (work_q == NULL)
                    st_worknull++;
                if (current_open_files == 0)
                    st_workfds++;
                if (tglimit <= 0)
                    st_worklimit++;
                pthread_mutex_unlock(&stat_lk);
            }
            pthread_cond_wait(&work_q_cv,&work_q_lk);
        }
        if (all_done != 0) {
            pthread_mutex_unlock(&work_q_lk);
            DP(DLEVEL1,("All_done was set to TRUE\n"));
            goto OUT;
        }
        work = work_q;
        work_q = work->next;  /* maybe NULL */
        work->next = NULL;
        current_open_files--;
        pthread_mutex_unlock(&work_q_lk);

        tid = 0;
        switch (work->tp) {
```

Code Example A-1 Source Code for `tgrep` Program

```
        case DIRT:
            pthread_mutex_lock(&cascade_q_lk);
            if (cascade_pool_cnt) {
                if (flags & FS_STATS) {
                    pthread_mutex_lock(&stat_lk);
                    st_cascade_pool++;
                    pthread_mutex_unlock(&stat_lk);
                }
                work->next = cascade_q;
                cascade_q = work;
                pthread_cond_signal(&cascade_q_cv);
                pthread_mutex_unlock(&cascade_q_lk);
                DP(DLEVEL2,("Sent work to cascade pool thread\n"));
            }
            else {
                pthread_mutex_unlock(&cascade_q_lk);
                err = pthread_create(&tid,&detached_attr,cascade,(void *)work);
                DP(DLEVEL2,("Sent work to new cascade thread\n"));
                if (flags & FS_STATS) {
                    pthread_mutex_lock(&stat_lk);
                    st_cascade++;
                    pthread_mutex_unlock(&stat_lk);
                }
            }
            break;
        case FILET:
            pthread_mutex_lock(&search_q_lk);
            if (search_pool_cnt) {
                if (flags & FS_STATS) {
                    pthread_mutex_lock(&stat_lk);
                    st_pool++;
                    pthread_mutex_unlock(&stat_lk);
                }
                work->next = search_q;  /* could be null */
                search_q = work;
                pthread_cond_signal(&search_q_cv);
                pthread_mutex_unlock(&search_q_lk);
                DP(DLEVEL2,("Sent work to search pool thread\n"));
            }
            else {
                pthread_mutex_unlock(&search_q_lk);
                err = pthread_create(&tid,&detached_attr,
                                     search_thr,(void *)work);
                pthread_setconcurrency_np(pthread_getconcurrency_np()+1);
```

Code Example A-1 Source Code for `tgrep` Program

```
                DP(DLEVEL2,("Sent work to new search thread\n"));
                if (flags & FS_STATS) {
                    pthread_mutex_lock(&stat_lk);
                    st_search++;
                    pthread_mutex_unlock(&stat_lk);
                }
            }
            break;
        default:
            fprintf(stderr,"tgrep: Internal error, work_t->tp not valid\n");
            exit(1);
        }
        if (err) {  /* NEED TO FIX THIS CODE. Exiting is just wrong */
            fprintf(stderr,"Could not create new thread!\n");
            exit(1);
        }
    }

 OUT:
    if (flags & TG_PROGRESS) {
        if (progress)
            fprintf(stderr,".\n");
        else
            fprintf(stderr,"\n");
    }
    /* we are done, print the stuff. All other threads are parked */
    if (flags & FC_COUNT) {
        pthread_mutex_lock(&global_count_lk);
        printf("%d\n",global_count);
        pthread_mutex_unlock(&global_count_lk);
    }
    if (flags & FS_STATS)
        prnt_stats();
    return(0); /* should have a return from main */
}

/*
 * Add_Work: Called from the main thread, and cascade threads to add file
 * and directory names to the work Q.
 */
int
add_work(char *path,int tp)
{
    work_t      *wt,*ww,*wp;
```

Code Example A-1 Source Code for `tgrep` Program

```
    if ((wt = (work_t *)malloc(sizeof(work_t))) == NULL)
        goto ERROR;
    if ((wt->path = (char *)malloc(strlen(path)+1)) == NULL)
        goto ERROR;

    strcpy(wt->path,path);
    wt->tp = tp;
    wt->next = NULL;
    if (flags & FS_STATS) {
        pthread_mutex_lock(&stat_lk);
        if (wt->tp == DIRT)
            st_dir_search++;
        else
            st_file_search++;
        pthread_mutex_unlock(&stat_lk);
    }
    pthread_mutex_lock(&work_q_lk);
    work_cnt++;
    wt->next = work_q;
    work_q = wt;
    pthread_cond_signal(&work_q_cv);
    pthread_mutex_unlock(&work_q_lk);
    return(0);
 ERROR:
    if (!(flags & FS_NOERROR))
        fprintf(stderr,"tgrep: Could not add %s to work queue. Ignored\n",
                path);
    return(-1);
}

/*
 * Search thread: Started by the main thread when a file name is found
 * on the work Q to be serached. If all the needed resources are ready
 * a new search thread will be created.
 */
void *
search_thr(void *arg) /* work_t *arg */
{
    FILE        *fin;
    char        fin_buf[(BUFSIZ*4)];  /* 4 Kbytes */
    work_t      *wt,std;
    int         line_count;
    char        rline[128];
```

Code Example A-1 Source Code for `tgrep` Program

```
    char        cline[128];
    char        *line;
    register char *p,*pp;
    int            pm_len;
    int         len = 0;
    long        byte_count;
    long        next_line;
    int         show_line;  /* for the -v option */
    register int slen,plen,i;
    out_t       *out = NULL;    /* this threads output list */

    pthread_yield_np();
    wt = (work_t *)arg; /* first pass, wt is passed to use. */

    /* len = strlen(string);*/  /* only set on first pass */

    while (1) {  /* reuse the search threads */
        /* init all back to zero */
        line_count = 0;
        byte_count = 0l;
        next_line = 0l;
        show_line = 0;

        pthread_mutex_lock(&running_lk);
        running++;
        pthread_mutex_unlock(&running_lk);
        pthread_mutex_lock(&work_q_lk);
        tglimit--;
        pthread_mutex_unlock(&work_q_lk);
        DP(DLEVEL5,("searching file (STDIO) %s\n",wt->path));

        if ((fin = fopen(wt->path,"r")) == NULL) {
            if (!(flags & FS_NOERROR)) {
                fprintf(stderr,"tgrep: %s. File \"%s\" not searched.\n",
                        strerror(errno),wt->path);
            }
            goto ERROR;
        }
        setvbuf(fin,fin_buf,_IOFBF,(BUFSIZ*4));  /* XXX */
        DP(DLEVEL5,("Search thread has opened file %s\n",wt->path));
        while ((fgets(rline,127,fin)) != NULL) {
            if (flags & FS_STATS) {
                pthread_mutex_lock(&stat_lk);
                st_line_search++;
```

Code Example A-1 Source Code for `tgrep` Program

```
            pthread_mutex_unlock(&stat_lk);
        }
        slen = strlen(rline);
        next_line += slen;
        line_count++;
        if (rline[slen-1] == '\n')
            rline[slen-1] = '\0';
        /*
        ** If the uncase flag is set, copy the read in line (rline)
        ** To the uncase line (cline) Set the line pointer to point at
        ** cline.
        ** If the case flag is NOT set, then point line at rline.
        ** line is what is compared, rline is what is printed on a
        ** match.
        */
        if (flags & FI_IGNCASE) {
            strcpy(cline,rline);
            uncase(cline);
            line = cline;
        }
        else {
            line = rline;
        }
        show_line = 1;  /* assume no match, if -v set */
        /* The old code removed */
        if (use_pmatch) {
            for (i=0; i<regexp_cnt; i++) {
                if (pmatch(pm_pat[i], line, &pm_len)) {
                    if (!(flags & FV_REVERSE)) {
                        add_output_local(&out,wt,line_count,
                                         byte_count,rline);
                        continue_line(rline,fin,out,wt,
                                      &line_count,&byte_count);
                    }
                    else {
                        show_line = 0;
                    } /* end of if -v flag if / else block */
                    /*
                    ** if we get here on ANY of the regexp targets
                    ** jump out of the loop, we found a single
                    ** match so do not keep looking!
                    ** If name only, do not keep searcthing the same
                    ** file, we found a single match, so close the file,
                    ** print the file name and move on to the next file.
```

Code Example A-1 Source Code for `tgrep` Program

```
                         */
                         if (flags & FL_NAMEONLY)
                             goto OUT_OF_LOOP;
                         else
                             goto OUT_AND_DONE;
                     } /* end found a match if block */
                 } /* end of the for pat[s] loop */
             }
             else {
                 if (bm_pmatch( bm_pat, line)) {
                     if (!(flags & FV_REVERSE)) {
                         add_output_local(&out,wt,line_count,byte_count,rline);
                         continue_line(rline,fin,out,wt,
                                       &line_count,&byte_count);
                     }
                     else {
                         show_line = 0;
                     }
                     if (flags & FL_NAMEONLY)
                         goto OUT_OF_LOOP;
                 }
             }
           OUT_AND_DONE:
             if ((flags & FV_REVERSE) && show_line) {
                 add_output_local(&out,wt,line_count,byte_count,rline);
                 show_line = 0;
             }
             byte_count = next_line;
         }
       OUT_OF_LOOP:
         fclose(fin);
         /*
         ** The search part is done, but before we give back the FD,
         ** and park this thread in the search thread pool, print the
         ** local output we have gathered.
         */
         print_local_output(out,wt);  /* this also frees out nodes */
         out = NULL; /* for the next time around, if there is one */
     ERROR:
         DP(DLEVEL5,("Search done for %s\n",wt->path));
         free(wt->path);
         free(wt);

         notrun();
```

Code Example A-1 Source Code for `tgrep` Program

```
        pthread_mutex_lock(&search_q_lk);
        if (search_pool_cnt > search_thr_limit) {
            pthread_mutex_unlock(&search_q_lk);
            DP(DLEVEL5,("Search thread exiting\n"));
            if (flags & FS_STATS) {
                pthread_mutex_lock(&stat_lk);
                st_destroy++;
                pthread_mutex_unlock(&stat_lk);
            }
            return(0);
        }
        else {
            search_pool_cnt++;
            while (!search_q)
                pthread_cond_wait(&search_q_cv,&search_q_lk);
            search_pool_cnt--;
            wt = search_q;  /* we have work to do! */
            if (search_q->next)
                search_q = search_q->next;
            else
                search_q = NULL;
            pthread_mutex_unlock(&search_q_lk);
        }
    }
    /*NOTREACHED*/
}

/*
 * Continue line: Special case search with the -C flag set. If you are
 * searching files like Makefiles, some lines might have escape char's to
 * contine the line on the next line. So the target string can be found, but
 * no data is displayed. This function continues to print the escaped line
 * until there are no more "\" chars found.
 */
int
continue_line(char *rline, FILE *fin, out_t *out, work_t *wt,
              int *lc, long *bc)
{
    int len;
    int cnt = 0;
    char *line;
    char nline[128];

    if (!(flags & FC_LINE))
```

Code Example A-1 Source Code for `tgrep` Program

```
        return(0);

    line = rline;
  AGAIN:
    len = strlen(line);
    if (line[len-1] == '\\') {
        if ((fgets(nline,127,fin)) == NULL) {
            return(cnt);
        }
        line = nline;
        len = strlen(line);
        if (line[len-1] == '\n')
            line[len-1] = '\0';
        *bc = *bc + len;
        *lc++;
        add_output_local(&out,wt,*lc,*bc,line);
        cnt++;
        goto AGAIN;
    }
    return(cnt);
}

/*
 * cascade: This thread is started by the main thread when directory names
 * are found on the work Q. The thread reads all the new file, and directory
 * names from the directory it was started when and adds the names to the
 * work Q. (it finds more work!)
 */

void *
cascade(void *arg)  /* work_t *arg */
{
    char        fullpath[1025];
    int         restart_cnt = 10;
    DIR         *dp;

    char        dir_buf[sizeof(struct dirent) + PATH_MAX];
    struct dirent *dent = (struct dirent *)dir_buf;
    struct stat   sbuf;
    char        *fpath;
    work_t      *wt;
    int         fl = 0, dl = 0;
    int         pm_file_len = 0;
```

Code Example A-1 Source Code for `tgrep` Program

```
    pthread_yield_np();  /* try toi give control back to main thread */
    wt = (work_t *)arg;

    while(1) {
        fl = 0;
        dl = 0;
        restart_cnt = 10;
        pm_file_len = 0;

        pthread_mutex_lock(&running_lk);
        running++;
        pthread_mutex_unlock(&running_lk);
        pthread_mutex_lock(&work_q_lk);
        tglimit--;
        pthread_mutex_unlock(&work_q_lk);

        if (!wt) {
            if (!(flags & FS_NOERROR))
                fprintf(stderr,"tgrep: Bad work node passed to cascade\n");
            goto DONE;
        }
        fpath = (char *)wt->path;
        if (!fpath) {
            if (!(flags & FS_NOERROR))
                fprintf(stderr,"tgrep: Bad path name passed to cascade\n");
            goto DONE;
        }
        DP(DLEVEL3,("Cascading on %s\n",fpath));
        if (( dp = opendir(fpath)) == NULL) {
            if (!(flags & FS_NOERROR))
                fprintf(stderr,"tgrep: Can't open dir %s, %s. Ignored.\n",
                        fpath,strerror(errno));
            goto DONE;
        }
        while ((readdir_r(dp,dent)) != NULL) {
            restart_cnt = 10;  /* only try to restart the interupted 10 X */

            if (dent->d_name[0] == '.') {
                if (dent->d_name[1] == '.' && dent->d_name[2] == '\0')
                    continue;
                if (dent->d_name[1] == '\0')
                    continue;
            }
```

Code Example A-1 Source Code for `tgrep` Program

```
            fl = strlen(fpath);
            dl = strlen(dent->d_name);
            if ((fl + 1 + dl) > 1024) {
                fprintf(stderr,"tgrep: Path %s/%s is too long. "
                        "MaxPath = 1024\n",
                        fpath, dent->d_name);
                continue;  /* try the next name in this directory */
            }
            strcpy(fullpath,fpath);
            strcat(fullpath,"/");
            strcat(fullpath,dent->d_name);

          RESTART_STAT:
            if (stat(fullpath,&sbuf)) {
                if (errno == EINTR) {
                    if (--restart_cnt)
                        goto RESTART_STAT;
                }
                if (!(flags & FS_NOERROR))
                    fprintf(stderr,"tgrep: Can't stat file/dir %s, %s. "
                            "Ignored.\n",
                            fullpath,strerror(errno));
                goto ERROR;
            }

            switch (sbuf.st_mode & S_IFMT) {
            case S_IFREG :
                if (flags & TG_FILEPAT) {
                    if (pmatch(pm_file_pat, dent->d_name, &pm_file_len)) {
                        DP(DLEVEL3,("file pat match (cascade) %s\n",
                                    dent->d_name));
                        add_work(fullpath,FILET);
                    }
                }
                else {
                    add_work(fullpath,FILET);
                    DP(DLEVEL3,("cascade added file (MATCH) %s to Work Q\n",
                                fullpath));
                }
                break;

            case S_IFDIR :
                DP(DLEVEL3,("cascade added dir %s to Work Q\n",fullpath));
                add_work(fullpath,DIRT);
```

Code Example A-1 Source Code for `tgrep` Program

```
                break;
            }
        }

      ERROR:
        closedir(dp);

      DONE:
        free(wt->path);
        free(wt);
        notrun();
        pthread_mutex_lock(&cascade_q_lk);
        if (cascade_pool_cnt > cascade_thr_limit) {
            pthread_mutex_unlock(&cascade_q_lk);
            DP(DLEVEL5,("Cascade thread exiting\n"));
            if (flags & FS_STATS) {
                pthread_mutex_lock(&stat_lk);
                st_cascade_destroy++;
                pthread_mutex_unlock(&stat_lk);
            }
            return(0); /* pthread_exit */
        }
        else {
            DP(DLEVEL5,("Cascade thread waiting in pool\n"));
            cascade_pool_cnt++;
            while (!cascade_q)
                pthread_cond_wait(&cascade_q_cv,&cascade_q_lk);
            cascade_pool_cnt--;
            wt = cascade_q;  /* we have work to do! */
            if (cascade_q->next)
                cascade_q = cascade_q->next;
            else
                cascade_q = NULL;
            pthread_mutex_unlock(&cascade_q_lk);
        }
    }
    /*NOTREACHED*/
}

/*
 * Print Local Output: Called by the search thread after it is done searching
 * a single file. If any oputput was saved (matching lines), the lines are
 * displayed as a group on stdout.
 */
```

Code Example A-1 Source Code for `tgrep` Program

```
int
print_local_output(out_t *out, work_t *wt)
{
    out_t       *pp, *op;
    int         out_count = 0;
    int         printed = 0;

    pp = out;
    pthread_mutex_lock(&output_print_lk);
    if (pp && (flags & TG_PROGRESS)) {
        progress++;
        if (progress >= progress_offset) {
            progress = 0;
            fprintf(stderr,".");
        }
    }
    while (pp) {
        out_count++;
        if (!(flags & FC_COUNT)) {
            if (flags & FL_NAMEONLY) {  /* Pint name ONLY ! */
                if (!printed) {
                    printed = 1;
                    printf("%s\n",wt->path);
                }
            }
            else {  /* We are printing more then just the name */
                if (!(flags & FH_HOLDNAME))
                    printf("%s :",wt->path);
                if (flags & FB_BLOCK)
                    printf("%ld:",pp->byte_count/512+1);
                if (flags & FN_NUMBER)
                    printf("%d:",pp->line_count);
                printf("%s\n",pp->line);
            }
        }
        op = pp;
        pp = pp->next;
        /* free the nodes as we go down the list */
        free(op->line);
        free(op);
    }

    pthread_mutex_unlock(&output_print_lk);
    pthread_mutex_lock(&global_count_lk);
```

Code Example A-1 Source Code for `tgrep` Program

```
    global_count += out_count;
    pthread_mutex_unlock(&global_count_lk);
    return(0);
}

/*
 * add output local: is called by a search thread as it finds matching lines.
 * the matching line, its byte offset, line count, etc. are stored until the
 * search thread is done searching the file, then the lines are printed as
 * a group. This way the lines from more then a single file are not mixed
 * together.
 */

int
add_output_local(out_t **out, work_t *wt,int lc, long bc, char *line)
{
    out_t       *ot,*oo, *op;

    if (( ot = (out_t *)malloc(sizeof(out_t))) == NULL)
        goto ERROR;
    if (( ot->line = (char *)malloc(strlen(line)+1)) == NULL)
        goto ERROR;

    strcpy(ot->line,line);
    ot->line_count = lc;
    ot->byte_count = bc;

    if (!*out) {
        *out = ot;
        ot->next = NULL;
        return(0);
    }
    /* append to the END of the list; keep things sorted! */
    op = oo = *out;
    while(oo) {
        op = oo;
        oo = oo->next;
    }
    op->next = ot;
    ot->next = NULL;
    return(0);

 ERROR:
    if (!(flags & FS_NOERROR))
```

Code Example A-1 Source Code for `tgrep` Program

```
        fprintf(stderr,"tgrep: Output lost. No space. "
                "[%s: line %d byte %d match : %s\n",
                wt->path,lc,bc,line);
    return(1);
}

/*
 * print stats: If the -S flag is set, after ALL files have been searched,
 * main thread calls this function to print the stats it keeps on how the
 * search went.
 */

void
prnt_stats(void)
{
    float a,b,c;
    float t = 0.0;
    time_t  st_end = 0;
    char    tl[80];

    st_end = time(NULL); /* stop the clock */
    printf("\n---------------- Tgrep Stats. --------------------\n");
    printf("Number of directories searched:           %d\n",st_dir_search);
    printf("Number of files searched:                 %d\n",st_file_search);
    c = (float)(st_dir_search + st_file_search) / (float)(st_end - st_start);
    printf("Dir/files per second:                     %3.2f\n",c);
    printf("Number of lines searched:                 %d\n",st_line_search);
    printf("Number of matching lines to target:       %d\n",global_count);

    printf("Number of cascade threads created:        %d\n",st_cascade);
    printf("Number of cascade threads from pool:      %d\n",st_cascade_pool);
    a = st_cascade_pool; b = st_dir_search;
    printf("Cascade thread pool hit rate:             %3.2f%%\n",((a/b)*100));
    printf("Cascade pool overall size:                %d\n",cascade_pool_cnt);
    printf("Cascade pool size limit:                  %d\n",cascade_thr_limit);
    printf("Number of cascade threads destroyed:      %d\n",st_cascade_destroy);

    printf("Number of search threads created:         %d\n",st_search);
    printf("Number of search threads from pool:       %d\n",st_pool);
    a = st_pool; b = st_file_search;
    printf("Search thread pool hit rate:              %3.2f%%\n",((a/b)*100));
    printf("Search pool overall size:                 %d\n",search_pool_cnt);
    printf("Search pool size limit:                   %d\n",search_thr_limit);
    printf("Number of search threads destroyed:       %d\n",st_destroy);
```

Code Example A-1 Source Code for `tgrep` Program

```
    printf("Max # of threads running concurrenly:     %d\n",st_maxrun);
    printf("Total run time, in seconds.               %d\n",
           (st_end - st_start));

    /* Why did we wait ? */
    a = st_workfds; b = st_dir_search+st_file_search;
    c = (a/b)*100; t += c;
    printf("Work stopped due to no FD's:  (%.3d)       %d Times, %3.2f%%\n",
           search_thr_limit,st_workfds,c);
    a = st_worknull; b = st_dir_search+st_file_search;
    c = (a/b)*100; t += c;
    printf("Work stopped due to no work on Q:         %d Times, %3.2f%%\n",
           st_worknull,c);
    if (tglimit == UNLIMITED)
        strcpy(tl,"Unlimited");
    else
        sprintf(tl,"   %.3d   ",tglimit);
    a = st_worklimit; b = st_dir_search+st_file_search;
    c = (a/b)*100; t += c;
    printf("Work stopped due to TGLIMIT:  (%.9s) %d Times, %3.2f%%\n",
           tl,st_worklimit,c);
    printf("Work continued to be handed out:          %3.2f%%\n",100.00-t);
    printf("----------------------------------------------------\n");
}
/*
 * not running: A glue function to track if any search threads or cascade
 * threads are running. When the count is zero, and the work Q is NULL,
 * we can safely say, WE ARE DONE.
 */
void
notrun (void)
{
    pthread_mutex_lock(&work_q_lk);
    work_cnt--;
    tglimit++;
    current_open_files++;
    pthread_mutex_lock(&running_lk);
    if (flags & FS_STATS) {
        pthread_mutex_lock(&stat_lk);
        if (running > st_maxrun) {
            st_maxrun = running;
            DP(DLEVEL6,("Max Running has increased to %d\n",st_maxrun));
        }
```

Code Example A-1 Source Code for `tgrep` Program

```
        pthread_mutex_unlock(&stat_lk);
    }
    running--;
    if (work_cnt == 0 && running == 0) {
        all_done = 1;
        DP(DLEVEL6,("Setting ALL_DONE flag to TRUE.\n"));
    }
    pthread_mutex_unlock(&running_lk);
    pthread_cond_signal(&work_q_cv);
    pthread_mutex_unlock(&work_q_lk);
}

/*
 * uncase: A glue function. If the -i (case insensitive) flag is set, the
 * target strng and the read in line is converted to lower case before
 * comparing them.
 */
void
uncase(char *s)
{
    char        *p;

    for (p = s; *p != NULL; p++)
        *p = (char)tolower(*p);
}

/*
 * usage: Have to have one of these.
 */

void
usage(void)
{
    fprintf(stderr,"usage: tgrep <options> pattern <{file,dir}>...\n");
    fprintf(stderr,"\n");
    fprintf(stderr,"Where:\n");
#ifdef DEBUG
    fprintf(stderr,"Debug     -d = debug level -d <levels> (-d0 for usage)\n");
    fprintf(stderr,"Debug     -f = block fd's from use (-f #)\n");
#endif
    fprintf(stderr,"          -b = show block count (512 byte block)\n");
    fprintf(stderr,"          -c = print only a line count\n");
    fprintf(stderr,"          -h = Do NOT print file names\n");
    fprintf(stderr,"          -i = case insensitive\n");
```

Code Example A-1 Source Code for `tgrep` Program

```
    fprintf(stderr,"          -l = print file name only\n");
    fprintf(stderr,"          -n = print the line number with the line\n");
    fprintf(stderr,"          -s = Suppress error messages\n");
    fprintf(stderr,"          -v = print all but matching lines\n");
#ifdef NOT_IMP
    fprintf(stderr,"          -w = search for a \"word\"\n");
#endif
    fprintf(stderr,"          -r = Do not search for files in all "
                                "sub-directories\n");
    fprintf(stderr,"          -C = show continued lines (\"\\\")\n");
    fprintf(stderr,"          -p = File name regexp pattern. (Quote it)\n");
    fprintf(stderr,"          -P = show progress. -P 1 prints a DOT on stderr\n"
                  "               for each file it finds, -P 10 prints a DOT\n"
                "              on stderr for each 10 files it finds, etc...\n");
    fprintf(stderr,"          -e = expression search.(regexp) More then one\n");
    fprintf(stderr,"          -B = limit the number of threads to TGLIMIT\n");
    fprintf(stderr,"          -S = Print thread stats when done.\n");
    fprintf(stderr,"          -Z = Print help on the regexp used.\n");
    fprintf(stderr,"\n");
    fprintf(stderr,"Notes:\n");
    fprintf(stderr,"      If you start tgrep with only a directory name\n");
    fprintf(stderr,"     and no file names, you must not have the -r option\n");
    fprintf(stderr,"      set or you will get no output.\n");
    fprintf(stderr,"      To search stdin (piped input), you must set -r\n");
    fprintf(stderr,"      Tgrep will search ALL files in ALL \n");
    fprintf(stderr,"      sub-directories. (like */* */*/* */*/*/* etc..)\n");
    fprintf(stderr,"      if you supply a directory name.\n");
    fprintf(stderr,"      If you do not supply a file, or directory name,\n");
    fprintf(stderr,"      and the -r option is not set, the current \n");
    fprintf(stderr,"      directory \".\" will be used.\n");
    fprintf(stderr,"      All the other options should work \"like\" grep\n");
    fprintf(stderr,"      The -p patten is regexp; tgrep will search only\n");
    fprintf(stderr,"      the file names that match the patten\n");
    fprintf(stderr,"\n");
    fprintf(stderr,"      Tgrep Version %s\n",Tgrep_Version);
    fprintf(stderr,"\n");
    fprintf(stderr,"      Copy Right By Ron Winacott, 1993-1995.\n");
    fprintf(stderr,"\n");
    exit(0);
}

/*
 * regexp usage: Tell the world about tgrep custom (THREAD SAFE) regexp!
 */
```

Code Example A-1 Source Code for `tgrep` Program

```
int
regexp_usage (void)
{
    fprintf(stderr,"usage: tgrep <options> -e \"pattern\" <-e ...> "
            "<{file,dir}>...\n");
    fprintf(stderr,"\n");
    fprintf(stderr,"metachars:\n");
    fprintf(stderr,"    . - match any character\n");
    fprintf(stderr,"    * - match 0 or more occurrences of previous char\n");
    fprintf(stderr,"    + - match 1 or more occurrences of previous char.\n");
    fprintf(stderr,"    ^ - match at beginning of string\n");
    fprintf(stderr,"    $ - match end of string\n");
    fprintf(stderr,"    [ - start of character class\n");
    fprintf(stderr,"    ] - end of character class\n");
    fprintf(stderr,"    ( - start of a new pattern\n");
    fprintf(stderr,"    ) - end of a new pattern\n");
    fprintf(stderr,"    @(n)c - match <c> at column <n>\n");
    fprintf(stderr,"    | - match either pattern\n");
    fprintf(stderr,"    \\ - escape any special characters\n");
    fprintf(stderr,"    \\c - escape any special characters\n");
    fprintf(stderr,"    \\o - turn on any special characters\n");
    fprintf(stderr,"\n");
    fprintf(stderr,"To match two diffrerent patterns in the same command\n");
    fprintf(stderr,"Use the or function. \n"
            "ie: tgrep -e \"(pat1)|(pat2)\" file\n"
            "This will match any line with \"pat1\" or \"pat2\" in it.\n");
    fprintf(stderr,"You can also use up to %d -e expressions\n",MAXREGEXP);
    fprintf(stderr,"RegExp Pattern matching brought to you by Marc Staveley\n");
    exit(0);
}

/*
 * debug usage: If compiled with -DDEBUG, turn it on, and tell the world
 * how to get tgrep to print debug info on different threads.
 */

#ifdef DEBUG
void
debug_usage(void)
{
    int i = 0;

    fprintf(stderr,"DEBUG usage and levels:\n");
    fprintf(stderr,"--------------------------------------------------\n");
```

Code Example A-1 Source Code for `tgrep` Program

```
    fprintf(stderr,"Level                   code\n");
    fprintf(stderr,"-------------------------------------------------\n");
    fprintf(stderr,"0                 This message.\n");
    for (i=0; i<9; i++) {
        fprintf(stderr,"%d                 %s\n",i+1,debug_set[i].name);
    }
    fprintf(stderr,"-------------------------------------------------\n");
    fprintf(stderr,"You can or the levels together like -d134 for levels\n");
    fprintf(stderr,"1 and 3 and 4.\n");
    fprintf(stderr,"\n");
    exit(0);
}
#endif

/* Pthreads NP functions */

#ifdef __sun
void
pthread_setconcurrency_np(int con)
{
    thr_setconcurrency(con);
}

int
pthread_getconcurrency_np(void)
{
    return(thr_getconcurrency());
}

void
pthread_yield_np(void)
{
/*    In Solaris 2.4, these functions always return - 1 and set errno to ENOSYS */
    if (sched_yield())  /* call UI interface if we are older then 2.5 */
        thr_yield();
}

#else
void
pthread_setconcurrency_np(int con)
{
    return;
}
```

Code Example A-1 Source Code for `tgrep` Program

```
int
pthread_getconcurrency_np(void)
{
    return(0);
}

void
pthread_yield_np(void)
{
    return;
}
#endif
```