DCMTK - Feature #161

Notizen zum Thema DCMTK und Multithreading

2001-11-19 00:00 - Marco Eichelberg

Status:	Closed	**Start date:**	
Priority:	High	**Due date:**	
Assignee:		**% Done:**	90%
Category:	Library and Apps	**Estimated time:**	0:00 hour
Target version:	3.6.2		
Module:	alle	**Compiler:**	
Operating System:			

Description

Generell: Unter Unix müssen Programme in MT-Anwendungen anstelle einiger libc-Funktionen thread-safe-Varianten verwenden, die an dem Suffix _r erkennbar sind. Unter Windows ist die gesamte libc thread safe, wenn man als "Multithread" kompiliert. Alle statischen Puffer werden da für jeden Thread separat instanziert. Der Sonderfall ist CygWin, was aussieht wie ein Unix, aber die Windows-libc verwendet und daher keine _r-Funktionen definiert.

Einige _r-Funktionen werden (zumindest unter Solaris) nur deklariert, wenn mit -D_POSIX_PTHREAD_SEMANTICS übersetzt wird. Bei einigen ändern sich dadurch die Parameter!

Es folgt eine Diskussion der MT-safe/unsafe-Funktionen in Posix und Unix98.

```
Page 32 of the 1996 Posix.1 standard says "All functions defined by
Posix.1 and the C standard shall be thread-safe, except that the following
functions need not be thread-safe:

asctime()
ctime()
getc_unlocked()*
getchar_unlocked()*
getgrid()
getgrnam()
getlogin()
getpwnam()
getpwuid()
gmtime()
localtime()
putc_unlocked()*
putchar_unlocked()*
rand()
readdir()
strtok()
ttyname()"

Note that a thread-safe XXX_r() version of the above are available, other
than those with an asterisk.  Also note that ctermid() and tmpnam() are
only thread-safe if a nonnull pointer is used as an argument.

However, POSIX and ANSI C specify only a small part of the "traditional
UNIX programming environment", though it's a start. The real danger in
reading the POSIX list is that most people don't really know what's
included. While an inclusive list would be better than an exclusive list,
that'd be awfully long and awkward.

The Open Group (OSF and X/Open) has extended the Single UNIX Specification
(also known as "SPEC1170" for it's 1,170 UNIX interfaces, or UNIX95) to
include POSIX.1b-1993 realtime, POSIX.1c-1995 threads, and various
```

```
extensions. It's called the Single UNIX Specification, Version 2; or
UNIX98. Within this calendar year, it's safe to assume that most UNIX
versions currently branded by The Open Group (as XPG3, UNIX93, UNIX95) will
extend their brand validation to UNIX98.

The interfaces specification part of the Single UNIX Specification, Version
2 (known as XSH5), in section 2.8.2, "Thread-safety", specifies that all
interfaces defined by THIS specification will be thread-safe, except for
"the following". There are two explicit lists, and one implicit. One is the
POSIX list already quoted by Rich Stevens. The second is an additional list
of X/Open interfaces:

basename      dbm_open    fcvt        getutxline    pututxline
catgets       dbm_store   gamma       getw          setgrent
dbm_clearerr  dirname     gcvt        l64a          setkey
dbm_close     drand48     getdate     lgamma        setpwent
dbm_delete    ecvt        getenv      lrand48       setutxent
dbm_error     encrypt     getgrent    mrand48       strerror
dbm_fetch     endgrent    getpwent    nl_langinfo
dbm_firstkey  endpwent    getutxent   ptsname
dbm_nextkey   endutxent   getutxid    putenv

The implicit list is a statement that all interfaces in the "Legacy"
feature group need not be thread-safe. From another section, that list is:

advance       gamma          putw        sbrk          wait3
brk           getdtablesize  re_comp     sigstack
chroot        getpagesize    re_exec     step
compile       getpass        regcmp      ttyslot
cuserid       getw           regex       valloc
<regexp.h>    <varargs.h>    <re_comp.h>
loc1          __loc1         loc2        locs

Obviously, this is still an exclusive list rather than inclusive. But then,
if UNIX95 had 1,170 interfaces, and UNIX98 is bigger, an inclusive list
would be rather awkward. (And don't expect ME to type it into the
newsgroup!)

On the other hand... beware that if you've got a system that doesn't claim
conformance to POSIX 1003.1c-1995 (or POSIX 1003.1-1996, which includes
it), then you're not guaranteed to be able to rely even on the POSIX list,
much less the X/Open list. It's reasonable to assume that any
implementation's libpthread (or equivalent, though that name has become
pretty much defacto standard) is thread-safe. And it's probably reasonable
to assume, unless specified otherwise, that "the most common" bits of libc
are thread-safe. But without a formal statement of POSIX conformance,
you're just dealing with "good will". And, even at that, POSIX conformance
isn't validated -- so without validation by the UNIX98 branding test suite,
you've got no real guarantee of anything.
```

Dies ist eine Liste aller Aufrufe von Funktionen, die evtl. nicht thread-safe sind, im öffentlichen Teil des DCMTK, Stand 2002-04-11 (updated JR)

```
============================================================
ctime (allesamt unproblematisch da in Kommandozeilenprogrammen)
============================================================
./imagectn/apps/imagectn.cc:        printf("Cleaned up after child (%d) %s", child, ctime(&t));
./imagectn/apps/imagectn.cc:               ctime(&t));
./dcmpstat/apps/dcmpsrcv.cc:             << ") " << ctime(&t);
./dcmpstat/tests/msgserv.cc:            COUT << ctime(&now);
============================================================
getenv
============================================================
./dcmdata/libsrc/dcdict.cc:char* getenv() {
./dcmdata/libsrc/dcdict.cc:    env = getenv(DCM_DICT_ENVIRONMENT_VARIABLE);
./dcmnet/libsrc/dulfsm.cc:        if ((tcpNoDelayString = getenv("TCP_NODELAY")) != NULL)
./dcmnet/libsrc/dulfsm.cc:    if ((TCPBufferLength = getenv("TCP_BUFFER_LENGTH")) != NULL) {
```

```
./dcmnet/libsrc/dcompat.cc:    if (((env = getenv("TMPDIR")) != NULL) && access(env, AMODES) == 0)
 {
./dcmnet/libsrc/dul.cc:    if ((tcpNoDelayString = getenv("TCP_NODELAY")) != NULL) {
./dcmnet/libsrc/dul.cc:    if ((TCPBufferLength = getenv("TCP_BUFFER_LENGTH")) != NULL) {
./dcmjpeg/libijg12/jmemmgr.c:extern char * getenv JPP((const char * name));
./dcmjpeg/libijg12/jmemmgr.c:    if ((memenv = getenv("JPEGMEM")) != NULL) {
./dcmjpeg/libijg16/jmemmgr.c:extern char * getenv JPP((const char * name));
./dcmjpeg/libijg16/jmemmgr.c:    if ((memenv = getenv("JPEGMEM")) != NULL) {
./dcmjpeg/libijg8/jmemmgr.c:extern char * getenv JPP((const char * name));
./dcmjpeg/libijg8/jmemmgr.c:    if ((memenv = getenv("JPEGMEM")) != NULL) {
============================================================
rand (allesamt unproblematisch da in Kommandozeilenprogrammen)
============================================================
./dcmpstat/apps/dcmmklut.cc:                i1 = (unsigned long)(rand() * factor);
./dcmpstat/apps/dcmmklut.cc:                i2 = (unsigned long)(rand() * factor);
./dcmpstat/apps/dcmmklut.cc:                i1 = (unsigned long)(rand() * factor);
./dcmpstat/apps/dcmmklut.cc:                i2 = (unsigned long)(rand() * factor);
============================================================
readdir (allesamt unproblematisch da in Kommandozeilenprogrammen)
============================================================
./dcmdata/apps/dcmgpdir.cc:            for (dp=readdir(dirp); dp!=NULL; dp=readdir(dirp)) {
./dcmpstat/apps/dcmprscu.cc:    for (dp=readdir(dirp);((result == EC_Normal)&&(dp != NULL)); dp=re
addir(dirp))
============================================================
strerror
============================================================
./dcmpstat/libsrc/dvpshlp.cc:              logconsole->lockCerr() << "wait for child failed: " <<
strerror(errno) << endl;
./dcmpstat/apps/dcmpsrcv.cc:           if (errno != ECHILD) CERR << "wait for child failed: " << s
trerror(errno) << endl;
./dcmpstat/apps/dcmpsrcv.cc:              CERR << "Cannot create association sub-process: " << str
error(errno) << endl;
./dcmpstat/apps/dcmpsrcv.cc:                     << "  reason: cannot create association sub-proces
s: " << strerror(errno) << endl
./dcmpstat/apps/dcmprscp.cc:           if (errno != ECHILD) CERR << "wait for child failed: " << s
trerror(errno) << endl;
./dcmdata/libsrc/cmdlnarg.cc:        ofConsole.lockCerr() << "INTERNAL ERROR: cannot map stderr to
 stdout: " << strerror(errno) << endl;
./dcmdata/libsrc/cmdlnarg.cc:        ofConsole.lockCerr() << "INTERNAL ERROR: cannot unbuffer stdo
ut: " << strerror(errno) << endl;
./dcmdata/libsrc/cmdlnarg.cc:        ofConsole.lockCerr() << "INTERNAL ERROR: cannot unbuffer stde
rr: " << strerror(errno) << endl;
./dcmdata/libsrc/dcuid.cc:       ofConsole.lockCerr() << "sysinfo: " << strerror(errno) << endl;
./imagectn/libsrc/dbindex.cc:        CERR << "DB_lseek: cannot get current position: " << strerror
(errno) << endl;
./imagectn/libsrc/dbindex.cc:        CERR << "DB_lseek: cannot get end of file position: " << stre
rror(errno) << endl;
./imagectn/libsrc/dbindex.cc:        CERR << "DB_lseek: cannot reset current position: " << strerr
or(errno) << endl;
./imagectn/libsrc/dbindex.cc:   CERR << msg << ": " << strerror(errno) << endl;
./imagectn/libsrc/dbstore.cc:        << "IMAGECTN_DB_ERROR: " << strerror(errno) << endl;
./imagectn/libsrc/dbstore.cc:        << strerror(errno) << endl;
./imagectn/libsrc/dbutils.cc:            CERR << phandle->indexFilename << ": " << strerror(errno)
 << endl;
./imagectn/apps/scemove.cc:     fname, strerror(errno));
./imagectn/apps/sceget.cc:      fname, strerror(errno));
./imagectn/apps/imagectn.cc:    errmsg("cannot access %s: %s", opt_configFileName, strerror(errno)
);
./imagectn/apps/imagectn.cc:        errmsg("wait for child failed: %s", strerror(errno));
./imagectn/apps/imagectn.cc:        errmsg("wait for child failed: %s", strerror(errno));
./imagectn/apps/imagectn.cc:           strerror(errno));
./imagectn/apps/tiquery.cc:        imgFile, strerror(errno));
./imagectn/apps/ti.cc:                strerror(errno));
./ofstd/libsrc/ofthread.cc:  const char *str = strerror(code);
./ofstd/libsrc/ofthread.cc:  const char *str = strerror(code);
./ofstd/libsrc/ofthread.cc:  const char *str = strerror(code);
./ofstd/libsrc/ofthread.cc:  const char *str = strerror(code);
```

```
./ofstd/libsrc/ofthread.cc:  const char *str = strerror(code);
./wlistctn/libsrc/wrklstdb.cc:        << "return code: " << strerror(errno) << endl;
./wlistctn/tests/wltest.cc:        strerror(errno));
./wlistctn/tests/wltest.cc:     errmsg("Cannot open file: %s: %s", queryFilename, strerror(errno))
;
./wlistctn/apps/wlistctn.cc:        errmsg("%s: %s", baseWorklistDBPath, strerror(errno));
./wlistctn/apps/wlistctn.cc:                errmsg("wait for child failed: %s", strerror(errno));
./wlistctn/apps/wlistctn.cc:                errmsg("wait for child failed: %s", strerror(errno));
./wlistctn/apps/wlistctn.cc:            errmsg("Cannot create association sub-process: %s", strerr
or(errno));
./dcmnet/libsrc/dimse.cc:            dataFileName, strerror(errno));
./dcmnet/libsrc/dimse.cc:               dataFileName, strerror(errno));
./dcmnet/libsrc/dulfsm.cc:      sprintf(buf1, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dulfsm.cc:        sprintf(buf1, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dulfsm.cc:          sprintf(buf1, "TCP Initialization Error: %s", strerror(errno))
;
./dcmnet/libsrc/dulfsm.cc:          sprintf(buf2, "TCP Initialization Error: %s", strerror(errno))
;
./dcmnet/libsrc/dulfsm.cc:              sprintf(buf1, "TCP Initialization Error: %s", strerror(err
no));
./dcmnet/libsrc/dulfsm.cc:      sprintf(buf1, "TCP I/O Error (%s) occurred in routine: %s", strerr
or(errno), "requestAssociationTCP");
./dcmnet/libsrc/dulfsm.cc:      sprintf(buf1, "TCP I/O Error (%s) occurred in routine: %s", strerr
or(errno), "ReplyAssociationTCP");
./dcmnet/libsrc/dulfsm.cc:          sprintf(buf1, "TCP I/O Error (%s) occurred in routine: %s", st
rerror(errno), "sendAssociationRJTCP");
./dcmnet/libsrc/dulfsm.cc:          sprintf(buf1, "TCP I/O Error (%s) occurred in routine: %s", st
rerror(errno), "sendAbortTCP");
./dcmnet/libsrc/dulfsm.cc:          sprintf(buf1, "TCP I/O Error (%s) occurred in routine: %s", st
rerror(errno), "sendReleaseRQTCP");
./dcmnet/libsrc/dulfsm.cc:          sprintf(buf1, "TCP I/O Error (%s) occurred in routine: %s", st
rerror(errno), "sendReleaseRPTCP");
./dcmnet/libsrc/dulfsm.cc:        sprintf(buf1, "TCP I/O Error (%s) occurred in routine: %s", stre
rror(errno), "writeDataPDU");
./dcmnet/libsrc/dulfsm.cc:        sprintf(buf2, "TCP I/O Error (%s) occurred in routine: %s", stre
rror(errno), "writeDataPDU");
./dcmnet/libsrc/dcompat.cc:  CERR << s << ": " << strerror(errno) << endl;
./dcmnet/libsrc/dcompat.cc:  CERR << s << ": " << strerror(errno) << endl;
./dcmnet/libsrc/dcompat.cc
./dcmnet/libsrc/dcompat.cc:char *strerror(int errornum)
./dcmnet/libsrc/dul.cc:        sprintf(buf3, "TCP Initialization Error: %s, accept failed on socke
t %d", strerror(errno), sock);
./dcmnet/libsrc/dul.cc:        sprintf(buf4, "TCP Initialization Error: %s, setsockopt failed on s
ocket %d", strerror(errno), sock);
./dcmnet/libsrc/dul.cc:        sprintf(buf1, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dul.cc:          sprintf(buf1, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dul.cc:      sprintf(buf1, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dul.cc:          sprintf(buf1, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dul.cc:          sprintf(buf2, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dul.cc:          sprintf(buf3, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dul.cc:          sprintf(buf4, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/libsrc/dul.cc:          sprintf(buf5, "TCP Initialization Error: %s", strerror(errno));
./dcmnet/apps/movescu.cc:       errmsg("Cannot open file: %s: %s", fname, strerror(errno));
./dcmnet/apps/findscu.cc:       errmsg("Cannot open file: %s: %s", fname, strerror(errno));
./dcmnet/apps/storescu.cc:  errmsg("Cannot open file: %s: %s", fname, strerror(errno));
./dcmnet/include/dcompat.h:char *strerror(int e);
============================================================
wait3
============================================================
./dcmpstat/libsrc/dvpshlp.cc:    child = wait3(&status, options, &rusage);
./dcmpstat/apps/dcmpsrcv.cc:        child = wait3(&status, options, &rusage);
./dcmpstat/apps/dcmprscp.cc:        child = wait3(&status, options, &rusage);
./imagectn/apps/imagectn.cc:    child = wait3(&status, options, &rusage);
./wlistctn/apps/wlistctn.cc:        child = wait3(&status, options, &rusage);
./dcmnet/include/dcompat.h:int wait3(int *statusp, int options, struct rusage *rusage);
```

Related issues:

Related to DCMTK - Bug #600: OFStandard::searchDirectoryRecursively() relies ...	**Closed**	**2014-06-26**

History

#1 - 2012-08-02 15:37 - Jan Schlamelcher

Habe dieses script zum untersuchen auf thread unsafe funktionen gefunden: http://git.savannah.gnu.org/cgit/gsasl.git/tree/tests/threadsafety
Orignial Post dazu:http://blog.josefsson.org/2009/06/23/thread-safe-functions/

#2 - 2013-08-22 18:37 - Jörg Riesmeier

- Category set to Library and Apps

- Priority changed from Normal to High

- Target version set to 3.6.2

- % Done changed from 0 to 60

#3 - 2014-09-22 17:02 - Michael Onken

- Private changed from No to Yes

#4 - 2015-01-15 13:51 - Michael Onken

- Private changed from Yes to No

#5 - 2017-03-27 17:29 - Jan Schlamelcher

The remaining issues are:
- rand -- may be ignored, since only unit tests ~~and ofuuid~~ are affected, where security is not an issue (ofuuid is already using a mutex around rand()).
- ~~readdir~~ -- fixed recently.
- getenv -- should be ok since we never call putenv and it is mostly called during program startup.
- ~~cuserid~~ -- fixed recently
- ~~wait3~~ -- already been fixed, waitpid is already used instead if available.

#6 - 2017-03-27 19:50 - Jan Schlamelcher

- % Done changed from 60 to 90

#7 - 2017-03-28 10:35 - Jan Schlamelcher

- Status changed from New to Closed

Remaining issues have been fixed by recent commit #a4a69b0167a08, other issues turned out to already have been fixed/accounted for on higher levels in the code (mutexes around the non reentrant functions, #ifdefs that already prefer reentrant alternatives if available, etc.).