Commit 11a11ec

Father Chrysostomos committed on Sep 28, 2010

```
[perl #75174] Clone dir handles

On systems that support fchdir, use it to clone dir handles.

On other systems, at least for now, don't give the new thread a copy
of the handle. This is not ideal, but better than crashing.
```

blead · v5.43.10 ••• if-0.0602

1 parent 6034bce commit 11a11ec

3 files changed +224 -2 lines changed

Top

Filter files...

- MANIFEST
- sv.c
- t/op
  - threads-dirh.t

3 files changed +224 -2 lines changed

Search within code

MANIFEST

```
@@ -4653,6 +4653,7 @@ t/op/symbolcache.t See if undef/delete works on
stashes with functions
4653 4653  t/op/sysio.t            See if sysread and syswrite work
4654 4654  t/op/taint.t            See if tainting works
4655 4655  t/op/threads_create.pl      Ancillary file for t/op/threads.t
     4656 + t/op/threads-dirh.t     Test interaction of threads and dir handles
4656 4657  t/op/threads.t          Misc. tests for perl features with threads
4657 4658  t/op/tiearray.t         See if tie for arrays works
```

```
4658  4659   t/op/tie_fetch_count.t      See if FETCH is only called once on tied
                                         variables
```

sv.c

```
                @@ -10838,11 +10838,101 @@ Perl_fp_dup(pTHX_ PerlIO *const fp, const
                char type, CLONE_PARAMS *const param)
10838  10838    DIR *
10839  10839    Perl_dirp_dup(pTHX_ DIR *const dp)
10840  10840    {
       10841  + #ifdef HAS_FCHDIR
       10842  +     DIR *ret;
       10843  +     DIR *pwd;
       10844  +     register const Direntry_t *dirent;
       10845  +     char smallbuf[256];
       10846  +     char *name = NULL;
       10847  +     STRLEN len = -1;
       10848  +     long pos;
       10849  + #endif
       10850  +
10841  10851        PERL_UNUSED_CONTEXT;
       10852  +
       10853  + #ifdef HAS_FCHDIR
10842  10854        if (!dp)
10843  10855        return (DIR*)NULL;
10844        -     /* XXX TODO */
10845        -     return dp;
       10856  +     /* look for it in the table first */
       10857  +     ret = (DIR*)ptr_table_fetch(PL_ptr_table, dp);
       10858  +     if (ret)
       10859  +     return ret;
       10860  +
       10861  +     /* create anew */
       10862  +
       10863  +     /* open the current directory (so we can switch back) */
       10864  +     if (!(pwd = PerlDir_open("."))) return (DIR *)NULL;
       10865  +
       10866  +     /* chdir to our dir handle and open the present working directory */
       10867  +     if (fchdir(my_dirfd(dp)) < 0 || !(ret = PerlDir_open("."))) {
       10868  +     PerlDir_close(pwd);
```

```
10869 +     return (DIR *)NULL;
10870 +     }
10871 +     /* Now we should have two dir handles pointing to the same dir. */
10872 +
10873 +     /* Be nice to the calling code and chdir back to where we were. */
10874 +     fchdir(my_dirfd(pwd)); /* If this fails, then what? */
10875 +
10876 +     /* We have no need of the pwd handle any more. */
10877 +     PerlDir_close(pwd);
10878 +
10879 + #ifdef DIRNAMLEN
10880 + # define d_namlen(d) (d)->d_namlen
10881 + #else
10882 + # define d_namlen(d) strlen((d)->d_name)
10883 + #endif
10884 +     /* Iterate once through dp, to get the file name at the current posi-
10885 +        tion. Then step back. */
10886 +     pos = PerlDir_tell(dp);
10887 +     if ((dirent = PerlDir_read(dp))) {
10888 +     len = d_namlen(dirent);
10889 +     if (len <= sizeof smallbuf) name = smallbuf;
10890 +     else Newx(name, len, char);
10891 +     Move(dirent->d_name, name, len, char);
10892 +     }
10893 +     PerlDir_seek(dp, pos);
10894 +
10895 +     /* Iterate through the new dir handle, till we find a file with the
10896 +        right name. */
10897 +     if (!dirent) /* just before the end */
10898 +     for(;;) {
10899 +         pos = PerlDir_tell(ret);
10900 +         if (PerlDir_read(ret)) continue; /* not there yet */
10901 +         PerlDir_seek(ret, pos); /* step back */
10902 +         break;
10903 +     }
10904 +     else {
10905 +     const long pos0 = PerlDir_tell(ret);
10906 +     for(;;) {
10907 +         pos = PerlDir_tell(ret);
10908 +         if ((dirent = PerlDir_read(ret))) {
```

```
                10909 +         if (len == d_namlen(dirent)
                10910 +          && memEQ(name, dirent->d_name, len)) {
                10911 +             /* found it */
                10912 +             PerlDir_seek(ret, pos); /* step back */
                10913 +             break;
                10914 +         }
                10915 +         /* else we are not there yet; keep iterating */
                10916 +         }
                10917 +         else { /* This is not meant to happen. The best we can do is
                10918 +                   reset the iterator to the beginning. */
                10919 +         PerlDir_seek(ret, pos0);
                10920 +         break;
                10921 +         }
                10922 +     }
                10923 +     }
                10924 + #undef d_namlen
                10925 +
                10926 +     if (name && name != smallbuf)
                10927 +     Safefree(name);
                10928 +
                10929 +     /* pop it in the pointer table */
                10930 +     ptr_table_store(PL_ptr_table, dp, ret);
                10931 +
                10932 +     return ret;
                10933 + #else
                10934 +     return (DIR*)NULL;
                10935 + #endif
10846           10936     }
10847           10937
10848           10938     /* duplicate a typeglob */
```

t/op/threads-dirh.t

```
                  ••• @@ -0,0 +1,131 @@
                    1 + #!perl
                    2 +
                    3 + # Test interaction of threads and directory handles.
                    4 +
                    5 + BEGIN {
                    6 +     chdir 't' if -d 't';
```

```
+      @INC = '../lib';
+      require './test.pl';
+      $| = 1;
+
+      require Config;
+      if (!$Config::Config{useithreads}) {
+         print "1..0 # Skip: no ithreads\n";
+         exit 0;
+      }
+      if ($ENV{PERL_CORE_MINITEST}) {
+        print "1..0 # Skip: no dynamic loading on miniperl, no threads\n";
+        exit 0;
+      }
+
+      plan(6);
+ }
+
+ use strict;
+ use warnings;
+ use threads;
+ use threads::shared;
+ use File::Path;
+ use File::Spec::Functions qw 'updir catdir';
+ use Cwd 'getcwd';
+
+ # Basic sanity check: make sure this does not crash
+ fresh_perl_is <<'# this is no comment', 'ok', {}, 'crash when duping dirh';
+    use threads;
+    opendir dir, 'op';
+    async{}->join for 1..2;
+    print "ok";
+ # this is no comment
+
+ my $dir;
+ SKIP: {
+  my $skip = sub {
+    chdir($dir);
+    chdir updir;
+    skip $_[0], 5
+  };
```

```
47 +
48 +  if(!$Config::Config{d_fchdir}) {
49 +   $::TODO = 'dir handle cloning currently requires fchdir';
50 +  }
51 +
52 +  my @w :shared; # warnings accumulator
53 +  local $SIG{__WARN__} = sub { push @w, $_[0] };
54 +
55 +  $dir = catdir getcwd(), "thrext$$" . int rand() * 100000;
56 +
57 +  rmtree($dir);
58 +  mkdir($dir);
59 +
60 +  # Create a dir structure like this:
61 +  #   $dir
62 +  #     |
63 +  #     `- toberead
64 +  #           |
65 +  #           +---- thrit
66 +  #           |
67 +  #           +---- rile
68 +  #           |
69 +  #           `---- zor
70 +
71 +  chdir($dir);
72 +  mkdir 'toberead';
73 +  chdir 'toberead';
74 +  {open my $fh, ">thrit" or &$skip("Cannot create file thrit")}
75 +  {open my $fh, ">rile" or &$skip("Cannot create file rile")}
76 +  {open my $fh, ">zor" or &$skip("Cannot create file zor")}
77 +  chdir updir;
78 +
79 +  # Then test that dir iterators are cloned correctly.
80 +
81 +  opendir my $toberead, 'toberead';
82 +  my $start_pos = telldir $toberead;
83 +  my @first_2 = (scalar readdir $toberead, scalar readdir $toberead);
84 +  my @from_thread = @{; async { [readdir $toberead ] } ->join };
85 +  my @from_main = readdir $toberead;
86 +  is join('-', sort @from_thread), join('-', sort @from_main),
```

```
87  +      'dir iterator is copied from one thread to another';
88  +  like
89  +    join('-', "", sort(@first_2, @from_thread), ""),
90  +    qr/(?<!-rile)-rile-thrit-zor-(?!zor-)/i,
91  +   'cloned iterator iterates exactly once over everything not already seen';
92  +
93  +  seekdir $toberead, $start_pos;
94  +  readdir $toberead for 1 .. @first_2+@from_thread;
95  +  is
96  +    async { readdir $toberead // 'undef' } ->join, 'undef',
97  +   'cloned dir iterator that points to the end of the directory'
98  +  ;
99  +
100 +  # Make sure the cloning code can handle file names longer than 255 chars
101 +  SKIP: {
102 +   chdir 'toberead';
103 +   open my $fh,
104 +     ">floccipaucinihilopilification-"
105 +    . "pneumonoultramicroscopicsilicovolcanoconiosis-"
106 +    . "lopadotemachoselachogaleokranioleipsanodrimypotrimmatosilphiokarabo"
107 +    . "melitokatakechymenokichlepikossyphophattoperisteralektryonoptokephal"
108 +    . "liokinklopeleiolagoiosiraibaphetraganopterygon"
109 +     or
110 +      chdir updir,
111 +      skip("OS does not support long file names (and I mean *long*)", 1);
112 +   chdir updir;
113 +   opendir my $dirh, "toberead";
114 +   my $test_name
115 +     = "dir iterators can be cloned when the next fn > 255 chars";
116 +   while() {
117 +    my $pos = telldir $dirh;
118 +    my $fn = readdir($dirh);
119 +    if(!defined $fn) { fail($test_name); last SKIP; }
120 +    if($fn =~ 'lagoio') {
121 +     seekdir $dirh, $pos;
122 +     last;
123 +    }
124 +   }
125 +   is length async { scalar readdir $dirh } ->join, 257, $test_name;
126 +  }
```

```
127 +
128 +  is scalar @w, 0, 'no warnings during all that' or diag @w;
129 +  chdir updir;
130 + }
131 + rmtree($dir);
```

Comments 0

Please sign in to comment.