$Id: README.FTB,v 1.2 2008/12/12 05:23:59 phargrov Exp $

Beginning in release 0.8.0, we have begun integration of BLCR with
the "Fault Tolerance Backplane" software from the Coordinated
Infrastructure for Fault Tolerant Systems (CIFTS) project. In short,
the FTB is a publish/subscribe system for sharing of fault information.
You can find more information about CIFTS and FTB from
  http://www.mcs.anl.gov/research/cifts/index.php

If you have the FTB software installed on your Linux system, then you
can configure BLCR with the option "--with-ftb=/usr/local/ftb" to get
FTB support in BLCR. Of course, you may need to replace "/usr/local/ftb"
to match you actual FTB installation. Additionally, you will need to
be sure that the FTB libraries are either in your LD_LIBRARY_PATH or
in a system location. You can apply the same guidance that we give for
the BLCR libraries in the BLCR Admin Guide.

Note that if you are running FTB-0.6, there is a patch at the end of
this file that resolves some known FTB bugs that could have the effect
of making the BLCR+FTB combination less usable.

In BLCR version 0.8.0, we have begun the integration by having BLCR
publish simple events to the FTB for each Checkpoint and Restart
operation it performs. The event schema for BLCR is not yet fixed,
and you should not rely on the current events being available in
any future BLCR release until this file is updated to say otherwise.

The current BLCR schema:

```
event_space: "FTB.checkpoint_sw.blcr"
event_name: "CHKPT_BEGIN"
  severity: "INFO"
  event_type: 1
  payload: none
  description: Event generated at start of each checkpoint request
event_name: "CHKPT_END"
  severity: "INFO"
  event_type: 2
  payload: event_handle of corresponding CHKPT_BEGIN
  description: Event generated for each successful checkpoint request
event_name: "CHKPT_ERROR"
  severity: "ERROR"
  event_type: 2
  payload: event_handle of corresponding CHKPT_BEGIN; 4-byte errno value
  description: Event generated for each failed checkpoint request
event_name: "RSTRT_BEGIN"
  severity: "INFO"
  event_type: 1
  payload: none
  description: Event generated at start of each restart request
event_name: "RSTRT_END"
  severity: "INFO"
  event_type: 2
  payload: event_handle of corresponding RSTRT_BEGIN
  description: Event generated for each successful restart request
event_name: "RSTRT_ERROR"
  severity: "ERROR"
```

  event_type: 2
  payload: event_handle of corresponding RSTRT_BEGIN; 4-byte errno value
  description: Event generated for each failed restart request

Note that one should not assume there will always be a CHKPT_END or CHKPT_ERROR for every CHKPT_BEGIN, and similarly for RSTRT events. This is because it is possible for the requesting process to be killed by certain errors, preventing them from reporting the ERROR event. However, one can assume no more than one _END or _ERROR event (and never both) per _BEGIN event.

If running FTB-0.6, we strongly recommend applying the following patch.
Without it, at least four tests in BLCR's testsuite will often fail:
  cs_enter_leave
  cs_enter_leave2
  cr_try_enter
  dup.ct
and your syslog will be flooded with "skipping a socket" messages.
This patch represents items #43, #46 and #47 in the CIFTS trac database.

```
--- FTB-0.6-orig/src/manager_lib/network/network_sock/ftb_network_tcp.c	2008-09-02 22:53:16.000000000 -0700
+++ FTB-0.6/src/manager_lib/network/network_sock/ftb_network_tcp.c	2008-12-10 22:59:51.000000000 -0800
@@ -37,6 +37,7 @@
 #include <time.h>
 #include <sys/ioctl.h>
 #include <net/if.h>
+#include <fcntl.h>

 #include "ftb_def.h"
 #include "ftb_client_lib_defs.h"
@@ -201,6 +202,7 @@
     hp = FTBNI_gethostbyname(addr->name);
     if (hp == NULL) {
         FTB_WARNING("cannot find host %s", addr->name);
+        close(entry->fd);
         return FTB_ERR_NETWORK_NO_ROUTE;
     }
     memset((void *) &sa, 0, sizeof(sa));
@@ -208,11 +210,22 @@
     sa.sin_family = AF_INET;
     sa.sin_port = htons(addr->port);
     if (setsockopt(entry->fd, IPPROTO_TCP, TCP_NODELAY, (char *) &optval, sizeof(optval))) {
+        close(entry->fd);
         return FTB_ERR_NETWORK_GENERAL;
     }
     if (connect(entry->fd, (struct sockaddr *) &sa, sizeof(sa)) < 0) {
+        close(entry->fd);
         return FTB_ERR_NETWORK_NO_ROUTE;
     }
+#ifdef FD_CLOEXEC
+    else {
+        int rc = fcntl(entry->fd, F_GETFD);
+        if ((rc < 0) || (fcntl(entry->fd, F_SETFD, rc | FD_CLOEXEC) < 0)) {
+            FTB_WARNING("Failed to set FD_CLOEXEC");
+            /* non-fatal */
+        }
+    }
```

```
+#endif
     entry->dst = (FTB_location_id_t *) malloc(sizeof(FTB_location_id_t));
     FTBNI_UTIL_WRITE(entry, &(FTBN_config_location.FTB_system_id), sizeof(uint32_t));
     FTBNI_UTIL_WRITE(entry, &FTBN_my_location_id, sizeof(FTB_location_id_t));
@@ -345,6 +358,7 @@

     FTBU_list_for_each(pos, FTBNI_connection_table, temp) {
             FTB_connection_entry_t *entry = (FTB_connection_entry_t *) pos;
+            close(entry->fd);
             free(entry->dst);
             free(entry);
     }
```