ROOT Tutorials – Session 10

Remote File Access, Networking SQL & Threads

Fons Rademakers

Introduction

- The following items will be covered:
  - Remote ROOT file access
  - Networking
  - SQL interface
  - Threads

Remote File Access

Remote Access

- Several classes deriving from TFile providing remote file access:
  - TNetFile performs remote access via the special rootd daemon
  - TWebFile performs remote read-only access via an Apache httpd daemon
  - TRFIOFile performs remote access via the rfiod daemon (RFIO is part of the CERN SHIFT software). RFIO has an interface to the Castor mass storage system

Remote Access (cont.)

- TDCacheFile performs remote access via the dcached daemon which provides access to the dCache mass storage system developed by DESY and Fermilab
- TChirpFile performs remote access via a Chirp server which is used in the Condor/VDT Grid software

Access Transparency

```
TFile *f1 = TFile::Open("local.root")

TFile *f2 = TFile::Open("root://cdfsga.fnal.gov/bigfile.root")

TFile *f3 = TFile::Open("rfio:/castor.cern.ch/alice/aap.root")

TFile *f4 = TFile::Open("dcache://main.desy.de/h1/run2001.root")

TFile *f5 = TFile::Open("chirp://hep.wisc.edu/data1.root")

TFile *f5 = TFile::Open("http://root.cern.ch/geom/atlas.root")
```

The `rootd` Daemon

- Daemon optimized for ROOT file access
- Performance typically better than NFS and AFS
- Easy to setup without superuser permissions:
  - `rootd -p 5151`
- Can also be started via (x)inetd
  - By default listens on port 1094 (assigned by IANA)
  - Supports anonymous mode
- Supports several authentication methods:
  - Clear text passwd
  - SRP
  - Kerberos 5
  - Globus
  - SSH
  - UidGid

Authentication

- The authentication methods are governed by two files on the client side:
  - $ROOTSYS/etc/system.rootrc
  - $HOME/.rootauthrc
- And by one file on the remote side:
  - $ROOTSYS/etc/system.rootdaemonrc

Exercise

- Try opening the /opt/h1data/dstarp1a.root remote file on the machine of your neighbor

```
TFile *f =
TFile::Open("root://pchret23XX//opt/h1data/dstarp1a.root");
```

The remote server accepts ssh authentication, make the change in the system.rootrc so you can connect the file

Networking

Networking Classes

- Provide a simple but complete set of networking classes
  - TSocket, TServerSocket, TMonitor, TInetAddress, TMessage
- Operating system independent
  - Thanks to TSystem

Setting Up a Connection

Server side

```
// Open server socket waiting for connections on specified port
TServerSocket *ss = new TServerSocket(9090, kTRUE);

// Accept a connection and return a full-duplex communication socket
TSocket *sock = ss->Accept();

// Close the server socket (unless we will use it later to wait for
// another connection).
ss->Close();
```

Client side

```
// Open connection to server
TSocket *sock = new TSocket("localhost", 9090);
```

Sending Objects

Client Side

```
TH1 *hpx = new TH1F("hpx","This is the px distribution",100,-4,4);
TMessage mess(kMESS_OBJECT);
mess.WriteObject(hpx);
sock->Send(mess);
```

Server Side

```
TMessage *mess;
while (sock->Recv(mess)) {
   if (mess->What() == kMESS_OBJECT) {
      if (mess->GetClass()->InheritsFrom("TH1")) {
         TH1 *h = (TH1 *)mess->ReadObject();
         . . .
      }
   } else if (mess->What() == kMESS_STRING)
      . . .
   delete mess;
}
```

Waiting for Objects

- To wait for multiple sockets you can use a TMonitor object:

```
TMonitor *mon = new TMonitor;
mon->Add(socket1);
mon->Add(socket2);
…
while (1) {
   TMessage *mess;
   TSocket  *s;

   s = mon->Select();
   …
}
```

Exercise

- See hclient.C, hserv.C
- Run hserv.C and 2 hclient.C's
  - First start hserv.C, then the hclient.C's
- Ask you neighbor to run hserv.C, modify your hclient.C so that you will talk to your neighbors hserv.C

GRID Networking - Long Fat Pipes

- Long fat pipes are WAN links with a large bandwidth*delay product
- For optimal performance keep pipe full
- By default this is not the case
  - maximum TCP buffer size is 64KB
  - for a pipe with a 192KB bandwidth*delay product the pipe is empty 60% of the time

TCP Window Scaling (RFC 1323)

- A solution is to use a TCP buffer size equal to the bandwidth*delay product
- This support for large TCP buffers (window scaling) is described in RFC 1323

- Problem: system administrators are needed to change maximum TCP buffer sizes on source and destination machines, e.g. for Linux:
  - echo 200000 > /proc/sys/net/core/rmem_max

Parallel Sockets

- Buffer is striped over multiple sockets in equal parts
- Ideal number of parallel sockets depends on bandwidth*delay product (assuming default 64KB TCP buffer size). No system manager needed to tune network

- Same performance as with large buffers

Parallel Sockets in ROOT

- Parallel socket classes, TPSocket and TPServerSocket, that derive from TSocket and TServerSocket

```
// Open server socket waiting for connections on specified port
TServerSocket *ss = new TPServerSocket(9090, kTRUE);

// Accept a connection and return a full-duplex communication socket
TSocket *sock = ss->Accept();
```

- TNetFile and rootd daemon support parallel sockets

Parallel FTP

- The TFTP class supports parallel sockets and rootd daemon for fast WAN file transfers (same performance as gridftp and bbftp)
- Supports all standard ftp commands
- Anonymous ftp
- Performance, CERN - GSI:
  - wu-ftp: 1.4 MB/s
  - TFTP: 2.8 MB/s

Exercise

- Create a TFTP object connecting to your neighbors machine
- Cd to the directory /opt/h1data
- Get the file dstarp1a.root

Main Networking Features

- Objects can be passed by value over a network connection
- Easy multiplexing on multiple sockets via TMonitor or via the main event loop
- Support for non-blocking sockets
- Most important socket options settable via TSocket::SetOption() (buffer sizes, non-blocking, OOB, keep alive, etc.)

SQL Interface

SQL Interface

- RDBMS access via a set of abstract base classes
  - TSQLServer, TSQLResult and TSQLRow
- Concrete implementations for MySQL and Oracle exist
  - TMySQLServer, TMySQLResult and TMySQLRow
  - TOracleServer, TOracleResult and TOracleRow

SQL Interface for TTree's

- Also TTree's can be queried via this interface
  - TTreeResult and TTreeRow
- A TTreeResult is returned by the TTree::Query() method
- Via these classes it is trivial to access an RDBMS via the interpreter and scripts
- There is also available an ODBC interface (based on Java's JDBC) to access databases

SQL Interface Usage

```
{
   TSQLServer *db = TSQLServer::Connect("mysql://localhost/test",
                                        "nobody", "");
   TSQLResult *res = db->Query("select count(*) from runcatalog "
                               "where tag&(1<<2)");

   int nrows   = res->GetRowCount();
   int nfields = res->GetFieldCount();
   for (int i = 0; i < nrows; i++) {
      TSQLRow *row = res->Next();
      for (int j = 0; j < nfields; j++) {
         printf("%s\n", row->GetField(j));
      }
      delete row;
   }

   delete res;
   delete db;
}
```

Performance Comparison

SQL

```
CREATE TABLE runcatalog (
   dataset        VARCHAR(32) NOT NULL,
   run            INT NOT NULL,
   firstevent     INT,
   events         INT,
   tag            INT,
   energy         FLOAT,
   runtype        ENUM('physics'
                  'cosmics','test'),
   target         VARCHAR(10),
   timef          TIMESTAMP NOT NULL,
   timel          TIMESTAMP NOT NULL,
   rawfilepath    VARCHAR(128),
   comments       VARCHAR(80)
)
```

C++/ROOT

```
class RunCatalog : public TObject {
public:
   enum ERunType { kPhysics, kCosmics,
                   kTest };

   char         fDataSet[32];
   Int_t        fRun;
   Int_t        fFirstEvent;
   Int_t        fEvents;
   Int_t        fTag;
   Float_t      fEnergy;
   ERunType     fRunType;
   char         fTarget[10];
   UInt_t       fTimeFirst;
   UInt_t       fTimeLast;
   char         fRawFilePath[128];
   char         fComments[80];
};
```

Performance Comparison Filling 500000 Entries

MySQL

- 177 s real-time
- 0.3 MB/s
- 54.6 MB DB file

TTree

- 42 s real-time (43 via rootd)
- 3.4 MB/s
- 11.5 MB DB file (compression level 1)

All results on PII 366, 256 MB RAM, RH 6.1

Performance Comparison Select of 2 Columns

MySQL

- 4.5 s real-time
- 12.1 MB/s

TTree

- 2.8 s real-time (2.9 via rootd)
- 4.1 MB/s (19.5 MB/s)

```
SELECT dataset,rawfilepath FROM runcatalog
WHERE tag&7 AND (run=490001 OR run=300122)
```

Performance Comparison

- ROOT TTree's are in this case a factor 4 smaller
- Filling time of TTree's is 4.2 times faster
- Query time of TTree's is 2 times faster
- However, MySQL and especially Oracle have the typical advantages of: locking, transactions, roll-back, SQL, etc.

More RDBMS Interfaces

- In addition to original MySQL interface:
- Oracle

  by Michael Dahlinger of GSI
  http://www.gsi.de/computing/root/OracleAccess.htm
- PostgreSQL

  by Gian Paolo Ciceri
- SAPDB

  by Marc Hemberger
- RDBC, a version of JDBC on top of ODBC

  by Valeriy Onuchin

Very Large Databases

- A VLDB can be made by using a combination of the ROOT object store and an RDBMS
- The RDBMS is typically used as catalog to keep track of the many ROOT object stores
- This is exactly what the POOL project tries to achieve

Threads

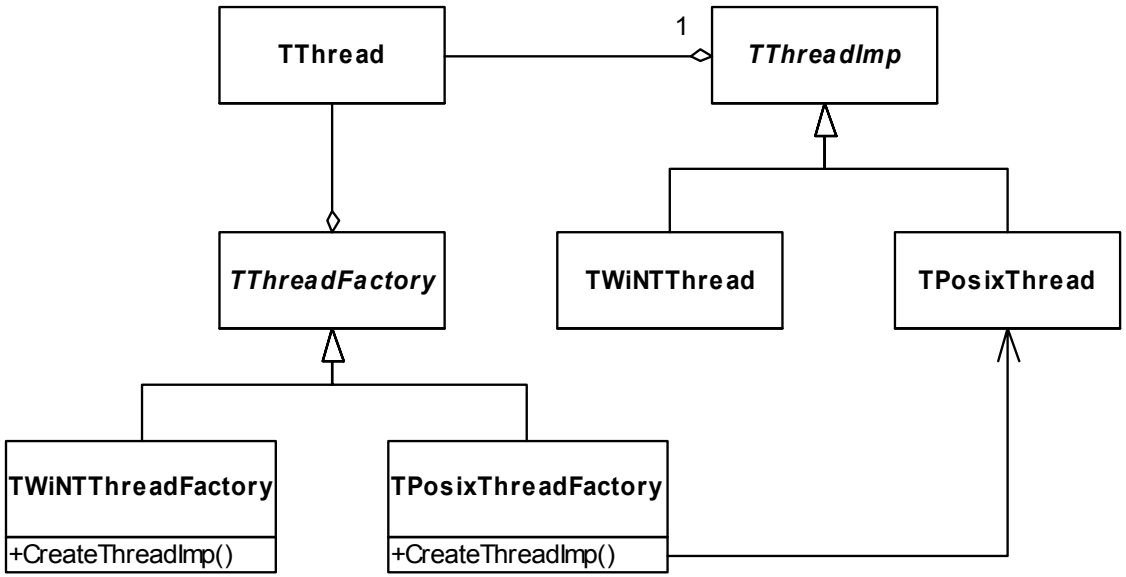
Threads

- Threads allow different tasks to run in a single process
- Threads share the processes address space
- Threads are much more convenient to program than asynchronous tasks
- On SMP systems threads can be scheduled to run on different CPU's
- However, threads are fairly recent additions to most operating systems
  - Many "legacy" libraries are not thread safe (C-lib, X11, etc.)

Threads in ROOT

- The following thread classes are supported:
  - TThread, TMutex, TCondition, TSemaphore, TRWLock, TLockGuard and TThreadPool
- Via a factory pattern most of these classes have a pointer to the actual machine dependent implementation (either for Posix or WinNT):
  - TPosixThread, TPosixMutex, TPosixCondition

Thread Creation Pattern



ROOT and Thread Safety

- ROOT and CINT are not thread safe:
  - Many globals in CINT
  - Many globals in ROOT (gDirectory, etc)
  - No locking in containers
- Solution:
  - Put locks around CINT and container access:
    - Central lock before calling into CINT
    - Central lock before calling into X11
  - And
    - Store special ROOT globals as thread specific data

Thread Safety Implementation

- Introduction of TVirtualMutex and TLockGuard classes in libCore
- Introduction of two global mutexes:
  - gCINTMutex and gContainerMutex
- After loading of libThread they will point to real TMutex objects, 0 otherwise
- Mutexes placed with TLockGuard via zero-cost macro (when not compiled with thread support)