Exercise 1)

Use ACLiC to compile the script you used for the exercises in the previous lecture. How much acceleration do you get?
# Exercises for Lecture 8

```c
void doit(UInt_t rgNr = 1, Int_t nrEvents = 200000000)
{
    if (gRandom) delete gRandom;
    switch (rgNr) {
        case (2):
            gRandom = new TRandom2();
            break;
        case (3):
            gRandom = new TRandom3();
            break;
        default:
            gRandom = new TRandom();
            break;
    }

    TH1D * hist = new TH1D("hist","TRandom",500,-10,10);

    TStopwatch *st = new TStopwatch();
    st->Start();
    for (Int_t i=0; i<nrEvents; i++) hist->Fill(gRandom->Gaus(0,1));
    st->Stop();

    TF1* gs = new TF1("gs","gaus",-10,10);
    hist->Fit("gs");
    Double_t normchi2 = gs->GetChisquare()/gs->GetNDF();
    printf("%s : %.1fs %.2f mus/event %.4f
", gRandom->GetName(), st->CpuTime(), 1e6*st->CpuTime()/nrEvents, normchi2);
}
```

The original macro
#include "TRandom.h"
#include "TRandom2.h"
#include "TRandom3.h"
#include "TH1D.h"
#include "TStopwatch.h"
#include "TF1.h"

void doit(UInt_t rgNr = 1, Int_t nrEvents = 200000000)
{
    if (gRandom) delete gRandom;
    switch (rgNr) {
    case (2):
        gRandom = new TRandom2();
        break;
    case (3):
        gRandom = new TRandom3();
        break;
    default:
        gRandom = new TRandom();
        break;
    }

    TH1D * hist=new TH1D("hist","TRandom",500,-10,10);

    TStopwatch *st=new TStopwatch();

    st->Start();
    for (Int_t i=0; i<nrEvents; i++) hist->Fill(gRandom->Gaus(0,1));
    st->Stop();

    TF1* gs = new TF1("gs","gaus",-10,10);
    hist->Fit("gs");
    Double_t normchi2 = gs->GetChisquare()/gs->GetNDF();
    printf("%s : %.1fs %.2f mus/event %.4f
", gRandom->GetName(), st->CpuTime(), 1e6*st->CpuTime()/nrEvents, normchi2);
}
CINT-MACRO:

root[0] .L ex8_1.C
root[1] doit(1)
....

ACLiC:

root[0] .L ex8_1.C+
Info in <TUnixSystem::ACLiC>: creating shared library
   /Users/messchendorp/Documents/rootCourse/.le8_1_C.so
root[1] doit(1)
....
Exercises for Lecture 8

CINT-MACRO (.L ex8_1.C):
Random : 469.7s $2.35 \mu s/event$ 2.0890
Random2 : 543.7s $2.72 \mu s/event$ 0.9439
Random3 : 494.7s $2.47 \mu s/event$ 0.9839

ACLiC (.L ex8_1.C+):
Random : 147.6s $0.74 \mu s/event$ 2.0890
Random2 : 209.9s $1.05 \mu s/event$ 0.9439
Random3 : 152.5s $0.76 \mu s/event$ 0.9839

~2.5-3.3 faster!!

Full compilation (.ex8_1.exe):
Random : 140.3s $0.70 \mu s/event$ 2.0890
Random2 : 211.6s $1.06 \mu s/event$ 0.9439
Random3 : 147.1s $0.74 \mu s/event$ 0.9839

Similar speed as with ACLiC
Exercise 2)

Complete the script below. Determine the speed (μs/event) for each of the 5 methods to fill a histogram when you run it in CINT or ACLiC mode. Comment on the difference …

```c
Double_t mygaus(Double_t* c, Double_t* par)
{
  Double_t x = c[0];
  return par[0]*exp(-0.5*(x-par[1])*(x-par[1])/par[2]/par[2]);
}

void compare()
{
  TF1* gs1 = new TF1("gs1", "gaus", -10, 10); gs1->SetParameters(1,0,1);
  TF1* gs2 = new TF1("gs2", "mygaus", -10, 10, 3); gs2->SetParameters(1,0,1);

  hist1->FillRandom(g1,nEvents); // method 1
  hist2->FillRandom(g2,nEvents); // method 2
  for(Int_t i=0;i<nEvents;i++) hist3->Fill(gRandom->Gaus(0,1)); // method 3
  for(Int_t i=0;i<nEvents;i++) hist4->Fill(gs1->GetRandom()); // method 4
  for(Int_t i=0;i<nEvents;i++) hist5->Fill(gs2->GetRandom()); // method 5
}```
```c
#include "TH1D.h"
#include "TF1.h"
#include "TRandom3.h"
#include "TStopwatch.h"

Double_t mygaus(Double_t* c, Double_t *par)
{
    Double_t x=c[0];
    return par[0]*exp(-0.5*(x-par[1])*(x-par[1])/par[2]/par[2]);
}

void compare(Int_t nrEvents=1)
{
    if (gRandom) delete gRandom;
    gRandom=new TRandom3();

    TH1D *hist1=new TH1D("hist1","Method 1",500,-10,10);
    TH1D *hist2=new TH1D("hist2","Method 2",500,-10,10);
    TH1D *hist3=new TH1D("hist3","Method 3",500,-10,10);
    TH1D *hist4=new TH1D("hist4","Method 4",500,-10,10);
    TH1D *hist5=new TH1D("hist5","Method 5",500,-10,10);

    TF1* gs1 = new TF1("gs1","gaus",-10,10);   gs1->SetParameters(1,0,1);
    TF1* gs2 = new TF1("gs2",mygaus,-10,10,3); gs2->SetParameters(1,0,1);

    TStopwatch *st=new TStopwatch();
    st->Start(); hist1->FillRandom("gs1",nrEvents); st->Stop();
    printf("Method 1 : %.1fs %.2f mus/event
",st->CpuTime(), 1e6*st->CpuTime()/nrEvents);

    st->Start(); hist2->FillRandom("gs2",nrEvents); st->Stop();
    printf("Method 2 : %.1fs %.2f mus/event\n",st->CpuTime(), 1e6*st->CpuTime()/nrEvents);

    st->Start(); for (Int_t i=0; i<nrEvents; i++) hist3->Fill(gRandom->Gaus(0,1)); st->Stop();
    printf("Method 3 : %.1fs %.2f mus/event\n",st->CpuTime(), 1e6*st->CpuTime()/nrEvents);

    st->Start(); for (Int_t i=0; i<nrEvents; i++) hist4->Fill(gs1->GetRandom());st->Stop();
    printf("Method 4 : %.1fs %.2f mus/event\n",st->CpuTime(), 1e6*st->CpuTime()/nrEvents);

    st->Start(); for (Int_t i=0; i<nrEvents; i++) hist5->Fill(gs2->GetRandom());st->Stop();
    printf("Method 5 : %.1fs %.2f mus/event\n",st->CpuTime(), 1e6*st->CpuTime()/nrEvents);
}
```
## Exercises for Lecture 8

### CINT-MACRO (.L ex8_2.C):

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (µs/event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method 1 (FillRandom, “gaus”)</td>
<td>0.66</td>
</tr>
<tr>
<td>Method 2 (FillRandom, “mygaus”)</td>
<td>0.66</td>
</tr>
<tr>
<td>Method 3 (Loop, Fill, Gaus(0,1))</td>
<td>2.42</td>
</tr>
<tr>
<td>Method 4 (Loop, Fill, “gaus”)</td>
<td>1.91</td>
</tr>
<tr>
<td>Method 5 (Loop, Fill, “mygaus”)</td>
<td>1.89</td>
</tr>
</tbody>
</table>

### ACLiC (.L ex8_2.C+):

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (µs/event)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method 1 (FillRandom, “gaus”)</td>
<td>0.67</td>
</tr>
<tr>
<td>Method 2 (FillRandom, “mygaus”)</td>
<td>0.64</td>
</tr>
<tr>
<td>Method 3 (Loop, Fill, Gaus(0,1))</td>
<td>0.75</td>
</tr>
<tr>
<td>Method 4 (Loop, Fill, “gaus”)</td>
<td>0.69</td>
</tr>
<tr>
<td>Method 5 (Loop, Fill, “mygaus”)</td>
<td>0.67</td>
</tr>
</tbody>
</table>
Conclusions from Exercises 8

1) The use of pre-defined or user-defined functions does not matter in performance.

2) Using macros with for-loops in CINT cause a significant drop in performance. Avoid making a “lot of calls” in a CINT macro.

3) Compiled code with ACLiC gives similar performance in all cases, independent of using a “lot of calls” or not in your macro.

4) The performance of a CINT macro is similar to ACLiC code if you minimize the number of calls in the macro.
ROOT Lecture 9
Networking and Threads
Why networking in a ROOT analysis?

1) **Performance**
   Divide the analysis work/tasks among several computers.

2) **Monitoring and Control**
   Monitor results and change parameters on-the-fly.

3) **Flexibility**
   Run your certain parts of the analysis anywhere you like.
TCP/IP: Transmission Control Protocol/Internet Protocol. A standard network protocol which allows to setup a reliable full-duplex communication link between two or more computers.
Some TCP/IP examples in Unix

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>telnet</td>
<td>in.telnetd</td>
<td>Terminal emulation</td>
</tr>
<tr>
<td>ssh</td>
<td>sshd</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>ftp</td>
<td>in.ftpd</td>
<td>File Transport Protocol</td>
</tr>
<tr>
<td>http browsers</td>
<td>httpd</td>
<td>Hyper Text Transport Protocol (WWW)</td>
</tr>
<tr>
<td>TNetFile</td>
<td>rootd</td>
<td>ROOT file access via internet</td>
</tr>
</tbody>
</table>
Class **TNetFile** : public **TFile**;

```
TNetFile("root://kvir03.kvi.nl/rootcourse/myrootfile.root","...")
```

Server address running “rootd”

Some remarks:

1) “rootd” has to run on server: \$\{(ROOTSYS)/bin/rootd\}

2) Connecting to a “rootd” requires the remote user id and password.
Building your own client-server application

A TCP connection:

kvis67.kvi.nl – Network Port 51677

kvir03.kvi.nl – Network Port 9090

Communication link
Building your own client-server application

Port numbers: 16-bit "phone-number"

0-1023: well-known ports. Controlled by Internet Assigned Numbers Authority (IANA). Example: port 80 = Web server. Never use these numbers for your server!!

1024-49151: registered ports. Not controlled by IANA. Check before using a number in this range. Example: port 1094 = rootd.

49152-65535: dynamic or private ports. Not listed by IANA. Dynamical ports generated by client applications always use numbers in this range.

http://www.iana.org/assignments/port-numbers
A TCP connection is uniquely defined by a 4-tuple, i.e. addresses (IP-address+portnr) of both endpoints. These numbers are called a socket-pair.
Building your own client-server application

Setting up a TCP connection

Server “listen socket”: it waits for client connection

```
TServerSocket *listenSocket = new TServerSocket(9090)
TSocket *srvSocket = listenSocket->Accept()
/* waits for connection before returning “connection” socket */
```

Server
Local: kvir03.kvi.nl/9090
Remote: */*
Building your own client-server application

Setting up a TCP connection

Client rings the server

Server “listen socket”: it receives a client connection

Client
Local: kvis67.kvi.nl/ 51677
Remote: kvir03.kvi.nl/9090

Server
Local: kvir03.kvi.nl/9090
Remote: */*

```c
TSocket *clntSocket=new TSocket(“kvir03.kvi.nl”,9090)
/* returns socket if connection established */
```
Building your own client-server application

Setting up a TCP connection

Client
Local: kvis67.kvi.nl/ 51677
Remote: kvir03.kvi.nl/9090

Server
Local: kvir03.kvi.nl/9090
Remote: */*

TSocket *clntSocket = new TSocket(“kvir03.kvi.nl”,9090)

TSocket *srvSocket = listenSocket->Accept()

Local: kvir03.kvi.nl/9090
Remote: kvis67.kvi.nl/51677

Clint establishes connection: “connection” sockets created
Building your own client-server application

Communication between client and server
(byte-stream)

Client

str

Server

clntSocket->**Send** ("Hello server")

Char_t str[32];
srvSocket->**Recv**(str,32)

**Note:**
1) **Recv()** “blocks” till data available
2) **Send()** does not block by default
Building your own client-server application

Communication between client and server (byte-stream)

```
Char_t str[32];
clntSocket->Recv(str,32)
```

```
srvSocket->Send(“Hello client”)```

**TCP/IP is a fully duplex protocol**
Building your own client-server application

Closing connection between client and server

```c
Int_t nrBytes=srvSocket->Recv(str,32);
if (nrBytes==0) /* Client closes conn. */
{
    srvSocket->Close();
}
```
Sending ROOT-type messages via TCP

```cpp
class TMessage : public TBuffer ;

TMessage(UInt_t what=kMESS_ANY)

kMESS_ANY          generic message type
kMESS_STRING       string message type
kMESS_OBJECT       object message type
kMESS_CINT         cint command follows
kMESS_ZIP          compress message
kMESS_ACK          message has to be acknowledged before “Send(..)” returns
```

**Example:** (create message which will contain a compressed object)

```cpp
TMessage *m=new TMessage(kMESS_OBJECT|kMESS_ZIP)
```
Sending ROOT objects via TCP

The sender…**WriteObject**

```
TH1D *his = new TH1D(...);
...
TMessage mess(kMESS_OBJECT);
mess.WriteObject(his);
socket->Send(mess);
```

The receiver…**ReadObject**

```
TMessage *mess;
socket->Recv(mess);
TH1D *his = (TH1D *)mess->ReadObject(mess->GetClass());
...
```
Only one client can connect to server !!!
Create detached process which handles communication with client in parallel: TThread
Threads

UNIX:

[messchendorp@KVIS67 messchendorp] xterm &
[1] 23238
[messchendorp@KVIS67 messchendorp] ps ax | grep xterm
23241 std S  0:00.17 xterm

ROOT:

class TThread : public TNamed ;

TThread( const char name, void *process_name, void *process_arguments=0,
    Epriority priority=kNormalPriority )
void runServer() {
    TServerSocket *listenSocket = new TServerSocket(...);
    while (1) {
        TSocket *sock = listenSocket->Accept();
        TThread *thread = new TThread("clntThread", handleRequest, sock);
        thread->Run(); /* Runs thread as detached process */
        TThread::Ps(); /* Similar to Unix “ps” */
    }
}

void *handleRequest(void *arg) {
    TSocket *sock = (TSocket *) arg;
    Int_t nrBytes=sock->Recv(...);
    ...
    sock->Close();
    (TThread::Self())->Delete();
}
1) **Parallelism and Performance**
   Threads could increase performance due to parallel processing. Not only on multi CPU computers!

2) **Sharing process resources**
   Threads share the same resources (globally defined variables), which allows an easy “communication” between the different threads.
1) **Sharing process resources**

Threads share the same resources, which might lead to *race* problems!

A thread producer could modify shared data at the same time a thread consumer reads it! The read data array might not be complete or corrupted…
Threads and Threats

Solution...put a lock on it! 

```
TThread::Lock() and TThread::UnLock()
```

```c++
void FillArray() {
    TThread::Lock();
    /* manipulate array */
    TThread::UnLock();
}

void ReadArray() {
    TThread::Lock();
    /* read array */
    TThread::UnLock();
}
```

Rare cases for which a lock doesn’t solve the problem
Threads and Threats

1) **Sharing process resources**
   Threads share the same resources, which might lead to *race* problems!

2) **ROOT CINT**
   Threads do not work with CINT! So use ACLiC or compile your code.

3) **Thread-compatible ROOT**
   Your ROOT version might not been build with thread support. If so, recompile!
Exercise for Lecture 9

Write a fully-functional client-server program. The functionality of the server and a client are given in the next slides. A template of the server and client code can be found on http://kvir03.kvi.nl/rootcourse
A client-server program

**Main Server:**
1) Starts DAQ Thread
2) Wait for Client connections and create Client Thread

**The Server**

**DAQ Thread:**
Fills continuously a 1D histogram with random numbers

**Client Thread:**
On request sends 1D histogram to client
A client-server program

```cpp
TH1D his("his","",5000,-4,4);

void exampleServer(UInt_t portNumber=9090) {
  TThread *daqThread=new TThread("daqThread", handleDAQ);
  daqThread->Run();

  TServerSocket *listenSocket = new TServerSocket(portNumber);

  while (1) {
    cout << "Waiting for connection..." << endl;
    TSocket *srvSocket = listenSocket->Accept();

    TThread *clntThread = new TThread("clntThread",
          handleClnt, (void *) srvSocket);
    clntThread->Run();
  }
}

void handleDAQ(void *arg) {
  {...}
}

void handleClnt(void *arg) {
  {...}
}
```
Client’s functionalities:

**openConnection** (serverName, portNumber):
- Connects to server (server creates Client Thread)

**closeConnection** ():
- Closes communication with server (server terminates Client Thread)

**drawHisto** ():
- Obtain the histogram from server (server sends TH1 object to client)
- Plot it
A client-server program

Create global objects

Connect to server

Disconnect from server by closing socket

Draw histogram via TCPIP

The Client

```cpp
TSocket *sock=NULL;

void openConnection(Char_t *host="localhost",
                    UInt_t portnumber=9090)
{
    sock = new TSocket(host, portnumber);
}

void closeConnection()
{
    sock->Close();
}

void drawHisto()
{
    ....
}```