Multithreading - 2

I. Semeniouk
LLR, CNRS – Ecole Polytechnique

Credits:
I. Hrivnacova(IJCLab), A. Dotti, M. Asai (SLAC)

Geant4 Tutorial,
23 - 27 November 2020, Orsay

Outline

- What is thread-safety
- Geant4 MT utilities

Thread Safety

Thread Safety (1)

- Consider a function that reads and writes a shared resource (a global variable in this example).

```
double sharedVariable;

int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

Thread Safety (2)

- Now consider two threads that execute the function at the same time. Concurrent access to the shared resource

```
double sharedVariable;
```

```
int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

```
int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

Thread Safety (3)

- `result` is a local variable, exists in each thread separately not a problem
- T1 starts, arrives here and then is halted to the shared resource

```
double sharedVariable;

int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

```
int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

Thread Safety (4)

- Now T2 starts and arrives here, the shared resource value is not yet updated, what is the expected behavior? What is happening?

```
double sharedVariable;
```

```
int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

```
int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

Thread Safety (5)

- Use mutex / locks to create a barrier. T2 will not start until T1 reaches UnLock
- However mutex significantly reduces performances (general rule in Geant4: not allowed in methods called during the event loop)

```
double sharedVariable;
```

```
int doSomeFunction() {
  int result = 0;
  Lock(&mutex);
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  UnLock(&mutex);
  return result;
}
```

```
int doSomeFunction() {
  int result = 0;
  Lock(&mutex);
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  UnLock(&mutex);
  return result;
}
```

Thread Safety (6)

- Do we really need to share sharedVariable?
- If not, declare it “thread local”, each thread then has its own copy
- Simple way to “transform” your code, but very small CPU penalty, no memory usage reduction
- General rule in Geant4: do not use unless really necessary!

```
double G4ThreadLocal
sharedVariable;

int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

```
double G4ThreadLocal
sharedVariable;

int doSomeFunction() {
  int result = 0;
  if ( sharedVariable > 0 ) {
    result = sharedVariable;
    sharedVariable = -1;
  }
  else {
    doSomethingElse();
    sharedVariable = 1;
  }
  return result;
}
```

Geant4 MT Utilities

Geant4 MT Types

- To hide platform dependent and POSIX definitions, there are introduced Geant4 type definitions (typedef) for MT related types & definitions
- Instead of using __thread keyword, use G4ThreadLocal, eg.

```
static G4ThreadLocal G4double value;
```

Setting the Number of Threads

- Default: the number of threads = 2
- Use /run/numberOfThreads or G4MTRunManager::SetNumberOfThreads() to change this default value
  - If you want to exploit fully your machine you can set the number of all logical cores of your machine using G4Threading::G4GetNumberOfCores()
- You can overwrite the setting in your application via setting the environment variable G4FORCENUMBEROFTHREADS
  - Must be done before starting the application
  - The special keyword MAX can be used to use all system cores
- The number of threads cannot be changed after run has been initialized

Tuning the Output

- When running an application in MT mode the output from workers is interlaced with the output from master and is preceded with the prefix string G4WTn >
  - Where n is thread Id (0, 1,2, ...)

```
G4WT1 > ### Run 0 start.
G4WT0 > ### Run 0 start.
G4WT1 > ... open Root analysis file : ED_t1.root - done
G4WT0 > ... open Root analysis file : ED_t0.root - done
G4WT0 > >>> Start event: 1
G4WT1 > >>> Start event: 0
G4WT1 >
--------->Chamber1HitsCollection: in this event:
G4WT1 > Chamber hit in layer: 0   time [s]: 1.37346e-08   position
[mm]: (80.6632,45.2255,-6000.1)
G4WT1 > Chamber hit in layer: 1   time [s]: 1.60253e-08   position
[mm]: (95.0864,52.1524,-5500.1)
G4WT1 > Chamber hit in layer: 2   time [s]: 1.83168e-08   position
[mm]: (109.993,60.1137,-5000.1)
....
```

Tuning the Output (2)

- This default behavior can be changed using the commands
- /control/cout/setCoutFile [filename]
  - Send G4cout stream to a per-thread file.
  - Use “***Screen***” to reset to screen
  - Analogous command is available for G4cerr
- /control/cout/useBuffer [true|false]
  - Send G4cout/G4cerr to a per-thread buffer that will be printed at the end of the job
- /control/cout/prefixString [string]
  - Add an per-thread identifier to each output line from threads, the thread id is appended to this prefix (default: G4WTn)
- /control/cout/ignoreThreadsExcept [id]
  - Show output only from thread “id”

Lock Mechanism

- To add a lock mechanism (remember: will spoil performances but may be needed with non thread-safe code):

```
#include “G4AutoLock.hh”

namespace {
   G4Mutex myMutex = G4MUTEX_INITIALIZER;
}

void myFunction() {
  // enter critical section
  G4AutoLock lock(&myMutex);
     //will automatically unlock when
     //out of scope
  return;
}
```

Other MT Utilities

- Few classes/utilities have been created to help handling of objects.
  - G4Cache : Allows to create a thread-local variable in shared class
  - G4ThreadLocalSingleton : for thread-private “singleton” pattern
  - G4AutoDelete : automatically delete thread objects at the end of the job
- See more details in Chapter 2.14 of Users’s Guide For Toolkit Developers

Conclusions

- Parallelism is a tricky business:
  - User code has to be thread-safe
  - Race conditions may appear (better: they will very probably appear)
- Locking mechanism and other utilities are provided with Geant4 to make migration to multithreading easier
- Experience is needed for complex applications
  - Bugs may often seem “random” and difficult to reproduce
  - A new hyper news user forum has been created (category Multithreading) to address all possible questions
- Ask an expert!