```
// ==================== EventAction.cc Class ===================================
//
// Developed by Bryan V. Egner, Darren E. Holland, and Julie V. Logan
// Modified by Darren Holland 2020-11-02
// ============================================================================
//
// This file track the total energy deposited during an event and outputs the
// result to a thread specific file (to avoid race conditions)
// ============================================================================
//
#include "B4aEventAction.hh"
#include "B4RunAction.hh"
#include "G4Event.hh"
//
#include "G4Threading.hh"
#include "G4RunManager.hh"
#include "G4Event.hh"
#include "G4Run.hh"
#include "G4SDManager.hh"
#include "G4HCofThisEvent.hh"
#include "G4UnitsTable.hh"
#include "G4SystemOfUnits.hh"
//
#include "Randomize.hh"
#include <iomanip>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <math.h>
//
#include <Settings.hh>
#include <mutex>
#include <thread>
#include "G4AutoLock.hh"
#include <string>
#include <chrono>
#include <boost/filesystem.hpp>

using namespace std;
namespace {
G4Mutex aMutex=G4MUTEX_INITIALIZER;}

// On creation, declare and define variables to track energy deposited
B4aEventAction::B4aEventAction(B4RunAction* runAction)
 : G4UserEventAction()
{}

// When finished, destroy the instance
B4aEventAction::~B4aEventAction()
{}

// Used to get thread file names (denoted by extension ".thread")
void B4aEventAction::GetFilesOfTypeInDirectory(const boost::filesystem::path
&directoryPath, const string &fileExtension,
std::vector<boost::filesystem::path> &list)
{
    if(!boost::filesystem::exists(directoryPath) || !
boost::filesystem::is_directory(directoryPath))
      {
      std::cerr << directoryPath << "is not a directory." << std::endl;
      return;
      }
    boost::filesystem::recursive_directory_iterator it(directoryPath);
    boost::filesystem::recursive_directory_iterator endit;
    while(it != endit)
      {
      if(boost::filesystem::is_regular_file(*it) && it->path().extension() ==
fileExtension) list.push_back(it->path());
      ++it;
      }
}

// At the start of a new local run (aka new particle), reset the total deposited
energy
void B4aEventAction::BeginOfEventAction(const G4Event* event)
{
    // Get thread number
    thread_local int Threadindex=-1;
    // If thread file exists (aka not first event+step) then change index number
    thread_local thread::id threadid = std::this_thread::get_id();
    thread_local stringstream tstr;
    tstr << threadid;
    thread_local string tstr_str = tstr.str();
    thread_local string tstr_str_cmp = "./" + tstr.str() + ".thread";
    // Match current thread to index (thread 1, 2, 3, etc)
    for (thread_local int ii = 0; ii < ThreadNum.size(); ii++ ) {
        if ( tstr_str_cmp.compare(ThreadNum[ii].string()) == 0 ) {
            Threadindex = ii;
            }
    }
    // Create file with event information
    // First time thread is used, create a thread file
    if ( Threadindex == -1 ) {
        ofstream ofile;
        // Create file named ThreadName.thread
        ofile.open ("./" + tstr_str + ".thread", fstream::out | fstream::out);
// ascii file
        ofile.close();
        // Pause to make sure all other thread files are created
        std::this_thread::sleep_for(std::chrono::seconds(1));
        GetFilesOfTypeInDirectory(".",".thread",ThreadNum);
        // Get index corresponding to current thread number
        for (thread_local int ii = 0; ii < ThreadNum.size(); ii++ ) {
            if ( tstr_str_cmp.compare(ThreadNum[ii].string()) == 0 ) {
            Threadindex = ii;
            }
        }
    }
    // Initialize thread's total deposited energy for this event
    eventTotEdepDetector[Threadindex] = 0.;
    // Get run and event numbers (to double check data is written correctly)
    runNum[Threadindex] = G4RunManager::GetRunManager()->GetCurrentRun()-
>GetRunID();
    Evt[Threadindex] = G4RunManager::GetRunManager()->GetCurrentEvent()-
>GetEventID();
}

// Function for locking a thread's output file until it is done writing.
// This should avoid output race conditions
void B4aEventAction::writetofile(G4String fname, const G4double IPE, const
G4double Edep, G4int runNum, G4int eid )
{
    // Create output stream and write to file
    G4AutoLock l(&aMutex);
    l.lock();
    ofstream ofile;
    // Write results to run number (aka measurement angle)
    ofile.open ("./" + to_string(runNum) + "/" + fname, fstream::out |
fstream::app);     // ascii file
    ofile.flush();
    ofile << fixed << setprecision(6);
    // Write the initial energy, total deposited energy by that particle, run
number (should be the
    // same in each file if no race occured) and event id (monotonically
increasing for single thread)
    ofile << IPE << "\t" << "\t" << Edep << "\t" << "\t" << runNum << "\t" << "\
t" << eid << "\n";
    ofile.close();
}

// At the end of the event, write output event data (local or cluster)
void B4aEventAction::EndOfEventAction(const G4Event* event)
{
    // Set index to -1 if not thread file not created (should never be true
since occurs after BeginOfEventAction)
    thread_local int Threadindex=-1;
    thread_local thread::id threadid = std::this_thread::get_id();
    thread_local stringstream tstr;
    tstr << threadid;
    thread_local string tstr_str = tstr.str();
    thread_local string tstr_str_cmp = "./" + tstr.str() + ".thread";
    // Get thread index number
    for (thread_local int ii = 0; ii < ThreadNum.size(); ii++ ) {
        if ( tstr_str_cmp.compare(ThreadNum[ii].string()) == 0 ) {
            Threadindex = ii;
        }
    }
    // Only write output if energy was deposited
    if(eventTotEdepDetector[Threadindex] != 0) {
        // Get initial (source) energy
        static G4ThreadLocal G4double Initial_Particle_Energy = event-
>GetPrimaryVertex()->GetPrimary()->GetKineticEnergy();
        G4ThreadLocal G4int eID = G4RunManager::GetRunManager()-
>GetCurrentEvent()->GetEventID();
        // Create output filename
        static G4ThreadLocal G4String fname_out_final= Settings::fname_out + "_"
+ tstr_str + ".ww";
        // Write the output to thread file

writetofile(fname_out_final,Initial_Particle_Energy,eventTotEdepDetector[Threadi
ndex],runNum[Threadindex],eID);
    }
}
```