Imię i nazwisko:..... **H2** czas: 60 min

Przyjąć, że udostępniona jest przestrzeń nazw std i boost

W rozwiązaniach można używać dowolnych udogodnień z biblioteki standardowej C++ oraz bibliotek Boost.

Zadanie 1 (3pkt)

Zaproponuj poprawę wydajności aplikacji poniżej. Kodu poza ramką proszę nie poprawiać, nie jest istotny.

```
using Data = int; //symuluje dane do óobrbki
static const Data END_DATA = 0;
static const Data VALID_DATA = 1;
int main() {
   DataSynch data;
   data.data_ = 1;
   Writer w1(data), w2(data);
   Reader r1(data), r2(data), r3(data), r4(data);
   thread th1( ref(w1) ), th2( ref(w2) );
   thread th3( ref(r1) ), th4( ref(r2) ), th5( ref(r3) ), th6( ref(r4) );
   th1.join(); th2.join(); th3.join(); th4.join(); th5.join(); th6.join();
   return 0;
}
```

```
struct DataSynch { //dane do obrobki w roznych watkach
   Data data_;
   mutex read_;
   mutex write_;
};

struct Reader {
   Reader(DataSynch& data) : data_(data) {}

   void operator()() {
     bool finish = false;
     while(!finish) {
       lock_guard<mutex> lock(data_.read_);
       //tylko czyta wspoldzielone dane
       finish = (data_.data_ == END_DATA);
     }
   }
   DataSynch& data_;
};

struct Writer {
   Writer(DataSynch& data) : data_(data) {}

   void operator()() {
      static const int NUM = 10; //symuluje wielkosc danych do obrobki
      for(int i=0;i<NUM;++i) {
        lock_guard<mutex> lock1(data_.read_);
        lock_guard<mutex> lock2(data_.write_);
        //czyta i pisze wspoldzielone dane
        data_.data_ = VALID_DATA; //symuluje modyfikacje danych
      }
      //koniec zapisu, wstawiana dana 'pusta'
      {
         lock_guard<mutex> lock1(data_.read_);
         lock_guard<mutex> lock2(data_.write_);
         data_.data_ = END_DATA;
      }
   }
   DataSynch& data_;
};
```

Pytanie 1 (0.5pkt)

Jakie treści proponujesz dodać do ZPR w przyszłych edycjach?

Pytanie 2 (0.5pkt)

Ile godzin poświęciłeś na wykonanie projektu z ZPR?

Ile godzin poświęciłeś na pozostałe elementy przedmiotu (wykłady, kolokwia, zadanie dodatkowe, itd.)?

Zadanie 2 (1pkt)

Podaj wydruk. Liczba liter Twojego nazwiska (stała NAME_SIZE jest użyta w zadaniu):

```
const int NAME_SIZE = [          ];

using PF = std::function<int (int, int)>;
vector<PF> vpf;
vpf.push_back([](int x, int y){ return x + y; });
vpf.push_back([](int x, int y){ return 2*x + y; });
vpf.push_back([](int x, int y){ return x + 2*y; });
vpf.push_back([](int x, int y){ return 2*x + 2*y; });

cout << vpf[0](NAME_SIZE, 1) << endl;
```

Zadanie 3 (2pkt)

Jakie korzyści daje użycie alokatorów dla małych obiektów, którego przykładami są `std::pmr::unsynchronized_pool_resource`, `std::pmr::synchronized_pool_resource`?

- Czy warto stosować taki alokator, gdy przetwarzamy graf z 10^9 wierzchołkami, a każdy wierzchołek jest obiektem o wielkości 16B, przechowywanym na stercie? Odpowiedz tak lub nie.
- Oszacuj (zgrubnie) zużycie pamięci w powyższym przykładzie, dla alokatora ogólnego przeznaczenia, który wymaga 48B na przydzielony blok. Podaj szacunek w MB.

Zadanie 4 (1pkt)

Podaj wydruk.

```
using Graph = boost::adjacency_list<boost::vecS, boost::vecS, boost::bidirectionalS>;
using Edge = pair<int, int>;
   enum { A, B, C, D, E, F, NUM_VERTICES };
const vector<Edge> EDGES =
   { Edge(A,B), Edge(B,A), Edge(C,A), Edge(D,A), Edge(E,F), Edge(F,E) };
Graph g(EDGES.begin(), EDGES.end(), NUM_VERTICES);
for_each( vertices(g).first, vertices(g).second,
        [&](Graph::vertex_descriptor v){
           for_each(out_edges(v, g).first, out_edges(v, g).second,
                  [&](Graph::edge_descriptor e){ cout << target(e, g) << '␣';});
        });
        cout << endl;
```

Uwagi do prowadzącego (R.Nowak):

Zadanie 5 (2pkt)

Spróbuj naprawić poniższy kod. Klient zamieszcza następujące wycinki z ¨logów¨:

```
// id: 4 data: 2
// id: 1276891168 data: 32518
// id: 7 data: -1
// id: 10 data: 0
// id: 3 data: 2
// id: 7 data: 1
// id: 4 data: 1
// id: 1 data: 1
// id: 3 data: 1
//
// done
// id: 3 data: 0
// a.out: kol2.cpp:18: void Task::update(): Assertion 'data.load() > 3' failed.
//
// id: -1331408640 data: 32717
// terminate called after throwing an instance of 'std::out_of_range'
// what(): map::at
// [1] 18826 IOT instruction (core dumped) ./a.out
```

```
const int TASK_DONE = 3;
struct Task {
   int id;
   std::atomic<int> data;
   bool done;
   Task(int id) : id(id), data(0), done(false) {}
   void update() {
      if (done) {
         assert(data.load() > TASK_DONE);
         std::cout << "done" << std::endl;
      } else {
         std::cout << "id: " << id << " data: " << data.load() << std::endl;
      }
   }
};
class TaskMap {
public:
   std::shared_ptr<Task> get_or_insert(int id) {
      std::scoped_lock guard(m_);
      auto task = std::shared_ptr<Task>(new Task(id));
      return task_map.try_emplace(id, task).first->second;
   }
   std::shared_ptr<Task> pop(int id) {
      std::scoped_lock guard(m_);
      auto task = task_map.at(id);
      task_map.erase(id);
      return task;
   }
private:
   std::mutex m_;
   std::map<int, std::shared_ptr<Task>> task_map;
};
void updater(TaskMap& task_map) {
   for (int i = 0; i < 1000; ++i) {
      // get ID from client
      int id = rand() % 10 + 1;
      auto& task = *task_map.get_or_insert(id);
      // do some work
      int update = rand() % 4 - 1;
      task.data.fetch_add(update);
      if (task.data.load() > TASK_DONE) {
         task.done = true;
         task_map.pop(id);
      }
      task.update();
   }
}
int main() {
   TaskMap task_map;
   auto updater_ = [&task_map]{
      updater(task_map);
   };
   std::thread thrd1(updater_); std::thread thrd2(updater_); std::thread thrd3(updater_);
   thrd1.join(); thrd2.join(); thrd3.join();
   return 0;
}
```

Uwagi do prowadzącego (Ł. Neumann):