Home / My courses / PDP-IE / Exam / Exam 2021-01-23

Question **1**

Not yet answered

Marked out of 1.00

I hereby agree that I have not, and will not, receive any help from anyone for all the duration of the exam, with respect to the subject of the exam.

(you should answer True to take the exam)

Select one:

○ True

○ False

Question **2**

Not yet answered

Marked out of 3.00

Consider the following code for implementing a future mechanism (the *set()* function is guaranteed to be called exactly once by the user code):

```
template<typename T>
class Future {
    T val;
    bool hasValue;
    mutex mtx;
    condition_variable cv;
public:
    Future() :hasValue(false) {}
    void set(T v) {
        cv.notify_all();                // statement 1
        unique_lock<mutex> lck(mtx);    // statement 2
        hasValue = true;                // statement 3
        val = v;                        // statement 4
    }
    T get() {
        unique_lock<mutex> lck(mtx);
        while(!hasValue) {
            cv.wait(lck);               // statement 5
        }
        return value;
    }
};
```

or Java equivalent

```
class Future<T> {
    T val;
    boolean hasValue = false;
    final Lock mtx = new ReentrantLock();
    final Condition cv = mtx.newCondition();

    public void set(T v) throws InterruptedException {
        cv.signalAll();        // statement 1
        mtx.lock();         // statement 2
        hasValue = true;    // statement 3
        val = v;            // statement 4
        mtx.unlock();
    }
    public T get() throws InterruptedException {
        mtx.lock();
        while(!hasValue) {
            cv.await();     // statement 5
        }
        T ret = value;
        mtx.unlock();
        return ret;
    }
};
```

What kind of concurrency issue does it present? How to fix them?

Select one or more:

☐

- ☐ [fix] a possible fix is to put in the order *statement 2*, *statement 4*, *statement 3*, and *statement* 1 those four lines
- ☐ [issue] a call to *get()* can return an uninitialized value if simultaneous with the call to *set()*
- ☐ [issue] simultaneous calls to *get()* and *set()* can make future calls to *get()* deadlock
- ☐ [issue] a call to *get()* can deadlock if called before *set()*
- ☐ [fix] a possible fix is to unlock the mutex just before line marked *statement 5* and to lock it back just afterwards
- ☐ [fix] a possible fix is to remove the line marked *statement 2* (for Java: together with its corresponding *mtx.unlock()*)
- ☐ [issue] a call to *get()* can deadlock if simultaneous with the call to *set()*
- ☐ [issue] a call to *get()* can deadlock if called after *set()*
- ☐ [fix] a possible fix is to interchange lines marked *statement 3* and *statement 4*
- ☐ [fix] a possible fix is to interchange lines marked *statement 1* and *statement 2*

Question **3**

Not yet answered

Marked out of 3.00

Consider the following code for computing the product of two matrices (assuming the number of columns of a is equal to the number of rows of b).

```
void computeOneElement(
    std::vector<std::vector<int> > const& a,
    std::vector<std::vector<int> > const& b,
    size_t row, size_t col,
    std::vector<std::vector<int> >& rez,
    std::mutex& mtx)
{
    mtx.lock();
    int sum = 0;
    for(size_t i=0 ; i<b.size() ; ++i) {
        sum += a[row][i]*b[i][col];
    }
    rez[row][col] = sum;
    mtx.unlock();
}

std::vector<std::vector<int> > matrixProd(
    std::vector<std::vector<int> > const& a,
    std::vector<std::vector<int> > const& b,
    size_t nrThreads)
{
    std::mutex mtx;
    size_t outNrRows = a.size();
    size_t outNrCols = b[0].size();
    std::vector<std::vector<int> > rez(outNrRows);
    for(std::vector<int>& row : rez) {
        row.resize(outNrCols);
    }
    size_t begin = 0;
    size_t step = (outNrRows+nrThreads-1)/nrThreads; // statement 1
    std::vector<std::thread> threads;
    threads.reserve(nrThreads);
    for(size_t th=0 ; th<nrThreads ; ++th) {
        size_t end = begin+step; // statement 2
        threads.emplace_back([begin,end,outNrCols,&a,&b,&rez,&mtx]() {
            for(size_t i=begin ; i<end ; ++i) {
                for(size_t j=0 ; j<outNrCols ; ++j) {
                    computeOneElement(a, b, i, j, rez, mtx);
                }
            }
        });
        begin = end;
    }
    for(std::thread& th : threads) {
        th.join();
    }
    return rez;
}
```

or java equivalent

```
void computeOneElement(int[][] a, int[][] b,
    int row, int col,
    int[][] rez,
    RecursiveMutex mtx)
{
    mtx.lock();
    int sum = 0;
    for(int i=0 ; i<b.length ; ++i) {
        sum += a[row][i]*b[i][col];
    }
    rez[row][col] = sum;
    mtx.unlock();
}

int[][] matrixProd(int[][] a, int[][] b, int nrThreads)
{
    final RecursiveMutex mtx = new RecursiveMutex();
    final int outNrRows = a.length;
    final int outNrCols = b.get(0).length;
    int[][] rez = new int[][outNrRows];
    for(int i=0 ; i<rez.length : ++i) {
        rez[i] = new int[outNrCols];
    }
    int begin = 0;
    final int step = (outNrRows+nrThreads-1)/nrThreads; // statement 1
    Thread[] threads(nrThreads);
    for(int th=0 ; th<nrThreads ; ++th) {
        final int begin1 = begin;
        int end = begin+step; // statement 2
        final int end1 = end;
        threads[th] = new Thread(() -> {
            for(int i=begin1 ; i<end1 ; ++i) {
                for(int j=0 ; j<outNrCols ; ++j) {
                    computeOneElement(a, b, i, j, rez, mtx);
                }
            }
        });
        begin = end;
    }
    for(Thread th : threads) {
        th.join();
    }
    return rez;
}
```

Identify the issues with this code and how to fix them (fixes marked fix-A are to be considered only as far as the issues marked issue-A are concerned, and similarly for B)

Select one or more:

- ☐ [issue-B] There are elements of the output matrix that are computed twice
- ☐ [fix-B] In *statement 1* put *step = outNrRows/nrThreads*
- ☐ [issue-B] There are elements of the output matrix that are not computed
- ☐ [issue-A] The result may be incorrect because of race conditions between the concurrent threads
- ☐ [fix-A] Make the output matrix *std::vector<std::vector<std::atomic<int>>>*
- ☐

[issue-B] The program will attempt to access non-existent elements

- ☐ [fix-B] After *statement 2* add *if(end>a.size()) end=a.size()*
- ☐ [fix-A] Remove the mutex mtx and all references to it
- ☐ [fix-B] In *statement 1* put *step = outNrRows/nrThreads* and after *statement 2* add *if(end>a.size()) end=a.size()*
- ☐ [issue-A] There is essentially no parallelism because no two threads can access the matrices at the same time

Question **4**

Not yet answered

Marked out of 3.00

We want a distributed program that computes the sum of two matrices of the same size.
You shall implement two functions (in C++, Java or C\#):

vector<vector<int> > primes(vector<vector<int> > const& a, vector<vector<int> > const& b, int nrProcs);

that will run on the process 0 of the *MPI_COMM_WORLD* communicator, where *a* and *b* are the input matrices (assumed of equal length) and *nrProcs* is the
number of processes in the world communicator.

void worker(int myId, int nrProcs);

will run on all other processesin the world communicator, with *myId* representing the rank and *nrProcs* the number of processes.

Maximum size for new files: 2MB

Files

You can drag and drop files here to add them.

Accepted file types

Image (JPEG) .jpe .jpeg .jpg
Image (PNG) .png
PDF document .pdf
Text file .txt

◄ Sample for the exam

Jump to...