Homework 3 Wet

Due Date: 7/6/18 23:30

Teaching assistant in charge:

- Qasem Sayah

Important: the Q&A for the exercise will take place at a public forum Piazza only. Critical updates about the HW will be published in pinned notes in the piazza forum. These notes are mandatory and it is your responsibility to be updated. A number of guidelines to use the forum:

- Read previous Q&A carefully before asking the question; repeated questions will probably go without answers
- Be polite, remember that course staff does this as a service for the students
- You're not allowed to post any kind of solution and/or source code in the forum as a hint for other students; In case you feel that you have to discuss such a matter, please come to the reception hour
- When posting questions regarding hw3, put them in the hw3 folder

Only the TA in charge, can authorize postponements. In case you need a postponement, contact him directly at 234123cs@gmail.com .

Synchronizing the Factory

Introduction

In this assignment we will implement a class named Factory, there are 3 kinds of visitors that can get the products produced in this factory:

(1) Simple Buyer – the simple buyer is interested in buying only one product and is always in a hurry so if he can't buy a product at this very moment he leaves.
(2) Company – the company must buy N oldest available products, so it will wait until it can get N products together, after getting the products the company will wait until it can return the undervalued products.
(3) Thief – the thief steals from oldest produced products as much as he can up to N, before leaving the factory he leaves his fake id as a mark.

this factory should provide multiple functionalities that we expect from a real factory:

(1) Producing new products
(2) Sell one product to Simple Buyer
(3) Sell multiple products to Company
(4) Get stolen by Thief
(5) Accept returned products from Company
(6) Open/Close the factory for visitors
(7) Open/Close products returning service

In this assignment we will implement 4 kinds of threads that behave differently from one another, these threads access the factory in synchronized way but in different levels of priorities:

(1) Thief thread – the most prior.
(2) Company thread – the second prior.
(3) Simple buyer thread – the least prior.
(4) Production Thread – not specified, he accesses in a regular synchronized way.

* there's no order between the same kind of threads.
* for example if we created a Thief thread and a Company thread at the same time, it should be guaranteed that they will access the factory in the above order.

Read the whole assignment before you start coding, Reread the relevant introduction parts when reading start* methods.

Implementation Requirements

Implement the following Factory methods as defined in Factory.h:

`Factory();`

Constructor, the constructed factory and it's returning service are opened by default.

`~Factory();`

Destructor.

`void startProduction(int num_products, Product* products, unsigned int id);`

Starts a new production thread with the given id, which should call the following member method with the given parameters:

- `void produce(int num_products, Product* products);`

Parameters:

- num_products > 0 and is the number of elements in products
- products is a valid Product array with num_products elements (see Product.h)
- id is a new identifier of a Production thread

`void produce(int num_products, Product* products);`

adds the given new produced products to the factory.

Parameters:

- num_products > 0 and is the number of elements in products
- products is a valid Product array with num_products elements

`void finishProduction(int id);`

Finishes the production thread with the given id.

Parameters:

- id is a valid identifier of a Production thread

`void startSimpleBuyer(unsigned int id);`

Starts a new Simple buyer thread with the given id, which should call the following member methods:

- `int tryBuyOne();`

Parameters:

- id is a new identifier of a Simple buyer thread

```
int tryBuyOne();
```

tries to buy the oldest produced product from the available products in the factory, the operation succeeds if there's at least one available product and there's no other thread working on the factory at the moment.

Parameters:

- None

Return value:

- the id of the bought product if the operation succeeded
- -1 if the operation failed

```
int finishSimpleBuyer(unsigned int id);
```

Finishes the Simple buyer thread with the given id.

Parameters:

- id is a valid identifier of a Simple buyer thread

Return value:

- the returned value of `tryBuyOne()`

```
void startCompanyBuyer(int num_products, int min_value, unsigned int
id);
```

Starts a new Company thread with the given id, which should call the following member methods with the relevant parameters:

- `std::list<Product> buyProducts(int num_products);`
- `void returnProducts(std::list<Product> products,unsigned int id);`

Parameters:

- num_products > 0 and is the number of products the thread company have to buy
- min_value is the minimum value for accepting a product, all products with less value than min_value will be returned to the factory.
- id is a new identifier of a Company thread

```
std::list<Product> buyProducts(int num_products);
```

the company buys the oldest num_products products from the factory, if it can't access it will wait until it can buy num_products together.

Parameters:

- num_products > 0 and is the number of products to buy

Return value:

- list of the bought products from oldest produced to newest

```
void returnProducts(std::list<Product> products, unsigned int id);
```

the company with the given id returns the given list of products to the factory (as the order of the list and together as the same operation), the factory considers the returned products as a newly produced products.

Parameters:

- products is the list of products to return
- id is the identifier of the returning company

```
int finishCompanyBuyer(unsigned int id);
```

Finishes the Company thread with the given id.

Parameters:

- id is a valid identifier of a Company thread

Return value:

- the number of products the company returned

```
void startThief(int num_products, unsigned int fake_id);
```

Starts a new Thief thread with the given fake_id, which should call the following member methods with the given parameters:

- `int stealProducts(int num_products,unsigned int fake_id);`

Parameters:

- num_products > 0 and is the number of products to steal
- fake_id is a new identifier of a Thief thread

```
int stealProducts(int num_products,unsigned int fake_id);
```

the thief steals up to num_products from oldest products as much as possible, if he can't access the factory he will wait until he can enter to steal something (or nothing if he enters and the factory is empty), the factory will register about all stolen products by this thief attached with his fake_id.

Parameters:

- num_products > 0 is the maximum number of products to steal

Return value:

- the actual number of stolen products

```
int finishThief(unsigned int fake_id);
```

Finishes the thief thread with the given fake_id.

Parameters:

- fake_id is a valid identifier of a thief thread

Return value:

- the number of products stolen by this thief

```
void closeFactory();
```

closes the factory for visitors, only Producer thread can access the factory, other threads wait/leave as defined earlier for each kind.
* assume that this method will be called only after finishing all threads, but new threads can be created after it and before `openFactory()`.

```
void openFactory();
```

opens the factory for visitors.

```
void closeReturningService();
```

closes the returning service. In other words Company threads can't return products and have to wait for the returning service to be opened again.
* assume that this method will be called only after finishing all Company threads, but new threads can be created after it and before `openReturningService()`.

```
void openReturningService();
```

opens the returning service.

```
std::list<std::pair<Product, int>> listStolenProducts();
```

return a list of all the filed thefts in the order they happened, each element in the list is a pair of stolen Product and the fake_id of the thief who stole it.

```
std::list<Product> listAvailableProducts();
```

return a list of all the available products from oldest produced to newest.

Important Notes and Tips

- **Don't modify the signatures of the methods defined in the Factory.h header file.** We will test your code according to this interface.
- **Don't modify Product.h, you will not submit it.**
- You may add any class members you need (integers, boolean, data structures, ...), but **your implementation should use only mutexes and condition variables as synchronization primitives.** To put it another way, you are not allowed to use semaphores or any synchronization primitives (besides pthread_mutex_t and pthread_cond_t) that are provided in the pthreads library.
- You can use as many mutexes and condition variables as you need.
- In your implementation, you can ignore any errors that the pthreads functions may return; we will not check such cases in our tests.
- Assume that all id's/fake_id's are valid, we will not check cases such that the given id is already used when creating threads or the given id is not a valid thread id when finishing threads.
- Assume that the array passed to Production Thread will not be freed until the main is done.
- Assume `num_products` passed to Factory methods is always > 0.
- Use only standard POSIX threads and synchronization functions. You are not allowed, of course, to use the C++11 thread support or the C++11 atomic operations libraries.
- You should try to make your implementation as concurrent and as efficient as possible. The performance of your solution can affect your grade. However, efficiency must not come at the cost of the correctness of the required implementation!
- You can use any of the C++11 STL containers library. We will build your code with "g++ -pthread -std=c++11", so you should too.
- **Make sure your code is running on csl3 rather than your virtual machine. We will check your code only on csl3.** [csl3 default compiler is GCC 5.5, so it fully supports C++11.]
- csl3 may not be reachable outside the Technion network; but you can connect to csl2 from any network and then ssh to csl3.

Submission

- You should edit and submit all Factory.h and Factory.cxx that were provided in the course website under the "HW3 Wet" section
- You should electronically submit a zip file that contains the above files.
- You should **not** submit a printed version of the source code. However, you should document your source code!
- We only require you to submit the implementation of your classes, but not any test programs that use these classes. Of course, this does not mean you will not need a test program for debugging!
- A file named submitters.txt which includes the ID, name and email of the participating students. The following format should be used:

```
Linus Torvalds linus@gmail.com 234567890
Ken Thompson ken@belllabs.com 345678901
```

Important Note: Make the outlined zip structure exactly. In particular, the zip should contain only the following files (no subdirectories):

```
zipfile -+
         |
         +- Factory.h
         |
         +- Factory.cxx
         |
         +- submitters.txt
```

If you missed a file and because of this, the exercise is not working, **you will get 0 and resubmission will cost 10 points.** In case you missed an important file (such as the file with all your logic) we may not accept it at all. In order to prevent it you should open the zip file in a new directory and try to build and test your code in the new directory, to see that it behaves as expected.

Important Note: when you submit, **retain your confirmation code and a copy of the file(s)**, in case of technical failure. Your confirmation code is **the only valid proof** that you submitted your assignment when you did.

Have a Successful Journey,
The course staff