Circus
There's a cheerful clown who owns a chain of restaurants.

Due to the difficult situation on the IT market, he also decided to take part in the game and fired all programmers.
However, he did not foresee that they would leave the order handling system in a deplorable state.

So he asks students to create a new software for him, free of charge, to handle orders.
suitable for the available equipment and to which other workers have become accustomed.

Description of restaurant operation
This is what the day of operation of a single restaurant looks like.

In the restaurant there are machines for making products.
At the beginning of the day, a system is started, which includes machines and a certain number of employees.

Customers come to the restaurant who, in order to make an order, must use the system.
They can learn from it about the currently available menu and identifiers of placed orders pending execution.
Through the system, they can also place an order and receive a device informing about the readiness of the order for collection.
Then, when the order is prepared, they can, by returning the device to the system, pick up the finished products.

At the end of the day, the system is shut down and employee reports on the progress of their work are collected.

In addition, during the day, as happens in Clown restaurants, any machine can fail.
In this case, customers whose order cannot be fulfilled will be informed accordingly.
via the notification device, and a product that can no longer be produced disappears from the menu.
Because people with repair privileges only work at night, no one will ever try to use a broken machine on that day.

It also happens that customers do not receive their orders for various reasons.
As the Clown tries to be economical, if the customer does not pick up the finished order for the time specified at the beginning of the day,
then the products from the order can be given to other customers. After this time, the client irretrievably loses the opportunity
receipt of your order.

Task
Your task is to implement the System and CoasterPager classes matching the interface provided in the 'system. hpp' file.

Products are instances of subclasses of the Product class.

An order will be called a multiset of product names.
We call the order completed when all the necessary products are collected by the worker.

Machines are instances of subclasses of the Machine class. The Machine class implements methods:
getProduct -- returns unique_ptr for the Product. Potentially blocking at the time of product

manufacture. If the machine has failed raises the MachineFailure exception.
returnProduct -- allows you to return a product of the right type to the machine, e. g. if the customer does not collect it. The BadProductException is raised when you try to return a product of the wrong type. Whether the machine has failed has no effect on the operation of this method.
start -- starts the machine, it must be done before the machine is used for the first time.
stop -- stops the machine (no need to call in case of machine failure).
Machine Class methods do not guarantee security.
There are no two machines producing the same type of product.
Once a machine is broken, it will always raise an exception when calling getProduct.

The employee report is a structure containing the following fields:
collectedOrders -- vector of completed orders
abandonedOrders -- vector of orders that were not received by the client
failedOrders -- vector of orders that could not be fulfilled due to machine failure
failedProducts -- a vector of product names he was waiting for that could not be produced due to machine failure.

Notification devices are instances of the CoasterPager class.
CoasterPager provides 4 methods:
wait() -- which blocks until the order is ready, and in case of failure raises FulfillmentFailure.
wait(timeout) -- j. w. but blocks maximum for ms timeout.
getId -- returns a unique order number. Order numbers should correspond to the order in which they were placed, i. e. if the customer placed order A and then order B, then A should have a smaller number than B.
isReady -- informs if the order is ready.
An instance of the System class is responsible for handling the whole system.
It has the following methods:
constructor -- assumes a day-specific mapping of product names available in the menu to restaurant machine pointers, number of employees and clientTimeout, i. e. the time (in ms) after which a ready, unclaimed order may no longer be issued.
shutdown -- shutting down the system. Customers can no longer place new orders. It waits for the issuance of pending orders, collects employee reports and returns them in the form of a vector. When the system is closed, the machines should be stopped.
getMenu -- returns a vector of product names available in the menu. Returns an empty vector when the system is closed.
getPendingOrders -- returns a vector of order numbers, which have not yet been received by the client, have not passed the waiting time for them or have not failed any machine that the executing worker was waiting for.
order -- takes a vector of product names ordered by the customer and returns a unique_ptr pager that the customer can use to receive the order. If the shutdown method has already been called, the RestaurantClosedException is raised, while if the customer orders a product that is not available in the menu, the BadOrderException is raised.
collectOrder -- takes unique_ptr on a pager and returns a vector of products ordered by the client.
It raises the following exceptions:
FulfillmentFailure -- if one of the machines needed to complete the order broke down before the orderer picked up the product.
OrderNotReadyException -- if the order is not yet ready (but may still be processed).
OrderExpiredException -- if the order was not received by the client during clientTimeout.
BadPagerException -- in any other case when the order cannot be delivered to the customer (e. g. when someone has already received our order).
getClientTimeout -- returns the time after which a finished, unclaimed order may no longer be issued.

For vectors returned by the above methods, the order of the elements is irrelevant.

Additional requirements

The system is to enable parallel execution of 'numberOfWorkers' orders, by representing employees in separate threads.

Employees wait for orders and then fulfill them by collecting products from the machines.
It is enough that each employee will process only one order at a time.
When the employee completes the order, he should wait for the customer, pick up and verify the pager, then issue the order and return to waiting for the next orders.
If the machine breaks down while waiting for the product, the employee is obliged to inform the customer about this incident, using the machine, return all collected products to the appropriate machines and return to waiting for further orders.
If the customer does not pick up the order during clientTimeout, the employee should also return all collected products to the machines.

Since customers are sensitive to being ignored, orders should be handled in the order they are placed, i. e. if two orders have been placed and the first one (placed earlier) has a non-empty intersection with the second (placed later), then products of any type, from the common part of both orders, returned by machines earlier, should go to the first order.
For an order placed earlier, we consider it with a smaller number.

Example:
Content of orders:
Order_1: Chips, Chips, Juice
Order_2: Fries, Juice, Burger
Machines return in sequence (subsidiary number, smaller number means earlier return of the product):
Machine_Fries: Fries_1, Fries_2, . . .
Juice machine: Juice 1, Juice 2, . . .
Burger Machine: Burger_1, Burger_2. . .
When receiving orders, customers receive accordingly:
Order_1: French fries_1, French fries_2, Juice_1
Order_2: French fries_3, Juice_2, Burger_1

Additionally, due to the way the machines work, the Clown requires that if a worker needs a given product, when the product can be picked up from the machine (getProduct will not block) the product must be picked up immediately.
(This means that the worker should be waiting at the same time for all the products he needs to order).
For the same reason, the employee can only collect the products that are needed for the current order.

Admittedly, no one would dare fake a pager from the Clown Restaurant, so we are sure that the pagers that the system receives in collectOrder were issued by the restaurant. However, trick attempts and petty scams are common, so you should protect the restaurant from losses by issuing an order, only to the first customer who reports to pick up the order with a given number.