CSE691 HW4 (45 points)

Due: 11:59pm, April 14, Thursday (11:59pm)

Consider a plant, where there are ***m* part workers** (each will be implemented with a thread) whose jobs are to produce five types of parts (A, B, C, D, E). Each part worker will produce 5 pieces of all possible combinations, such as (2,0,0,2,1), (0,2,0,0,3), (5,0,0,0,0), etc., given that it takes 500, 500, 600, 600, 700 microseconds (*us*) to make each part of type A, B, C, D, E, respectively. For example, it takes a part worker 500*2 +600*2+700 *us* to make a (2,0,0,2,1) part combination. Each part worker will attempt to load the produced parts to a buffer area, which has a capacity for 5, 5, 4, 3, 3 pieces of type A, B, C, D, E parts, respectively. That is**, the buffer capacity is (5,5,4,3,3).** It will take a part worker 200, 200, 300, 300,400 *us* to move a part of type A, B, C, D, E, respectively, to/from the buffer. Each part combination, such as (1,0,2,1,1) is referred to as a **load order**.

The current number of parts of each type in the buffer, such as (5,2,1,3,2) is referred to as **buffer state**. A part worker will load the number of parts of each type to the buffer, restricted by the buffer's capacity of each type. For example, if a load order is (1,1,0,2,1) and the buffer state is (5,2,1,3,2), then the part worker can place a type B part, and a type E part to the buffer; thus, the updated load order will be (1,0,0,2,0) and the updated buffer state will be (5,3,1,3,3). The part worker will **wait** near the buffer area for the buffer space to become available to complete the load order. If the wait time reaches **MaxTimePart** *us* (maximum wait time for a part worker), the part worker will stop waiting, move the un-loaded parts back, and randomly re-generate a new load order, which must re-use the previously un-loaded parts (that got moved back). Recall that it takes a part worker 20, 20, 30, 30, 40 *us* to move a part of type A, B, C, D, E, respectively. A part worker will then repeat the process to produce a brand-new load order.

In addition, there are ***n* product workers** (each implemented as a thread) whose jobs are to take the parts from the buffer area and assemble them into products. Each product assembly needs five pieces of parts each time; however, the five pieces will be from exactly two or three types of parts, such as (1,2,2,0,0), (1,0,3,1,0), (1,0,0,0,4), (0,2,3,0,0), etc. with equal occurrence probability. For example, a product worker will not generate an order of (1,1,2,1,0), (5,0,0,0), etc. Each such legal combination from a product worker is referred to as a **pickup order**. The time it takes a product worker to move a part of type A, B, C, D, E from the buffer is 200, 200,300,300, 400 *us*, respectively. Like that for part workers, partial fulfillment policy is adopted. If the current buffer state is (4,0,2,1,3) and a pickup order is (1,1,0, 0,3), then the updated buffer state will be (3,0,2,1,0) and the updated pickup order will be (0,1,0,0,0). Note that when a product worker goes to the buffer area to pick up parts, the product worker will pick up parts and load them on the cart. At this moment, the numbers of parts on the cart will be (1,0,0,0,3), which is referred to as **cart state**. The product worker will wait next to the buffer area, looking to complete the pickup order. Once all needed parts are obtained, they will be moved back to assembly area and then assembled into products. The move time for parts of each type has been described. The assembly time needed for parts of type A, B, C, D, E, will be 600, 600, 700, 700, 800 *us*, respectively. If the wait time reaches **MaxTimeProduct** *us* (maximum wait time for a product worker), the product worker will move back the parts already picked up and randomly re-generate a load order which has to re-use all the parts that were moved back. The product worker will then re-produce a brand-new pickup order. If the parts moved back after timeout event are (1,1,0,0,0). During the next iteration, if a new pickup order (2,1,2,0,0) is generated, then the real pickup order that the product worker will bring to the buffer area is (1,0,2,0,0), while the parts (1,1,0,0,0), which were brought back during the last iteration due to timeout event, will be staying at local area; (1,1,0,0,0) will be referred to as **local state**.

Develop a simulation program of the activity of the above-described plant. Your implementation should be designed to improve the performance of the plant, while ensuring a fair treatment to all workers. Each part work or product worker is said to have completed one **iteration** when a load order or pickup order is completed, or if timeout event occurs such that an order is aborted. Your program should allow each worker to finish 5 iterations. Clearly you need to protect the shared resource, buffer, with proper lock/mutex. Every time when a part worker

thread or a product worker thread gain the access to the shared resource, we need to print information as shown below to a file call *log.txt* . Note that each product worker thread will also print the total number of completed products.

Create two global constants (of type int): MaxTimePart (maximum wait time for a part worker) and MaxTimeProduct (maximum wait time for a product worker).

The following is a sample main function.

```
const int MaxTimePart{ 18000 }, MaxTimeProduct{ 20000 };
//Different times might be used during grading; What will be good times to get good performances?

int main() {

	const int m = 20, n = 16; //m: number of Part Workers
		//n: number of Product Workers
		//Different numbers might be used during grading.


	vector<thread> PartW, ProductW;
	for (int i = 0; i < m; ++i) {
		PartW.emplace_back(PartWorker, i + 1);
	}
	for (int i = 0; i < n; ++i) {
		ProductW.emplace_back(ProductWorker, i + 1);
	}
	for (auto& i : PartW) i.join();
	for (auto& i : ProductW) i.join();

	cout << "Finish!" << endl;


	return 0;
}
```