I. Design

Roughly the structure of this program is:

```
main(): Interface   ----  multiple threads: Generator
                    ----  multiple threads: Operator
                    ----  thread: Timer
```

The `Pthread` interface is used.

I.i Interface

In this program, generators and operators are threads created by `main()`. `main()` constantly calls `scanf()` to read the input from the hardware with a while-loop, so that the user can input his command at any time to do control. That is, the interface is separated from the working threads so that the "pause/resume" and the other user-controlled functionality can be implemented. Since the generators and operators requires no input, there will be no interference.

I.ii Input Buffer

The input buffer is designed following these guidelines:

1) The size of the buffer is 10.
2) It shall be a random access structure, that is, the input/ output order will not be recorded.
3) All materials of the same kind are regarded identical.
4) The in-buffer statuses of all 3 kinds of materials are recorded respectively.
5) A counter is used to count how many slots of the buffer have been occupied.

Roughly the structure of the buffer is:

```
item buffer[BUFFER_SIZE];
int buffer_quantity[MATERIAL_TYPE];
bool buffer_status[MATERIAL_TYPE]; //is in-buffer or not
int counter;
```

I.iii Output Queue

The output queue is designed following these guidelines:

1) The size of the queue is pseudo-unlimited (depends on the data type of the queue);
2) The quantities of all 3 kinds of products are recorded respectively.
3) The latest output is recorded and updated constantly.

4) The output order of the products is not recorded.

5) All products of the same kind are regarded identical.

Roughly the structure of the output queue is:

```
long output_quantity[PRODUCT_TYPE];
PRODUCT latest_product;
```

I.iv Tools

The tools are designed under following guidelines:

1) The number of tools is decided by the user (read by `scanf()` in `main()`), but the acceptable range shall be 2~100 to guarantee the program really works.

2) All tools are considered identical.

3) 2 tools is called a set. A semaphore `tool` is used to show currently available sets.

4) Both grabbing tools and releasing tools should be atomic.

5) The operator can only grab 2 tools concurrently at one time.

Roughly the structure of tools is:

```
sem_t tool;
pthread_mutex_t mutex_tool;
int num_tools;
int free_tools;
```

I.v Generator

The generator threads constantly generates materials with while-loops. They are designed following these guidelines:

1) There shall be 3 generators with unique generator id respectively.

2) Each generator spends random time (1.0~2.0 sec) to generate a material.

3) These generators work without a fixed order.

4) Once the buffer is full, check whether all 3 types of materials are in buffer. If not, reduce all in-buffer materials to not more than 3 to make space of a possible different input. It is done to avoid deadlock.

5) Generators are blocked or woken up by a pair of semaphores `full` and `empty`.

6) The actions of generating a material and put it into buffer shall be atomic, controlled by a mutex `mutex_buffer`.

7) Every time when a generator successfully finishes a running in the loop, it changes the active flag to tell the timer thread that the program is normally running.

8) A generator shall be blocked but not canceled when paused.

Roughly the structure of a generator thread is:

```
while(1) {
    if not paused {
        spend time to generate;
        if the buffer's full, block the generator; else decrease semaphore
empty;
        Atomic actions {
            put the material into the buffer;
            do reduction if necessary;
        }
        increase semaphore full;
        mark active flag to be true;
    }
    if paused {
        Atomic actions {
            block the generator until the value of the condition variable is
changed;
        }
    }
}
```

I.vi Operator

The operator threads constantly produce and output products with while-loops. They are designed following these guidelines:

1) The number of operators is decided by the user (read by `scanf()` in `main()`), but the acceptable range shall be 1~100 to guarantee the program really works.

2) Each operator spends random time to get tools (1.0~2.0 sec) and generate a product (0.01~1.0 sec).

3) An operator shall first attempt to grab tools and then attempt to get materials.

4) The materials that can produce a same kind product as the latest one shall hold the lowest priority. For example, if the latest product in output queue is A, then the generator shall consider which product to produce next starting with B, then C, and lastly A.

5) The operator get 2 different materials concurrently at one time. If it is impossible do that, the operator will wait until possible. This is controlled by a semaphore `material_available.`

6) The actions of getting materials, producing a product and releasing the tools shall be

atomic, using the mutex `mutex_buffer`.

7) The operator outputs the newly produced product into the output queue only if (a) it is different from the latest product and (b) the differences between all 3 products in the queue will be less than 10 after this output. Else it will be discarded immediately.

8) The output operation shall be atomic, using a mutex `mutex_output`.

9) Operators are blocked or woken up by a pair of semaphores `full` and `empty`.

10) Every time when an operator successfully finishes a running in the loop, it changes the active flag to tell the timer thread that the program is normally running.

11) An operator shall be blocked but not canceled when paused.

Roughly the structure of the operator is:

```
while(1) {
   if not paused {
      spend time to get tools;
      if no more sets of tools are available, block; else decrease semaphore 
tool;
       Atomic actions {
         grab tools;
      }
      if the materials in buffer are not enough for producing, block; else 
decrease semaphore material_available;
      if the buffer's empty, block; else decrease semaphore full by 2;
      Atomic actions {
         get materials;
         spend time to produce;
         produce the product;
         if not any type of materials in buffer is used up after above 
operations, increase the semaphore material_available;
         Atomic actions {
            release tools;
         }
      }
      increase the semaphore empty by 2;
      Atomic actions {
         output or discard product;
      }
      mark active flag to be true;
```

```
    }
    if paused {
        Atomic actions {
            block the operator until the value of the condition variable is
changed;
        }
    }
}
```

I.vii Timer

The timer thread constantly counts the run time with while-loops to detect deadlocks. It is designed following these guidelines:

1) The timer should check the active flag once a second and mark it as false.

2) If the program is inactive (marked by the active flag) for no less than 20 sec, a deadlock will be considered occurred and the threads will be restarted, the buffer will be cleared and the mutexes and semaphores will be re-initialize, without changing the information of works already done.

3) The timer shall increase the number of deadlocks once a timeout occurs.

Roughly the structure of the timer is:

```
while(1) {
    resting interval, which is 1 sec;
    if not active {
        add to the timeout counter;
    }
    else {
        timeout counter = 0;
        mark the active flag false;
    }
    if timeout {
        increase number of deadlocks;
        cancel all thread, re-initialize and restart all threads;
    }
}
```

II. Manual

II.i Input

Inputs shall be done during initialization:

Input	Functionality	Prompt
Interger between 1~100	Specify the number of operators	`<?> Please set the number of operators:`
Interger between 2~100	Specify the number of tools	`<?> Please set the number of tools:`

During the program running, the user can do operations by input:

(a brief manual is shown right after initialization)

Input	Functionality
`u`	Pause/resume
`m`	See how many materials have been generated
`b`	See the current status of the input buffer
`p`	See how many products have been produced
`q`	See the current status of the output queue
`d`	See how many times that deadlocks occurred
`h`	Show the brief manual again
`restart`	Restart the process (also throws the stored information of the previous works)
`exit`	Exit the process
`status`	See other detail information (not mentioned in the brief manual, used for testing)

II.ii Output

Except the requested outputs mentioned above, other outputs are printed automatically and constantly, including:

1) Notifications of newly generated materials and newly produced products. Start with `<*>`.

2) Notifications of a product being put into the output queue. Start with `<*>`.

3) Notifications of discarding operations and errors. These conditions may or may not have the process crashed or working incorrectly. Start with `<!>`.

4) Notifications of pause/resume. Start with `<*>`.

5) Prompts for inputs. Start with `<?>`.

For example:

```
<*> Output: Product A
<!> Illegal to output A; it will be discarded.
```