CSE 3100 Systems Programming	Fall 2023
Homework #8	

A Makefile is provided. Submit your work to gradescope.

Problem 1. (100 points) Fast Food (Producer and Consumer)

In this problem, we write code to simulate a producer and consumer problem using multiple threads.

To make the problem easier to understand, we use a fast food restaurant analogy. Assume we have a fast food restaurant serving only one type of drink, one type of fries, and one type of burger. Customers order their food online and each customer can only order one drink, one fries and one burger. Since there are only type of drink, one type of fries and one type of burger, all orders are identical. Each of the these three items will be put into a queue implemented by a linked-list, which is accessed by the restaurant employees. Each employee will remove one item from the front of the linked-list and prepare the item, and put the prepared item into a sliding tray so that the item will slide down to the bottom for customers to pick up. The sliding tray has 3 columns, one for drinks, one for fries and one for burgers. A customer will pick up the food if all the 3 columns have items at the bottom of the tray. Since all the orders are identical, any customer can pick a order when it is available. Moreover, the sliding tray can hold limited number of items in each of the column. Because of this limitation, an employee needs to wait before he can put an item in the tray if the column for that item is full.

To simplify the problem, we assume the number of customers is `n_consumer` and the number of employees is `n_producer`, and these are known before hand. Moreover, we assume all customers already made their orders online before the employees started preparing food. After that, the customers show up at the restaurant to pick up their orders.

Note that multiple customers could be ordering simultaneously, and multiple employees could access the linked-list simultaneously. This implies that we need to use a mutex to protect the linked-list. Moreover, we need to make sure that all the 3 items a customer ordered are added to the linked-list together, otherwise, under certain conditions, both customers and employees can get stuck, and we will have hungry customers.

We use a 2-dimensional buffer to model the sliding tray. Specifically, we define the following structure for it.

```
typedef struct {
        int size;       //max # of items in each column
        int buf[MAX][3];
        int remain;     //remaining # of items to serve
        int counts[3];  //# of items in each column
        pthread_mutex_t mutex;
        pthread_cond_t produce_cond;
        pthread_cond_t consume_cond;
}two_d_buffer;
```

Here `size` is the maximum number of items each column can hold. `remain` is the number of items that still need to be produced, and is initialized to `3*n_customers`. Note this variable can be used for producer threads to determine when to exit. The 2-dimensional array `buf[MAX][3]` holds items (drinks, fries and burgers). The array `counts[3]` indicates the number of items in each column. Figure 1 shows an example of a 2-dimensional buffer with size 5. Note we use mutex to protect the 2-dimensional buffer, and we use two condition variables to synchronize the producing and consuming activities. Specifically, we need to implement two functions for the 2-dimensional buffer. The first one is

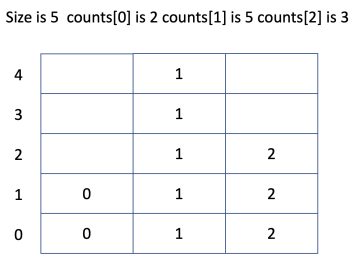

Figure 1: 2d-buffer with size 5

```
void add_to_buffer(int item, int col, two_d_buffer *p)
```

This function tries to add the `item` at the end of the column `col`. The function should block if that column is full. When the item added successfully, the function should update `remain` associated with the 2d buffer, and try to wake up the thread blocked on function `remove_from_buffer()`.
The second function is

```
void remove_from_buffer(int *a, int *b, int *c, two_d_buffer *p)
```

This function tries to remove all 3 items from the first row of the buffer, the removed items from column 0, 1 and 2, are pointed by a, b and c. The function should block if not all the items are present in the first row. If all 3 items are successfully removed from the buffer, we need to make sure the rest of the items will slide down to the right positions, and the function should try to wake up the threads that are blocked on `add_to_buffer()`.
The following function simulates the time needed to prepare different food items. And it is used in producer threads.

```
void prepare(int item)
{
        usleep((item + 1)*100);
}
```

We need to implement both the consumer thread function, and the producer thread function. Both of them share the following structure.

```
struct thread_data
{
        int id;
        list_t *p;
        two_d_buffer *q;
        int total;                      //total items produced by a producer
        pthread_barrier_t *p_barrier;
};
```

Here `p` points to a linked-list, as defined below.

```
typedef struct {
    node    *head, *tail;
    pthread_mutex_t mutex;
} list_t;
```

Note the linked-list keeps track of both the head and the tail so that we can use it conveniently as a queue. Note a mutex is used since there might be multiple threads try to access the linked-list simultaneously.

Note `p_barrier` points to a barrier, which will be used to ensure no producers start producing before all the orders are put in the queue represented by the linked-list.

Below is the function for consumer threads.

```
void* thread_consume(void* threadarg)
{
        struct thread_data* my_data = (struct thread_data*) threadarg;
        int id = my_data->id;
        list_t *p = my_data->p;

        node *n1 = create_node(0);
        node *n2 = create_node(1);
        node *n3 = create_node(2);

        //TODO
        //fill in code below to add n1, n2 and n3 to the linked-list pointed by p

        pthread_barrier_t *p_barrier = my_data->p_barrier;
        pthread_barrier_wait(p_barrier);

        two_d_buffer *q = my_data->q;
        int a, b, c;

        remove_from_buffer(&a, &b, &c, q);
        printf("consumer %04d (%d %d %d)\n", id, a, b, c);
        pthread_exit(NULL);
}
```

Below is the function for producer threads.

```
void* thread_produce(void* threadarg)
{
        struct thread_data* my_data = (struct thread_data*) threadarg;
        list_t *p = my_data->p;
        pthread_barrier_t *p_barrier = my_data->p_barrier;
        pthread_barrier_wait(p_barrier);
        two_d_buffer *q = my_data->q;

        int done = 0;
        while(!done)
        {
                //TODO
                //fill in code below
        }
        pthread_exit(NULL);
}
```

Find the "TODO"s in the starter code food.c and fill in the necessary code. We do not need to change anything in linked-list.h. The main() function is located in food.c. Three command line arguments are required.

The first one is the number of customers(consumers), the second is the number of employees(producers), and the third is the size of the 2-dimensional buffer. Below is an example output.

```
$ ./food 5 1 1
consumer 0001 (0 1 2)
consumer 0004 (0 1 2)
consumer 0003 (0 1 2)
consumer 0000 (0 1 2)
consumer 0002 (0 1 2)
total = 15
```

In this example, there are 5 customers (consumers), and they all made orders for 1 drink, 1 fries and 1 burger. In total they ordered 15 items. There is only one employee(producer) serving all the items. Moreover, the buffer size is 1. From the output, we can see every customer picked up their order in an arbitrary order. For example, the first line of output shows that consumer 0001 is the first to pick up an order (0 1 2). The total number of items served is 15 as shown in the last line of the output.

Submit food.c to gradescope.

Enjoy!