Xi Liu, xl3504, hw4

1.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

	int i = 0;

	/* ADD SOME THINGS HERE */
	pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
	pthread_cond_t c = PTHREAD_COND_INITIALIZER;

	void thread_exit()
	{
		pthread_mutex_lock(&m);
		i = 1;
		pthread_cond_signal(&c);
		pthread_mutex_unlock(&m);
	}

	void
	*foo(void *arg)
	{
		printf("I am foo!!!\n");
		/* ADD SOME CODE HERE */
		thread_exit();
		return NULL;
	}

	void
	*boo(void *arg)
	{
		/* ADD SOME CODE HERE */
		pthread_mutex_lock(&m);
		while(i == 0)
		{
			pthread_cond_wait(&c, &m);
		}
		pthread_mutex_unlock(&m);
		printf("I am boo!!!\n");
	}
```

```
int
main(int argc, char** argv)
{
        pthread_t p1, p2;
        pthread_create(&p1, NULL, &foo, NULL);
        pthread_create(&p2, NULL, &boo, NULL);

        // wait for threads to finish
        // before exiting
        pthread_join(p1, NULL);
        pthread_join(p2, NULL);
        printf("main: end\n");
        exit(0);
}
```

2. 1

Variable Name	Variable Type	Initial Value	Description
Paper	int	0	If the smoker has paper, then paper is 1; otherwise, paper is 0.
Tobacco	int	0	If the smoker has tobacco, then tobacco is 1; otherwise, tobacco is 0.
Matches	int	0	If the smoker has matches, then matches is 1; otherwise, match is 0.
remaining_ingredient	int	0	Before the procedure chooseIngredients() run, remaining_ingredient is 0, indicates that nothing is selected; After an execution of chooseIngredients() that randomly selects 2 of the 3 ingredients, if remaining_ingredient

			is 1, indicates that paper is the remaining ingredient; if remaining_ingredient is 2, indicates that tobacco is the remaining ingredient; if remaining_ingredient is 3, indicates that matches is the remaining ingredient;
smoker_i->ingredient	int	0	i = 1, 2, 3 smoker->ingredient is the ingredient that the smoker has. If smoker->ingredient is 0, indicates that the smoker process has not been initialized If smoker->ingredient is 1, indicates that the smoker process has paper If smoker->ingredient is 2, indicates that the smoker process has

			tobacco If smoker->ingredient is 3, indicates that the smoker process has matches
smoker_complete	int	0	smoker_complete is initialized as 1, since no smoker started smoking, they are set to be “complete”. if a smoker completed smoking, smoker_complete is immediately set to 1; before a call to chooseIngredients, smoker_complete is set to 0, which indicates that the smoker would start to smoke.
mutex	pthread_mutex_t		Mutual exclusion, ensure that only one thread can modifies its state at a given time.
c	pthread_cond_t		Conditional variable, Used like a queue to communicate between threads.

2.2

//3 smokers can smoke at the same time

```
int paper = 0;
int tobacco = 0;
int matches = 0;
```

```
int remaining_ingredient = 0;
int smoker_complete = 1;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;


Agent()
{//the agent places 2 of the ingredients on the table

        acquire(&mutex);
        while(smoker_complete = = 1)
        {
            smoker_complete = 0;
            chooseIngredients(&paper, &tobacco, &matches);

            if(paper & tobacco)
            {
                remaining_ingredient = 3; //matches
            }
            else if(paper & matches)
            {
                remaining_ingredient = 2; //tobacco
            }
            else if(tobacco & matches)
            {
                remaining_ingredient = 1; //paper
            }
        }
        release(&mutex);
    }


matchSmoker()
{//this smoker has a lot of matches
        acquire(&mutex)
        while(remaining_ingredient != 3)
        {
            pthread_cond_wait(&c, &m);
        }

        //the smoker has the remaining ingredient which is matches
        smoke();
        paper = 0;
        tobacco = 0;
```

```
        smoker_complete = 1;
        pthread_cond_signal(&c); //signal the agent
        release(&mutex);
}
```

3.

```
Class WaitingRoom
{
    int n_used = 0; //chair used
    pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
    pthread_cond_t c = PTHREAD_COND_INITIALIZER;
    Queue<Integer> q = new Queue<Integer>();
    int myturn = 0;
    int call = 0;

    int enter()
    {
        acquire(&mutex);
        if(n_used = = NCHAIRS)
        {
            release(&mutex);
            return WR_FULL;
        }
        else
        {
            n_used++;
            call++;
            q.enqueue();
            while(myturn != call)
            {
                pthread_cond_wait(&c, &mutex);
            }
            n_used--;
            q.dequeue();
            release(&mutex);
            return MY_TURN;
        }
    }
```

```
    callNextCustomer()
    {
        acquire(&mutex);
        if(n_used = = 0)
        {
            release(&mutex);
            return WR_EMPTY;
        }
        else
        {
            pthread_cond_signal(&c);
            release(&mutex);
            return WR_BUSY;
        }
    }
}

Class BarberChair
{
    int state = EMPTY;
    int customer = 0;
    pthread_cond_t barber_up = PTHREAD_COND_INITIALIZER;
    pthread_cond_t customer_done = PTHREAD_COND_INITIALIZER;
    pthread_cond_t sit_in_chair = PTHREAD_COND_INITIALIZER;



    void napInChair()
    {
        acquire(&mutex);
        if(costumer = = 0)
        {
             state = BARBER_IN_CHAIR;
            //when no customer present, barber sits in chair
            //and falls asleep
        }
        while(customer = = 0)
        {
            pthread_cond_wait(&barber_up, &mutex);
             //barber remains sleeping if no customer
        }

        release(&mutex);
    }
```

```
void wakeBarber()
{
    acquire(&mutex);
    customer = 1;
    pthread_cond_signal(&barber_up);
    release(&mutex);
}

void sitInChair()
{
    acquire(&mutex);
    while(state != EMPTY)
    {
        pthread_cond_wait(&sit_in_chair, &mutex);
    }
    if(LONG_HAIR_CUSTOMER_IN_CHAIR != NULL)
    {
        state = LONG_HAIR_CUSTOMER_IN_CHAIR;
    }
    while(state != SHORT_HAIR_CUSTOMER_IN_CHAIR)
    {
        pthread_cond_wait(&customer_done, &mutex);
    }
    state = EMPTY;
    pthread_cond_signal(&customer_done, &mutex);
    release(&mutex);
}

void cutHair()
{
    acquire(&mutex);
    while(state != LONG_HAIR_CUSTOMER_IN_CHAIR)
    {
        pthread_cond_wait(&customer_done, &mutex);
    }
    if(SHORT_HAIR_CUSTOMER_IN_CHAIR != NULL)
    {
        state = SHORT_HAIR_CUSTOMER_IN_CHAIR;
    }
    release(&mutex);
}

void tellcustomerDone()
```

```
    {
        acquire(&mutex);
        while(state != EMPTY)
        {
            pthread_cond_wait(&customer_done, &mutex);
        }
        release(&mutex);
    }
}
```