```c
/*objective:
 * This lab's objective is to simulate a restaurant that get's orders from customers, where the restaurant has burger cooks and fryer cooks to take orders from customers.
 * so the goal is to do it using multithreading concepts in operating systems. For each burger cook, fryer cook and customer a thread is created. once the cooks make the maximum number of burgers then simulation completes.
 *
 * Design:
 * Structs were used to store the parameters of a burger cook, fryer cook and a customer
 * another struct was used to store all the data. I used the linked lists provided by the instructor to add or remove burger or fries.
 * while adding or removing items from the list, I used mutexes to lock and unlock while adding or removing.
 *
 * Build instructions:
 * to build: make
 * to run: ./burger *textfile name*
 * textfile names : parameters,parameters 1
 *
 * Analysis:
 * I used mutexes instead of semaphores to lock and unlock while adding or removing. as it was easy to fill the order.
 * all threads starts at the same time. usage of a flag to terminate the simulation when the day is over was very helpful.
 * I used a different text file with different parameters, and verified that the ratios of the customer orders matched.
 * before adding a burger, the size of list shouldn't be zero and the size becomes the maximum no of burgers and similarly for the fries.
 *
 * Conclusion:
 * This was a very challenging lab as more than understanding the concept, debugging the multi-threads was the hardest.
 * very good assignment to understand and apply the concept of multi-threading.
 *
 *
*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include "llist.h"
#include <semaphore.h>
#include <unistd.h>

//Struct Defintions for Burger, Fryer, customer
typedef struct {
    int no of cooks;
    int time taken;
    int no of burgers;
}burger_cooks;

typedef struct{
    int no of cooks;
    int time taken;
    int no of servings;
}fryer_cooks;

typedef struct{
    int n burgers;
    int n fries;
    int wait time;
    int allitems ordered;
    int order_completed;
```

```
54  }customers;
55
56  typedef struct{
57      burger_cooks *burger data;
58      fryer_cooks *fryer data;
59      customers *customer data;
60  }data;
61
62  //Function Definitions
63  void read first line(FILE*,burger_cooks *,char *argv[]);
64  void read second line(FILE*,fryer_cooks *,char *argv[]);
65  void init customer threads(FILE*,int,customers *customer,char *argv[]);
66  void create threads(data *);
67
68
69  //Global Variables
70
71  //Linked lists for Burgers and Fryers
72  list burgerlist;
73  list fryerslist;
74
75
76  //warmer trays
77  pthread_mutex_t warmer tray;
78
79
80  int max burgers = 0;
81  int max fries = 0;
82  int no of customers;
83  int end of dayflag=0; // TODO ???? rename, or at least comment what this flag is for
84
85  int main(int argc, char* argv[]) {
86      llInit(&burgerlist);
87      llInit(&fryerslist);
88
89
90      FILE* simulator= fopen(argv[1],"r");
91      pthread mutex init(&warmer tray,NULL);
92
93      burger_cooks *burger cook = malloc(sizeof(burger_cooks));
94      fryer_cooks *fryer cook = malloc(sizeof(fryer_cooks));
95
96
97      read first line(simulator,burger cook,argv);
98      read second line(simulator,fryer cook,argv);
99
100     fscanf(simulator,"%d",&no of customers);
101     //printf("%d\n",no of customers);
102     customers *customer=malloc(no of customers*sizeof(customers));
103
104     init customer threads(simulator,no of customers,customer,argv);
105
106     data *burgerplace = malloc(sizeof(data));
107     burgerplace->burger data=burger cook;
108     burgerplace->fryer data=fryer cook;
109     burgerplace->customer data=customer;
110
111
112     printf("\nThe Burger Place");
113     printf(" Burger Cooks - Number:%d",burgerplace->burger data->no of cooks);
114     printf(" Time:%d",burgerplace->burger data->time taken);
115     printf(" Total:%d",burgerplace->burger data->no of burgers);
116
117
118     printf("\nFry cooks - Number:%d",burgerplace->fryer_data->no_of_cooks);
```

```
119         printf(" Time:%d",burgerplace->fryer data->time taken);
120         printf(" Total:%d",burgerplace->fryer data->no of servings);
121
122         for(int i=0; i < no of customers; i++){
123         printf("\nCustomer %d - Burgers:%d Fries:%d
            Wait:%d",i,burgerplace->customer data[i].n burgers,burgerplace->customer data[i].n
             fries,burgerplace->customer data[i].wait time);
124         }
125
126         create threads(burgerplace);
127
128         printf("\n\nSimulation Finished:\n");
129         printf("Max Burgers in the burger warmer:%d\n",max fries);
130         printf("Max Fries in the fry warmer:%d\n",max burgers);
131
132         for(int i=0;i<no of customers;i++){
133         printf("Customer %d had their order filled %d
            times\n",i,burgerplace->customer data[i].order completed);
134         }
135
136         fclose(simulator);
137         free(burger cook);
138         free(fryer cook);
139         free(customer);
140         free(burgerplace);
141     }
142     /*
143      * read the first line and creates the number of threads
144      */
145     void read first line(FILE* simulator, burger_cooks *burger cook, char *argv[]){
146         int parameter 1; //Burger cook or fryers
147         int parameter 2; //time taken to make one burger or one serving of fries
148         int parameter 3; // no of burgers or number of serving of fries
149
150         fscanf(simulator,"%d",&parameter 1);
151         fscanf(simulator,"%d",&parameter 2);
152         fscanf(simulator,"%d",&parameter 3);
153
154         burger cook->no of cooks = parameter 1;
155         burger cook->time taken= parameter 2;
156         burger cook->no of burgers=parameter 3;
157         //printf("%d\n",burger cook.no of cooks);
158         //printf("%d\n",parameter 2);
159         //printf("%d\n",parameter 3);
160     }
161     /*
162      * Reads the paramters for Fryer cooks and call the create fryer function
163      */
164     void read second line(FILE* simulator, fryer_cooks *fryer cook,char *argv[]){
165         int parameter 1; //Burger cook or fryers
166         int parameter 2; //time taken to make one burger or one serving of fries
167         int parameter 3; // no of burgers or number of serving of fries
168
169         fscanf(simulator,"%d",&parameter 1);
170         fscanf(simulator,"%d",&parameter 2);
171         fscanf(simulator,"%d",&parameter 3);
172
173         fryer cook->no of cooks = parameter 1;
174         fryer cook->time taken = parameter 2;
175         fryer cook->no of servings= parameter 3;
176     }
177     /*
178      * reads the values for customers and calls for creating a thread for each customer
179      */
180     void init_customer_threads(FILE* simulator, int no_of_customers, customers
```

```
*customer, char *argv[]){

    int parameter 1; //number of burgers each customer buys
    int parameter 2; //number of servings of the fries
    int parameter 3; //amount of time the customers waits after ordering

    for(int i=0; i<no of customers;i++){
        fscanf(simulator,"%d",&parameter 1);
        fscanf(simulator,"%d",&parameter 2);
        fscanf(simulator,"%d",&parameter 3);
        customer[i].n burgers= parameter 1;
        customer[i].n fries= parameter 2;
        customer[i].wait time= parameter 3;
    }
}
/*
 * test thread, to check if a thread is actually created
 */
void *example thread(void *x){
    printf("sucessfull\n");
    pthread exit(0);
}
/*
 * each burger cook cooks 100 burgers and wait time for each is 1000 us
 */
void *burgercook thread(void *x){

    burger_cooks *burger cook = (burger_cooks*)x;
    //printf("Burger Cook HELLO\n");
    //push 100 burgers for each cook
    for (int i=0; i< burger cook->no of burgers; i++){
        usleep(burger cook->time taken);
        pthread mutex lock(&warmer tray);
        llPushFront(&burgerlist, "B");
        //printf("%d",llSize(&burgerlist));
        if(max burgers <= llSize(&burgerlist))
        {
            max burgers =llSize(&burgerlist);
        }
        pthread mutex unlock(&warmer tray);
    }
    pthread exit(0);
}
/*
 * each fryer cooks serves 125 servings and wait time for each servings is 2000us
 */
void *fryercook thread(void *x){
    fryer_cooks *fryer cook = (fryer_cooks*)x;
//  printf("Fryer Cook HELLO\n");
    for(int i=0; i<fryer cook->no of servings;i++){
        usleep(fryer cook->time taken);
        pthread mutex lock(&warmer tray);
        llPushFront(&fryerslist, "F");
        if(max fries <=llSize(&fryerslist)){
            max fries = llSize(&fryerslist);
        }
        pthread mutex unlock(&warmer tray);
    }
    pthread exit(0);
}
void *customer thread(void *x){
    customers *customer = (customers*)x;
 //   printf("Customers HELLO\n");
    while(end of dayflag){
        int burger_count = 0;
```

```
245             while(burger count != customer->n burgers){
246                 if(llSize(&burgerlist)!=0){
247                     pthread mutex lock(&warmer tray);
248                     llPopFront(&burgerlist);
249                     burger count = burger count + 1;
250                     if(burger count == customer->n burgers){
251                         customer->allitems ordered = customer->allitems ordered + 1;
252                     }
253                 pthread mutex unlock(&warmer tray);
254                 }
255                 if(end of dayflag==0){
256                     pthread mutex unlock(&warmer tray);
257                     pthread exit(0);
258                 }
259             }
260         int fries count = 0;
261         while(fries count != customer->n fries){
262             if(llSize(&fryerslist)!=0){
263                 pthread mutex lock(&warmer tray);
264                 llPopFront(&fryerslist);
265                 fries count = fries count + 1;
266                 if(fries count == customer->n fries){
267                     customer->allitems ordered = customer->allitems ordered + 1;
268                 }
269                 pthread mutex unlock(&warmer tray);
270                 }
271             if(end of dayflag==0){
272                 pthread mutex unlock(&warmer tray);
273                 pthread exit(0);
274             }
275         }
276         if(customer->allitems ordered == 2){
277             usleep(customer->wait time);
278             customer->order completed += 1;
279             customer->allitems ordered=0;
280         }
281         if(end of dayflag==0){
282             pthread mutex unlock(&warmer tray);
283             pthread exit(0);
284             }
285         }
286         pthread exit(0);
287     }
288     /*
289      * creates threads for n burger cooks
290      */
291     void create threads(data *burgerplace){
292         //int no of threads = burger cook->no of cooks;
293         //printf("%d\n",no of threads);
294 
295         pthread_t burger cookthread[burgerplace->burger data->no of cooks];
296         pthread_t fryer cookthread[burgerplace->fryer data->no of cooks];
297         pthread_t customerthread[no of customers];
298 
299         for(int i=0; i<burgerplace->burger data->no of cooks; i++){
300            if(pthread create(&(burger cookthread[i]), NULL, &burgercook thread,
           (void*)(burgerplace->burger data))){
301                     printf("error");
302             }
303         }
304         for(int i=0; i<burgerplace->fryer data->no of cooks; i++){
305             if(pthread create(&(fryer cookthread[i]), NULL, &fryercook thread,
            (void*)(burgerplace->fryer data))){
306                     printf("error");
307                 }
```

```
308         }
309         for(int i=0;i<no of customers;i++){
310             if(pthread create(&(customerthread[i]), NULL, &customer thread,
            (void*)((burgerplace->customer data)+i))){
311                     printf("error");
312                 }
313         }
314
315         end of dayflag =1;
316         for(int i=0; i<burgerplace->burger data->no of cooks;i++){
317             pthread join(burger cookthread[i],NULL);
318         }
319         for(int i=0; i<burgerplace->fryer data->no of cooks;i++){
320             pthread join(fryer cookthread[i],NULL);
321         }
322         end of dayflag=0;
323         for(int i=0;i<no of customers;i++){
324             pthread join(customerthread[i],NULL);
325         }
326     }
327
328
```