نام و نام خانوادگی: هلیا سادات هاشمی پور **شماره دانشجویی:**9831106

نام استاد:استاد علیزاده **تاریخ:**10آذر 1400

-۲

توضیح الگوریتم کلی

با توجه به توضیحات مطرح شده در دستور کار و کلاس برای جلوگیری از deadlock این برنامه به این صورت کار می‌کند که هنگام اجرای هر ترد، چوب‌ها چک می‌شوند و اگر دو چوب بود که قفل نشده بود، آن ترد که در واقع نماینده‌ی فیلسوف مورد نظر است آن دو چوب را قفل می‌کند؛ تا زمانی که موقع فکر کردن او فرا برسد. در هر مرحله، زمان فکر کردن فیلسوف به صورت عددی رندوم و کوچکتر از ۵ در نظر گرفته می‌شود. همچنین پس از این که زمان مشخصی گذشت که فیلسوف در حال غذا خوردن بود، غذا خوردن او به طور کامل تمام شده و پس از اتمام خوردن همه‌ی ۵ فیلسوف برنامه به پایان می‌رسد.

توضیح کد

ابتدا کتابخانه‌های مورد استفاده و مقادیر ثابت و mutex مربوط به چوب‌ها که از ایجاد شرایط مسابقه جلوگیری می‌کند را تعریف کردیم.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <pthread.h>
4 #include <unistd.h>
5
6 #define EAT_TIME 5
7 #define CHOPSTICKS_NUM 5
8 #define PHILSOOPH_NUM 5
9
10 pthread_mutex_t chopsticks[CHOPSTICKS_NUM];
11
12
```

در تابع main نیز همانطور که از کامنت‌ها مشخص است، یک آرایه‌ی پنج‌تایی ids ساختیم که هر خانه‌ی آن مقدار برابر با شماره‌ی خانه است؛ این کار باعث مشخص شدن شماره‌ی هر فیلسوف می‌شود سپس تردها را ساختیم و با پیمایش روی آن‌ها خانه‌ی متناظر با شماره‌ی پیمایش را به عنوان پوینتری که باید به هندلر ترد داشته باشیم، می‌دهیم. در واقع ترد ها را یکی‌یکی صدا میزنیم که تابع هندلر را اجرا کنند.

```
69
70 void main() {
71   pthread_t philosophs[PHILSOOPH_NUM];
72   int ids[5];
73   for (int i = 0; i < CHOPSTICKS_NUM; i++) {
74     ids[i] = i;
75     //make all chopsticks a mutex object, null make a mutex by default attributes
76     pthread_mutex_init(&chopsticks[i], NULL);
77   }
78
79   for (int i = 0; i < PHILSOOPH_NUM; i++) {
80     // run a thread for each philsooph by philsoph_handler and get ids as args of
   this function
81     pthread_create(&philosophs[i], NULL, philosoph_handler, &ids[i]);
82   }
83   //wait for all philsooph(thread) to finish their job
84   for (int i = 0; i < PHILSOOPH_NUM; i++) {
85     pthread_join(philosophs[i], NULL);
86   }
87   printf("*********finished********* \n");
88 }
nn
```

در هندلر مربوط به تردها هم چنین عمل کردیم:

همانطور که در ابتدا توضیح داده شد و در کامنت‌ها هم مشاهده میکنید در آرایه‌ی مربوط به چوب‌ها پیمایش میکنیم تا چوبی که لاک نشده است را پیدا کنیم و فیلسوف مورد نظر با قفل کردن آن در واقع مشتول به غذا خوردن شود.

```
12
13 void *philosoph_handler (void* args)
14  {
15   int id = *((int*)args);
16   int try1;
17   int try2;
18   int timePassed = 0;
19   int possible1 = 1;
20   int possible2 = 1;
21   while(timePassed < EAT_TIME){
22   printf("Philosoph[%d] is Thinking \n", id);
23   sleep(rand() % 5);
24   printf("Philosoph[%d] is Hungry \n", id);
25   //this loop try to find 2 CHOPSTICKS
26   while(1) {
27     possible1 = 1;
28     possible2 = 1;
29     //try to lock  chopsticks If the chopsticks is not already locked, lock it as
   first one
30     for (try1 = 0; try1 < CHOPSTICKS_NUM; try1++) {
31           possible1 = pthread_mutex_trylock(&chopsticks[try1]);
32           if(possible1 == 0){
33            for (try2 = CHOPSTICKS_NUM-1; try2 >= 0; try2--) {
34         //try to find another fork, if there isnt any free fork this function return
   0 and doesnt block thread
35         possible2 = pthread_mutex_trylock(&chopsticks[try2]);
36         if (possible2 == 0){ //it's possible
37                 break;
38                 }
39         }
```

```
40          //Two chopsticks are availab
41                  if (possible1 == 0 && possible2 == 0){
42                          break;
43                  }
44            }
45          }
46
47      if (possible2 == 0) {
48        break; // there two chopsticks for eating so break this loop
49      } else {
50        pthread_mutex_unlock(&chopsticks[try1]);//because there aren't two chopsticks
   we put locked chopstick on table by unlocking related mutex object
51        sleep(1);
52      }
53    }
54
55
56    int eat_time = rand() % 5 + 2;
57    sleep(eat_time);
58    timePassed += eat_time;
59
60    //after eating we put booth fork on table by unlocking related mutex object
61    pthread_mutex_unlock(&chopsticks[try1]);
62    if (possible2 == 0)
63          pthread_mutex_unlock(&chopsticks[try2]);
64    printf("Philosoph[%d] is Eating by chopstick[%d] and chopstick[%d]\n", id, try1,
   try2);
65 }
66          printf("Philosoph[%d] finished \n", id);
67 }
68
```

- **خروجی کد**

همانطور که از کد مشخص بود، ابتدا تمامی فیلسوف‌ها شروع به فکر کردن می‌کنند، بعد به نوبت شروع به چک کردن چوب‌ها کرده و در صورت در دسترس بودن ۲ چوب، خوردن را تا فرا رسیدن زمان دوباره‌ی تفکر، ادامه می‌دهند.

```
Philosoph[2] is Thinking
Philosoph[1] is Thinking
Philosoph[0] is Thinking
Philosoph[1] is Hungry
Philosoph[3] is Hungry
Philosoph[2] is Hungry
Philosoph[1] is Eating by chopstick[0] and chopstick[4]
Philosoph[1] is Thinking
Philosoph[4] is Hungry
Philosoph[0] is Hungry
Philosoph[3] is Eating by chopstick[1] and chopstick[3]
Philosoph[3] is Thinking
Philosoph[1] is Hungry
Philosoph[3] is Hungry
Philosoph[0] is Eating by chopstick[1] and chopstick[3]
Philosoph[0] is Thinking
Philosoph[4] is Eating by chopstick[0] and chopstick[4]
Philosoph[4] finished
Philosoph[0] is Hungry
Philosoph[3] is Eating by chopstick[1] and chopstick[3]
Philosoph[3] finished
Philosoph[2] is Eating by chopstick[0] and chopstick[4]
Philosoph[2] finished
Philosoph[1] is Eating by chopstick[1] and chopstick[3]
Philosoph[1] finished
Philosoph[0] is Eating by chopstick[0] and chopstick[4]
Philosoph[0] finished
********finished*********
```

نکته‌ی قابل توجه در این خروجی، این است که چون حلقه‌ای که به دنبال چوب اول است از خانه‌ی ۰ و حلقه‌ای که مربوط به چوب دوم است از خانه‌ی ۴ شروع به پیمایش می‌کنند و با توجه به این نکته که تعداد چوب‌ها فرد است، در حالی که برای خوردن به تعداد چوب‌های زوج نیاز است، هیچ گاه از چوب خانه‌ی ۲ استفاده نمی‌شود.

اگر بخواهیم از تمامی چوب‌ها حداقل یکبار استفاده کنیم، می‌توانیم چنین عمل کنیم که به هر فیلسوف چوب منتاظر با آی دی او را اختصاص دهیم و تنها در حلقه به دنبال چوب دوم بگردیم.

البته لازم به ذکر است الگوریتم پیاده شده هم تناقضی با موارد خواسته شده و مطرح شده در دستورکار ندارد و توضیح داده شده صرفا جهت ارائه‌ی پیشنهاد ثانویه بود.