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

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

ا-

- **توضیح کد**

ابتدا کتابخانه‌های مورد استفاده، سپس تعداد readerها و تعدادی که میخواهیم شمارنده افزایش یابد را تعریف کردیم.

سپس یک struct برای برطرف کردن مشکل race condition و قفل کردن پراسه‌ها تعریف کردیم که در آن از semaphore استفاده شده است: به این صورت که یک آبجکت سمافور به نام write_lock و count وجود دارد. این struct در حافظه‌ی مشترک استفاده می‌شود.

در دو خط بعد هم اسم و اندازه‌ی حافظه را تعریف کردیم که اندازه‌ی آن همان اندازه‌ی struct نوشته شده است.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <sys/mman.h>
 4 #include <sys/stat.h>
 5 #include <sys/shm.h>
 6 #include <sys/types.h>
 7 #include <sys/wait.h>
 8 #include <fcntl.h>
 9 #include <time.h>
10 #include <unistd.h>
11 #include <semaphore.h>
12
13 #define READER_NUM 2
14 #define MAX_NUM 10
15
16
17 struct Counter
18 {
19     sem_t write_lock;
20     int count;
21 };
22
23 const int SIZE = sizeof(struct Counter);
24 char* NAME = "shared memory";
25
```

اکنون به سراغ توابع می‌رویم. یک تابع برای نوشتن و یک تابع برای خواندن داریم.

Writer:

shared memory را با اسم و اندازه‌ی تعریف شده ایجاد می‌کنیم و با استفاده از mmap به شکل آرایه به آن دسترسی پیدا میکنیم. جنس این آرایه از struct Counter تعریف شده است که در واقع تک‌خانه‌ی مصرفی ما می‌باشد.

کارهای مورد نظر را از طریق memory_object روی مموری انجام می‌دهیم.

سپس در یک حلقه‌ی while always true تا وقتی که count کوچکتر از مقدار تعریف شده باشد، مقدار آن را افزایش می‌دهیم.

برای جلوگیری از race condition موقع نوشتن روی مموری درسترسی reader ها را قفل می‌کنیم، در واقع با استفاده از حلقه که به تعداد reader ها تکرار می‌شود، تک‌تک آن‌ها را در حالت wait قرار می‌دهیم تا این که پس از نوشتن و اتمام کار مورد نظر دوباره با استفاده از حلقه، آن‌ها را post می‌کنیم تا دوباره امکان دسترسی پیدا کنند.

هر بار پس از نوشتن، قبل از اینکه به خواننده‌ها امکان دسترسی بدهیم، مقدار count را چاپ می‌کنیم.

```
26 int writer() {
27     int shm = shm_open(NAME, O_RDWR, 0666);
28     struct Counter *memory_object = (struct Counter *)mmap(0, SIZE,
   PROT_READ | PROT_WRITE, MAP_SHARED, shm, 0);
29     while(1){
30         //comment for loop -> race condition
31         for(int i = 0; i<=READER_NUM ; i++){
32         sem_wait(&(memory_object->write_lock));
33         }
34
35         (memory_object->count)++;
36         fprintf(stdout,"writer writer1, new value:%d\n", memory_object-
   >count);
37         fflush(stdout);
38         //comment for loop -> race condition
39         for(int i = 0; i<=READER_NUM; i++){
40         sem_post(&(memory_object->write_lock));
41         }
42
43         if((memory_object->count) > MAX_NUM){
44            exit(EXIT_FAILURE);
45         }
46     }
47 }
```

Reader:

در ابتدا خیلی شبیه به تابع Writer عمل می‌کنیم و یک ممووری با همان نام و سایز و آرایه را تعریف میکنیم.

سپس در حلقه تا جایی به خواندن ادامه می‌دهیم که count به مقدار تعریف شده برسد.

در گام اول wait برای آبجکت سمافور صدا میزنیم اگر در تابع wait مقدار این آبجکت به صفر رسیده باشد، در واقع قفل شده باشد در این خط لاک میشویم و تا زمانی که در تابع write تابع sem_post صدا زده نشود، خط بعد اجرا نخواهد شد.

وقتی writer سمافور را پست کرد آن گاه reader ها به نوبت به shared memory دسترسی پیدا میکنند ودمقدار داخل آن را میخوانند و چاپ می‌کنند.

```
49 void reader(const char* readerName){
50     int shm = shm_open(NAME, O_RDWR, 0666);
51     struct Counter *memory_object = (struct Counter *)mmap(0, SIZE,
   PROT_READ | PROT_WRITE, MAP_SHARED, shm, 0);
52     while(1){
53         //comment wait -> race condition
54         sem_wait(&(memory_object->write_lock));
55         fprintf(stdout,"reader %s, value:%d\n", readerName, memory_object-
   >count);
56         fflush(stdout);
57         //comment post -> race condition
58         sem_post(&(memory_object->write_lock));
59         if((memory_object->count) > MAX_NUM){
60            exit(EXIT_FAILURE);
61         }
62     }
63 }
64
```

Main:

ابتدا حافظه را تعریف می‌کنیم. سپس با استفاده از fork دو فرزند برای خواندن و نوشتن تعریف می‌کنیم. با شرط‌هایی که قرار میدهیم در واقع با چک کردن خروجی تابع fork، اگر فرزند بود تابع مورد نظر را اجرا میکنیم. پس از اتمام کار تمامی پردازه‌ها به اندازه ۱ ثانیه صبر می‌کنیم و ابتدا خواننده‌ها و سپس نویسنده را kill می‌کنیم.

```
65 int main(){
66
67     shm_unlink(NAME);
68
69     int shm = shm_open(NAME, O_CREAT | O_RDWR, 0666);
70     ftruncate(shm, SIZE);
71     struct Counter *memory_object = (struct Counter *)mmap(0, SIZE,
   PROT_WRITE | PROT_READ, MAP_SHARED, shm, 0);
72     sem_init(&(memory_object->write_lock), 1, 3);
73     memory_object->count = 0;
74     munmap(memory_object, SIZE);
75     close(shm);
76
77
..
78     int writer_child = fork();
79     if(writer_child == 0) {
80         writer();
81         exit(0);
82     }
83
84     int reader1 = fork();
85     if(reader1 == 0) {
86         reader("reader1");
87         exit(0);
88     }
89
90     int reader2 = fork();
91     if(reader2 == 0) {
92         reader("reader2");
93         exit(0);
94     }
95
96     sleep(1);
97     kill(reader1, SIGKILL);
98     kill(reader2, SIGKILL);
99     kill(writer_child, SIGKILL);
100 }
101
```

- خروجی:

خروجی بدون **race condition** :

- همانطور که مشاهده می‌کنید مقداری که هر بار خواننده‌ها چاپ میکنند دقیقا برابر آخرین مقدار نوشته است.

```
reader reader1, value:0
reader reader1, value:0
reader reader2, value:0
reader reader2, value:0
reader reader2, value:0
reader reader1, value:0
reader reader1, value:0
reader reader1, value:0
reader reader2, value:0
reader reader2, value:0
reader reader2, value:0
reader reader1, value:0
reader reader1, value:0
reader reader2, value:0
reader reader1, value:0
reader reader2, value:0
writer writer1, new value:1
writer writer1, new value:2
writer writer1, new value:3
writer writer1, new value:4
writer writer1, new value:5
writer writer1, new value:6
writer writer1, new value:7
writer writer1, new value:8
writer writer1, new value:9
writer writer1, new value:10
writer writer1, new value:11
reader reader1, value:11
```