شرح آزمایش:

زمانی که فرآیند ها به صورت همزمان اجرا می شوند و منابع بین آن ها مشترک است احتمال بروز شرایط مسابقه وجود دارد که در آن برنامه الزاماً در هر بار اجرا، پاسخ یکسانی تولید نخواهد کرد. برای جلوگیری از این مساله، نیاز به همگام‌سازی است. در این آزمایش هدف بررسی بیشتر این مساله است.

بخش اول:

سمافور یک متغیر عدد صحیح است که از طریق دو عملیات اتمی wait() و signal() دسترسی یا تغییر می‌یابد. در زبان C، عملیات‌های مربوطه به ترتیب با sem_wait() و sem_post() انجام می‌شوند. در ادامه برنامه‌ای برای همگام‌سازی فرایندها با استفاده از سمافورها آورده شده است تا پیاده‌سازی sem_wait() و sem_post() برای جلوگیری از شرایط مسابقه درک شود.

برنامه زیر دو نخ ایجاد می‌کند یکی برای افزایش مقدار متغیر مشترک و دیگری برای کاهش مقدار آن. هر دو نخ از متغیر سمافور استفاده می‌کنند تا اطمینان حاصل شود که فقط یکی از نخ‌ها در بخش بحرانی خود در حال اجرا است.

```c
#include <pthread.h>
#include <stdio.h>
#include <semaphore.h>
#include <unistd.h>
void *fun1();
 void *fun2();
 int shared=1; //shared variable
 sem_t s; //semaphore variable
 int main()
 {
 sem_init(&s,0,1);
 //initialize semaphore variable - 1st argument is address of variable, 2nd is number of processes sharing semaphore, 3rd argument is the initial value of semaphore variable
 pthread_t thread1, thread2;
 pthread_create(&thread1, NULL, fun1, NULL);
 pthread_create(&thread2, NULL, fun2, NULL);
 pthread_join(thread1, NULL);
 pthread_join(thread2,NULL);
 printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable
 }
```

```
 void *fun1()
{
    int x;
    sem_wait(&s); //executes wait operation on s
    x=shared;//thread1 reads value of shared variable
    printf("Thread1 reads the value as %d\n",x);
    x++;  //thread1 increments its value
    printf("Local updation by Thread1: %d\n",x);
    sleep(1); //thread1 is preempted by thread 2
    shared=x; //thread one updates the value of shared variable
    printf("Value of shared variable updated by Thread1 is: %d\n",shared);
    sem_post(&s);
}

void *fun2()
{
    int y;
    sem_wait(&s);
    y=shared;//thread2 reads value of shared variable
    printf("Thread2 reads the value as %d\n",y);
    y--;  //thread2 increments its value
    printf("Local updation by Thread2: %d\n",y);
    sleep(1); //thread2 is preempted by thread 1
    shared=y; //thread2 updates the value of shared variable
    printf("Value of shared variable updated by Thread2 is: %d\n",shared);
    sem_post(&s);
}
```

مقدار نهایی متغیر مشترک برابر با ۱ خواهد بود. وقتی که هر یک از نخ‌ها عملیات wait را اجرا می‌کند، مقدار متغیر سمافور s به صفر می‌رسد. بنابراین، نخ دیگر (حتی اگر نخ در حال اجرا را موقتی از اجرا خارج کند) قادر نخواهد بود تا عملیات wait را روی s به طور موفقیت‌آمیز اجرا کند. به این ترتیب، نمی‌تواند مقدار ناسازگار متغیر مشترک را بخواند. این امر اطمینان می‌دهد که در هر لحظه فقط یکی از نخ‌ها در بخش بحرانی خود در حال اجرا است. خروجی برنامه به صورت زیر نشان داده شده است.

```
Thread1 reads the value as 1
Local updation by Thread1: 2
Value of shared variable updated by Thread1 is: 2
Thread2 reads the value as 2
Local updation by Thread2: 1
Value of shared variable updated by Thread2 is: 1
Final value of shared is 1
```

فرآیند، متغیر سمافور s را با استفاده از تابع sem_init() به ۱ مقداردهی اولیه می‌کند زیرا از سمافور باینری استفاده شده است. اگر چندین نمونه از منبع در دسترس باشد، می‌توان از سمافور شمارشی استفاده کرد. سپس، فرآیند دو نخ ایجاد می‌کند. thread1 متغیر سمافور را با فراخوانی sem_wait() به دست می‌آورد. سپس، دستورات در بخش بحرانی خود را اجرا می‌کند. از تابع sleep(1) استفاده می‌کنیم تا نخ thread1 را موقتی از اجرا خارج کنیم و نخ thread2 را شروع کنیم. این سناریو شبیه‌سازی یک شرایط واقعی است. اکنون، هنگامی که thread2 تابع sem_wait() را اجرا می‌کند، قادر نخواهد بود این کار را انجام دهد زیرا thread1 قبلاً در بخش بحرانی خود قرار دارد. در نهایت، thread1 تابع sem_post() را فراخوانی می‌کند. حالا thread2 می‌تواند با استفاده از sem_wait() متغیر s را به دست آورد. این امر همزمانی بین نخ‌ها را تضمین می‌کند.

تمرین:

برنامه‌ای بنویسید که همگام‌سازی بین چندین نخ را برقرار کند. نخ‌ها سعی می‌کنند به منبعی به اندازه ۲ دسترسی پیدا کنند. به سوالات زیر پاسخ دهید.

- مقدار اولیه متغیر سمافور چیست؟
- چرا از تابع pthread_join() در برنامه استفاده می‌کنیم؟
- چرا پارامتر چهارم در pthread_create() برابر با NULL است؟
- اهمیت استفاده از sleep(1) در توابع fun1 و fun2 چیست؟
- چگونه از سمافورهای شمارشی استفاده کنیم؟

بخش دوم: مساله خوانندگان-نویسندگان را پیاده سازی کنید.

بدین منظور فرض کنید دو فرآیند reader و یک فرآیند writer وجود دارند که به ترتیب به خواندن مقدار بافر یا به روزرسانی آن می پردازند. بین این فرآیند ها همانند روشی که در آزمایش قبل فراگرفتید یک حافظه مشترک در نظر بگیرید و در آن مقدار اولیه صفر را بنویسید. توجه داشته باشید که فرآیند writer دسترسی خواندن و نوشتن داشته باشد و فرآیند reader فقط دسترسی خواندن داشته باشد .

فرآیند writer با هر بار دسترسی به بافر مقدار موجود را یک واحد افزایش می دهد. writer بعد از دسترسی به بافر پیغامی چاپ می کند و در آن شماره فرآیند خودش(PID) و مقدار count را اعلام می کند.

هر reader نیز به طور مداوم مقدار بافر را می‌خواند و در پیغامی شماره فرآیند خودش و مقدار count را اعلام می کند. توجه داشته باشید که هر دو reader میتوانند با هم به بافر دسترسی داشته باشند.

شرط پایان این است که مقدار count به یک مقدار بیشینه دلخواه برسد.

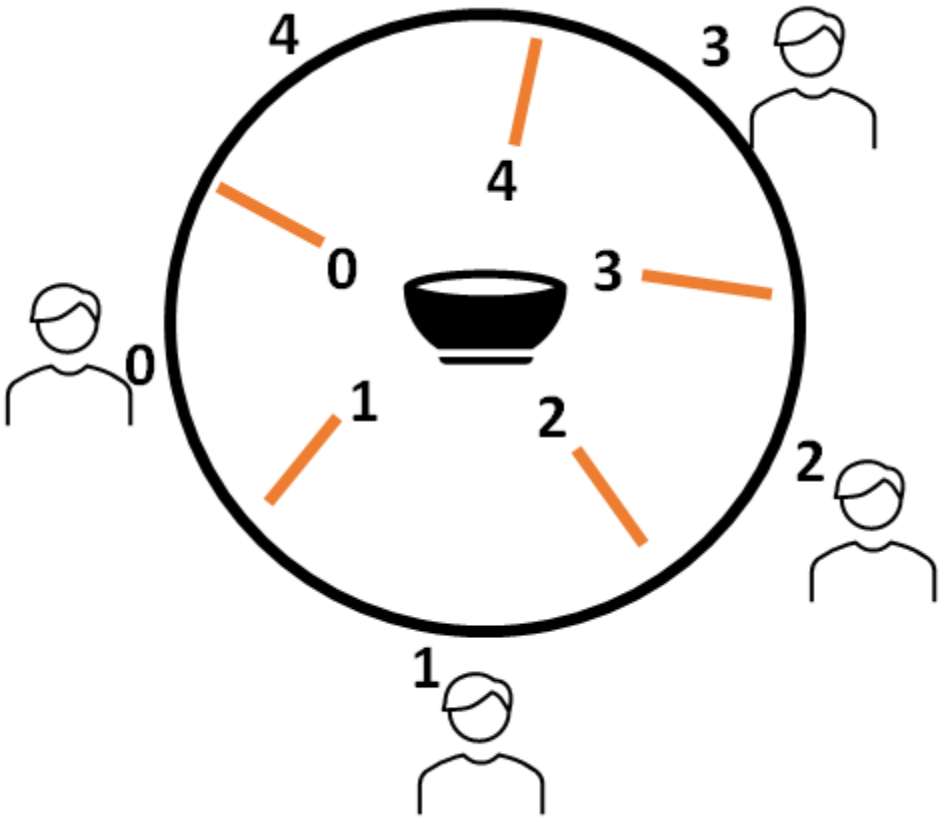
برنامه مربوطه را بصورت کامل نوشته و سپس اجرا کنید. آیا مشکلی وجود دارد؟ در صورت وجود ناهماهنگی چه راهکاری ارائه می کنید؟

راهنمایی: برای همگام سازی فرآیند های reader و writer می توانید از روش های همگام‌سازی استفاده کنید. در این صورت وقتی اولین reader به بافر دسترسی می‌یابد باید آن را lock کند و وقتی آخرین reader کارش تمام شد lock را رها میکند . فرآیند writer زمانی می تواند مقداری بنویسد که فرآیند reader به بافر دسترسی نداشته باشد و تا اتمام عملیات نوشتن، فرآیند readerقادر به خواندن نیست.

بخش سوم: مساله فیلسوف‌های غذاخور

این یک مساله کلاسیک در مبحث همگام‌سازی فرآیند ها است. این مسئله یک نمایش ساده از شرایطی است که تعدادی منبع در اختیار تعدادی فرآیند است و قرار است از پیش آمدن بن بست یا قحطی جلوگیری شود. میزی در نظر بگیرید که ۵ فیلسوف دور آن نشسته اند و ۵ چوب غذا برای غذا خوردن وجود دارد (بین هر دو صندلی یک چوب قرار دارد).

مسئله‌ی فیلسوفان غذاخور به این صورت است که پنج فیلسوف وجود دارند که دو کار انجام می‌دهند: فکر کردن و غذا خوردن. این فیلسوفان یک میز را به اشتراک می‌گذارند که برای هر کدام یک صندلی دارد. در مرکز میز، یک کاسه برنج قرار دارد و روی میز ۵ عدد چاپستیک (چوب غذاخوری) قرار داده شده است (به شکل زیر مراجعه کنید).



هدف این بخش، پیاده سازی این مسئله به زبان C است. بدین منظور هر چاپستیک را با یک سمافور نمایش بدهید.

```
sem_t chopstick[5];
```

تمرین: کد مسئله فیلسوفان غذاخور را پیاده‌سازی کنید و خروجی برنامه را به مدرس خود تحویل دهید.

خروجی کد شما می تواند مانند زیر باشد.

در اینجا فیلسوف (نخ) ۰ ابتدا سعی می‌کند غذا بخورد. بنابراین، او ابتدا سعی می‌کند چاپستیک سمت چپ را بردارد که موفق می‌شود. سپس چاپستیک سمت راست را برمی‌دارد. از آنجا که او هر دو چاپستیک را برداشته است، فیلسوف ۰ شروع به غذا خوردن می‌کند. اکنون، به تصویر ابتدایی آزمایش مراجعه کنید. اگر فیلسوف ۰ شروع به غذا خوردن کند، این به این معنی است که چاپستیک‌های ۰ و ۱ مشغول هستند، بنابراین فیلسوف‌های ۱ و ۴ نمی‌توانند غذا بخورند تا زمانی که فیلسوف ۰ چاپستیک‌ها را پایین بگذارد. حالا خروجی را بخوانید، فیلسوف بعدی می‌خواهد غذا بخورد. او سعی می‌کند چاپستیک سمت چپ (یعنی چاپستیک ۱) را بردارد، اما موفق نمی‌شود زیرا چاپستیک ۱ در حال حاضر در دست فیلسوف ۰ است. به همین ترتیب، می‌توانید بقیه خروجی را درک کنید.

سوال: آیا ممکن است بن بست رخ دهد؟ در صورت امکان چگونگی ایجاد آن را توضیح دهید.