دانشگاه صنعتی شاهرود
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

سیستم‌های عامل

استفاده از سمافور

استاد: علیرضا تجری

استفاده از سمافور

- مسأله فرآیند نویسنده و فرآیند خواننده
- مسأله تولیدکنندگان و مصرف‌کنندگان
- مسأله غذا خوردن فیلسوف‌ها
- مشکلات سمافور
- معرفی مانیتور

مسأله فرآیند نویسنده و فرآیند خواننده / تولیدکننده – مصرف کننده

- دو فرآیند داریم. فرآیند اول، مقداری را محاسبه می‌کند و برای فرآیند دوم می‌فرستد. فرآیند دوم، از آن مقدار استفاده می‌کند.
  - فرآیند اول: نویسنده / تولید کننده
  - فرآیند دوم: خواننده / مصرف کننده
  - یک متغیر مشترک / (حالت توسعه یافته: استفاده از یک بافر مشترک)
- تا زمانی که نویسنده مقدار جدیدی را در متغیر مشترک ننوشت، خواننده نمی‌تواند از مقدار آن متغیر استفاده کند.
- تا زمانی که خواننده از مقدار متغیر مشترک را نخواند، نویسنده نمی‌تواند مقدار جدیدی را در متغیر مشترک بنویسد.

مسأله فرآیند نویسنده و فرآیند خواننده

- ساختار اولیه کد نویسنده و خواننده

یک متغیر مشترک

فرآیند تولید کننده

```
while(1){
    تولید داده جدید
    نوشتن مقدار جدید در متغیر مشترک

}
```

فرآیند مصرف کننده

```
while(1){
    خواندن مقدار جدید از متغیر مشترک
    استفاده از مقدار جدید

}
```

مسأله فرآیند نویسنده و فرآیند خواننده

- ساختار اولیه کد نویسنده و خواننده

یک متغیر مشترک

فرآیند تولید کننده

```
while(1){
    تولید داده جدید
    نوشتن مقدار جدید در متغیر مشترک
```

فرآیند مصرف کننده

```
while(1){
    خواندن مقدار جدید از متغیر مشترک
    استفاده از مقدار جدید

}
```

باید برای این قسمت‌ها فکری کنیم.

مسأله فرآیند نویسنده و فرآیند خواننده

- ساختار اولیه کد نویسنده و خواننده

متغیر مشترک Data

در ابتدا، متغیر برای نوشتن آماده است.

فرآیند تولید کننده

```
while(1){
    تولید داده جدید
    آیا متغیر برای نوشتن آماده است؟
    نوشتن مقدار جدید در متغیر مشترک
    متغیر برای خواندن آماده است و
    برای نوشتن آماده نیست
}
```

فرآیند مصرف کننده

```
while(1){
    آیا متغیر برای خواندن آماده است؟
    خواندن مقدار جدید از متغیر مشترک
    متغیر برای نوشتن آماده است و
    برای خواندن آماده نیست
    استفاده از مقدار جدید
}
```

مسأله فرآیند نویسنده و فرآیند خواننده

متغیر مشترک Data

- ساختار کد نویسنده و خواننده

```
Semaphore canRead = 0;
Semaphore canWrite = 1;
```

فرآیند تولید کننده

```
while(1){
    compute new_value
    wait(canWrite);
    Data = new_value
    signal(canRead);
}
```

فرآیند مصرف کننده

```
while(1){
    wait(canRead);
    new_value = Data;
    signal(canWrite);
    use new_value
}
```

مسأله نویسندگان و خوانندگان / تولیدکنندگان و مصرف‌کنندگان

- چند فرآیند نویسنده و چند فرآیند خواننده داریم.
  - تعدادی داده مشترک
  - مثل نرم‌افزارهای مدیریت پایگاه داده
- در هر لحظه،
  - فقط یک نویسنده می‌تواند در داده‌های مشترک بنویسد.
  - چند خواننده می‌توانند داده‌های مشترک را بخوانند.
- سؤال: اگر در یک لحظه چند خواننده و چند نویسنده بخواهند از داده‌های مشترک استفاده کنند، اولویت با کدام است؟

مسأله نویسندگان و خوانندگان / تولیدکنندگان و مصرف‌کنندگان

- سؤال: اگر در یک لحظه چند خواننده و چند نویسنده بخواهند از داده‌های مشترک استفاده کننده، اولویت با کدام است؟
  - مسأله اول خواننده: اولویت با خواننده‌ها است.
    - تا زمانی که یک خواننده وجود دارد، نویسندگان نمی‌توانند بنویسند.
  - مسأله اول نویسنده: اولویت با نویسنده‌ها است.
    - تا زمانی که یک نویسنده وجود دارد، خوانندگان نمی‌توانند بخوانند.
- در هر دو حالت، گرسنگی به وجود می‌آید.

مسأله نویسندگان و خوانندگان / تولیدکنندگان و مصرف‌کنندگان

مسأله اول خواننده

- تعدادی متغیر مشترک وجود دارد.
- برای دسترسی به متغیرهای مشترک، باید تعداد خواننده‌های فعال را بدانیم.
  - تعداد خواننده‌های فعال هم یک متغیر مشترک است.
  - دسترسی به این متغیر مشترک هم نیاز به حفاظت دارد.
- از دو سمافور استفاده می‌کنیم. یک برای محافظت از متغیرهای مشترک و دیگری برای محافظت از متغیر تعداد خواننده‌ها.

مسأله نویسندگان و خوانندگان / تولیدکنندگان و مصرف‌کنندگان

- سمافورها و متغیرهای مورد نیاز

```
Semaphore rw_mutex = 1;
Semaphore mutex = 1;
int readCount = 0;
```

  - سمافور rw_mutex: برای محافظت از متغیرهای مشترک اصلی
  - سمافور mutex: برای محافظت از متغیر readCount
  - متغیر readCount: تعداد خواننده‌ها
- نکته مهم: فقط زمانی که **readCount** برابر با 0 است، فرآیندهای نویسنده می‌توانند بنویسند.

مسأله نویسندگان و خوانندگان / فرآیندهای نویسنده

```
Semaphore rw_mutex = 1;
Semaphore mutex = 1;
int readCount = 0;
```

```
do{

   wait(rw_mutex);

   نوشتن منبع مشترک

   signal(rw_mutex);

}while(true);
```

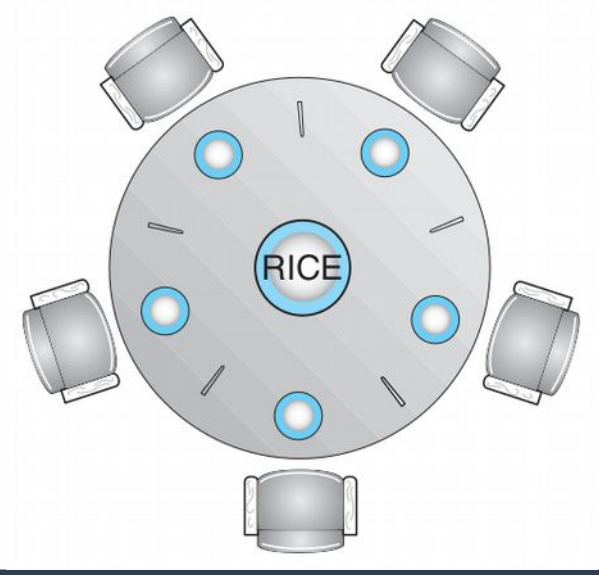
مسأله نویسندگان و خوانندگان / فرآیندهای خواننده

```
Semaphore rw_mutex = 1;
Semaphore mutex = 1;
int readCount = 0;
```

```
do{
    wait(mutex);
    readCount++;
    if(readCount == 1){
        wait(rw_mutex);
    }
    signal(mutex);

    خواندن منبع مشترک

    wait(mutex);
    readCount--;
    if(readCount == 0){
        signal(rw_mutex);
    }
    signal(mutex);

}while(true);
```

مسأله غذا خوردن فیلسوف‌ها

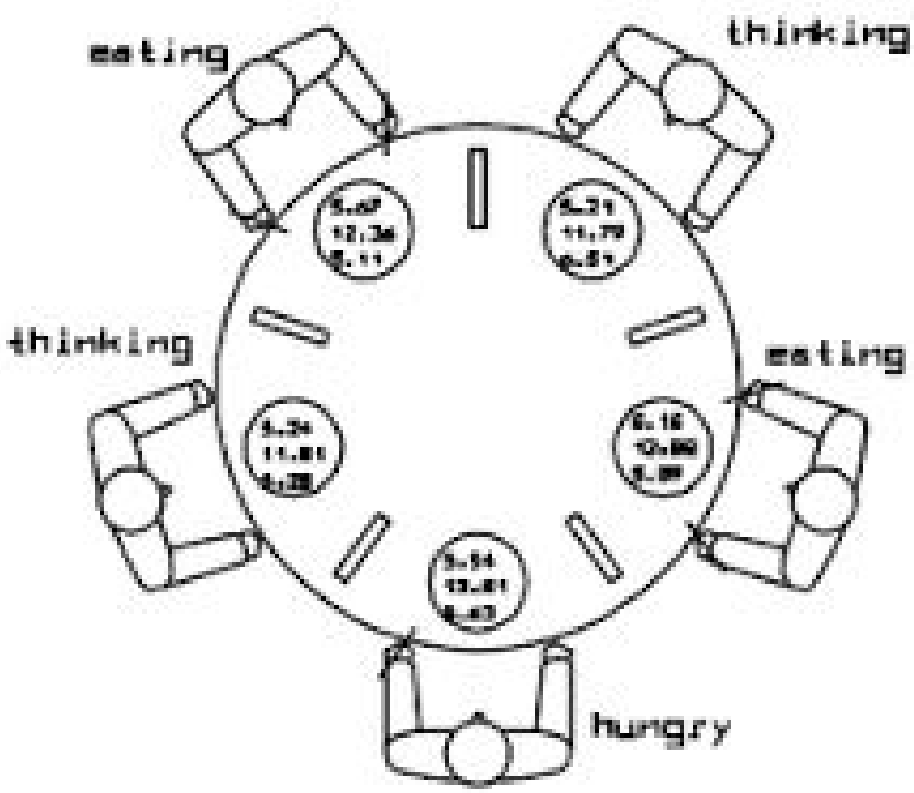
- ۵ فیلسوف دور یک میز نشسته‌اند و می‌خواهند غذا بخورند.
- هر فیلسوف برای غذا خوردن به دو چوب نیاز دارد.
- هر فیلسوف ابتدا چوب سمت راست را بر می‌دارد و سپس چوب سمت چپ را بر می‌دارد و غذا می‌خورد. سپس فکر می‌کند.
- فیلسوف: فرآیند
- غذا: منابع



مسأله غذا خوردن فیلسوف‌ها

- مدل کردن مسأله: برای هر چوب یک سمافور در نظر می‌گیریم.

```
semaphore chopstick[5];
```



مسأله غذا خوردن فیلسوف‌ها / گرسنگی دارد

```
do {
   wait(chopstick[i]);
   wait(chopstick[(i+1) % 5]);
      . . .
   /* eat for awhile */
      . . .
   signal(chopstick[i]);
   signal(chopstick[(i+1) % 5]);
      . . .
   /* think for awhile */
      . . .
} while (true);
```

مشکلات سمافور

- طراحی راه‌حل نیاز به دقت فراوان دارد.
- فرض کنید که یک مسأله وجود دارد که برای راه‌حل آن باید چند فرآیند همکار نوشته شود. هر فرآیند را یک نفر می‌نویسد.
- اگر یک نفر، کد اشتباه بنویسد، بر روی کد بقیه افراد هم اثر می‌گذارد.

```
wait(S);
...
signal(S);
```

```
wait(S);
...
wait(S);
```

```
signal(S);
...
wait(S);
```

```
signal(S);
...
signal(S);
```

معرفی مانیتور

- یک راه حل دیگر برای همگام سازی فرآیندها (نخ‌ها)
- توسط سیستم‌عامل پشتیبانی می‌شود.
- معمولاً زبان‌های برنامه‌نویسی از آن پشتیبانی می‌کنند.
  - قابلیت پیاده‌سازی با کمک سمافور
- یک کلاس است که در هر لحظه، فقط یک نسخه از تابع آن اجرا می‌شود.
- این کلاس بین فرآیندها به اشتراک گذاشته می‌شود.
- مثال: توابع **synchronized** در جاوا