دانشگاه صنعتی شاهرود
دانشکده مهندسی کامپیوتر و فناوری اطلاعات

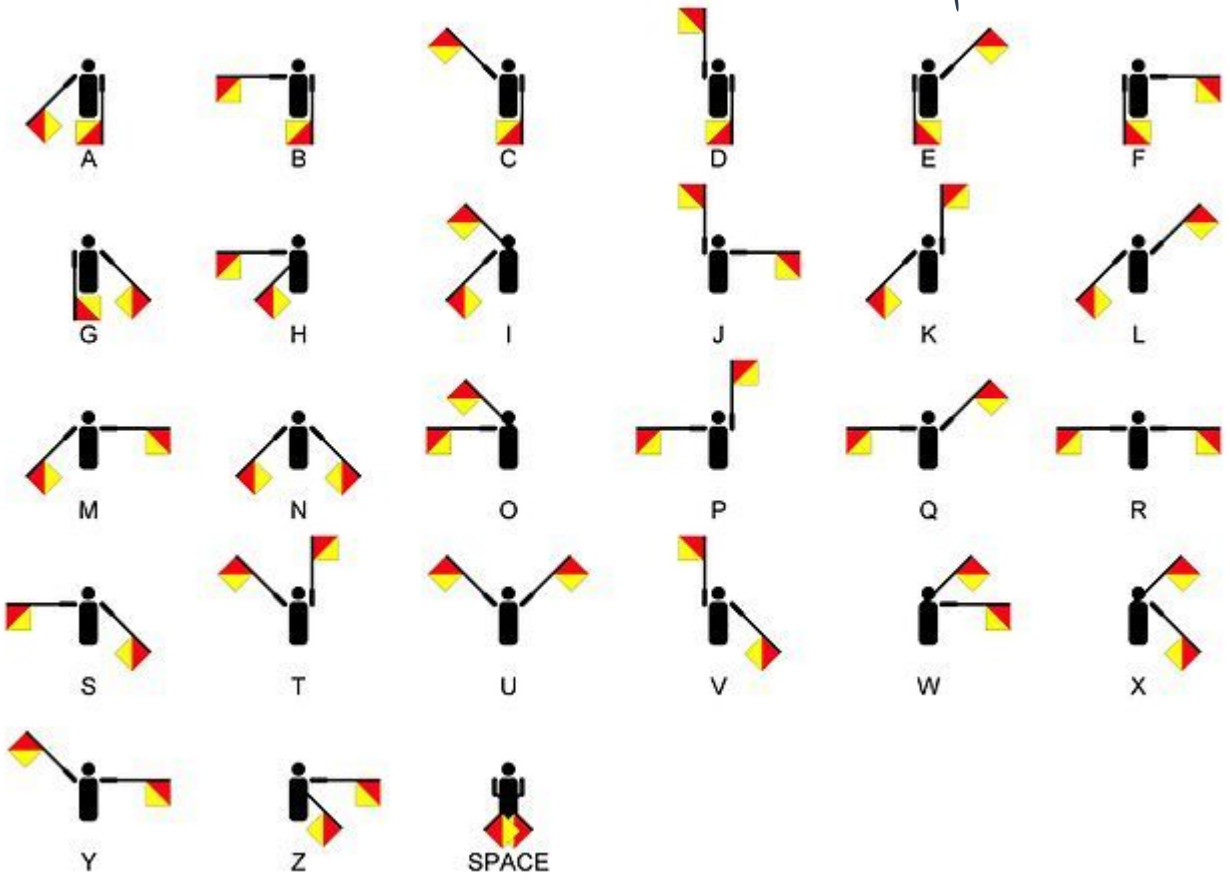
سیستم‌های عامل

معرفی سمافور (Semaphore)

استاد: علیرضا تجری

سمافور Semaphore – قبل از بحث سیستم‌عامل

- یک روش ارسال پیام بین دو شخص
  - با کمک علامت (پرچم)



سمافور در سیستم‌عامل

- مکانیزمی برای هماهنگی فرآیندها (یا نخ‌ها)
- یک راه‌حل برای مسأله ناحیه بحرانی
- این راه‌حل توسط سیستم‌عامل پشتیبانی می‌شود.
  - تقریباً همه سیستم‌عامل‌ها
  - تقریباً همه زبان‌های برنامه‌نویسی

سمافور در سیستم‌عامل

- یک متغیر مشترک به نام سمافور (S)
  - از نوع Integer
- دو تابع که روی این متغیر عملیات انجام می‌دهند.

```
wait (S){
   while(S<=0);
   S--;
}
```

```
signal (S){
   S++;
}
```

سمافور در سیستم‌عامل

- دو تابع **wait** و **signal**، توابع اتمیک هستند.
  - کل تابع اجرا می‌شود / کل تابع اجرا نمی‌شود.
  - چطور امکان‌پذیر است؟
    - با غیر فعال کردن وقفه‌ها در هنگام اجرای کد این توابع
  - در پیاده‌سازی واقعی
    - در تک پردازنده‌ها

سمافور در سیستم‌عامل

- انواع سمافور
  - سمافور شمارنده
    - متغیر S می‌تواند هر مقدار صحیحی را بگیرد.
  - سمافور باینری
    - متغیر S فقط می‌تواند مقادیر 0 و یا 1 را بگیرد.
      - شبیه به mutex

محافظت از ناحیه بحرانی با کمک سمافور

- یک متغیر مشترک سمافور **S** بین فرآیندها تعریف می‌شود.
  - مقدار اولیه آن، 1 است.
- در بخش ورود، از **wait(S)** استفاده می‌شود.
- در بخش خروج، از **signal(S)** استفاده می‌شود.

```
do {
        entry section
                critical section
        exit section
                remainder section
} while (true);
```

محافظت از ناحیه بحرانی با کمک سمافور

```
S = 1
```

P0

```
do{
    wait(S);
      critical section
    signal(S);
    remainder section
}while(true);
```

P1

```
do{
    wait(S);
      critical section
    signal(S);
    remainder section
}while(true);
```

محافظت از ناحیه بحرانی با کمک سمافور

```
S = 1
```

P0

```
do{
    wait(S);
      critical section
    signal(S);
    remainder section
}while(true);
```

P1

```
do{
    wait(S);
      critical section
    signal(S);
    remainder section
}while(true);
```

```
wait (S){
    while(S<=0);
    S--;
}
```

```
signal (S){
    S++;
}
```

سمافور شمارنده

- برای دسترسی به منبعی که تعدادی از آن وجود دارد، از این سمافور استفاده می‌شود.
- مقدار اولیه سمافور، تعداد منابع است.
- هر فرآیندی که می‌خواهد از منبع استفاده کند، روی آن **wait** می‌کند.
  - با اینکار، مقدار سمافور یک واحد کاهش می‌یابد.
- هر فرآیندی که می‌خواهد منبع را آزاد کند، آن را **signal** می‌کند.
  - با اینکار، مقدار سمافور یک واحد افزایش می‌یابد.
- وقتی مقدار سمافور به 0 برسد، یعنی همه منابع در حال استفاده هستند.
  - فرآیند جدید که می‌خواهد از منبع استفاده کند (با فراخوانی wait) باید صبر کند.

هماهنگ کردن فرآیندها با کمک سمافور

- فرض کنید
  - فرآیند p1 باید تابع p1Func را اجرا کند.
  - فرآیند p2 باید تابع p2Func را اجرا کند.
  - تابع p2Func باید بعد از تابع p1Func اجرا شود.
- چطور اینکار را انجام بدهیم؟

p1

```
…
p1Func();
…
```

p2

```
…
p2Func();
…
```

هماهنگ کردن فرآیندها با کمک سمافور

- چطور اینکار را انجام بدهیم؟
  - با کمک یک سمافور

```
synch = 0
```

p1

```
…
p1Func();
signal(synch);
…
```

p2

```
…
wait(synch);
p2Func();
…
```

پیاده‌سازی واقعی سمافور / در پردازنده‌های تک هسته‌ای

انتظار مشغولی ندارد!

و

خاصیت انتظار محدود را دارد!

پیاده‌سازی واقعی سمافور

```
typedef struct {
        int value;
        struct process *list;
} semaphore;
```

پیاده‌سازی واقعی سمافور

```
wait(semaphore *S) {
        S->value--;
        if (S->value < 0) {
                add this process to S->list;
                block();
        }
}
```

فراخوانی سیستمی

باعث می‌شود که فرآیند از حالت running
به حالت waiting برود.

پیاده‌سازی واقعی سمافور

```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

فراخوانی سیستمی

باعث می‌شود که فرآیند از حالت waiting
به حالت ready برود.

بن‌بست Deadlock

- فرض کنید که مقدار اولیه **Q** و **S**، برابر با 1 است.
- این دو فرآیند به چه صورتی اجرا می‌شوند؟

```
S = 1;
Q = 1;
```

P1

```
wait(S);
wait(Q);

…

signal(S);
signal(Q);
```

P2

```
wait(Q);
wait(S);

…

signal(S);
signal(Q);
```

بن‌بست Deadlock

چند فرآیند در بن‌بست هستند،

اگر هر یک منتظر رخ دادن رخدادی باشد

که در دیگری باید انجام شود.

بن‌بست Deadlock

\- اول تو قطع کن
\+ نه، اول تو قطع کن

گرسنگی Starvation قحطی

گرسنگی زمانی رخ می‌دهد که

یک فرآیند هیچ وقت وارد ناحیه بحرانی نشود.

(بقیه فرآیندها وارد ناحیه بحرانی می‌شوند)