操作系统及Linux内核

西安邮电大学

信号量简介

信号量(Semaphore)

信号量是荷兰学者Dijkstra在1965年提出一种卓有成效的进程同步机制，为表示临界资源的实体，广泛应用于单处理机、多处理机及计算机网络中解决进程同步与互斥问题。

- 整型信号量
- 记录型信号量
- AND型信号量
- 信号量集

整型信号量

信号量定义为一个整型量 S；S≤0表示该资源已被占用； S>0表示资源可用。

进程企图进入临界区时， S<=0禁止进入， S>0可以进入。

```
//申请资源操作原语
wait(int &S){  //P(S)
    while (S<=0);
    S = S-1;
}
```

```
//释放资源操作原语
signal(int &S){ //V(S)
   S = S+1;
}
```

- wait 操作 ---- P操作
- signal操作 ---- V操作

整型信号量--应用示例

```
wait(int &S){  //P(S)
    while (S<=0);
    S = S-1;
}
```

```
signal(int  &S){ //V(S)
    S = S+1;
}
```

存在问题：S<=0时，会让进程(线程)处于“忙待”状态。

```
int S = 1;  //信号量
……
void *p1(void *p){
    ……
    wait(S);
    Critical Section;
    signal(S);
    ……
}
```

```
void *p2(void *p){
    ……
    wait(S);
    Critical Section;
    signal(S);
    ……
}
```

记录型信号量

```
typedef struct{
    int             value;
    list<PROC>   L;
}semaphore;
```

- value>0时，value为资源可用数目
- value<0，|value|为已阻塞进程的数目
- L为阻塞进程队列首指针

```
wait(int &S){  //P(S)
    S.value = S.value -1;
    if (S.value<0);
        block(S.L);  //阻塞到队尾，
   //程序计数器定位在wait之后
}
```

```
signal(int &S){ //V(S)
    S.value = S.value+1;
    if (S.value<=0);
        wake(S.L);  //唤醒队首
}
```

记录型信号量-应用示例

```
wait(int &S){  //P(S)
    S.value = S.value -1;
    if (S.value<0);
        block(S.L);  //阻塞到队尾
}
```

```
signal(int &S){ //V(S)
    S.value = S.value+1;
    if (S.value<=0);
        wake(S.L);  //唤醒队首
}
```

```
semaphore S;   S.value = 1;  ……
void *p1(void *p){
    ……
    wait(S);
    Critical Section;
    signal(S);
    ……
}
```

```
void *p2(void *p){
    ……
    wait(S);
    Critical Section;
    signal(S);
    ……
}
```

AND型信号量

申请n种资源，每种1个，或者全部分配，或者不分配，避

```
Swait(semaphore S_1, …, semaphore S_n){
    if (S_1.value>=1 && … && S_n.value>=1)
        for (int i=1; i<=n; i++)   S_i.value = S_i.value –1;
    else{
        for (int i=1; i<=n && S_i.value>=1; i++); //寻找第1个不满足资源
        [自我阻塞到S_i.L中，将程序计数定位到Swait操作起点] }
}
```

```
Ssignal(semaphore S_1, …, semaphore S_n){
    for (int i=1; i<=n; i++)  {
        S_i.value = S_i.value +1;   [唤醒S_i.L中所有阻塞进程] }
}
```

信号量集

申请n种资源，资源S_i(1<=i<=n)申请d_i个资源且资源个数

```
Swait(semaphore S_1,int t_1,int d_1,…, semaphore S_n,int t_n,int d_n){
   if (S_1.value>=t_1 && … && S_n.value>=t_n)
      for (int i=1; i<=n; i++)   S_i.value = S_i.value – d_i;
   else{
      for (int i=1; i<=n && S_i.value>=t_i ; i++); //寻找第1个不满足资源
      [阻塞到S_i.L中，程序计数定位到Swait操作起点] }
}
```

```
Ssignal(semaphore S_1, int d_1 , …, semaphore S_n, int d_n){
    for (int i=1; i<=n; i++)  {
        S_i.value = S_i.value +d_i;   [唤醒S_i.L中所有阻塞进程 ] }
}
```

信号量集—特殊情况

- Swait(S_1, 1, 1, …, S_n, 1, 1)，即所有t_i、d_i都等于1：

  等价于AND信号量的Swait(S_1, …, S_n)。

- Swait(S, 1, 1)：退化为记录型信号量，但执行效率低。

- Swait(S, 1, 0)：特殊信号量，S.value>=1时允许多进程进入临界区；S<=0时阻止所有进程进入临界区，相当于一个开关 。

记录型信号量VS信号量集

记录型信号量	信号量集
1种资源S	多种资源S_1, …, S_n
每次申请1个	每次每种申请d_i个
S.value<1不分配（阻塞）	S_i.value少于t_i不分配
先减1后判断	先判断，后减d_i(1≤i≤n)
自我阻塞后程序计数定位到wait()操作之后	阻塞到S_i.L(第1个不满足)，程序计数定位到Swait()起点
signal唤醒第1个阻塞进程	Ssignal唤醒S_i.L(1≤i≤n)所有阻塞进程

信号量机制的应用

解决同步与互斥问题的基本策略

① 在问题域里寻找临界资源，可能是**物理资源**，如打印机、扫描仪等，也可能是**逻辑资源**，如共享变量、共享文件、控制信号等；

② 对于每一类临界资源，定义一个信号量，信号量的初值为资源的数量；

③ 使用临界资源前先通过wait(S)申请资源，其中S为资源对应的信号量；

④ 访问完（新生成）临界资源后，通过signal(S)释放一个临界资源。

信号量机制的应用

Linux的信号量 semaphore.h

- ✓ **定义信号量：** sem_t sem;
- ✓ **信号量初始化：**

int sem_init(sem_t *sem, int pshared, unsigned int value)

形参：

sem：待初始化信号量变量地址；

pshared：信号量能否在多进程间共享，只能取0；

value ：信号量初始化值。

- ✓ **wait、signal操作：**

sem_wait(sem_t *sem) ; sem_post(sem_t *sem) ;

- ✓ **销毁信号量：** sem_destroy(sem_t *sem) ;

信号量机制的应用

应用实例：苹果--橘子问题

桌上有一个空盘子，爸爸专向盘中放苹果，妈妈专向盘中放橘子，儿子专等吃盘中的橘子，女儿专等吃盘中的苹果。规定当盘子空时，一次只能放一个水果。

- **找临界资源：**盘子、苹果和桔子。
- **定义信号量：**

plate： 表示盘子，开始只有1个，初值为1
apple： 表示苹果，开始不存在，初值为0
orange： 表示桔子，开始不存在，初值为0

信号量机制的应用

```
void *father(void *p){
    while(1){
        sem_wait(&plate);
        Father puts apple…
        sem_post(&apple);
    } return NULL;
}
```

```
void *mother(void *p){
    while(1){
        sem_wait(&plate);
        Mother puts orange…
        sem_post(&orange);
    } return NULL;
}
```

```
void *daughter(void *p){
    while(1){
        sem_wait(&apple);
        Daughter gets apple…
        sem_post(&plate);
    } return NULL;
}
```

```
void *son(void *p){
    while(1){
        sem_wait(&orange);
        Son gets orange…
        sem_post(&plate);
    } return NULL;
}
```

小结

信号量机制

定义

- 表示资源的实体，操作系统支持

类型

- 整型信号量
- 记录型信号量
- ADD型信号量
- 信号量集

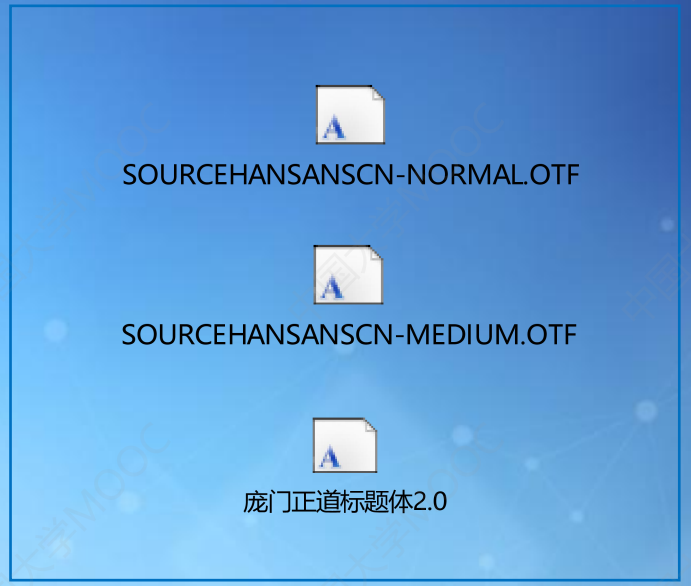
策略

- 寻找临界资源
- 定义信号量
- 使用前申请
- 使用后释放

字体：中文：思源黑体≥24
英文：新罗马 ≥24

配色

层级  文件管理



特殊字体双击安装