SYNCHRONIZÁCIA V OS

Princípy operačných systémov

Vlákna

- "odľahčený" proces
- každý proces má aspoň jedno vlákno
- zdieľajú adresný priestor procesu
- nezdieľané informácie:
  - Sada aplikačných registrov
  - Program Counter register
  - Zásobník
- informácie o vlákne uložené v TCB
- User-Level vs. Kernel-Level vlákna

Vlákna - modely

- Mapovanie vlákien medzi User Space a Kernel Space
- One-to-One model
- Many-to-One model
- Many-to-Many model

Vlákna - pozitíva

- výborný reakčný čas
- zdieľanie systémových prostriedkov
- nízky overhead
- škálovateľnosť

Vlákna - negatíva

- nevyhnutnosť synchronizácie
- komplexnosť - komplikácie napr. pri ladení
- súboj o systémové prostriedky

Základné pojmy synchronizácie

- Kritická sekcia (critical section) – sekcia (miesto v pamäti, alebo prístup k prostriedku) kde je potrebné koordinovať prístup pre zabezpečenie integrity systému.
- Súbeh (race condition) - kritický stav v počítačových systémoch, pri ktorom správanie sa systému je závislé od nedostatočne kontrolovaných udalostí. Príklady:
  - Viacúlohové systémy
  - Distribuované systémy
  - Súborové systémy

Synchronizácia - úvod

- problém s príkazom priradenia pri prepínaní kontextu:

```
int lock = 0;
```

```
while (1) {
    if (lock == 0) {
        lock = 1;
        // critical section
        lock = 0;
    }
}
```

```
while (1) {
    if (lock == 0) {
        lock = 1;
        // critical section
        lock = 0;
    }
}
```

Klasický synchronizačný algoritmus

```
int turn;
```

Thread 0:

```
turn = 1;
while (turn == 1) { // busy wait }
// critical section
...
// end of critical section
```

Thread 1:

```
turn = 0;
while (turn == 0) { // busy wait }
// critical section
...
// end of critical section
```

Petersnovo riešenie

```
bool flag[2] = {false, false};
int turn;
```

Thread 0:

```
flag[0] = true;
turn = 1;
while (flag[1] && turn == 1) { // busy wait }
// critical section
...
// end of critical section
flag[0] = false;
```

Thread 1:

```
flag[1] = true;
turn = 0;
while (flag[0] && turn == 0) { // busy wait }
// critical section
...
// end of critical section
flag[1] = false;
```

Test and Set (atomic instruction)

```
int TestAndSet(int &lock)
{
    int original = *lock;
    *lock = 1;
    return original;
}
```

```
int lock = 0;

while (TestAndSet(&lock) == 1) { /* Busy wait */ }
// Critical section
lock = 0;
```

Fetch and Add (atomic instruction)

```
int FetchAndAdd(int &lock, int inc)
{
    int original = *lock;
    *lock = original + inc;
    return original;
}

int box     = 0;
int service = 0;

int ticket  = FetchAndAdd(&box, 1);

while (service != ticket) { /* WAIT */ }
// critical section
FetchAndAdd(&service, 1);
```

Compare and Swap (atomic instruction)

```
boolean CompareAndSwap(int &M, int old, int new) {
    if (*M != old) {
        return false;
    }
    *M = new;
    return true;
}

OPENED = 0;
LOCKED = 1;
int lock = OPENED;

while (CompareAndSwap(&lock, OPENED, LOCKED) == false) { /* WAIT */ }
// critical section
CompareAndSwap(&lock, LOCKED, OPENED);
```

Semafór P=wait; V=signal;

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

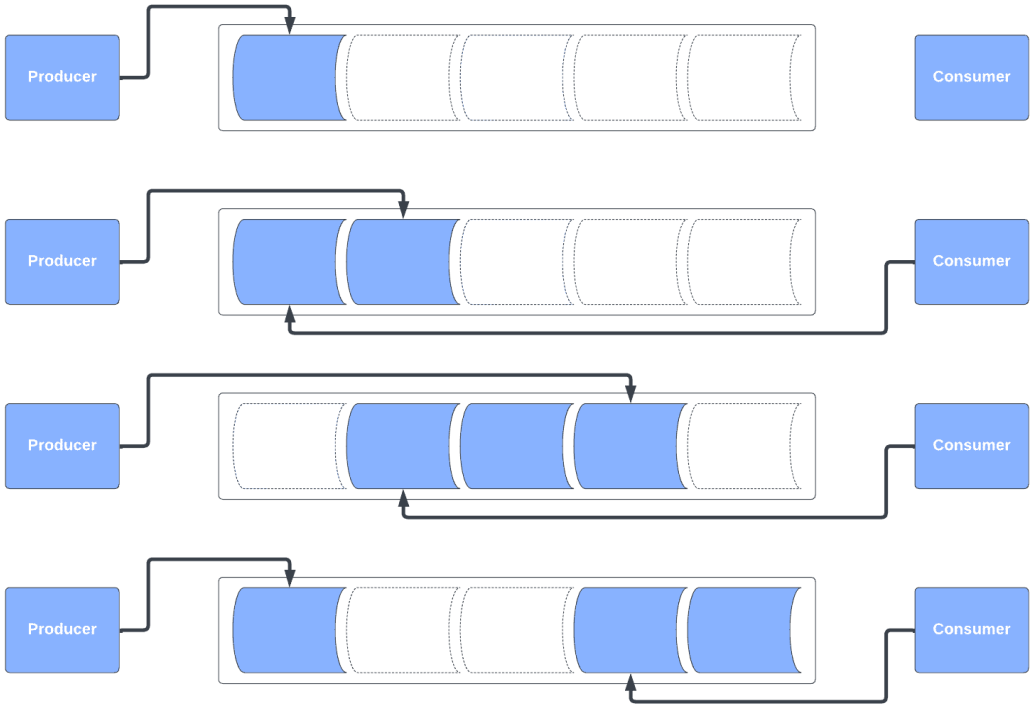
```
signal(S) {
    S++;
}
```

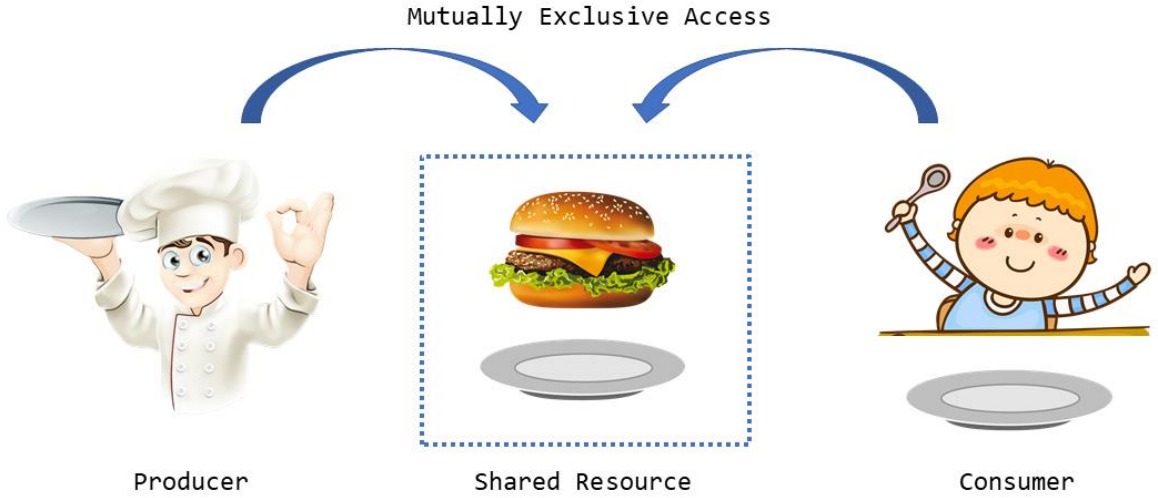
```
mutex M;

wait(&M);
// critical section
signal(&M);
```

Limitácie:

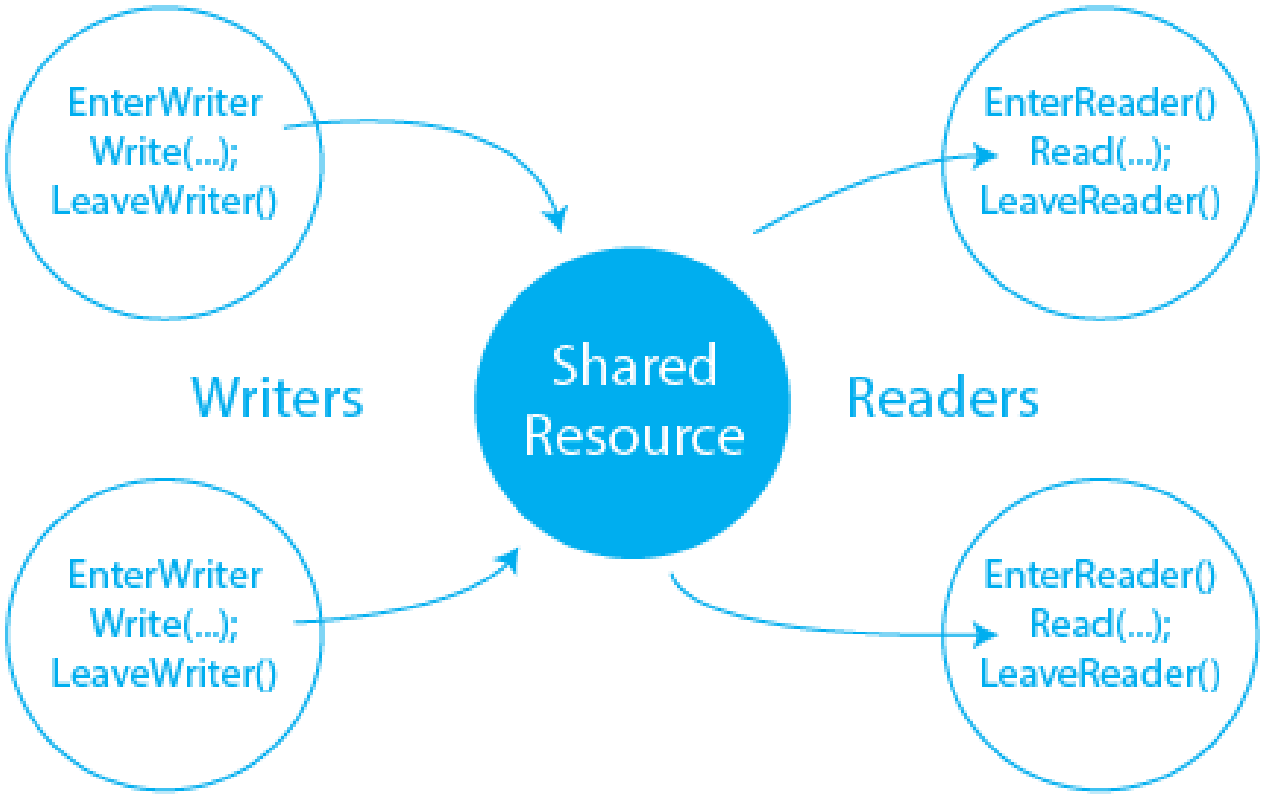
- Inverzia priorít
- Deadlock
- Zložitosť implementácie

Producent a konzument





Zapisovateľ a čitatelia



Stolujúci filozófovia