



Operating Systems

Synchronization Tools-Part2

Seyyed Ahmad Javadi

sajavadi@aut.ac.ir

Fall 2021

Interrupt-based Solution

اولین کاربر از نمایشگر
critical section

- Entry section: disable interrupts
- Exit section: enable interrupts

→ In progress, mutual Exclusion

نیز! حون عکس
و مخواه
و باره p2 دید
و مخواه
نیز! فوت نیز! فوت
نیز! فوت نیز! فوت
نیز! فوت نیز!

- Will this solve the problem?
 - What if the critical section is code that runs for an hour?

- در این صورت - می تواند در طی ساعت میانه یک اینترکپت را در یک ساعت از نیز! فوت نیز! فوت نیز! فوت نیز!
- Can some processes starve -- never enter their critical section.

- What if there are two CPUs?

! بازگشت از
critical section → core 1 داشت
و مخواه از p1 و p2
جنون می داشت
و مخواه از p3 و p4
جنون می داشت

چندرآمد بود!
چند روز می خواست!



Software Solution 1

اولین راه حل نرم افزاری

- Two process solution.
متد برای دو
- Assume that the load and store machine-language instructions are **atomic**; that is, cannot be interrupted.
معنی دلک درست، احتمال
آنکه در میان این
عملیات خود بگیرد،
نداشته باشد!
- The two processes share one variable:
 - int ***turn***;
 - ***turn*** indicates whose turn it is to enter the critical section.

Algorithm for Process P_i

```
while (true) {
```

```
    while (turn == j);
```

! نیازی به store, load
سپاهیا turn = j باید باشد
(نیازی نیست) ! سه کوتاه

```
/* critical section */
```

```
turn = j;
```

```
/* remainder section */
```

```
}
```



Algorithm for P₀ and P₁

Initially turn = 0

turn=0 دارای CPU، P0 و P1 می باشد و قسم
برای تردد و مقدم دارد اما ممکن نهاده کنند
برای اینکه در یک قسم خود را در یک قسم خود را در گیری
برای اینکه در یک قسم خود را در یک قسم خود را در گیری

while (TRUE) {

entry section ← while (turn != 0) ; /* loop */;
critical_region();
exit section ← turn = 1;
noncritical_region();
}

(a)

(a) Process 0.

while (TRUE) {

while (turn != 1) ; /* loop */;
critical_region();
turn = 0;
noncritical_region();
}

(b)

(b) Process 1.

برای برآوردن دسترسی 'proc' بسیار سخت است

!while (turn != 0) { deny ↓ → grant ↓ → critical section
CPU is busy waiting دسترسی ممکن نیست

Correctness of the Software Solution

■ Mutual exclusion is preserved

- P_i enters critical section only if:

$turn = i$

البيت این برای 2 بوده است!
برای چند بوده ممکن نیست!

- turn cannot be both 0 and 1 at the same time

محل دست برای استارتم → turn !
محل دست برای استارتم → turn !

■ It **wastes** CPU time

- So we should avoid busy waiting as much as we can.

■ Can be used only when the waiting period is expected to be **short**.

busy waiting ← قدرت حاصل کردن
critical section
(busy waiting) queue ← سرویس دهنده از آن
آن

Correctness of the Software Solution (cont.)

! Obj \neq multi Exclusion

- However there is a problem in the above approach!

- What about the **Progress requirement?**

نیاز
برای این خواهد

- What about the **Bounded-waiting requirement?**

دارد
نیاز دوچرخه برای
از لختهای سرورها
کار می‌شوند و بروج کوئی
bound = 1



Correctness of the Software Solution (cont.)

- P_0 leaves its critical region** and sets turn to 1, enters its non-critical region.
- P_1 enters its critical region**, sets turn to 0 and leaves its critical region.
- P_1 enters its non-critical region**, quickly finishes its job and goes back to the while loop.

بازه می فرود و (در)
critical section

بدانه ترتیب بایستی همگزینی نداشته باشد (No race condition)

- Since turn is 0, process 1 **has to wait** for process 0 to finish its non-critical region so that it can enter its critical region.

بدانه حالات ممکن طرح ای، ناجیی →
جگلی خود رفته است اما باز هم بقیع نہیں
P1 شرط دارد ناجیی جگلی بچے! ⇐ progress ندارد!
- This violates the **second condition (progress)** of providing mutual exclusion.

P_0

Initially turn = 0

```
while (TRUE) {
    while (turn != 0)
        critical_region();
    turn = 1;
    noncritical_region();
}
```

P_1

```
while (TRUE) {
    while (turn != 1)
        critical_region();
    turn = 0;
    noncritical_region();}
```



How About this solution?

```
//Algorithm for Pi  
while (true) {
```

```
{turn = i;  
while (turn == j);  
    ذرت اون لکی
```

→ entry section

```
/* critical section */
```

turn = j;

→ exit section
info = 1 condition = 0
البيانات كلها متساوية

```
/* remainder section */
```

}



Peterson's Solution

- The previous solution solves the problem of one process blocking another process while its outside its critical section.
- Peterson's Solution is a neat solution with busy waiting, that defines the procedures for entering and leaving the critical region.

وَآنْ مُرْجِعِي 8 - دَوْتَابُولَدَاهَ دَارِيمْ.
! atomic store, load - 8

اندیشی *bounded waiting* نزاره ! نیرا بوطره ها به پتروسون تاسی پری برای اجرای (critical section) طرد ! اینجری هکنه کی تاسی باجی بتری طرد ! صدای صدم 1 دخوت آفر
لو بده ! (مچ انبات ریاضی نزاره برای اینکه از زمانی که مد درزش به 5 زمانه که 10 بید بیرون و خوبت مد بیم رفع انبات و چور نزد داده بگیری صدم 10 po
بلی میگیرم بده !)

دلی اون در کادلیرو دره !
انبات بلد باشی !

Peterson's Solution (cont.)

- Two process solution
- Assume that the **load** and **store** machine-language instructions are **atomic**; that is, cannot be interrupted.
- The two processes share two variables:
 - int turn;
 - boolean flag[2]
- The variable **turn** indicates whose turn it is to enter the critical section.
- The **flag array** is used to indicate if a process is ready to enter the critical section.
 - $\text{flag}[i] = \text{true}$ implies that process P_i is ready!



Algorithm for Process P_i

```
while (true) {
```

```
    flag[i] = true;  
    turn = j; بعم تعارف میزند!  
    while (flag[j] && turn == j); آنکه ورود دارد ذبشن هم بار
```

→ entry
section

```
/* critical section */
```

```
flag[i] = false;
```

P_0	P_1
$flag[0]=1$	$flag[1]=1$
$turn=1$	$turn=0$
$while(flag[0] \&& turn==1);$	$while(flag[1] \&& turn==0);$

هر من کن اول P_1 نزدیکی اجری لایتینگ
را پس کشند. سپه کمیاد بیرون چشم چشم
کنی و قطع صلحی P_0 $flag[i]$ میگویند
ماکانه در P_1 اجری شود!
critical section

```
/* remainder section */
```

```
}
```

هر P_0 و P_1 در حالت انتظاری صلحی طبق اجرا میگوند، اونتی کن اجری اینها ممکن است $j = turn$ خیر اجری باشد!



Correctness of Peterson's Solution

- Provable that the three CS requirement are met:

- Mutual exclusion is preserved

میتوان هر دو while loop با یک turn و flag[i] = false را در یک critical section بارگیرید.

P_i enters CS only if:

either $\text{flag}[j] = \text{false}$ or $\text{turn} = i$

هر دو while loop با یک turn و flag[i] = false را در یک critical section بارگیرید.
دو دلیل این است که while loop با یک turn و flag[i] = false را در یک critical section بارگیرید.

- Progress requirement is satisfied

برای P_0 خروج از P_1 برای خود داشت. P_1 خروج از P_0 برای خود داشت.

برای P_0 با خروج از P_1 داشت. P_1 با خروج از P_0 داشت. P_0 خروج از P_1 داشت. P_1 خروج از P_0 داشت.

- Bounded-waiting requirement is met

پس از از خروج از P_1 داشت. P_0 خروج از P_1 داشت. P_1 خروج از P_0 داشت. P_0 خروج از P_1 داشت. P_1 خروج از P_0 داشت.

پس از خروج از P_1 داشت. P_0 خروج از P_1 داشت. P_1 خروج از P_0 داشت. P_0 خروج از P_1 داشت. P_1 خروج از P_0 داشت.

* این رویی را بدانند حالی صدق کنند! چون pipeline ممکن است جایی در سورکت کهنه بود! یعنی جایی که خود این رویی داشته باشد!

Peterson's Solution and Modern Architecture

- Although useful for demonstrating an algorithm, Peterson's Solution **is not guaranteed to work on modern architectures.**
 - To improve performance, **processors and/or compilers may reorder operations that have no dependencies.**
- Understanding why it **will not work** is useful for better **understanding race conditions.**
- **For single-threaded this is ok as the result will always be the same.**
- **For multithreaded the reordering may produce inconsistent or unexpected results!**