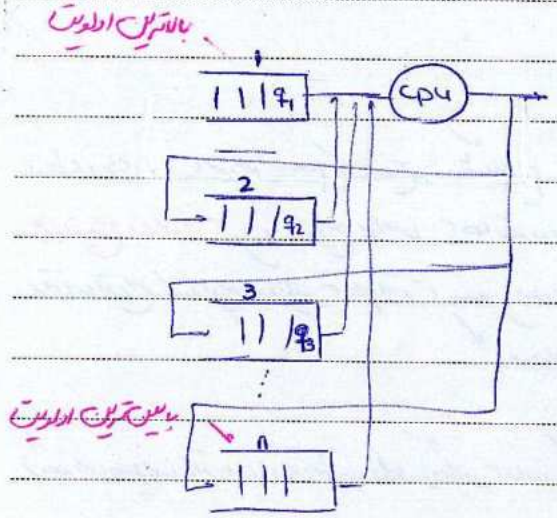
Subject:
Year. Month. Date. ( )

در هم زمانی OS قبل از اینکه پردازنده را به فرآیند بدهد Timer را صفر می‌کند تا کم زمانی به درستی اجرا شود.
(point) q باید طوری انتخاب شود که ۱۰ برابر زمانی باشد که OS نیاز دارد ولی اگر خیلی بزرگ باشد شبیه FCFS
می‌شود که خوب است ولی هرچه کوچکتر باشد بالانس تر است و شبیه SPN می‌شود ولی سیستم محدودیت دارد.

روش Feedback Queue : (پیشنهاد)



در این روش به جای یک صف از چندین
صف استفاده می‌کند کسی وارد می‌شود وارد صف اول
می‌شود و یک q دارد که اگر کافی نباشد به صف دوم
می‌رود که در صف دوم q می‌تواند همان قبلی باشد یا نباشد
اولویت با صف اول است و تا زمانی که فرآیندی در صف
اول است نوبت به صف دوم نمی‌رسد. احتمال اینکه کسی
در صف اول نباشد زیاد است چون برگشت به صف اول
نداریم اگر در صف دوم کارش انجام شد چه بهتر اگر انجام نشد
به صف سوم می‌رود و می‌تواند q دیگری داشته باشد
صف آخر یک q داشته باشد چون اگر q آن تمام شود کجا برود صف دیگری نداریم که بفرستیم صبر کند تا انجام
فرآیند به طور کامل و کامل انجام شود
این روش دیگر الگوریتم SPN را اجرا می‌کند چون اولویت با کوتاه‌ترها است. اگر فرآیندی طولانی باشد مثلاً
است چهار حالت اگر سطح شود به دلیل اینکه زمان سرویس را بداند دارد SPN را شبیه‌سازی می‌کند
هرچه صف بیشتر شود وظیفه OS بیشتر می‌شود
این صف جزء پارامترهای OS هستند و در کنار OS ذخیره می‌شوند

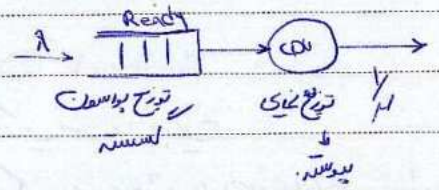
← تا حالا بحث بودن روش‌ها را با مثال نشان دادیم ولی باید به طور علمی اثبات کنیم. برای اثبات از چندین روش
استفاده می‌کنیم ولی الان ساده‌ترین نوع آن گفته می‌شود.

الان با روش آماری کار می‌کنیم ولی این بسیار پیچیده است و ما آشنایش را انتخاب می‌کنیم

انجمن علمی مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

یک صف داشتیم و پردازنده‌ای که به آن خدمت می‌کرد. بسته به اینکه چه توزیع آمدی و چه سرعتی را برای این در نظر بگیریم
نحوه رسان می‌شود. پس روشی که انتخاب می‌کنیم FCFS است حالا ساده‌ترین توزیع عمل را برای داریم



از طرفی برای زمان سرویس هم یک توزیع در نظر بگیریم ← توزیع نمایی

چرا توزیع پواسون ← توزیع پواسون حافظه ندارد و می‌داند قبلاً چه اتفاقی افتاده است. برایش مهم نیست
وقتی با توزیع کار داریم میانگین مهم است ← در واحد زمان چند تا فرآیند وارد صف می‌شوند (λ میانگین تعداد)
میانگین زمان سرویس: فرآیند چقدر است (1/μ میانگین زمان)

فرآیند خیلی نامرتب وارد می‌شوند ولی میانگین دارند. چون نامرتب بودن آنها را همان توزیع مشخص می‌کند.

روبه‌رو ورود فرآیند است که گسسته است (1, 2, 3, ...) ← وقتی 2.5 تا فرآیند داریم ولی 5 تا فرآیند داریم
زمان بین ورود فرآیندها چقدر است؟ عدد ندارد بلکه پیوسته است. توزیع آن پیوسته است و تفاوت زمان همان
توزیع آن مشخص می‌کند

میانگین تعداد

$$P(k) = (\lambda T)^k e^{-\lambda T} / k!$$

$k = 0, 1, \ldots$

← احتمال ورود k فرآیند در فاصله زمانی T

بدست آوردن میانگین این توزیع:

$$E(k) = \sum_{k=0}^{\infty} k P(k) = \lambda T$$

واریانس: $\sigma^2 = \lambda T$ ← چقدر از میانگین دور یا نزدیک هستیم

← کم ← خیلی به میانگین نزدیک، زیاد ← از میانگین دور هستیم

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

توزیع نمایی:

τ : زمان سرویس

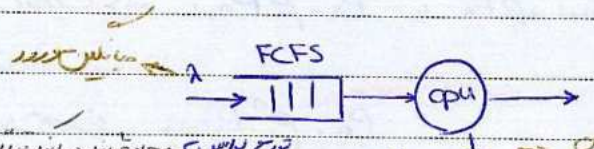
$f(\tau) = \mu e^{-\mu\tau}$

اگر برای رویدادها توزیع پواسون داشته باشیم زمان بین آنها نمایی است.

$E(\tau) = \int_0^\infty \tau f(\tau)\, d\tau = \frac{1}{\mu}$ → میانگین

$\sigma_\tau^2 = \frac{1}{\mu^2}$ → واریانس

جلسه یازدهم 28, 7,



$\rho = \frac{\lambda}{\mu}$ ← مقدار بار نسبت عکس دارد.

← توانایی سرویس دادن

توانایی سرویس دهی باید از مقدار ورود بالاتر باشد $\lambda < \mu$

در غیر اینصورت صف به سمت بی نهایت می رود حتی مساوی هم نمی تواند باشد

به حداقل زمان های بیکاری خواهد رسید اما همیشه $\rho < 1$

مشغول نیست ← نسبت ورود به نسبت توانایی سرویس

مشخصات اساسی یک صف محدود

← ظرفیت مشخصی دارد که نمی تواند تا بی نهایت افزایش پیدا کند

میانگین طول صف

- میانگین زمان انتظار

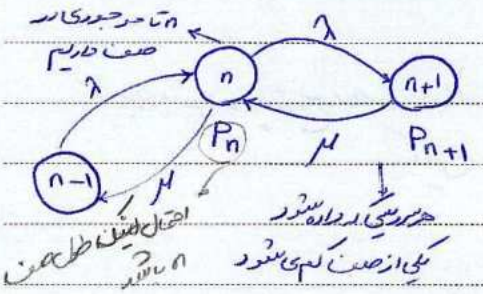
- " " " کل

- احتمال مسدود شدن صف

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

- میانگین توان عملیاتی (Throughput) ← چند تا کار در واحد زمان



صف نمی‌تواند از حالت n-1 به حالت n+1 برود
تغییر چندتایی انجام نمی‌دهد بلکه تغییر حالت یکی یکی است.

$\lambda P_n = \mu P_{n+1}$ ← تابع بشت است

$P_{n+1} = \rho P_n$ یا $P_n = \rho P_{n-1}$ => با استفاده از ترتیب ρ => خاصیت توزیع پواسن برای روابط برقرار می‌شود.

$P_n = \rho^n P_0$ => احتمال خالی بودن صف

عدد ثابتی است.

تابع اندازه‌گیری نیست ولی دانیم چقدر است

گفتیم صف نامحدود است اما برای سادگی مسئله حرمان را پیش می‌گیریم پس صف را نامحدود در نظر می‌گیریم

$$\sum_{i=0}^{\infty} P_i = 1 \Rightarrow \sum_{i=0}^{\infty} \rho^i P_0 = P_0 \underbrace{\sum_{i=0}^{\infty} \rho^i}_{\text{تصاعد هندسی}} = \frac{P_0}{1-\rho} = 1 \Rightarrow \underline{P_0 = 1-\rho}$$

$$E(n) = \sum_{i=0}^{\infty} i P_i = \frac{\rho}{1-\rho}$$ ← میانگین طول صف نامحدود

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر
تهیه و تنظیم امیر قلاوری

اینجا برای صف نامحدود بود حالا می‌آییم برای صف محدود (طولش را N می‌گیریم). هر جا نمی‌گفتیم (∞) استفاده کردیم حالا محدودش می‌کنیم و N قرار می‌دهیم و مقادیر زیر بدست می‌آید.

$P_0 = (1-\rho) / (1-\rho^{N+1})$

$P_n = (1-\rho)^n / (1-\rho^{N+1})$

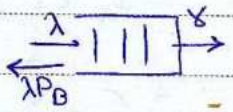
$P_N = (1-\rho)\rho^N / (1-\rho)^{N+1}$ ← احتمال اینکه صف پر شود

PAPCO

Subject:
Year. Month. Date. ( )

$P_B = P_N$ ⇝ چون وقتی که صف پر شود بقیه ترافیک‌ها block می‌شوند

وقتی که صف پر شود ترافیک‌های جدید که وارد می‌شوند پذیرفته نخواهند شد.



$\gamma = \lambda(1 - P_B)$ ← توان عملیاتی

* به خاطر توزیع‌های انتخاب شده تمام اینها از دقت دور است

یک رابطه‌ای مستقل از نوع توزیع است و همیشه صادق است.

n طول صف

w زمان انتظار

$\Rightarrow \quad n = w \cdot \lambda$ ⇝ رابطه لیتل

از روی این رابطه به راحتی می‌توان زمان انتظار پیدا می‌شود:

$$w = \frac{n}{\lambda}$$

$$\text{زمان کل} = w + \frac{1}{\lambda}$$

μ ← میزان پردازنده برای سرویس دادن است پس از این استفاده می‌کنیم و از λ که زمان افزایش می‌کند استفاده می‌کنیم

تا زمانی ترافیک وارد صف می‌شود که صف پر نشده باشد و λP_B تا از آخر اینجا انجام نخواهند شد

↳ احتمال block شدن

تمرین) از زمان‌بندی HRRN استفاده کنیم

	زمان ورود	زمان سرویس
P_1	t_1	r_1
P_2	t_2	r_2
P_3	t_3	r_3

کسری که منظوری باشد که امتیاز می‌دهیم $1 + \frac{t_w}{t_s}$

از این رابطه در حالت کلی استفاده می‌کنیم ولی اینجا برای کنیم

ترتیب اجرای فرایندها:

$t_1 < t_2 < t_3$

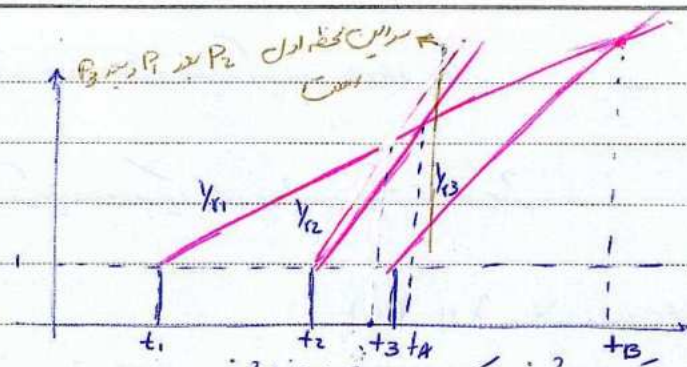
$r_2 < r_3 < r_1$

$1 + \frac{t_w}{t_s}$ ← در واقع تابع تعیین اولویت است

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:

Year. Month. Date. ( )



$t \le t_A$

$t_A \le t \le t_B$

$t \ge t_B$

زمان پایان سرویس فرآیند قبلی

توابع اولویت سه خط هستند چونکه باید جمع شدند هیچ کدام از صفر شروع نمی‌شوند

$R_1 = 1 + \frac{t_w}{t_s} = 1 + \frac{t - t_1}{r_1}$ ← سلایم‌ترین شیب

در این زمان وارد شده

در زمان انتظار است نقطه‌ای که از شروع کرده منهای زمانی که وارد شده

$R_2 = 1 + \frac{t - t_2}{r_2}$ ← تیزترین شیب

$R_3 = 1 + \frac{t - t_3}{r_3}$

در نمودار در یک لحظه اولویت با کسی است که بالاتر است یعنی شیب بیشتری دارد.

$t \le t_A$ ⇝ $\overrightarrow{P_1, P_2, P_3}$

$t_A < t \le t_B$ ⇝ $\overrightarrow{P_2, P_1, P_3}$

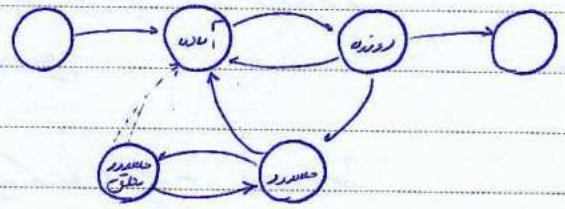
$t_B < t$ ⇝ $\overrightarrow{P_2, P_3, P_1}$ ⇝ وقتی که تمام شدند ادامه پیدا کنند

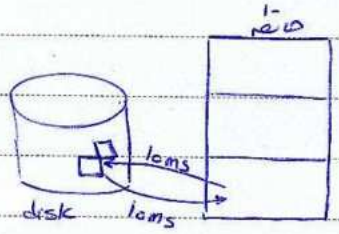
وقتی که تمام شدند سرویس فرآیند که تمام شده از این ترتیب حذف می‌شود.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

تمرین) مفهوم زمانی swaping





حافظه این سرتا partition ثابت دارد پس در لحظه 3 تا فرآیند بیشتری نتوانند در حافظه باشند و اگر بیشتر شد باید به حالت مسدود بطلق بروند

برای وارد کردن فرآیند بعدی یکی را از حافظه دیسک میدیم و فرآیند را از دیسک به حافظه میدیم.

زمان دسترسی به دیسک 4ms ← برای هر بایت اطلاعات است

زمان انتقال 6ms

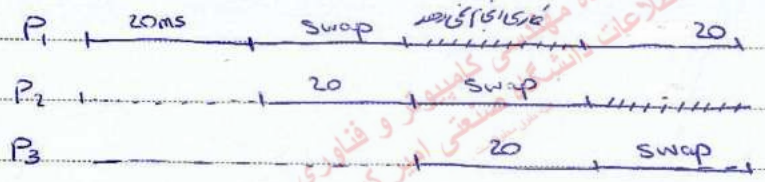
پس هرکدام از اینها برای انتقال به حافظه 10ms طول می‌کشد (یعنی جمع زمان دسترسی + زمان انتقال)

هم) برای بهره‌دهی از CPU یک فرآیند چقدر به پردازنده اجازه دهد؟

← پردازنده بیکار نباشد و کاری برای انجام داشته باشد

برای انتقال هر فرآیند باید اول یکی وایسادن کنیم یکی را باید وارد کنیم پس هر انتقال 20ms طول می‌کشد

زمان یک swap، 20ms است و برای اینکه فرآیند جدی آماده کار شود باید فرآیند در حال اجرا 20ms کار کند که در این عمل swap انجام شود که وقتی عمل swap انجام شده فرآیند جدی 20ms انجام دهد.



دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

پردازنده هیچ وقت بیکار نیست و همیشه چیزی برای اجرا است. swap ها نمی‌توانند هم انجام دهند چون فقط یک دسترسی به دیسک می‌توانیم داشته باشیم

Subject:
Year. Month. Date. ( )

b) اگر هر کاربر یک درخواست در ثانیه بفرستد حداکثر تعداد کاربران؟

یعنی زمان پاسخ باید ۱ ثانیه باشد چون اگر بیشتر باشد متوجه می‌شود

$\frac{1s}{20ms} = 50$

در هر ثانیه به 50 کاربر پاسخ می‌دهیم

c) شخص قبل را برای حالتی بکشید که زمان سرویس 10ms باشد

جلسه دوازدهم 7,30

تمرین) دو صفحه را در نظر بگیرید که هم زمان اجرا شود چه اتفاقی می‌افتد.

که اینجا بالا دیدین tally را شخص کن

```
const int n=10;
int tally;
void total()
{
    int count;
    for (count=1; count<=n; count++)
    {
        tally++;
    }
}

void main()
{
    tally=0
    Parbegin(total(), total());
    write(tally);
}
```

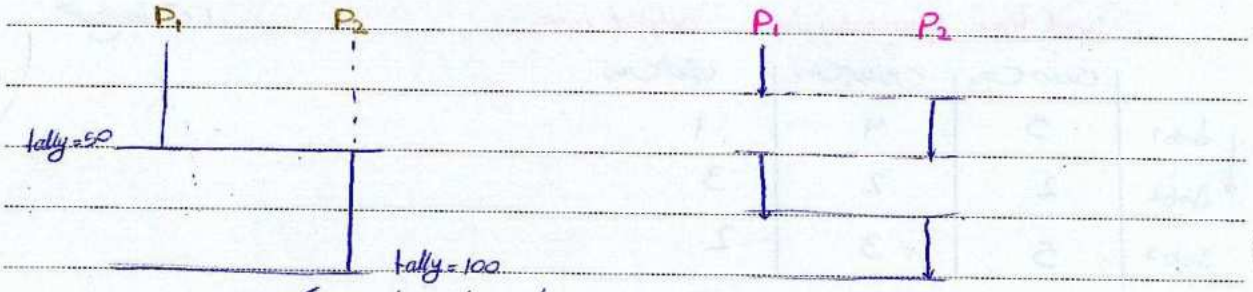
P_1 R_2

دو تا کپی را همزمان اجرا کن

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

P_1 و P_2 چون به صورت همزمان هستند زمان بین آنها تقسیم می‌شود ما هیچی نمی‌دانیم و هر ترکیبی از این‌ها می‌تواند باشد. ترکیب‌های مختلفی که می‌تواند وجود داشته باشد کران بالا و پایین هیچیک کران بالا کاملاً مشخص است چون P_1 اجرا می‌شود و P_2 هم اجرا می‌شود ← 100



هر دست دیگری را بکشیم از این کمتر می‌شود

tally++ = tally+1; ← برای اجرا باید به زبان سطح پایین تبدیل شود

Reg1 = tally;
Reg = Reg1+1; ← پایین‌ترین سطح
tally = Reg1;

چیزی که اجرا می‌شود اینست

این را compile کردیم (خودش compiler شدیم)

اگر کم زمانی را در نظر می‌گیریم برای چیزهایی که اجرا می‌شود تاثیر دارد.

اگر در حال افزایش tally تصادفاً کم زمانی در وسط کار یعنی بعد از اجرای Reg=Reg1+1 تمام شد، بخوردمان می‌گوییم چه اشکالی دارد، دفعه‌ی بعد که نوبتم شد می‌نویسم ← ذهن خیال باطل چون نوبت را به کس دیگری دادیم مثلاً کس دیگر منتظر بود که 11 را ببیند وقتی کران 10 را می‌بیند

Reg2 = tally 10

Reg2 = 11

tally = 11

پخ با تکرار لوپ ⇐ tally = 15

دوباره نوبت به P_1 می‌رسد ← tally = 11

به نظر می‌آید 50 جواب باشد برای کران پایین ولی جواب این نیست چونکه جواب 2 است، حالا چرا؟

کم زمانی دقتی می‌اجازه ما تفحص می‌شود خیلی مشکلات می‌تواند ایجاد کند و باعث این گمراهی می‌شود.

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

کارها را در یک رشته به آن می‌دهند
رشته‌ای

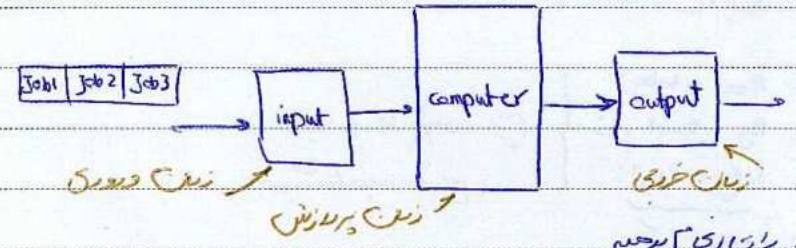
تمرین) راجع به یک مسئله خیلی تئوری (در درس گفته نشده)

جدول زیر زمان‌های لازم برای ورود و پردازش و خروج سه کار را در یک سیستم batch نشان می‌دهد. حداقل کل زمان مصرفی برای اجرای هر سه کار چقدر است؟ (ترتیب ورود تعیین کننده ترتیب پردازش و خروج است)

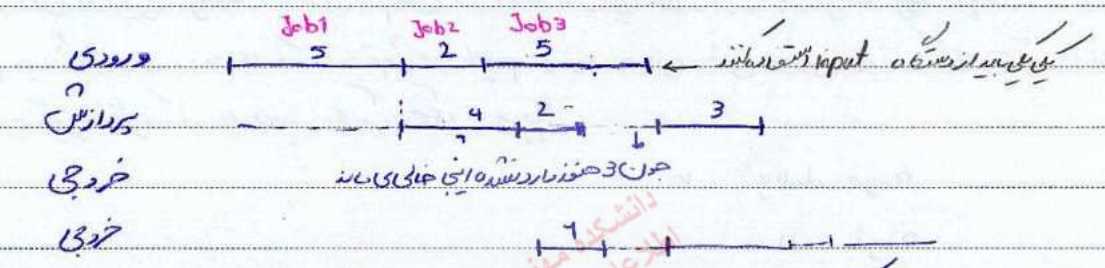
Input time processing time output time

	زمان ورودی	زمان پردازش	زمان خروجی
Job1	5	4	1
Job2	2	2	3
Job3	5	3	2

سیستم batch



این سیستم‌ها قطعاً تعاملی نیستند
جدول ورودی را می‌گیرد و از آن در
خروجی را تحویل می‌دهند
می‌توانیم به این سیستم یک سری Job برای انجام بدهیم
که کدام از این‌ها زمان‌های لازم دارند که خوانده شوند ← زمان ورودی



پردازش باید وقتی شروع شود که ورودی به طور کامل خوانده شود یعنی پردازش Job1 بعد از 5 واحد زمانی شروع می‌شود ولی بعد از آن چون Job2 به طور کامل وارد نشده آن را پردازش نمی‌کنیم ولی چون بعد از آن Job3 هنوز کامل وارد نشده نمی‌توانیم آن را شروع کنیم پس سیستم تا مدتی بیکار می‌ماند و وقتی Job3 کامل آمد شروع به پردازش آن می‌کند.

دانشکده مهندسی صنایع و فناوری اطلاعات خلیج فارس - مدرس امیر بیگی

Subject:
Year. Month. Date. ( )

تمرین) دو تا فرآیند P_1 و P_2 به طور همزمان اجرا می‌شوند

P_1 — loop بی‌نهایت

```
while (TRUE){
  print "A";
  print "B";
}
```

P_2

```
while (TRUE);
  print "C";
  print "D";
}
```

پایان هم زمانی

A CDCD

اگر خروجی را می‌خواست تست کند چون هر ترکیبی امکان داشت

$(AB)^*(CD)^*$

می‌تواند جواب باشد چون هم زمانی در پایان هر loop تمام می‌شود

$A(CD)^*B$

یک A، print می‌کنیم هم زمانی ما تمام می‌شود ولی P_2 چندین بار تکرار می‌شود

BCAD ← قسمتی از خروجی

از یک لحظه‌ای شروع می‌کنیم B را print می‌کند هم زمانی تمام می‌شود C را print می‌کند و A را از سر می‌گیرد

$C^*A^*B^*C^*$

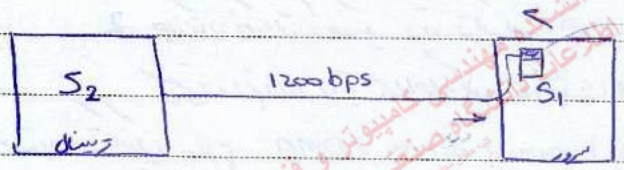
این امکان ندارد باشد

مثال دیگر) در مورد کنترل دسترسی‌ها و خروجی‌ها با وقفه

دو تا سیستم با هم در ارتباط هستند

سیستم اصلی

با اصل وقفه در مکان حافظه ذخیره می‌کنیم



S_2 برای S_1 مثل ترمینال عمل می‌کند و پردازش‌ها را S_1 انجام می‌دهد

سرعت ارتباط بین اینها را خیلی کم نوشته ← 1200 bitps ، واحد ترمیمی این دو سیستم، کاراکتر است.

هر کاراکتر 10bit است

دریافت هر کاراکتر با یک وقفه معوض می‌شود با این سرعت 120 کاراکتر در ثانیه می‌توانیم ارسال کنیم

دانشکده مهندسی کامپیوتر و فناوری
آموزشکده مهندسی صنعتی امیرکبیر

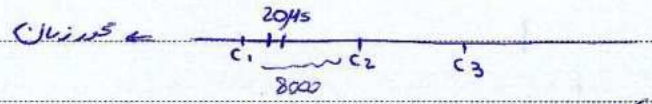
Subject:
Year. Month. Date. ( )

وقفه حاصل می‌شود ← باید داخل روتین وقفه برویم و آن را اجرا کنیم

روال وقفه خواندن پورت و ذخیره در حافظه ← 20 μs

سوال اول: آیا این سیستم مشکلی پیدا می‌کند؟

فاصله ورود دو کاراکتر $8000\,\mu s = 8\,ms = \frac{1}{120}\,s$

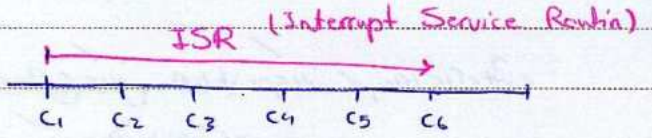


در 8000 می‌خواهیم، 20 تا برای این کار مصرف کنیم ← هیچ مشکلی پیش نمی‌آید.

حالت دوم:

در حالت دوم می‌بینید شبکه را بررسی می‌کند به طوری که به ازای هر 4μs یک کاراکتر وارد می‌شود

1 char per 4μs



کاراکتر اول یک وقفه تولید می‌کند، برنامه هیچ فرقی نمی‌کند، C1 بررسی وقفه را راه می‌اندازد

برای این که به روتین وقفه برای C1 رسیدگی کند 9 تا char را از دست داده و همچنین تا ISR فعال می‌شود دوباره

یک char جدید وارد می‌شود و همیشه ISR در حال اجراست.

راه حل ها: 1- کندش کنیم و اجازه ندهیم هر 4μs بیاید (در صورتی که امکان پذیر باشد)

2- چند تا کاراکتر را در یک وقفه تولید کنیم و همه را با هم به خودشون بریم و یا این چنین

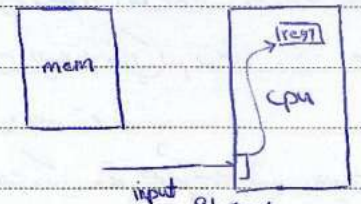
سیستمی داشته باشد روتین را اجرا می‌کند و کار دیگری می‌تواند بکند

راه حل اصلی: استفاده از تکنیکی به نام DMA (Direct memory access) ← دسترسی مستقیم به حافظه

دسترسی غیر مستقیم به حافظه داریم

دسترسی port به حافظه غیر مستقیم است

و توسط CPU است.



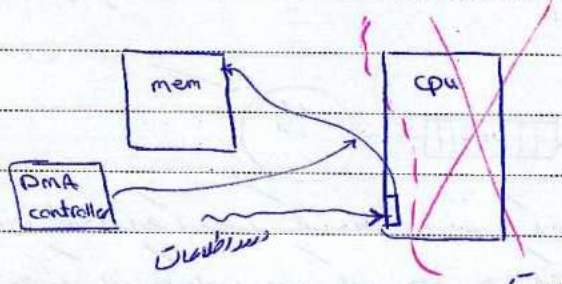

```
ld  reg1,M1
inc address
```

جدول می‌خواهیم ولی همان خانه‌ها بنویسیم که

PAPCO

Subject:
Year. Month. Date. ( )

به ازای هر چیزی که وارد می‌شود باید چندین Instruction اجرا کنیم
جای DMA می‌گوییم که واسطه را بردار و ارتباط را مستقیم برقرار کن
چون لازم نیست Instruction اجرا کنی و هر چی امر خودش می‌رود برجاش پس خیلی بهتر اجرا می‌شود ولی
کمی افزار جانبی برای انجام این کار می‌خواهد



کنترل DMA controller در دست OS است
هر وقت load یا char آمد یک Interrupt به من بده این کار انجام نمی‌شود بلکه در حافظه ذخیره می‌شود (عملا در این جا
حافظه‌ش buffer می‌گیرند)
مثال: یک وقفه برای load و ورودی

جلسه سیزدهم 7,8,

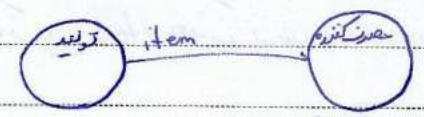
هماهنگی فرایندها: process synchronizing

چند تا فرایند هستند دارای منابع مشترک پس باید به گونه‌ای هماهنگ شوند که مزاحم کار یکدیگر نشوند یعنی استفاده
مشترک از فرایندها به صورت حفاظت شده باشد.
این هماهنگی ← توسط خود فرایندها
← توسط OS

monitor

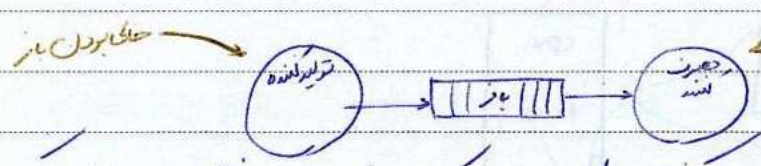
دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

مثال از هماهنگی فرایندها
مثال تولیدکننده (producer) و مصرف‌کننده (consumer)
در فرایندی یک چیزی تولید می‌کند که فرایند
دیگر آن را مصرف می‌کند برای اینکه با هم
کار کنند باید با هم هماهنگ باشند
مصرف‌کننده اگر بخواهد چیزی را مصرف کند باید قبلش تولید شده باشد



Subject:
Year. Month. Date. ( )

پس تولید باید قبل از مصرف باشد و اگر چیزی تولید نشده باشد که نمی‌تواند مصرف کننده آن را مصرف کند.
اگر تولید کننده بطور مرتب تولید کند و مصرف کننده بطور مرتب مصرف کند
این مثال محدود کننده است و
مصرف ← تولید
" ← تولید
و البته این در بهم خیلی زیاد است

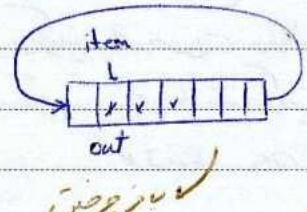


در این جا تولید کننده کاری ندارد که مصرف کننده item را مصرف کرده یا نه کار خود را انجام می‌دهد و تولیدی می‌کند چه مصرف کننده مصرف کند چه نکند و مصرف کننده هم به هر سرعتی که بخواهد می‌تواند item ها را مصرف کند اگر خالی نباشد

هماهنگی: برای تولید کننده: باز پر نباشد
" مصرف کننده: آزمون خالی بودن باز

← در این جا هماهنگی از بین نرفته بلکه سست‌تر شده و نسبت به حالت قبل که پله پله به هم وابسته بودند کمتر شده

پیاده‌سازی باز



باز چرخشی

یک آرایه است و pointer داریم که نشان می‌دهد کجای باز را می‌توانیم input کنیم و از طرف دیگر یک pointer برای مصرف کننده است که نشان می‌دهد تا کجا خوانده شده و عقب‌تر از pointer اول است این دو بیشتر به هم هستند ولی چند خانه جای از باز که خوانده شده آزاد می‌شود. این باز به صورت باز حلقوی می‌تواند تا ابد ادامه دهد

هر کدام از خانه‌های باز یک item هستند که ساختاری دارند و هر چیزی می‌تواند باشد

PAPCO

Subject:
Year. Month. Date. ( )

```
typedef item ____ ;
item buffer[n] ;        ⇝ آرایه‌ای از item تعریف شد
int in, out ;           ⇝ pointer ست
```

وقتی بافر خالی است in=out است، آزمون خالی بودن : in=out

به نظر پر بودن هم همین است اگر همین باشد فرق پر و خالی بودن چه فرقی دارند؟

فرض کنیم ← پر کنیم تا جایی که به قبل out برسیم و یک خانه مانند به out می‌گوییم پر است و برای این که یک خانه فرق بگذاریم پس داریم: آزمون پر بودن : in+1=out

↳ (in+1) mod n = out جون حالت چرخشی دارد باید محاسبه شود

اگر تعداد بافر زیاد باشد از دست دادن یک مکان (خانه) خیلی مهم نیست

PRODUCER:

```
while (True) {          ⇝ حلقه‌ای که تا بی‌نهایت تولید کند
  ...
  produce an item in nexp      ⇝ item را تولید کن در متغیر nextp قرار بده
  ...
  while ((in+1) mod n = out));  ⇝ تا زمانی که پر هست اولین حلقه باشد تا یک مکان
  buffer[in] = nextp;          خالی شود
  in = (in+1) % n;             ⇝ یک واحد pointer اضافه می‌کند
}
```

CONSUMER

```
while (True){
  while (in == out);           ⇝ تا زمانی که خالی هست دور خودت بچرخ
  nextc = buffer[out];
  out = (out+1) % n;
  ...
  consume the item in nextc;   ⇝ item موجود در nextc را مصرف کن
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

update کردن این جا هم صدق می‌کند

متغیرهای مشترک توسط OS تعریف کرد که بین فرایندها مشترک باشد در این حالت مشکلی پیش می‌آید زمانی مشکل پیش می‌آید که دو فرایند با هم بخواهند متغیر را تغییر دهند

← می‌خواهیم راه حلی بدهیم که مکان از دست رفته نداشته باشیم؟

یک متغیر counter در این جا تعریف می‌کنیم که تعداد خانه‌های پر بافر را به ما می‌گوید

اگر counter=0 ← خالی

اگر counter=n ← پر

و قسمت جای کد قبلی به صورت زیر تغییر می‌کند،

```
producer: while (counter==n);
          buffer [in] = nextp;
I         counter = counter+1

consumer: while (counter==0)
          nextc = buffer [out];
II        counter = counter-1;
```

این راه حل کار نمی‌کند: چون counter را هر دو دستکاری می‌کنند و کافیست در ازای یک counter اشتباه شود ← پر و خالی بودن را عوض می‌کنیم

روی یک چیزی بلوک می‌کنیم که یک خانه خالی شده را به بافر می‌دهیم و یک چیزی را از بافر می‌خوانیم

صدق نشده پس یک Hare از دست می‌دهیم

← دستورات I و II باید به زبان سطح پایین اجرا شوند

$$counter = counter + 1 \begin{cases} reg1 = counter \\ reg1 = reg1 + 1 \\ counter = reg1 \end{cases}$$

$$counter = counter - 1 \begin{cases} reg2 = counter \\ reg2 = reg2 - 1 \\ counter = reg2 \end{cases}$$

دانشکده مهندسی کامپیوتر و علوم
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

در producer هستیم و مقدار counter=6 است

counter = 6

reg1 = 6

reg = 7 ⇝ قبل از نوشتن کم و زیادی انجام شده

⇝ الان باید counter=7 باشد

consumer counter = 6

reg2 = 6

reg2 = 5

counter = 5 ⇝ کاملا واضح است این عدد غلط است

در حالت اصلی باید به 6 می‌رسیدیم

← در روش اول هر دو فرآیند قفل می‌خوانند و چون رستاری می‌کنند به مشکل برمی‌خوریم فقط ممکن است پیام خالی بودن را زودتر یا دیرتر بفهمیم

← در این جا از یک متغیر مشترک integer (part, printer) استفاده کردیم در صورتی که هرجای قرار باشد پس استفاده از یک منبع مشترک بدون حفاظت

همه مشکل اصلی در این جا بود

فرآیندها باید در استفاده مشترک از منابع با هم هماهنگی کنند در صورتی که هر هماهنگی باشند فقط منبع مشترک اگر اگاه باشد

حفاظت شده با فرآیند کار کنند و قبل از دستکاری حتما تست می‌کنند

— یک مشکل کلی است برای همه منابع پس باید یک راه حل کلی باشد و تنها در مورد این متغیر نباشد

هر منبع مشترکی که استفاده می‌شود باید حتما یک controller باشد تا ضد بان به سراغ منابع مشترک می‌آوریم و درخواست‌های خود را به OS می‌دهیم و OS این را handle می‌کند چون همه چیز را می‌بیند این توانایی را دارد و همه چیز را به دست OS می‌سپاریم برایش سالم است چون همه جا را می‌تواند ببیند

مشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی خواجه نصیر

Subject:
Year. Month. Date. ( )

اینکه OS این کار را انجام دهد و یا خودمان مزایا و معایبی دارد.

← اگر زیاد به سراغ OS برویم کار می‌کند ولی سیستم را کند می‌کند و کارهایش را چون کلی انجام دهد برای یک کار کوچک که زیادی را اجرا می‌کند چون باید همه حالت‌ها را در نظر بگیرد.

مسئله را به صورت کلی بیان می‌کنیم:

مسئله ناحیه بحرانی (Critical Section problem)

ناحیه بحرانی: جایی که می‌خواهیم حفاظت شده باشد در مثال قبل ناحیه بحرانی موارد زیر است

counter = counter + 1 و counter = counter - 1

برای حل این یک حل برای ناحیه بحرانی تعریف می‌کنیم

حل برای ناحیه بحرانی:

۱) تا فرآیند را در نظر می‌گیریم که می‌خواهیم با هم همانند کار کنند

← می‌توانند مختلف باشند ولی به منبع مشترکی نیاز دارند

```
void P(int i)
{
  while (True) {
    enter_critical (i);
    /* critical_section i */ ;
    exit critical (i);
    /* noncritical_section */ ;
  }
}
```

این جا چک می‌کند آیا می‌تواند وارد ناحیه بحرانی شود یا نه

کد اصلی ما در این جا هست که در این بخش به منبع مشترک دسترسی پیدا می‌کند

در این جا به منبع مشترک کاری ندارد

حفاظت بین سطور

به بقیه کار اضافه می‌کند چون در ناحیه بحرانی نیستیم

بهترین حالت است که تا فرآیند از یک منبع مشترک را تا آخر حال استفاده هستند

```
void main()
{
  parbegin(P(1), ..., P(n));
}
```

راه‌حلی که برای همه موارد کار کند وجود ندارد ولی خیلی زیاد است و همه جا را در نظر می‌گیرد و در صورتی که خیلی دقت‌ها لازم نیست را انجام این راه‌حل به هزینه‌اش نمی‌ارزد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

چون این مشکل راه حل ساده‌ای ندارد برای دو فرآیند آنرا حل می‌کنیم سپس آنرا به n تعمیم می‌دهیم

جلسه چهاردهم

http://groups.google.com/group/OS_aut_fall_87

برای این هفته (1-2-3-12-13)

- راه حل های ناحیه بحرانی را برای دو فرآیند بررسی می‌کنیم

الگوریتم های دو فرآیندی برای مسئله ناحیه بحرانی

- برای درست بودن راه حل باید سه شرط برقرار باشد

شرط های راه حل صحیح:

1. اگر فرآیندی وارد ناحیه بحرانی شود دیگر فرآیندی نمی‌تواند وارد این ناحیه شود ← انحصار متقابل mutual exclusion

متقاضی فرآیند در ناحیه بحرانی باشد

2. انتظار محدود bounded wait ← برای وارد شدن به ناحیه بحرانی باید تضمین شود که میزان انتظار فرآیند محدود است

← این دو تا شرط برای صحیح بودن راه حل کافی است اما یک شرط دیگر هم بعداً می‌گذاریم

تلاش اول (first attemp)

```
P0 → while (True) {
        while (turn == 0);
        critical_section0();
        turn = 1;
        noncritical_section0();
     }
```

متغیر boolean

```
P1 → while (True) {
        while (turn == 1);
        critical_section1();
        turn = 0;
        noncritical_section1();
     }
```

فرض کنید که هر کدام به طور مشترک می‌کنند یکی باشد.

اگر turn = 0 باشد نوبت برای P0 است و می‌تواند وارد ناحیه بحرانی است اگر 0 نبود در این حلقه گیر می‌کند

ولی وقتی شرط برقرار بود وارد ناحیه بحرانی خودش می‌شود وقتی کارش در ناحیه بحرانی تمام شد نوبت را به P1 می‌دهد

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

در این روش اجنا رقابل وجود دارد چون turn یا صف است یا ! در یک لحظه فقط یک مقدار را دارد. نوبت بین P_0 و P_1 در رفت و آمد است و تعداد نوبت ها مساوی است چون هر کی نوبت را بدش به اون یکی می دهد ← کم توان چیع تا نیری در دست یا غلط بودن این که پیش نداره.

← شرط سوم برقرار است چون هر کسی صدا کند یک نوبت منتظر می شود چون نوبت بعد از این که کارش پیدا انجام شد به دیگری می دهد این که این یک نوبت چقدر طول می کشد ربطی به موضوع ندارد.

← اما از این راه حل خوشمان می آید چون شرط سوم که نگفتیم ندارد یعنی در کل راه حل خوبی نیست حالا چرا؟

فرض کنیم ناحیه بحرانی P_0 کم و برای P_1 زیاد باشد.

نوبت با P_0 و P_0 وارد ناحیه بحرانی می شود و کارش را انجام را انجام می کند و نوبت را به P_1 می دهد و کار ناحیه غیر بحرانی خود را تمام می کند

P_0 باید آنقدر انتظار می کشد که کار ناحیه غیر بحرانی P_1 تمام شود و خیلی طول می کشد ولی اگر P_1 کاری با ناحیه بحرانی ندارد پس باید P_0 بتواند کار خود را با ناحیه بحرانی انجام دهد و اینکه نوبت ها را مساوی می دهد اصلاً خوب نیست (چون P_1 اصلاً ناحیه بحرانی را نمی خواهد ولی ما صبر می کنیم تا بخواهد) اگر یکی تند تند نوبت بخواهد دیگری بیرون بیاید سریع تر را به سرعت می رسانند و به هر دری که به سرعت پایین می رسانند

نوبت با P_0	P_1 در ناحیه غیر بحرانی
ناحیه بحرانی P_0	
نوبت با P_1	
ناحیه غیر بحرانی P_0	
انتظار	

شرط سوم) پیشرفت Progress ← کسی که داوطلب ورود به ناحیه بحرانی نیست در رقابت شرکت نکند

این راه حل خوبی نیست چون شرط سوم را ندارد می خواهیم راه حلی بدهیم ← ستیزی تحریمی می کنیم که نشان دهیم کی داوطلب ورود به ناحیه بحرانی است.

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

(second attemp

```
P0 → while (True) {
    ① flag[0] = True;
    ② while (flag[1]);
        critical_section0();
        flag[0] = false;
        noncritical_section0();
    }
```

پایان بخش‌های

wait

```
P1 → while (True)
    ② flag[1] = True;
    ③ while (flag[0]);
        critical_section1();
        flag[1] = false;
        noncritical_section1();
    }
```

wait

flag ← نشان می‌دهد کسی داوطلب است و اگر کسی بخواهد وارد ناحیه بحرانی شود flag خود را true می‌کند و برای وارد شدن به ناحیه بحرانی flag دیگری را چک می‌کند

این راه‌حل از نظر اصل متقابل صحیح است و هر دو نمی‌توانند وارد ناحیه بحرانی شوند.
شرط سوم را هم دارد ← کسی که داوطلب نیست در رقابت نخواهد بود چون flag آن true نیست
آیا شرط دوم هم برقرار است؟

اصلا به شرط دوم نمی‌رسد و یک مشکل دیگری وارد کار می‌کند ← فرض کنیم ① کهربای برای P_0 است و flag خود را True می‌کند قبل از وارد شدن به ناحیه بحرانی کهربای تمام می‌شود و در بعدی وارد می‌شویم، نوبت به P_1 ② می‌رسد و flag خود را True می‌کند و وی در حلقه جدید که flag[0] ③ True است گیر می‌کند نمی‌تواند رد شود و صبر می‌کند تا کهربای تمام شود نوبت به P_0 می‌رسد ④ و چون flag[1] هم true است P_0 در حلقه گیر می‌کند و هر دو منتظر هستند flag دیگری false شود ← شبیه حالت بن بست است و راهی برای نجات ندارند و احتمال غیرصفر برای اتفاق افتادن این وجود دارد.

راه‌حل درست: این دو راه‌حل را با هم مخلوط می‌کنیم ← هم turn داریم و هم flag

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

تلاش سوم (third attemp)

```
P0 → while (True) {
     flag [0] = True;        قسمت ورود
     turn = 1;               ← برای نوبت دهی
     while (flag[1] && turn == 1)
     critical_section0();
     flag[0] = false;        خروج
     noncritical_section0();
     }
```

```
P1 → while (True)
     flag [1] = true;        قسمت ورود
     turn = 0;
     while (flag[0] && turn == 0);
     critical_section1();
     flag [1] = false;       خروج
     noncritical_section1();
     }
```

flag خود را True کند ← می‌خواهم وارد ناحیه بحرانی شوم

اگر کسی Turn را دارد اما flag ندارد ← نمی‌تواند جلوی کسی را برای ورود به ناحیه بحرانی بگیرد.

احتمال دارد طرف مقابل را دیگر پیشنهاد نداده

← اگر هر کس turn را قبل از ورود به ناحیه بحرانی به خودش بدهد (الگوریتم حریصانه) ممکن است طرف مقابل از حالتی بیند از که نوبت به دیگری نرسد. یعنی مثلاً P_0 چندین بار turn را بگیرد و به P_1 ندهد این چندین بار معلوم نیست چقدر است ← امکان اتفاق افتادن این غیرصفر است

اما اینکه نوبت را به دیگری بدهیم اگر داوطلب نباشد هیچ اتفاقی نخواهد افتاد و یک نوبت بیشتر معطل می‌شود و نه بیشتر چون کسی که الان turn را دارد اول turn را به من می‌دهد.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

محکم‌سازی پیچیده است و تنها تابعی که دارد این است که حداکثر یک نوبت می‌تواند من را عقب بیندازد

تعمیم به n فرآیند:

همین راه‌حل را می‌توان به n فرآیند تعمیم داد البته خیلی کجی است. اینکه این را بدانیم خوب است ولی اگر خودمان بخواهیم بنویسیم بیاوریم سخت است

در قبل دو حالت داشتیم ← می‌خواهند وارد ناحیه بحرانی شوند

می‌خواهند ← ← ← ←

Subject:
Year. Month. Date. ( )

idle — اما در این جا سه حالت را تعریف می‌کنیم ← نمی‌خواهد وارد شود

want_in — تمایل دارد وارد ناحیه بحرانی شود

in_CS — وارد ناحیه بحرانی شده

```
enum state {idle, want_in, in_cs}
state flag[n];
int turn;
enter_critical_section:
do { flag[i] = want_in;
     j = turn;
     while (j != i) {
         if (flag[j] != idle) j = turn;
         else j = (j+1) % n;
     }
     flag[i] = in_cs;
     j = 0;
     while (j<n && (j == i || flag[i] != in_cs)) j++;
     if (j>=n && (turn == i || flag[turn] == idle)) break;
     }
     turn = i;
```

ناحیه بحرانی

می‌بینیم تا جهت عالی باشد

شرط اختصاص متغیر

همه فرایندها را چک می‌کند تا in_CS نبیند اگر دید دوباره از اول اگر مشکلی نبود وارد ناحیه بحرانی می‌شود

while(true)

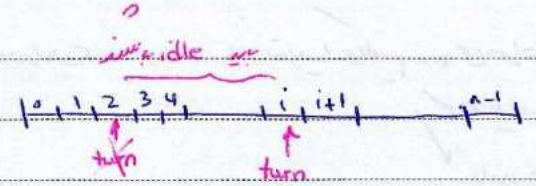
وقتی از ناحیه وارد ناحیه بحرانی شود ← turn را به خودش می‌دهد بعد از اینکه همه چیز را چک کرده باشد

آخرین کاری است که انجام می‌شود یعنی مطمئن بوده که می‌تواند وارد ناحیه بحرانی شود

انجمن علمی مهندسی کامپیوتر
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

ناحیه بحرانی چی کار می‌کنند؟



چک می‌کنند دونه دونه را از turn تا خودش که همه باید idle باشند اگر اینطور باشد می‌گوید حق با من است اگر idle نباشند در این ناحیه گرفتاری شود و انقدری بماند تا همه idle ببیند یعنی مخدوش حق خودش است اگر این حالت اتفاق افتاد flag خودش را به in_cs تغییر می‌دهد

اما هنوز این راه‌حل، راه‌حل کاملی نیست و این مشکلات را در ناحیه خروج حل می‌کنیم

```
exit_critical_section:
    j = (turn+1) % n;
    while (flag[j] = idle) j = (j+1) % n;
    turn = j;
    flag[i] = idle
```

همه را چک می‌کنیم ببینیم کی هست که idle نیست (اولین کسی که idle نیست کی است) turn را به اولی می‌دهیم اگر در این n تا فرآیند کسی وجود نداشت که idle نیست turn به خودم برمی‌گردم چون به نوبت کار می‌کنند حداکثر n-1 نوبت طول می‌کشد تا نوبت به من برسد

تا چک‌هایی که آخر می‌کنند برای ایجاد انحصار متقابل
شرط انتظار محدود وجود دارد حداکثر n-1 نوبت
واضح است شرط سوم هم هست چون اصلاً افراد idle را دخالت نمی‌دهد

دانشکده مهندسی کامپیوتر شاهرودی
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

جلسه پانزدهم 8,19

ارسال تمرین‌ها به Pedram@aut.ac.ir

راه‌حل جلسه قبل برای OS نداشت

الگوریتم نانوایی baker's algorithm ← مدل استفاده نمی‌شود و فقط گفته می‌شود
هر کس که می‌خواهد وارد ناحیه بحرانی شود شماره می‌دهیم (نوبت‌دهی) و بر اساس آن شماره می‌توانند سرویس بگیرند
یعنی وارد ناحیه بحرانی شوند

```
boolean choosing[n];
int number[n];
```

دیتا مشترک هستند یعنی داخلی نیستند و اصلا حساب نمی‌شوند

```
enter_critical_section:
    choosing[i] = True;
    number[i] = max(number[0], ..., number[n-1]) + 1;
    choosing = False;
    for (j=0; j<n; j++)
        while (choosing[j]);
        while (number[j] != 0 && (number[j], j) < (number[i], i));
}
```

شانس می‌گیرد که به دریافت number نوبت هستیم

نحوه شماره گیری

max یک لیست می‌گیرد و یک عدد اضافه می‌کند

علامت هر کس طوری است که بر روی j ← یعنی آن پروسس j

منتظر می‌ماند که چک شماره گیری است صبر می‌کند تا آن شماره‌اش را بگیرد

بیشتر تا 5 بیرون نمی‌رود تست می‌شود

برای سلسله‌مراتبی درشماره

یعنی number کمتر نباشد

شماره‌ی number یعنی از من کوچکتر است پس می‌توانیم وارد ناحیه بحرانی شویم

```
exit_critical_section:
    number[i] = 0;
```

باشگاه پژوهشی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

مرحله شماره گیری، مرحله طولانی است ← بدین تعداد max بگیرد ← مقابل زبان ماشین آن طولانی می‌شود پس در وسط کار می‌تواند قطع شود ← بنابین خاطر همه را باید به حساب آورد حتی کسی که در حال شماره گیری است چون ممکن است قبل از من باشد و وسط کار قطع شده

همچنین امکان شماره دهی مساوی وجود دارد ← این کار را می‌کند اگر دمتا شماره مساوی بود به شماره فرایند برای شکستن حالت تساوی نگاه کن ← اول شماره‌ها چک می‌شود اگر مساوی بود شماره فرایند چک می‌شود

یک مشکل الگوریتم بوجود شده: number حتا زمانی بالا می‌رود که overflow بشود وقتی overflow داریم چکار کنیم؟ حل آن بسادگی نیست ← دچار اشتباه می‌شویم

این راه حل تحت شرایطی کار می‌کند ← اگر زمان‌هایی وجود داشته باشد که هیچکس درخواست ورود به ناحیه بحرانی نیست در این صورت number حاضر می‌شود یعنی زمانی کار می‌کند که این‌ها دائم در رقابت باشند و از این زمان‌ها این وسط باشد در عمل هم این طوری یعنی دائم در حال رقابت نیستند

راه حل بسیار ساده

تا به حال در مورد دستورالعمل‌های ماشین فرض کردیم (بسیار ساده‌اند) اما الان در نظر می‌گیریم که مقداری دستورالعمل پیچیده دارد (دستورالعمل‌های غیرساده)

دستورالعمل زبان ماشین

tsl → Test_and_Set

test_and_set_logical

پیش از یک عمل ای‌ای می‌دهد

مقدار اولیه

فرض بر این است

① محتویات حافظه به reg

② به عدد غیرصفر بخواه ! ، tsl می‌کند

اگر دستم بگیریم cpu این دستورالعمل را دارد راه حل بسیار ساده می‌شود.

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

زبان assembly :

```
enter_critical_section
    tsl   reg2 , flag
    cmp   reg2 , #0
    jnz   enter_critical_section
    ret
exit_critical_section
    move  flag , #0
    ret
```

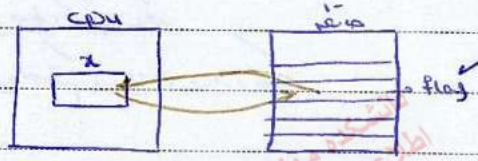
یعنی حسابی نباشد خطا در حسابی باشد Zero = 1 می‌شود

هر فرآیندی که وارد ناحیه بحرانی می‌شود flag را یک می‌کند مثلا flag = 1 باشد یعنی کسی وارد ناحیه بحرانی استفاده می‌کند پس اگر فرآیند جدیدی خواست وارد ناحیه بحرانی شود تا زمانی که flag صفر نشده نمی‌تواند وارد شود در ضمن حریک زمان دو تا فرآیند می‌توانند tsl انجام دهند

در tsl مهم زمانی به وسط کار رفع می‌شود چون یک دستور العمل واحد است.

فرض بر حضور CPU درست بود ← Intel این را ندارد و چیزی شبیه این دارد

swap →



محتویات این جا را جابه جا می‌کند

```
enter :
    mov   regx , 1
    swap  reg2 , flag
    cmp   reg2 , #0
    jnz   enter
    ret
```

```
exit :
    mov   flag , #0
    ret
```

برای هر منبع می‌توانیم ناحیه بحرانی زیادی تعریف کنیم پس کمتر تعداد flag زیادی داشته باشیم

انجمن علمی مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

این راه حل انحصار متقابل را کاملاً حفظ می کند. (تضمین شده و نمی تواند خراب کند)

مشکل این راه حل کجاست؟

شرط محدودیت زمانی را تضمین نمی کند (شرط سوم را دارد (کسی که داوطلب نیست، دخالت داده نمی شود) محدودیت زمانی تضمین نمی شود.

← نوبت به طور کاملاً تصادفی گرفته می شود یعنی اگر 5 تا فرآیند داریم یکی گرفت، 4 تای خواهند بین این 4 تا تصادفی است. احتمال غیرصفر دارد که یکی اصلاً نوبت بهش نرسد

← این راه حل ضمن سادگی، صحیح نیست

↳ دنبال راه حل صحیح نبودیم، دنبال چیزی بودیم که تحت شرایطی کار کند

این راه حل تحت چه شرایطی کار می کند؟ اگر به طور دقیق بخواهیم بگوییم که هیچ کس داوطلب نباشد کار می کند ولی اگر همیشه فرآیندها در رقابت باشند این راه حل کار نمی کند و در واقعیت این است که زمان هایی وجود دارد که هیچ کس داوطلب نباشد

← اگر در ناحیه بحرانی لیدر کند شدن بین سبب است و برای تمام این راه حل ها ممکن است اتفاق بیفتد

↳ قسمت ناحیه بحرانی باید خیلی کوچک باشد

این راه حل ماست و ما به دنبال راه حلی نیستیم که برای همه موارد کار کند

این راه حل زمانی درست کار می کند که ← دائم در حال رقابت نباشند

↳ ناحیه بحرانی کوتاه باشد

← نکته مشترک تمام راه حل های ارائه شده:

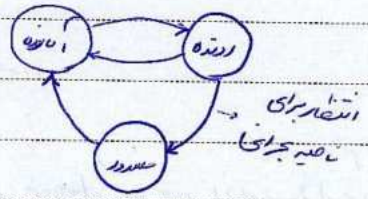
از انتظار مشغولی (busy waiting) استفاده می کنند

↳ برای انتظار از حلقه استفاده می کنند ← زمان پردازنده می جهت تلف می شود و هیچ کاری صورت انجام نمی شود ← راه حل های ارائه شده که فرآیندها را خواب می کنند

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

می‌خواهیم راه‌حلی ارائه کنیم که انتظار مشغول نداشته باشد و از حالت مسدود استفاده می‌کنیم.



زمانی که می‌خواهند منتظر ناحیه بحرانی باشند به حالت مسدود می‌روند

انتظار برای ناحیه بحرانی را مثل درخواست برای سرویس در نظر بگیریم

مفهومی بیان می‌شود که تحت عنوان یک ابزار پیاده‌سازی شده

سمافور (Semaphor در لغت) ← یک متغیر int به همراه صف که اندیس‌ای بگیریم چه موقع باید وارد ناحیه بحرانی شده و یک موقع باید استراحت کرد
ابزاری برای هماهنگی فرآیندها (ورود به ناحیه بحرانی)

یک ساختار داده‌ای است ← یک متغیر int
یک صف

در حالت مسدود فرآیندها بنابر سرویسی که می‌خواستند در یک صف منتظر می‌شوند این صف Semaphor را هم در این جا می‌گذاریم ← فرآیندی که به بقیه سرویس‌ها می‌شود

Semaphor ← یک مجوز برای وارد شدن به ناحیه بحرانی
← برای هماهنگی به کار می‌بریم

دانشکده مهندسی کامپیوتر شاوری
اطلاعات دانشگاه صنعتی امیرکبیر
۰۰۹۶-۸۳۸۲۶۰۰۷ 11365 8320469 ۰۰۸-۱۳۹۶۰۲۵-۱

Subject:
Year. Month. Date. ( )

جلسه شانزدهم 19,8,

تمام راه حل های ارائه شده از busy waiting استفاده می کنند

Semaphore (سمافور، راهنما)

ابزاری برای هماهنگی

ساختاری شامل یک integer و یک صف دارد. (از فرآیندها)

عملیاتی که روی این ساختمان داده هست را هم باید بگوییم

عملیات روی Semaphore

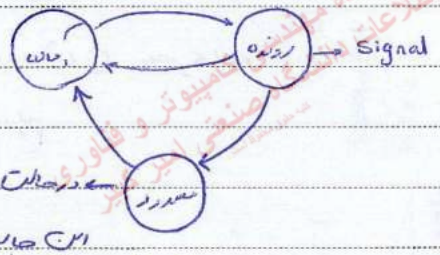
مقدار گذاری اولیه به مقدار integer را باید set کنیم و بعدی برای شروع به کار برسیم

عملیات مهم روی Semaphore ← wait

signal

wait ← یک واحد از integer کم می کند و نگاه می کند که آیا مقدار منفی شد یا نه. اگر منفی شد فرآیند مسدود می شود و در صف قرار می گیرد در غیر اینصورت فقط یک واحد کم می کند و عبور می کند

Signal ← یک واحد اضافه می کند و چک می کند که مثبت نباشد ← اگر مثبت نباشد یکی از فرآیندهای مسدود شده آزاد می شود (آن که در صف است آزاد می کند، آن که در نوبتش است)



← در حالت signal یکی از فرآیندهایی که این جا بود به حالت آماده می رود برای خود کسی که signal را اجرا کرده اتفاقی نمی افتد

دانشکده مهندسی کامپیوتر و فناوری اطلاعات صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

```
struct Semaphore {
        int Count;
        queueType queue;
    }

void wait (Semaphore s)
{
    s.count --;
    if (s.count < 0) {
        place this process in s.queue;
        block this process;
    }
}

void signal (semaphore s)
{ s.count ++;
    if (s.count <= 0) {
        remove a process P from s.queue;
        place Process P on ready list;
    }
}
```

یک صف انتظار برای ها

اگر منفی شد blocked می‌کند

Binary Semaphore

نوع دیگری از semaphore ← binary Sema ← به جای integer یک bool استفاده می‌کند و کار همین است

اصطلاحاً می‌گویند عملیات wait و signal اتمی هستند یعنی نمی‌شود متقاطع کرد (مثل Test & set) یا یک یک چه انجام شود و نمی‌شود تجزیه کرد. اگر wait می‌کنیم باید کل wait انجام شود و یا هیچی انجام نشود.
پیاده‌سازی همین چیزی راحت نیست مثل اینکه بخوایم زمان می‌گذاریم نباید

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

این استفاده از semaphore در ناحیه بحرانی است

حل مسئله ناحیه بحرانی (برای n فرآیند) ← نمی‌خواهیم busy waiting

انحصار متقابل

```
/* program mutualexclusion */
const int n = /* number of processes */
semaphore s = 1;
void P(int i)
{
  while (True) {
    wait(s);
    critical_section(i);
    signal(s);
    non_critical_section(i);
  }
}

void main()
{
  parbegin (P(1), P(2), ..., P(n));
}
```

مقدار اولیه 1 داریم و با این کار 1 به 0 می‌رود

وارد ناحیه بحرانی می‌شود، هر کس دیگری این کد اجرا می‌شود نمی‌تواند وارد ناحیه بحرانی شود

اگر صدا زده شد یکی از فرآیندهای منتظر را آزاد می‌کند اگر کسی نبود که آزاد کند count 1 می‌کند تا کسی آمد بتواند ناحیه بحرانی استفاده کند

این راه حل ساده هم کار می‌کند هم busy waiting نیست

← قدرت طلق عدد صحیح (s.count) نشان می‌دهد چند فرآیند در صف داریم.

1. انتظار محدود وارد چه کسی در صف می‌گذاریم چه نوبت دهی می‌کنیم
2. چه مقدار اولیه 1 بود فقط یک فرآیند می‌تواند وارد شود اگر 2 می‌دادیم، 2 تا فرآیند وارد می‌شوند
3. کسانی که داوطلب نیستند که در صف قرار نمی‌گیرند پس اصلا شرکت داده نمی‌شوند

دانشکده مهندسی کامپیوتر و فناوری
بازارچه دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

پیاده‌سازی این با شرط انحصاری بودن wait و signal خیلی سخت است ولی بدون این شرط حالا این راه‌حل کار نمی‌کند. مثلاً یک زمانبند 1 دیده wait را انجام داد ولی هنوز signal نکرده بود زمانبندی هم 1 می‌بیند و ... هم وارد ناحیه بحرانی می‌شود.

با داشتن این ابزار می‌توانیم کاربردهای دیگری نیز داشته باشیم

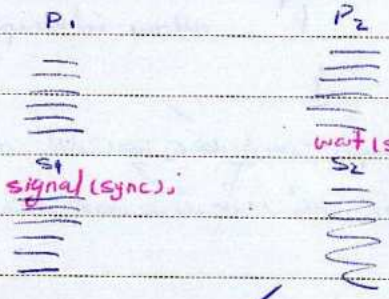
← دو فرآیند P_1 و P_2 داریم که دستورالعمل‌های به صورت همروند انجام می‌دهند

سیستم طوری است که S_1 باید زودتر از S_2 انجام شود (وقتی که S_1 چیزی را تولید می‌کند که S_2 باید از آن استفاده کند)

S_1 زودتر از S_2 اجرا شود: حکم

مسئله قابل تضمین نیست ← حکم زمانی ← با تعریف یک semaphore این کار را انجام می‌دهیم

Semaphore sync = 0 =>



قبل از S_2 wait می‌کنیم و S_1 را انجام می‌دهیم و جلو S_2 block می‌شود و وقتی S_1 انجام شد S_2 را آزاد می‌کند. الزاماً S_1 زودتر پیش اجرا می‌شود و sync = 1 می‌شود جلو wait می‌کنیم 0 می‌شود و منفی نیست S_2 حکم انجام می‌شود.

پیاده‌سازی:

دو راه‌حل وجود دارد:

راه‌حل اول: چیزی که این کار را قطع می‌کند وقفه حکم زمانی است پس اگر همه وقفه‌ها را از کار بیندازیم دیگر قطع نمی‌شود

البته باید برای از کار انداختن وقفه‌ها در System mode باشیم

```
void wait (semaphore S)
{
    inhibit interrupt;      → interrupt از کار انداختن
    s.count;
    if (s.count < 0) {
        place this ...
        block this ... , allow interrupts → interrupt به کار انداختن
    }
    else allow interrupts
}
```

دانشکده مهندسی کامپیوتر
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

← کی داره این کارها را انجام می‌دهد؟ یکی غیر از فرآیند چون در user mode است و عملاً این کارها را OS انجام می‌دهد و وقتی مسدود می‌کند خودش را مسدود می‌کند

```
void Signal (semaphore s)
{
  inhibit interrupts;
  s.count++;
  if (s.count <= 0) {
      remove a ... ;
      place ... ;
  }
  allow interrupts;
}
```

← راه حل خوبی نیست چون از کار انداختن interrupt خطرناک است چون ممکن است interrupt های مهمی بیایند که همان موقع باید بهشون پاسخ دهیم (مثلاً اینکه یک متغیری از max خودگذشت مثل درجه حرارت) و کی ممکن است استفاده درست نشد و بستگی به کاربرد داره.
ما کاری می‌کنیم که این را قفل نکنیم ← زمان ما در این فاصله از دست می‌رود

← پس راه حل مناسب‌تری داریم؟ همان جعبه جادویی

```
void wait (semaphore s)
{
  while (!testset (s.flag));   ← اینجا نوعی flag هم داره OS که عوض شده
  s.count--;
  if (s.count < 0) {
      place ... ;
      block ... ;
      set s.flag to zero;
  }
  else set s.flag to zero;
}
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

```
void signal                    s.flag
{
while (!testset (s.flag));
s.count ++;
if (s.count <=0) {
  remove ...
  place ...
}
  s.flag = 0;
}
```

signal(s), wait(s) ناحیه بحرانی تعریف کردیم یعنی گفتیم اول test & set کند که ببیند می‌تواند flag را بدست بیاورد یا نه

test & set (flag)	test & set (flag)
wait (s)	signal (s)
set flag to 0	set flag to 0

test&set ← چک می‌کنیم که flag را بدست آوردیم

این جا دیگر interrupt کار نداریم. با این روش ایمنی بودن نداریم ولی اثر ایمنی بودن را بدست می‌آوریم
این جا هم می‌گوییم چون داریم wait یا signal می‌کنیم چون می‌توانی flag را بدست آوردی پس
می‌توانی wait و signal کنی. حالا کم زمانی را از تا بگیرد مهم نیست چون می‌تواند flag را بگیرد پس
می‌تواند wait یا signal کند

ظاهر قضیه این است که داریم نقض غرض می‌کنیم. روش‌های قبلی برای ناحیه بحرانی که جلک مناسب بود و ما
در ناحیه بحرانی می‌ماندیم دقیقاً چقدر طول می‌کشد ولی در این جا دقیقاً می‌دانیم چقدر طول می‌کشد

Subject:
Year. Month. Date. ( )

داریم test & set برای جایی به کاری بریم که کوتاه است و بیخ کار آن تمام می‌شود ولی معطل می‌مانیم چون انجام عمل wait، signal کوتاه است.

اما ناحیه بحرانی اصلی را wait و signal کنترل می‌کنند

← برای حسد و شرکت کار می‌کنند

test & set (flag)

wait (s) → قفل را می‌بندد

flag = 0

ناحیه بحرانی

test & set (flag)

signal → از حالت حسد خارج شد

flag = 1

اجرای wait و signal ← پشتیبانی OS را می‌خواهد

← حسد و درصف قرار دادن

حتماً باید System call باشد نمی‌تواند باشد بلکه هرچیزی نمی‌تواند یک library باشد که از چند تا System call استفاده کند به طور صریح یا ضمنی از System Call استفاده می‌کند

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

جلسه هفدهم 97.8.21

مسئله حافظه محدود

1- تولیدکننده و مصرف کننده ← حافظه، کفا توسط باز بود حالا با کامفور انجام می‌دهیم

```
/* Program boundedbuffer */
const int sizeofbuffer = /* bufferSize */
Semaphor s = 1 ;
Semaphor n = 0 ;
Semaphor e = n ; + sizeofbuffer

void producer ( )
{
while (True) {
produce (s);
wait (e);
wait (s);
append (s);
signal (s) ;
signal (n);
}

void consumer( )
{
while (True) {
wait (n);
wait (s);
take ;
signal (s);
```

کد تولیدکننده

انتظار پر بودن

دسترسی باز کنترل می‌کند

آیتم را به باز append می‌کند یعنی در باز بگذار

اگر خالی باشد می‌خوابیم از آن چیزی برداریم

مقدار را از باز برمیدار

انتظار خالی بودن

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

signal(e); ← تعداد سلول های خالی را یکی زیاد می کند

consume();

void main

یعنی semaphore s برای دسترسی اعضای به کار رفته در نتیجه مقدار اولیه = 1

اگر می خواستیم دربار بگذاریم ← جلو می کردیم باز پر نباشد

" " از " بخوانیم ← " " " " خالی "

چون e برای تعداد سلول پر، نشان بودن است باید به اندازه سایز باز باشد چون الم اضافه کنیم یکی از ش کم می شود

می توانیم اضافه کنیم تا جایی که صفر بشود بعد منفی می شود و دیگر می توانیم از wait عبور کنیم

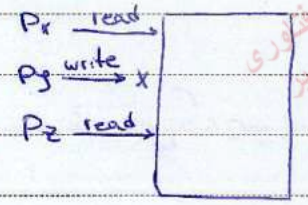
n برای تعداد سلول خالی بودن است ← تعداد سلول های پر شده را نشان می دهد

e تعداد سلول های خالی را نشان می دهد.

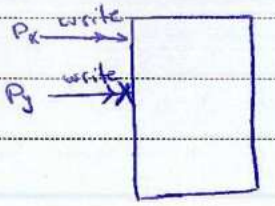
← پس همواره این ممانعت ها فقط s برای ناحیه بحرانی به کار رفته و بقیه برای تعداد سلول های پر است و این مثال می گوید که ممانعت برای ناحیه بحرانی فقط نیست

2. خواننده ها و نویسنده ها: Readers & writers prob.

اینجا یک object مشترک مثل file وجود دارد و ترد های مختلف می توانند از این object بخوانند یا بنویسند ولی همه همگی نمی توانند این کار را انجام دهند مثلاً در حین خواندن دیگری می تواند در آن بنویسد



برای read کردن همزمان می توانند انجام دهند چون اطلاعات عوض نمی شود و اگر یکی دارد read می کند فقط می توان در آن write کرد



ولی وقتی write می کنیم هیچ مشکلی نمی توانیم انجام دهیم چون اطلاعات قابل اعتماد نیست و عملیات write انحصاری است

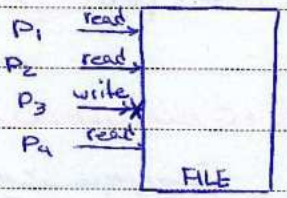
دانشگاه صنعتی امیر کبیر
دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیر کبیر

Subject:
Year. Month. Date. ( )

اینکه P_4 هم باید منتظر بماند یا نه بستگی به مسئله دارد:

1. سیستم عامل به صورتی است که هر کدام که صف بیشتر هم انجام شوند

2. شاید اینجا ما بخواهیم بخواهند write به چه کار اند



First readers & writes prob ← به P_4 اجازه خواندن دهیم

Second " " " " ← به P_4 اجازه ندهیم قبل از write بخواند

(اختصاری بودن write در هر دو مشترک است.)

First readers & writers prob: ←

دستور کنید ← هم read و هم write می‌کنند

خواننده نویسنده: P_w (نوشتن اختصاری است.)

wrt ← برای رشته‌ی اختصاری به کار بردیم پس مقدار اولیه‌اش یک است

```
Pw
wait(wrt);
writing();
signal(wrt);
```

از این جا رشته‌ی توالی خواندن را انجام می‌دهی

```
Pr
wait(mutex);
read count++;
if(read == 1)
    wait(wrt);
signal(mutex);
reading()
wait(mutex)
readcount--;
if readcount == 0
    signal(wrt);
signal(mutex);
```

برای تقید readcount است

جلوی نفی در حال write باشد

خروجی

خواننده: P_r

اگر خواننده اول باشد با نوشتن رفتارش متفاوت است

اول بودن ← باید جلوی کسی که کسی می‌نویسد

اول نبودن ← لازم نیست جلوی

readcount ← مقدار خواننده‌ها پس هر کدام خواهد

بخواند باید این حساب را یکی زیاد کند

← mutex دیگری به readcount را اختصاری می‌کنیم

پس مقدار اولیه‌اش یک است

بعد از خواندن باید ببینیم اگر آخرین خواننده هستی باید

آن کسی را که منتظر write است آزاد کنی

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیر کبیر

Subject:
Year. Month. Date. ( )

بخش آخر (Second readers & writers prob. را آخر حل کنید

3. فلاسفه در حال غذا خوردن : Dining Philosophers

این فیلسوف ها چینی هستند

هر کسی می خواهد غذا بخورد باید دو تا چوب در اختیار داشته باشد

دیگر نباید منتظر بماند

مرابط ها: فیلسوف ها (همیشه در حال خوردن نیستند یعنی وقتا متفکر می باشند)

منابع: چوب ها

Function : eating (عمل غذا خوردن) و thinking

برای هر chopstick یک semaphore

دقت کنید اگر بن بست پیش اومد چی میشه

```
Semaphore chopstick [5];
void P(i)
{
  while True {
  wait(chopstick[i]);
  wait (chopstick[(i+1) % 5]);
  eating ( );
  signal (chopstick [i]);
  signal ( chopstick [(i+1) % 5]);
  thinking ( );
}

void main ( )
{ parbegin (P(0), P(1), ..., P(4));
{
```

بالا دو خط wait کردن این دو را است

=>

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

اما این راه حل یک مشکل دارد.

semaphore ابزاری قوی برای هماهنگی است ولی مشکلات دارد

Subject:
Year. Month. Date. ( )

مشکلات Semaphore

1. کلاً Semaphore در اختیار برنامه‌نویس است و این باعث مشکلاتی می‌شود.

1. بکارگیری صحیح ← به ازای هر wait باید یک signal داشته باشیم و برعکس چون ممکن به deadlock برسد

wait(s);
≡ ⇒ صحیح
signal(s);

signal(s);
≡
wait(s);
این دو تا اصلاً متقابل

wait(s);
≡
wait(s);
بن بست

wait(s);
≡
wait(s);

2. بن بست. ممکن است پیش بیاید حتی اگر درست استفاده کنیم

هر دو فرآیند کاملاً قانونی هستند

P_1	P_2
① wait(p);	② wait(q);
④ wait(q);	③ wait(p);
signal(p);	signal(q);
signal(q);	signal(p);

① باید کم زمانی ④ ← مسدود چون wait در P_2 شده

③ ← نمی‌تواند رد شود ← مسدود تا کی؟ p را سیگنال کند

اینجا هر دو تا منتظر منابع برای خلاص می‌مانند

این مشکلات در مسئله Dining Philosopher وجود دارند ← همه یکدونه چوب بردارند منتظر دومی باشند در نتیجه هیچ وقت این مشکل حل نمی‌شود چطور نمی‌ذارند دیگری منتظر این منبع است

← می‌شود کاربرهای از کار خود بیکار کرد که به بدی نمی‌خورد

راه حل این مشکل اینست که اساساً اختیار به برنامه‌نویس ندهیم که هر کاری می‌خواهد بکند پس به سراغ ابزار سطح بالاتری می‌رویم ← monitore (مانیتور یا ناظر)

اینجا صف نشین هم نیستند بلکه هر کسی در جای خودش استفاده می‌شود و جای همدیگر را نمی‌گیرند

دانشکده مهندسی کامپیوتر و فناوری
باشگاه کامپیوتر دانشگاه صنعتی امیرکبیر

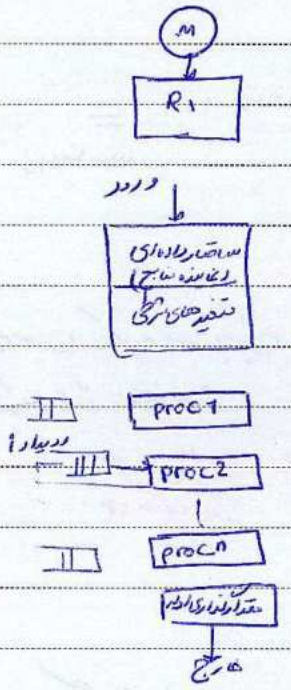
Subject:
Year. Month. Date. ( )

تمرین سری 9, 5, 6, 7, 14, 15

جلسه هجدهم 8, 26

یکی از ابزارهای دیگر جهت ایجاد هماهنگی بین فرایندهای باشد البته می‌تواند جایگزین سمافور شود. اختیارات محدودتر است و یکی از اشتباهات سمافور مدیریت نمی‌کند.

به طور کلی کنترل قابل پیدا منبع دارد و پس عملیات هم کنترل قابل دارد پس اختیارات محدودتر می‌شود



monitor ساختار

۱. هر موقع متصل به یک فرایند داخل monitor فعال باشد

در غیر این صورت اجازه متقابل منتظر می‌شود

تعداد این صف‌ها متغیر می‌تواند چندین تا باشد

اگر یکی از صف‌ها بیدار شد منتظر شد می‌توان فرایند دیگری را

وارد کرد و این اجازه متقابل را از بین می‌برد

فرایند قبلی یا خارج شده است یا مسدود

این صف‌ها متغیرهای سراسری هستند

مثال) تولیدکننده و مصرف‌کننده.

```
/* Program producer Consumer */
monitor boundedbuffer;
char buffer[N];
int nextin, nextout;
int count;
cond notfull, notempty
```

ساختمان داده‌های سابقه

→ می‌تواند هر چیزی غیر از char باشد

— semaphor mutex

condition

دانشکده مهندسی کامپیوتر و علوم اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

```
void append (char x)
{   wait(mutex)
    if (count==N)                    →  شرط پیروزی
        cwait(notfull)
    buffer[nextin]=x;
    nextin=(nextin+1)%N;
    count++;
    csignal(notempty)   signal(mutex)
}

void take(char x)
{   wait(mutex);
    if (count==0)
        cwait(notempty);
    X=buffer[nextout];
    nextout=[nextout+1)%N;
    count--;
    csignal(notfull);   signal(mutex);
}
    nextin=0;  nextout=0;  count=0;
```

این کد به producer بوده consumer ← monitor بوده

```
void Producer()
{
    char x;
    while (True) {
        produce(x);
        boundedbuffer.append(x);
    }}
```

دانشکده مهندسی کامپیوتر و فناوری اطلاعات دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

```
void Consumer
{ char x;
    while (True) {
    boundedbuffer take (x);
    consumer (x);
  }

void main ()
{
  parbegin (producer, consumer);
}
```

استفاده از این monitor اعضای هست این اعضای بولن که کجا مشخص شده است؟

در keyword ، monitor هست که وقتی این کلمه compiler می‌بیند، compiler بدونه این keyword می‌خواد که یک کلمه‌ی بعدی باید به آن اضافه شود و این یک کلمه‌ی بعدی می‌باشد و باید یکسری چیزها به آن اضافه شود تا صحت شود حالا به جز غیر دقیق متغیرهای * را اضافه می‌کند

← جمله‌ای که از monitor استفاده می‌شود باید از compiler به معنای خاصی برای monitor داشته باشد استفاده کند. در حالت دقیق‌تر باید در متن cwait هم چیزهایی اضافه می‌شود

مثال) فیلسوف‌ها ← غیر از حالت eating و thinking حالت hungry هم دارند

```
monitor Dining Philosophers;

enum itemation { hungry, eating, thinking }
int action state[5];
condition self[5];
```

دانشکده مهندسی کامپیوتر
دانشگاه صنعتی امیرکبیر

Subject:
Year. Month. Date. ( )

```
void pickup(i)
{ state[i] = hungry;
  test(i);
  if(state[i] != eating)
        cwait(self[i]);
}

void put down(i)
{ state[i] = thinking;
  test((i+1)%5);
  test((i+4)%5);
}

  test(i)
  {
     if (state[i] = hungry & state[(i+1)%5] != eating &
                        state[(i+4)%5] != eating) {
        state[i] = eating;
        csignal(self[i]);
     }
  }

for(i=0; i<5; i++)
{ state[i] = thinking;
}

void p(i)
{ while (True) {
  Dining Philosophers.pickup(i);
   eating();
   Dining philosophers.put down(i);
   thinking(i);
}}
```

دانشکده مهندسی کامپیوتر و فناوری
اطلاعات دانشگاه صنعتی امیر کبیر

P4PCO