Operating Systems

Isfahan University of Technology
Electrical and Computer Engineering Department

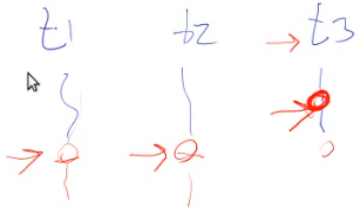
Zeinab Zali

Synchronization-Condition Variables-monitors

همگام سازی - متغیرهای شرط - مانیتور

Condition Variable

- there are many cases where a thread wishes to check whether a condition is true before continuing its execution
- To wait for a condition to become true, a thread can make use of what is known as a condition variable
- A condition variable is an explicit queue that threads can put themselves on when some state of execution (i.e., some condition) is not as desired (by waiting on the condition)
- some other thread, when it changes said state, can then wake one (or more) of those waiting threads and thus allow them to continue (by signaling on the condition)



-
Condition Variable چی هست؟
یه موقعی هست که نیاز داریم که یه جایی یه کاری انجام بشه و بعد ادامه کار رو انجام بدیم ینی اجرای یک کدی یا تردی رو مشروط به برقرار بودن یک شرط بکنیم تا وقتی که اون شرط برقرار نشده اون ترد در حالت بلاک بره و cpu بهش تعلق نگیره تا وقتی که اون شرط برقرار شد و ترد بتونه ادامه پیدا کنه
مثلا چندین تا ترد داریم و این ها توی یک نقطه ای باید بهم دیگه سینک بشن مثلا اگر ترد t1 , t2 ,t3 داشته باشیم اینا یه سری خطوطی واسه اجرا دارن و می خوایم ممطئن بشیم که از یه جایی به بعد توی ترد t1 , t2 حتما t3 هم به یک نقطه خاصی رسیده باشه -->ما نمی دونیم که سیستم اینارو چجوری اسکجول می کنه ممکنه که t1,t2 به اون خط رسیده باشند در حالی که t3 هنوز نرسیده باشه اون نقطه می خوایم یه کاری بکنیم که t1,t2 توی اون خط بلاک بشن و دیگه پیش نرن تا اینکه t3 برسه به اون نقطه مورد نظر و بعد که رسید t1,t2 بتونن به اجراشون ادامه بدن
متن توی این اسلاید:
Condition Variable: وقتی که یک تردی رو دوست داریم که یک شرط خاصی رو چک بکنه و اگر اون شرط برقرار بود اجراش ادامه پیدا بکنه و اگر برقرار نبود همون جا بلاک بشه در این صورت می تونیم از Condition Variable استفاده بکنیم
از طرف دیگه یک ترد دیگه میتونه وجود داشته باشه که اگر اون شرط برقرار شد تردی که منتظر برقرار شرط بوده رو بتونه wake اش بکنه
پس Condition Variable یک صفی میشه که تردهایی که خودشون تمایل داشته باشند رو می ریزه توی این صف تا وقتی که اون Condition که مدنظر است برقرار بشه و تا وقتی که اون Condition برقرارنیست این تردها می رن توی اون صف و دیگه اجرا نمیشن
و از طرف دیگه یک ترد دیگه می تونه در صورتی که شرط برقرار بود اون تردهایی که رفتن توی صف رو wakeشون بکنه حالا ممکنه یکشون رو wake بکنه یا همشون یا چندتاشون..
پس اینجا این حالتو داریم که میتونه یه جوری سیگنال بکنه که چندتا ترد با هم wake بشن یا می تونه هم جوری باشه اون سیگنال که یکیشون wake بشه فقط

CV definitions and routines

- pthread_cond_t c
- pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m): The wait() call is executed when a thread wishes to put itself to sleep
  - The responsibility of wait() is to release the lock and put the calling thread to sleep (atomically); when the thread wakes up (after some other thread has signaled it), it must re-acquire the lock before returning to the caller.
- pthread_cond_signal(pthread_cond_t *c): the signal() call is executed when a thread has changed something in the program and thus wants to wake a sleeping thread waiting on this condition

-
pthread_cond_t c: وقتی میخوایم CV رو بسازیم از این استفاده میکنیم
و برای wait کردن روی CV از تابع pthread_cond_wait استفاده میکنیم که باید دوتا ارگومان ورودی بگیره یکی همون CV است که توی تابع قبلی گفتیم اینو و یکی هم mutex است که ما همیشه این mutex رو نیاز داریم
وقتی که داریم wait می کنیم روی CV یا سیگنال میکنیم حتما باید قبلش هم با استفاده از یک mutex باید اون ناحیه ای که داره از CV استفاده میکنه رو قفلش بکنیم
پس CV غیر از اینکه خودش امکان این wait کردن و سیگنال کردن رو داره باید حتما وقتی که داریم ازش استفاده میکنیم اون ناحیه رو داخل یک lock قرارش بدیم و مثلا می تونیم از یک mutex استفاده بکنیم و با اون mutex بیایم قفلش بکنیم
و وقتی هم که میخوایم سیگنال بکنیم کافیه که فقط CV رو به سیگنال بدیم تا این سیگنال بدونه باید کجارو wake بکنه
فرق دیگه ای که این wait با wait های سمافور داره اینه که اون تردی که این wait رو داره فراخوانی میکنه حتما خودش رو می بره توی حالت lock و بلاک میشه در واقع ولی توی سمافور اگه اون مقدارش بیشتر از صفر بود اون lock نمیشد و می تونست رد بشه ولی اینجا توی wait همیشه اون تردی که wait رو فراخوانی کرده رو توی حالت بلاک می بره
پس این یک wait است که با اختیار یک ترد اون ترد رو می بره توی حالت بلاک ولی امکان این رو داریم که از جای دیگه با تردهای دیگه با یک فراخوانی سیگنال هرجایی که نیاز داشتیم رو اون تردی که رفته بود توی حالت wait رو بیدارش بکنیم

CV example 1

```
1   void *child(void *arg) {
2       printf("child\n");
3       // XXX how to indicate we are done?
4       return NULL;
5   }
6
7   int main(int argc, char *argv[]) {
8       printf("parent: begin\n");
9       pthread_t c;
10      Pthread_create(&c, NULL, child, NULL); // create child
11      // XXX how to wait for child?
12      printf("parent: end\n");
13      return 0;
14  }
```

Desired Output:

```
parent: begin
child
parent: end
```

مثال:
یک مین یک تردی رو می سازه و بعد میخوایم مطمئن بشیم که بدنه ترد حتما اجرا میشه کامل و بعد مینمون تموم بشه
کاری که قبلا می کردیم این بود که join می ذاشتیم روی ترد
ما اینجا می خوایم مثلا خودمون جوین رو پیاده سازی بکنیم
یا مثلا توی فانکشن ترد که داره اجرا میشه ممکنه به یک نقطه خاصی که رسید اون فانکشن ترد از اونجا به بعد دیگه به مین اجازه بدیم که اگر خواست اتمام پیدا کنه و اینطور نباشه که اجرای کامل فانکشن ترد رو بخوایم و اینجا میخوایم به صورت دستی بریم جلو به جای نوشتن جوین

اجرا:
در این کد اول حتما childمون کامل اجرا بشه و بعد parent خاتمه پیدا کنه

نکته: اون ناحیه ای که داره از CV استفاده می کنه رو باید قفل بکنیم و همینطور بخاطر متغییر شیر هم از mutex استفاده میکنیم --> متغییر done --> بخاطر همزمانی توی مقدار done توی چندتا ترد است پس از mutex قبلش استفاده میکنیم

```
1  int done  = 0;
2  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
3  pthread_cond_t c  = PTHREAD_COND_INITIALIZER;
4
5  void thr_exit() {
6
7
8
9
10 }
11
12 void *child(void *arg) {
13     printf("child\n");
14     thr_exit();
15     return NULL;
16 }
17
18 void thr_join() {
19
20
21
22
23 }
24
25 int main(int argc, char *argv[]) {
26     printf("parent: begin\n");
27     pthread_t p;
28     Pthread_create(&p, NULL, child, NULL);
29     thr_join();
30     printf("parent: end\n");
31     return 0;
32 }
```

بنابراین نیاز داریم که دو تابع اینجا بنویسیم:
یک تابع جوین و یک تابع exit و به این صورت استفاده میکنیم که بعد از ساختن ترد توی مین ابتدا جوین رو فراخوانی میکنیم و توی فرزند هم هر جایی که خواستیم exit رو فراخوانی میکنیم و اونجا به این معنی است که دیگه الان جایی که جوین فراخوانی شده می تونه ازش رد بشه و دیگه لازم نیست اینجا بلاک بشه برنامه روی خط جوین
پس هدف این است که این توابع جوین و exit رو به نحوی خودمون کدش رو بنویسیم

```
1   int done  = 0;
2   pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
3   pthread_cond_t c = PTHREAD_COND_INITIALIZER;
4
5   void thr_exit() {
6       pthread_mutex_lock(&m);
7       done=1;
8       pthread_cond_signal(&c)
9       pthread_mutex_unlock(&m)
0   }
1
2   void *child(void *arg) {
3       printf("child\n");
4       thr_exit();
5       return NULL;
6   }
7
8   void thr_join() {
9      pthread_mutex_lock(&m);
0      while(done==0)
1          pthread_cond(wait(&c,&m);
2      pthread_mutex_unlock(&m);
3   }
4
5   int main(int argc, char *argv[]) {
6       printf("parent: begin\n");
7       pthread_t p;
8       Pthread_create(&p, NULL, child, NULL);
9       thr_join();
0       printf("parent: end\n");
1       return 0;
2   }
```

What is the problem with this solution?

```
1   void thr_exit() {
2       Pthread_mutex_lock(&m);
3       Pthread_cond_signal(&c);
4       Pthread_mutex_unlock(&m);
5   }
6
7   void thr_join() {
8       Pthread_mutex_lock(&m);
9       Pthread_cond_wait(&c, &m);
10      Pthread_mutex_unlock(&m);
11  }
```

راه حل:
اینجا یک mutex تعریف کردیم که اینجا گلوبال است توی این برنامه

اگر کدمون رو به این صورت پیاده سازی میکردیم:
ینی done نمی ذاشتیم داخلش
در این حالت مشکلی که پیش میاد این است که:
این سناریو در نظر بگیر که این ترد ساخته میشه و والد این ترد رو می سازه و ترد زودتر از والد اجرا میشه ینی زودتر از والد اسکجولر میشه ینی اجرا میاد روی خط بچه و بعد thr_exit فراخوانی میشه و هنوز thr_join فراخوانی نشده باشه در این صورت اگر کد به صورت روبرو پیاده شده باشه اول m قفل میشه و بعد CV رو سیگنال میکنه در صورتیکه هنوز جایی wait فراخوانی نشده در این حالت CV به این صورت عمل میکنه که از این سیگنال رد میشه و عملا انگار کار خاصی انجام نمیده چون چیزی نبوده که بخواد wakeاش بکنه و بعد Exit و childمون کامل میشه
اینجا CV ما مقدار نداره
حالا که اجرای ترد کامل شد و والد میخواد جوین رو فراخوانی بکنه در این حالت ابتدا m رو لاک میکنه و بعد میاد روی خط wait و چون if هم اینجا نداریم پس می ره توی حالت لاک و چون ترد فرزند هم سیگنال رو قبلا فراخوانی کرده و رد شده دیگ این توی لاک می مونه پس برنامه توی مین دیگه جلو نمی ره هیچ وقت و توی همون خط جوین لاک میشه

پس اینجا نشون میده که حتما باید از اون done استفاده میکردیم و اونو چک میکردیم

What is the problem with this solution?

```
1   void thr_exit() {
2       done = 1;
3       Pthread_cond_signal(&c);
4   }
5
6   void thr_join() {
7       if (done == 0)
8           Pthread_cond_wait(&c);
9   }
```

اگر بچه هنوز اجرا نشده ینی done اش هنوز صفره اینجا wait بشه و خودشو ببره توی حالت بلاک این مال قسمت جوین است

توی این حالت اگر به جای while از if می رفتیم چی میشد؟ توی مثال بعدی جواب اینو میده

Producer/Consumer Problem

```
1   int loops; // must initialize somewhere...
2   cond_t  cond;
3   mutex_t mutex;
4
5   void *producer(void *arg) {
6       int i;
7       for (i = 0; i < loops; i++) {
8                                                   // p1
9                                                   // p2
10                                                  // p3
11                                                  // p4
12                                                  // p5
13                                                  // p6
14      }
15  }
16
17  void *consumer(void *arg) {
18      int i;
19      for (i = 0; i < loops; i++) {
20                                                  // c1
21                                                  // c2
22                                                  // c3
23                                                  // c4
24                                                  // c5
25                                                  // c6
26
27      }
28  }
```

مثال:
فعلا اندازه بافر رو یک در نظر می گیریم
تولیدکننده مرتبا می تونه مقداری رو قرار بده توی اون بافر و مصرف کننده هم باید از اون بافر مصرف کنه با توابع put , get
که put رو تولیدکننده استفاده میکنه و get رو مصرف کننده
تابع assert نشون دهنده خطا است مثلا برای put میگه اگر count != 0 شد باید خطا بده در غیر اینطوری که میره خط های پایینی رو اجرا میکنه
count در ابتدا صفر است ینی مقدار دهی اولیه اش را صفر می ذاریم چون بافر در ابتدا خالی است

نکته: برای تولیدکننده --> اگر count=1 بود ینی بافر ما الان پر است پس باید صبر بکنه که خالی بشه و بعد داخل بافر بریزه
برای مصرف کننده هم اگر count=0 بود باید صبر بکنه چون بافر خالی است و باید صبر بکنه تا بافر پر بشه
put(i) ینی مقدار i رو توی بافر قرار میده و بعد سیگنال می کنیم CV رو
توی مصرف کننده --> get می کنه یک مقداری رو از بافر و بعد سیگنال می کنیم CV رو

جلوتر کد کاملش هست....

Producer/Consumer Problem

```
1   int buffer;
2   int count = 0; // initially, empty
3
4   void put(int value) {
5       assert(count == 0);
6       count = 1;
7       buffer = value;
8   }
9
10  int get() {
11      assert(count == 1);
12      count = 0;
13      return buffer;
14  }
```

Producer/Consumer Problem

```
1   int loops; // must initialize somewhere...
2   cond_t  cond;
3   mutex_t mutex;
4
5   void *producer(void *arg) {
6       int i;
7       for (i = 0; i < loops; i++) {
8           Pthread_mutex_lock(&mutex);            // p1
9           if (count == 1)                        // p2
10              Pthread_cond_wait(&cond, &mutex);  // p3
11          put(i);                                // p4
12          Pthread_cond_signal(&cond);            // p5
13          Pthread_mutex_unlock(&mutex);          // p6
14      }
15  }
16
17  void *consumer(void *arg) {
18      int i;
19      for (i = 0; i < loops; i++) {
20          Pthread_mutex_lock(&mutex);            // c1
21          if (count == 0)                        // c2
22              Pthread_cond_wait(&cond, &mutex);  // c3
23          int tmp = get();                       // c4
24          Pthread_cond_signal(&cond);            // c5
25          Pthread_mutex_unlock(&mutex);          // c6
26          printf("%d\n", tmp);
27      }
28  }
```

Producer/Consumer broken solution with if

T_{c1}	State	T_{c2}	State	T_p	State	**Count**	*Comment*
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep		Ready	p1	Running	0	
	Sleep		Ready	p2	Running	0	
	Sleep		Ready	p4	Running	1	Buffer now full
	Ready		Ready	p5	Running	1	T_{c1} awoken
	Ready		Ready	p6	Running	1	
	Ready		Ready	p1	Running	1	
	Ready		Ready	p2	Running	1	
	Ready		Ready	p3	Sleep	1	Buffer full; sleep
	Ready	c1	Running		Sleep	1	T_{c2} sneaks in ...
	Ready	c2	Running		Sleep	1	
	Ready	c4	Running		Sleep	0	... and grabs data
	Ready	c5	Running		Ready	0	T_p awoken
	Ready	c6	Running		Ready	0	
c4	Running		Ready		Ready	0	Oh oh! No data

می خوایم کدی که صفحه قبلی زدیم رو چک بکنیم:

سناریو: توی این سناریو دوتا مصرف کننده داریم و یک دونه تولیدکننده
توی این جدول ما هرجایی که یکی از تردها می تونستن اجرا بشن و توی حالت ready بودن استیتشون رو ready نوشتیم و موقعی که cpu بهشون داده بشه استیتشون running است و وقتی که به دلیل wait می رن توی حالت بلاک رو استیتشون رو sleep گذاشتیم
فرض میکنیم توی لحظه اول یکی از مصرف کننده ها اجرا میشه مثلا Tc1 و بعد از اون به Tp میایم و cpu میدیم
توی نقطه p5 که سیگنال فراخوانی شد باید Tc1 بیاد ready بشه اما فرض میکنیم که همچنان cpu بهش تعلق نگرفته و cpu دوباره دست Tp است و بعد که میاد از اول و توی if می بینه که count=1 است پس بلاک میشه و دیگه نمی تونه بره جلو پس p3 میشه sleep
و بعد cpu داده میشه به Tc2
و بعد از این cpu دست Tc1 می افته ینی به Tc1 داده میشه و می خواد c4 اجرا بکنه و یک مقداری رو از توی بافر برداره ولی الان توی بافر چیزی نیست چون قبلش Tc2 اونو برداشته پس اینجا می ره توی تابع get و توی خط assert می بینه که count=0 است و توی این نقطه باعث میشه مصرف کننده قطع بشه و از برنامه بیاد بیرون پس اینجا مشکل ایجاد شد

باید چجوری این مسئله رو حل بکنیم الان که مشکل رفع بشه؟
اگر مصرف کننده به جای اون if می اومد while می ذاشت مشکل حل میشد --> چون بعد از اینکه به مصرف کننده cpu دوباره داده میشد همچنان باز توی حلقه بود و باید دوباره شرط حلقه رو چک می کرد

```
1   int loops;
2   cond_t  cond;
3   mutex_t mutex;
4
5   void *producer(void *arg) {
6       int i;
7       for (i = 0; i < loops; i++) {
8           Pthread_mutex_lock(&mutex);            // p1
9           while (count == 1)                     // p2
10              Pthread_cond_wait(&cond, &mutex);  // p3
11          put(i);                                // p4
12          Pthread_cond_signal(&cond);            // p5
13          Pthread_mutex_unlock(&mutex);          // p6
14      }
15  }
16
17  void *consumer(void *arg) {
18      int i;
19      for (i = 0; i < loops; i++) {
20          Pthread_mutex_lock(&mutex);            // c1
21          while (count == 0)                     // c2
22              Pthread_cond_wait(&cond, &mutex);  // c3
23          int tmp = get();                       // c4
24          Pthread_cond_signal(&cond);            // c5
25          Pthread_mutex_unlock(&mutex);          // c6
26          printf("%d\n", tmp);
27      }
28  }
```

```
1   int loops;
2   cond_t  cond;
3   mutex_t mutex;
4
5   void *producer(void *arg) {
6       int i;
7       for (i = 0; i < loops; i++) {
8           Pthread_mutex_lock(&mutex);            // p1
9           while (count == 1)                     // p2
10              Pthread_cond_wait(&cond, &mutex); // p3
```

TIP: USE WHILE (NOT IF) FOR CONDITIONS

When checking for a condition in a multi-threaded program, using a `while` loop is always correct; using an `if` statement only might be, depending on the semantics of signaling. Thus, always use `while` and your code will behave as expected.

```
18      int i;
19      for (i = 0; i < loops; i++) {
20          Pthread_mutex_lock(&mutex);            // c1
21          while (count == 0)                     // c2
22              Pthread_cond_wait(&cond, &mutex); // c3
23          int tmp = get();                       // c4
24          Pthread_cond_signal(&cond);            // c5
25          Pthread_mutex_unlock(&mutex);          // c6
26          printf("%d\n", tmp);
27      }
28  }
```

پس به طور کلی قانونی که گفته میشه این است که وقتی که داریم از CV ها استفاده می کنیم حتما سعی بکنیم از while استفاده بکنیم به جای if که توی حالت های خاص به همچین مشکل هایی نخورنیم و راه حلمون اشتباه نباشه

Producer/Consumer still a broken solution

T_{c1}	State	T_{c2}	State	T_p	State	**Count**	*Comment*
c1	Running		Ready		Ready	0	
c2	Running		Ready		Ready	0	
c3	Sleep		Ready		Ready	0	Nothing to get
	Sleep	c1	Running		Ready	0	
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	T_{c1} awoken
	Ready		Sleep	p6	Running	1	
	Ready		Sleep	p1	Running	1	
	Ready		Sleep	p2	Running	1	
	Ready		Sleep	p3	Sleep	1	Must sleep (full)
c2	Running		Sleep		Sleep	1	Recheck condition
c4	Running		Sleep		Sleep	0	T_{c1} grabs data
c5	Running		Ready		Sleep	0	Oops! Woke T_{c2}
c6	Running		Ready		Sleep	0	
c1	Running		Ready		Sleep	0	
c2	Running		Ready		Sleep	0	
c3	Sleep		Ready		Sleep	0	Nothing to get
	Sleep	c2	Running		Sleep	0	
	Sleep	c3	Sleep		Sleep	0	Everyone asleep...

سناریوی بعدی با داشتن while:
که باز باعث میشه راه حل ما راه حل درستی نباشه : خط اخر رو ببین همشون رفتن توی حالت sleep ینی یک حالتی شبیه بن بست اینجا پیش اومده

راه حل ما برای رفع این مشکل:
اینجا برای چک کردن پر بودن بافر و خالی بودن بافر از یک CV استفاده کردیم و راه درست این است که ما دوتا CV مختلف رو استفاده بکنیم --> صفحه بعدی...

Producer/Consumer with buffer size MAX

```
1   cond_t empty, fill;
2   mutex_t mutex;
3
4   void *producer(void *arg) {
5       int i;
6       for (i = 0; i < loops; i++) {
7           Pthread_mutex_lock(&mutex);             // p1
8           while (count == MAX)                    // p2
9               Pthread_cond_wait(&empty, &mutex); // p3
10          put(i);                                 // p4
11          Pthread_cond_signal(&fill);             // p5
12          Pthread_mutex_unlock(&mutex);           // p6
13      }
14  }
15
16  void *consumer(void *arg) {
17      int i;
18      for (i = 0; i < loops; i++) {
19          Pthread_mutex_lock(&mutex);             // c1
20          while (count == 0)                      // c2
21              Pthread_cond_wait(&fill, &mutex);   // c3
22          int tmp = get();                        // c4
23          Pthread_cond_signal(&empty);            // c5
24          Pthread_mutex_unlock(&mutex);           // c6
25          printf("%d\n", tmp);
26      }
27  }
```

Producer/Consumer with buffer size MAX

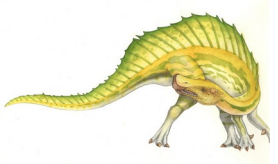
```
1   int buffer[MAX];
2   int fill_ptr = 0;
3   int use_ptr  = 0;
4   int count    = 0;
5
6   void put(int value) {
7       buffer[fill_ptr] = value;
8       fill_ptr = (fill_ptr + 1) % MAX;
9       count++;
10  }
11
12  int get() {
13      int tmp = buffer[use_ptr];
14      use_ptr = (use_ptr + 1) % MAX;
15      count--;
16      return tmp;
17  }
```

توابع put , get برای حالتی که اندازه بافر بیشتر از یک باشه هست مثلا مقدارش MAX است فرقش با قبلی ها این است که ما بافر رو به صورت چرخشی با put پرش میکنیم و توی مصرف کننده هم با get باید مقدار برداریم از توش و دیگه count رو صفر یا یک نمی کنیم بلکه کم یا زیاد می کنیم
و اینجا هم باید ایندکس بافر رو به درستی توی مصرف کننده یا تولیدکننده جلو ببریم

Monitors

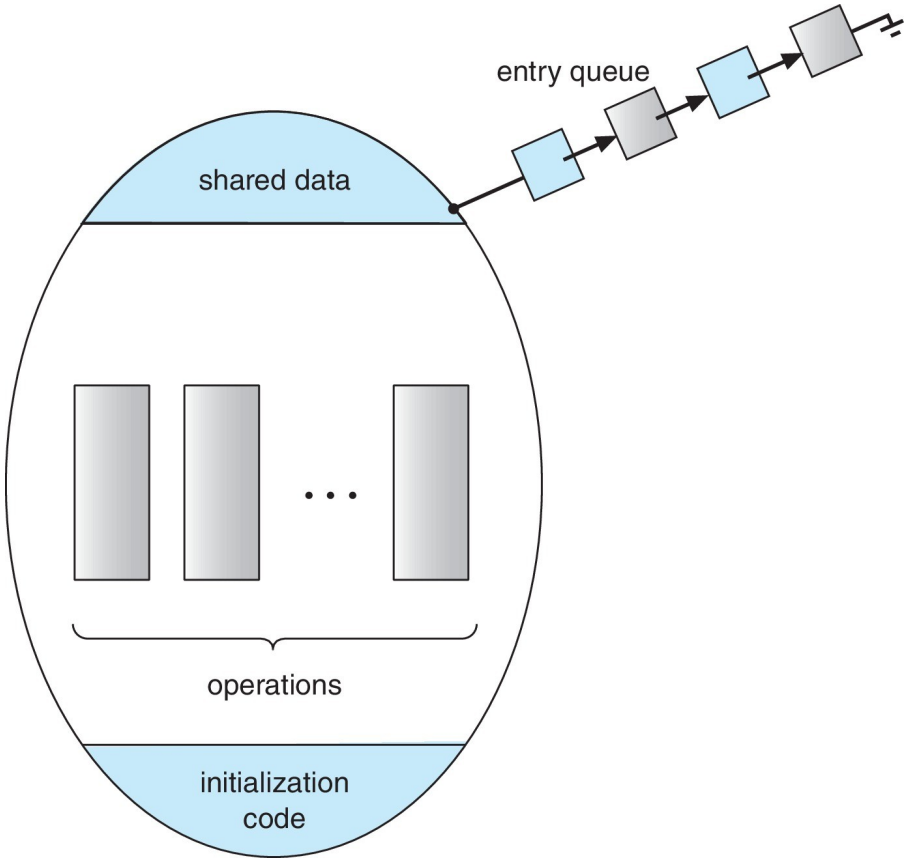
- A high-level abstraction that provides a convenient and effective mechanism for process synchronization
- *Abstract data type*, internal variables only accessible by code within the procedure
- Only one process may be active within the monitor at a time
- Pseudocode syntax of a monitor:

```
monitor monitor-name
{
  // shared variable declarations
  procedure P1 (…) { …. }

  procedure P2 (…) { …. }

  procedure Pn (…) {……}

  initialization code (…) { … }
}
```



-
یکی دیگر از ابزارهایی که برای همگام سازی پروسس ها در اختیارمون است Monitors است به چه صورت این مدل تعریف میشه؟
در واقع یک مانیتور شامل تعدادی shared variable و تعدادی procedure است که این shared variable ها فقط توسط این پروسیجرها قابل دسترسی و تغییر و اپدیت هستن و initialization code هم داریم مثل کانستراکتور کلاس که این shared variable ها رو برای دفعه اول initialize می کنه ولی از اون به بعد توسط پروسیجرها قابل اپدیت هستن
و این پروسیجرها هم خاصیتشون این است که اتومیک هستن ینی اگر دوتا پروسس توی مانیتور قرار بگیره ینی مثلا دوتا پروسس بخواد p1 رو از این مانیتور اجرا بکنه نمیتونه چون این مانیتور به صورتی نوشته شده که جلوگیری میکنه از این کار و در هر لحظه فقط یک پروسیجر می تونه اجرا بشه
نکته: فقط یک پروسس در یک زمان واحد می تونه داخل مانیتور باشه ینی تمام این پروسیجرها به طوری هستن که فقط یکیشون در هر لحظه قابل اجرا است و اگر پروسس های مختلفی بخوان توی یک پروسیجر باشن باز هم نمی تونن

Schematic view of a Monitor



اگر پروسس های مختلفی بخوان از این مانیتور استفاده بکنن در حالی که یکی از پروسس ها در حال استفاده از مانیتور باشه بقیه باید وارد صف بشن پس مثل این است که این مانیتور یک صف داره که اگر مانیتور ازاد نبود پروسس ها رو داخل صف قرار میده تا اینکه مانیتور ازاد بشه و یکی از پروسس ها بتونه از مانیتور استفاده بکنه
اپریشن ها در هر لحظه فقط یکیشون می تونه اجرا بشه

Monitor Implementation Using Semaphores

- Variables

```
semaphore mutex
mutex = 1
```

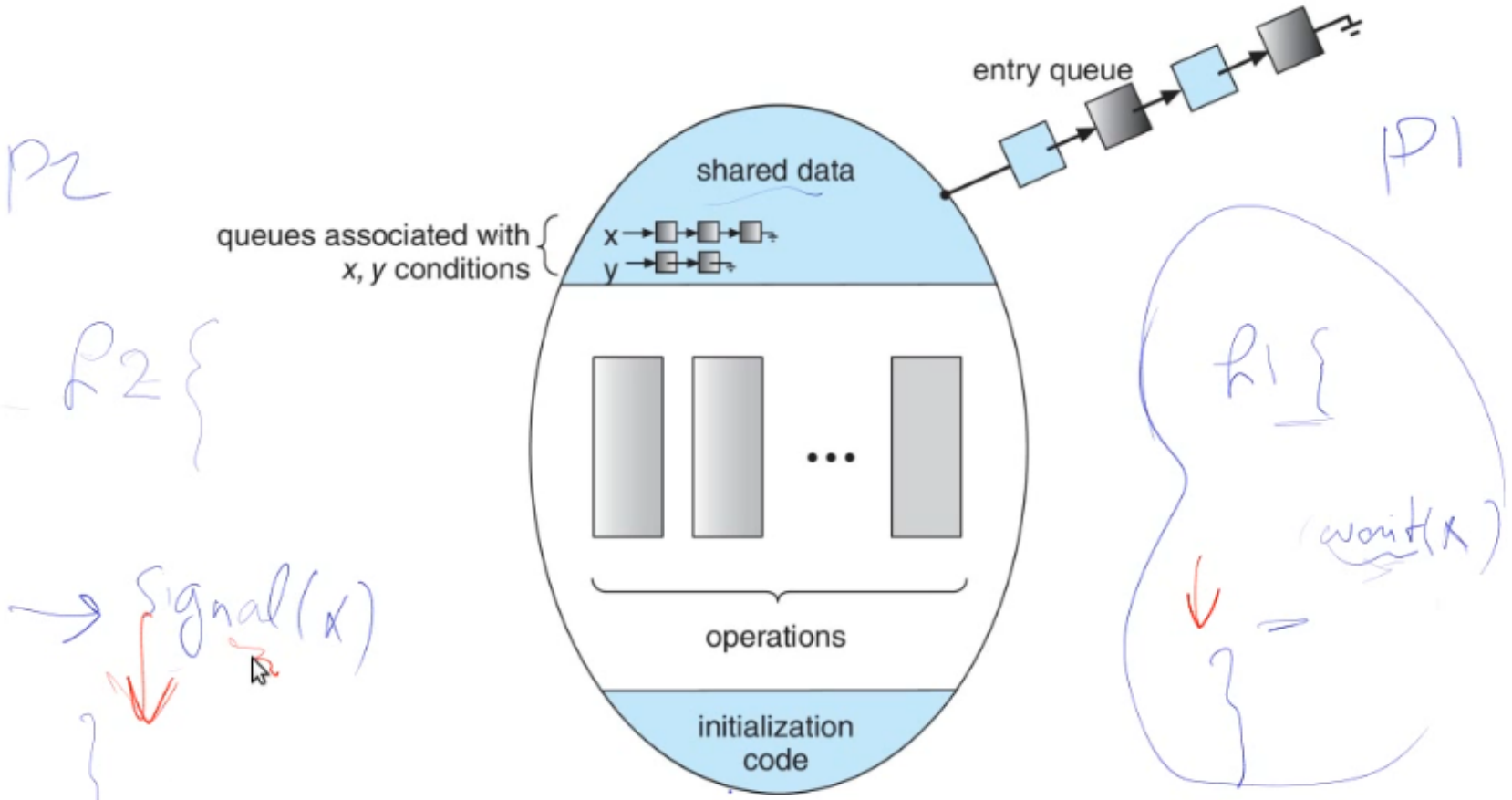
- Each procedure ***P*** is replaced by

```
wait(mutex);
        …
            body of P;
        …
signal(mutex);
```

- Mutual exclusion within a monitor is ensured

پیاده سازیش:
بدنه هر پروسیجری رو داخل یک Mutual exclusion گذاشته باشیم ینی یک wait وسیگنالی داشته باشیم قبل و بعد کل بدنه یک پروسیجر
و با یک سمافوری که مقدار اولیه اش یک است می تونیم بدنه این پروسیجرها رو پیاده سازی بکنیم

Monitor with Condition Variables



یک چیزی که می تونه خیلی مفید باشه توی مانیتورها استفاده از CV است
به چه صورت؟
با کمک این CV ها می تونیم اجازه این رو بدیم که اگر ما توی یکی از فانکشن های این تابع هستیم تحت شرایط خاصی بریم توی حالت waiting ینی پروسسی که خودش داره اون فانکشن رو فراخوانی میکنه مثلا فانکشن f1 خودش رو ببره توی حالت Sleep و منتظر برقراری یک شرطی باشه در این صورت mutex مربوط به این فانکشن ازاد میشه وقتی این پروسسی که f1 رو فراخوانی کرده می ره توی حالت Sleep و یک پروسس دیگه میتونه از اپریشن ها استفاده بکنه یا از مانیتور استفاده بکنه و پروسس های دیگه الان باید اون شرط رو برقرار کنن و وقتی که برقرار شد مثلا یک پروسس دیگه ای جایی داشته باشیم مثلا توی یکی از فانکشن های این مانیتور ممکنه اون همون CV رو سیگنالش بکنه که باعث میشه پروسس دیگه ای که توی حالت بلاک بوده wake بشه و بتونه وارد مانیتور بشه
ما می تونیم CV های مختلفی روی توی shared data مانیتور تعریف بکنیم برحسب نیازمون
نکته: اگر f1 اینجا wait شده باشه روی x <-- CV و یک پروسس دیگه ای داره یک فانکشن دیگه ای رو توی مانیتور اجرا می کنه که می رسه به خط سیگنال کردن x الان خود این پروسس p2 داخل مانیتور است و وقتی که x رو میاد سیگنال میکنه باعث میشه که p1 هم وارد مانیتور بشه و اتفاقی که می افته این است که باعث میشه همزمان هر دوتای این ها داخل مانیتور قرار بگیرند در صورتی که ما مانیتور رو اینطوری تعریف کرده بودیم که در هر لحظه فقط یک پروسس می تونه ازش استفاده بکنه
این که اینجا چجوری مسئله رو حل بکنیم وابسته به این است که ما مانیتور رو چجوری فرض کرده باشیم: دوتا فرض وجود داره:
بعضی ها می گن توی این حالت اشکالی نداره ینی p2 که خودش باعث wake شدن p1 شد باید بتونه ادامه پیدا بکنه ولی بعضی ها میگن نه ینی p2 بلاک بشه و اونی که سیگنال شده بتونه اجرا بشه و دو نوع پیاده سازی می تونه اینجا وجود داشته باشه

Usage of Condition Variable Example

- Consider P_1 and P_2 that need to execute two statements S_1 and S_2 and the requirement that S_1 to happen before S_2
  - Create a monitor with two procedures F_1 and F_2 that are invoked by P_1 and P_2 respectively
  - One condition variable "x" initialized to 0
  - One Boolean variable "done"
  - **F1:**

```
S1;
done = true;
x.signal();
```

  - **F2:**

```
if done = false
   x.wait()
S2;
```

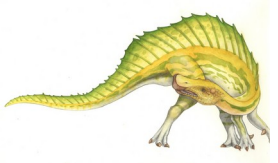
-
مثال:
توی دوتا از فانکشن های مانیتور استفاده شده ینی فانکشن F1 , F2
هدف این بوده که حتما done= true بشه و بعدا ادامه F2 اجرا بشه

Monitor Solution to Dining Philosophers

```
monitor DiningPhilosophers
{
   enum { THINKING; HUNGRY, EATING) state [5] ;
   condition self [5];

   void pickup (int i) {
           }

   void putdown (int i) {
   }
```

- Each philosopher "i" invokes the operations **pickup()** and **putdown()** in the following sequence:

```
DiningPhilosophers.pickup(i);

/** EAT **/

DiningPhilosophers.putdown(i);
```



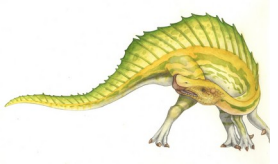
Monitor Solution to Dining Philosophers

```
monitor DiningPhilosophers
{
   enum { THINKING; HUNGRY, EATING) state [5] ;
   condition self [5];

   void pickup (int i) {
          state[i] = HUNGRY;
          test(i);
          if (state[i] != EATING) self[i].wait;
   }

   void putdown (int i) {
          state[i] = THINKING;
                  // test left and right neighbors
          test((i + 4) % 5);
          test((i + 1) % 5);
   }
```

اینجا برای هر فیلسوفی 3 تا استیت در نظر گرفتیم:
THINKING; HUNGRY, EATING
hungry: حالتی است که فیلسوف می خواد بخوره ولی چوبش ازاد نیست حالا یا یه دونه چوب یا دوتا چوبش و مجبوره که صبر بکنه
یه دونه CV هم تعریف میکنیم و به ازای هر فیلسوف یه دونه تعریف می کنیم و اسمشو گذاشتیم self و با این CV می خوایم مانع خوردن اون فیلسوف بشیم اگر چوب هاش ازاد نبود
فیلسوفی که قصد خوردن داره تابع pickup رو فراخوانی میکنه
توی تابع pickup باید چک بکنه که چوب های سمت راست و چپش ازاد هستن هر دو یا نه و اگر هر دو ازاد بودن هر دو برمیداره در غیر اینصورت هیچ کدوم رو برنمیداره پس ما اینجا باید چک بکنیم که فیلسوف های کناریش توی حالت eating هستن یا نه و اگر توی این حالت بودن با اینکه ما الان میخوایم غذا بخوریم ولی نمیتونیم بخوریم پس اینجا باید باید روی self بیایم wait بکنیم حالا کی این wait ازاد میشه و کی میشه این CV رو سیگنال کرد؟ اگر یکی از این دوتا همسایه های کناریش خوردنشون تموم بشه
پس توی تابع putdown که چوب ها رو می خوان زمین بذارن باید نگا کنیم اگر فیلسوف همسایه ای وجود داره که hungry است ینی میخواسته غذا بخوره ولی wait شده اون فیلسوف رو بیایم wake بکنیم یا CVاش رو سیگنال بکنیم
برای اینکه pickup , putdown رو بنویسیم به یک تابع دیگه هم نیاز داریم:
به اسم test و این test قراره هر بار بیاد چک بکنه که ایا فیلسوف iام الان می خواد غذا بخوره یا نه و کارهای لازم رو انجام بده و اگر مثلا میخواست غذا بخوره و کناری هاش ازاد بودن CV فیلسوف i ام رو سیگنال بکنه
تابع initialization_code هم صرفا برای اینکه مقدارهای اولیه استیت رو مشخص بکنیم که برای همه فیلسوف ها باید اینجا استیت رو روی thinking قرار بدیم

Solution to Dining Philosophers (Cont.)

```
    void test (int i) {
            if ((state[(i + 4) % 5] != EATING) &&
            (state[i] == HUNGRY) &&
            (state[(i + 1) % 5] != EATING) ) {
                 state[i] = EATING ;
            self[i].signal () ;
            }
    }

        initialization_code() {
           for (int i = 0; i < 5; i++)
               state[i] = THINKING;
         }
}
```



Solution to Dining Philosophers (Cont.)

- Each philosopher "i" invokes the operations **`pickup()`** and **`putdown()`** in the following sequence:

```
DiningPhilosophers.pickup(i);

/** EAT **/

DiningPhilosophers.putdown(i);
```

- No deadlock, but starvation is possible

-
اگر یک فیلسوفی خواست غذا بخوره می تونه pickup بکنه و eat رو انجام بده و بعد putdown بکنه

همینطور می تونیم یه کاری بکنیم که گرسنگی پش بیاد اینجا

End of Chapter 6