#### Concurrency: Condition Variables, PCP, 5DP CS 571: Operating Systems (Spring 2022) Lecture 7 Yue Cheng

Some material taken/derived from:

• Wisconsin CS-537 materials created by Remzi Arpaci-Dusseau.

neor

Licensed for use under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

# **Condition Variables**

## **Condition Variables (CV)**

- Definition:
  - An explicit queue that threads can put themselves when some condition is not as desired (by waiting on the condition)
- Other thread can wake one of those waiting threads to allow them to continue (by signaling on the condition)
- Pthread CV

>> pthread\_cond\_wait(pthread\_cond\_t \*c, pthread\_mutex\_t \*m); >> pthread\_cond\_signal(pthread\_cond\_t \*c);

# **CV-based Approach**

child.







#### child

void	thread_exit() {		
	<pre>Mutex_lock(&amp;m);</pre>	11	а
	<pre>Cond_signal(&amp;c);</pre>	11	b
	Mutex_unlock(&m);	11	С
}			







<pre>void thread_exit() {</pre>		<pre>void thread_join() {</pre>	
<pre>Mutex_lock(&amp;m);</pre>	// a	<pre>Mutex_lock(&amp;m);</pre>	// x
<pre>Cond_signal(&amp;c);</pre>	// b	Cond_wait(&c, &m);	// y
<pre>Mutex_unlock(&amp;m);</pre>	// c	<pre>Mutex_unlock(&amp;m);</pre>	// z
}		}	











```
void thread_exit() {
    done = 1;
    Cond_signal(&c);
}
void thread_join() {
    Mutex_lock(&m);
    // w
    Mutex_lock(&m);
    // x
    Cond_wait(&c, &m); // y
    Mutex_unlock(&m);
    // z
}
```











Condition Variable wait thread



Only one thread gets a signal

**Condition Variable** 



**Condition Variable** 

Condition Variable





#### Signal lost if nobody waiting at that time

## Guarantee

Upon signal, there has to be **at least one** thread waiting; If there are threads waiting, **at least one** thread will wake



```
int done = 0;
1
    pthread mutex t m = PTHREAD MUTEX INITIALIZER;
2
    pthread cond t c = PTHREAD COND INITIALIZER;
3
4
    void thr_exit() {
5
        Pthread mutex lock (&m);
6
        done = 1;
7
        Pthread_cond_signal(&c);
8
                                        Approach
        Pthread_mutex_unlock(&m);
9
10
11
    void *child(void *arg) {
12
        printf("child\n");
13
        thr exit();
14
        return NULL;
15
16
17
    void thr_join() {
18
        Pthread mutex lock (&m);
19
        while (done == 0)
20
             Pthread_cond_wait(&c, &m);
21
        Pthread mutex unlock (&m);
22
23
24
    int main(int argc, char *argv[]) {
25
        printf("parent: begin\n");
26
        pthread_t p;
27
        Pthread_create(&p, NULL, child, NULL);
28
        thr_join();
29
        printf("parent: end\n");
30
        return 0;
31
32
```

#### CV-based Parent-wait-for-child Approach

```
int done = 0;
1
    pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
2
    pthread cond t c = PTHREAD COND INITIALIZER;
3
4
    void thr_exit() {
5
        Pthread mutex lock (&m);
6
                                       CV-based Parent-wait-for-child
        done = 1;
7
        Pthread_cond_signal(&c);
8
                                      Approach
        Pthread_mutex_unlock(&m);
9
10
11
    void *child(void *arg) {
12
        printf("child\n");
13
                                        Rule of using CVs
        thr exit();
14
        return NULL; -> Always do 1. wait and 2. signal while holding the lock
15
16
17
    void thr_join() {
18
        Pthread mutex lock (&m);
19
                                                   Why: To prevent lost signal
        while (done == 0)
20
            Pthread_cond_wait(&c, &m);
21
        Pthread mutex unlock (&m);
22
23
24
    int main(int argc, char *argv[]) {
25
        printf("parent: begin\n");
26
        pthread_t p;
27
        Pthread_create(&p, NULL, child, NULL);
28
        thr_join();
29
        printf("parent: end\n");
30
        return 0;
31
                                                                             32
32
```

#### **Classical Problems of Synchronization**

- Producer-consumer problem
  - Semaphore version
  - CV-based version
- Readers-writers problem
- Dining-philosophers problem

#### **CV-based Producer-Consumer Implementation 1**

```
Single CV and if statement
                                                           int buffer;
                                                       1
                                                           int count = 0; // initially, empty
                                                       2
                                                       3
                                                           void put(int value) {
                                                       4
                                                               assert(count == 0);
                                                       5
cond t cond;
                                                             - count = 1;
                                                       6
mutex t mutex;
                                                             buffer = value;
                                                       7
                                                       8
void *producer(void *arg) {
                                                       9
    int i;
                                                           int get() {
                                                       10
    for (i = 0; i < loops; i++) {
                                                               assert(count == 1);
                                                       11
        Pthread mutex lock(&mutex);
                                               // p1
                                                               count = 0;
                                                       12
      if (count == 1)
                                               // p2
                                                               return buffer;
                                                       13
            Pthread cond wait (&cond, &mutex); // p3
                                                       14
        put(i);
                                               // p4
        Pthread cond signal (&cond);
                                               // p5
                                                                Put and Get routines
        Pthread mutex unlock(&mutex);
                                               // p6
                                                                    Single buffer
void *consumer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        Pthread_mutex_lock(&mutex);
                                               // c1
     → if (count == 0)
                                               // c2
            Pthread_cond_wait(&cond, &mutex); // c3
   \checkmark int tmp = get();
                                              // c4
        Pthread cond signal (&cond);
                                              // c5
        Pthread mutex unlock (&mutex);
                                               // c6
        printf("%d\n", tmp);
```

#### **CV-based Producer-Consumer Implementation 1**

#### Single CV and if statement

```
cond t cond;
mutex_t mutex;
void *producer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        Pthread mutex lock(&mutex);
                                               // p1
        if (count == 1)
                                               // p2
            Pthread cond wait (&cond, &mutex); // p3
        put(i);
                                               // p4
        Pthread cond signal (&cond);
                                               // p5
        Pthread mutex unlock(&mutex);
                                               // p6
```

```
}
```

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

#### Put and Get routines Single buffer

## What's the problem of this approach?

#### **CV-based Producer-Consumer Implementation 1**


oid *consumer(void *arg) {	
int i;	
for (i = 0; i < loops; i++) {	
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// cl
if $(count == 0)$	// c2
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
<pre>int tmp = get();</pre>	// c4
<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5
<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	// c6
<pre>printf("%d\n", tmp);</pre>	
}	



$T_{c1}$	State	T <sub>c2</sub> Stat	e   $T_p$	State	Count	Comment
c1	Running	Read	y	Ready	0	
c2	Running	Read	y	Ready	0	
c3	Sleep	Read	y	Ready	0	Nothing to get
	Sleep	Read	y p1	Running	0	
	Sleep	Read	y p2	Running		

}

V

<pre>void *consumer(void *arg) {</pre>	
int i;	
for $(i = 0; i < loops; i++) $ {	
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// cl
if (count $== 0$ )	// c2
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
<pre>int tmp = get();</pre>	// c4
<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5
<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	// c6
<pre>printf("%d\n", tmp);</pre>	
}	

void \*producer(void \*arg) { P running int i; for (i = 0; i < loops; i++) { Pthread\_mutex\_lock(&mutex); // p1 if (count == 1)// p2 Pthread\_cond\_wait(&cond, &mutex); // p3 put(i); 11 p4 Pthread\_cond\_signal(&cond); // p5 Pthread\_mutex\_unlock(&mutex); // p6 }

$T_{c1}$	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	0 0
	Sleep		Ready	p2	Running	$\circ$	
	Sleep		Ready	p4	Running		Buffer now full 🧲

<pre>void *consumer(void *arg) {</pre>	
int i;	
for $(i = 0; i < loops; i++) $ {	
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// cl
if (count $== 0$ )	// c2
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
<pre>int tmp = get();</pre>	// c4
<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5
<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	// c6
<pre>printf("%d\n", tmp);</pre>	
}	



$T_{c1}$	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

oid *consumer(void *arg) {	
int i;	
for (i = 0; i < loops; i++) {	
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// cl
if $(count == 0)$	// c2
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
<pre>int tmp = get();</pre>	// c4
Pthread_cond_signal(&cond);	// c5
<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	// c6
<pre>printf("%d\n", tmp);</pre>	

void \*producer(void \*arg) { P running int i; for (i = 0; i < loops; i++) {</pre> Pthread\_mutex\_lock(&mutex); // p1 if (count == 1) p2 Pthread\_cond\_wait(&cond, &mutex); p3 11 put(i); // p4 Pthread\_cond\_signal(&cond); // p5 Pthread\_mutex\_unlock(&mutex); // p6 }

$T_{c1}$	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	

}

V

<pre>void *consumer(void *arg) {</pre>	
int i;	
for (i = 0; i < loops; i++) {	
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// cl
if (count $== 0$ )	// c2
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
<pre>int tmp = get();</pre>	// c4
<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5
Pthread_mutex_unlock(&mutex);	// c6
<pre>printf("%d\n", tmp);</pre>	
}	

void \*producer(void \*arg) { P running int i; for (i = 0; i < loops; i++) { Pthread\_mutex\_lock(&mutex); // p1 if (count == 1) p2 Pthread\_cond\_wait(&cond, &mutex); p3 11 put(i); // p4 Pthread\_cond\_signal(&cond); // p5 Pthread\_mutex\_unlock(&mutex); // p6 }

$T_{c1}$	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	

void *consumer(void *arg) { C1 run	nable
int i;	
for $(i = 0; i < loops; i++) $ {	
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// c1
if (count $== 0$ )	// c2
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
<pre>int tmp = get();</pre>	// c4
<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5
Pthread_mutex_unlock(&mutex);	// c6
<pre>printf("%d\n", tmp);</pre>	
}	

$T_{c1}$	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

oid *co	onsumer(void *arg) { C2 runr	ning	I
int	i;	C	
for	(i = 0; i < loops; i++) {		
Г	Pthread_mutex_lock(&mutex);	//	c1
_	if $(count == 0)$	//	c2
	<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	11	c3
	<pre>int tmp = get();</pre>	11	c4
	Pthread_cond_signal(&cond);	11	c5
	Pthread_mutex_unlock(&mutex);	11	c6
	<pre>printf("%d\n", tmp);</pre>		
}			

$T_{c1}$	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	0 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	<u>^</u>	Sleep	1	$T_{c2}$ sneaks in

}

V

void *consumer(void *arg) { C2 run	ning
int i;	0
for $(i = 0; i < loops; i++) $ {	
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// c1
if (count $== 0$ )	// c2
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
<pre>int tmp = get();</pre>	// c4
Pthread_cond_signal(&cond);	// c5
<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	// c6
<pre>printf("%d\n", tmp);</pre>	
}	

$T_{c1}$	State	T <sub>c2</sub>	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

void *co	onsumer(void *arg) { C2 runr	ning
int	i;	0
for	(i = 0; i < loops; i++) {	
	<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// c1
	if $(count == 0)$	// c2
	<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
	<pre>int tmp = get();</pre>	// c4
	<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5
	Pthread_mutex_unlock(&mutex);	// c6
	<pre>printf("%d\n", tmp);</pre>	
}	Sector construction and a subsection of the subsection of Sector and Sector Secto	

$T_{c1}$	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	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

void *consumer(void *arg) { C2 runni									
int i;	0								
for $(i = 0; i < loops; i++) $ {									
<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// c1								
if (count == 0)	// c2								
<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3								
<pre>int tmp = get();</pre>	// c4								
<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5								
<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	// c6								
<pre>printf("%d\n", tmp);</pre>									
}									

$T_{c1}$	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	<b>c</b> 1	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	<b>c6</b>	Running		Ready	0	1

void *c	onsumer(void *arg) { C1 runr	ning
int	i;	0
for	(i = 0; i < loops; i++) {	
	<pre>Pthread_mutex_lock(&amp;mutex);</pre>	// c1
	if (count $== 0$ )	// c2
	<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	// c3
	<pre>int tmp = get();</pre>	// c4
	<pre>Pthread_cond_signal(&amp;cond);</pre>	// c5
	Pthread_mutex_unlock(&mutex);	// c6
	<pre>printf("%d\n", tmp);</pre>	
}		

$\langle$	$T_{c1}$	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
	$\checkmark$	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	pl	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

}

Y. Cheng

GMU CS571 Spring 2022

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

```
cond t cond;
1
    mutex_t mutex;
2
3
                                                  Single CV and while
    void *producer(void *arg) {
4
        int i;
5
        for (i = 0; i < loops; i++) {
6
             Pthread mutex lock (&mutex);
                                                      // p1
7
             while (count == 1)
                                                      // p2
8
                 Pthread_cond_wait(&cond, &mutex);
                                                      // p3
9
             put(i);
                                                      // p4
10
             Pthread_cond_signal(&cond);
                                                      // p5
11
            Pthread mutex unlock (&mutex);
                                                      // p6
12
         }
13
14
    }
                                                   What's the problem of this
15
    void *consumer(void *arg) {
16
                                                   approach?
        int i;
17
18
        for (i = 0; i < loops; i++) \{
             Pthread_mutex_lock(&mutex);
                                                      // c1
19
             while (count == 0)
                                                      // c2
20
                 Pthread_cond_wait(&cond, &mutex);
                                                      // c3
21
             int tmp = qet();
                                                      // c4
22
             Pthread_cond_signal(&cond);
                                                      // c5
23
             Pthread mutex unlock (&mutex);
                                                      // c6
24
             printf("%d\n", tmp);
25
         }
26
                                                                          49
27
    }
```

void *a int	consumer( ; i;	void *arg) {		C1 runnir	ıg	vo	id *produ int i;	ucer(void *a	rg) {		
for	(i = 0;	i < loops;	i++) {				for (i	= 0; i < loc	ops; i++) {		
	Pthread	l_mutex_lock(	<pre>&amp;mutex);</pre>		// c1		Pth	hread_mutex_1	lock(&mutex);	11	p1
	while (count == 0) /,						while (count == 1) //				p2
	Pth	read_cond_wa	it (&cond,	&mutex);	// c3			Pthread_com	nd_wait(&cond, &mutex);	11	pЗ
	int tmp	) = get();			// c4		put	t(i);		11	p4
	Pthread_cond_signal(&cond);						Pthread_cond_signal(&cond); // p				
	Pthread_mutex_unlock(&mutex);				// c6		Pthread_mutex_unlock(&mutex); // p6				
	<pre>printf("%d\n", tmp);</pre>						}				
}						}					
}	Ten	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		

voi	.d *a int	consumer t i;	(void *arg)		C2 runnir	ıg	void *produc int i;	er(void *a	rg) {	
	foi	r (i = 0)	; i < loops;	i++) {			for $(i =$	0; i < lo	ops; i++) {	
		Pthrea	d_mutex_lock	(&mutex)	;	// c1	Pthr	ead_mutex_	lock(&mutex);	// p1
		while	(count == 0)			// c2	whil	e (count =	= 1)	// p2
		Pt	hread_cond_wa	ait(&con	d, &mutex);	// c3		Pthread_co	nd_wait(&cond, &mutex);	// p3
	I	int tm	p = get();			// c4	put (	i);		// p4
		Pthrea	d_cond_signal	(&cond)	;	// c5	Pthr	ead_cond_s	ignal(&cond);	// p5
		Pthrea	d_mutex_unloc	ck(&mute	x);	// c6	Pthr	ead_mutex_	unlock(&mutex);	// p6
		printf	("%d\n", tmp)	;			}			
	}						}			
}		$T_{c1}$	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	

```
voi
void *consumer(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {</pre>
       Pthread_mutex_lock(&mutex);
                                            // c1
       while (count == 0)
                                            // c2
      Pthread_cond_wait(&cond, &mutex); // c3
                            // c4
       int tmp = get();
       Pthread_cond_signal(&cond);
                                            // c5
       Pthread_mutex_unlock(&mutex);
                                            // c6
       printf("%d\n", tmp);
                                                    }
    }
}
```

d *p	roducer(void *arg) { Prunni	inσ	
int	i;	18	
for	(i = 0; i < loops; i++) {		
	<pre>Pthread_mutex_lock(&amp;mutex);</pre>	11	p1
	while (count == 1)	11	p2
	<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	11	p3
C	put(i);	11	p4
	Pthread_cond_signal(&cond);	11	p5
	<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	11	p6
}			

$T_{c1}$	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	0 0
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	0 0
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full

```
voi
void *consumer(void *arg) {
   int i;
   for (i = 0; i < loops; i++) {</pre>
       Pthread_mutex_lock(&mutex);
                                            // c1
       while (count == 0)
                                            // c2
      Pthread_cond_wait(&cond, &mutex); // c3
                           // c4
       int tmp = get();
       Pthread_cond_signal(&cond);
                                            // c5
       Pthread_mutex_unlock(&mutex);
                                            // c6
       printf("%d\n", tmp);
                                                       3
                                                   }
}
```

d *produc int i:	cer(void *arg) { Prunni	ng	
for (i -	-0, $i < loope, i++)$		
TOT (T -	- 0; 1 < 100ps; 1++) {		
Pthi	read_mutex_lock(&mutex);	11	p1
whil	_e (count == 1)	11	p2
	<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	11	рЗ
put	(i);	11	p4
Pthi	<pre>read_cond_signal(&amp;cond);</pre>	//	p5
Pthi	read_mutex_unlock(&mutex);	11	p6
1			

$T_{c1}$	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	0 0
	Sleep	c2	Running		Ready	0	
	Sleep	c3	Sleep		Ready	0	Nothing to get
	Sleep		Sleep	p1	Running	0	0 0
	Sleep		Sleep	p2	Running	0	
	Sleep		Sleep	p4	Running	1	Buffer now full
	Ready		Sleep	p5	Running	1	$T_{c1}$ awoken

void *p: int	roducer(void *arg) { P sleep	oing	
for	(i = 0; i < loops; i++)		
	Pthread_mutex_lock (&mutex);	11	p1
	while (count == 1)	11	p2
	<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	11	pЗ
	put(i);	11	p4
	Pthread_cond_signal(&cond);	11	p5
	<pre>Pthread_mutex_unlock(&amp;mutex);</pre>	11	p6
}			

$T_{c1}$	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)

}

}

void *c	<pre>int i;</pre>								
int	for (i = 0; i < loops; i++) {								
for	Pthread_mutex_lock(&mutex);								
}	T <sub>c1</sub> c1 c2 c3	State Running Running Sleep Sleep Sleep Sleep Sleep Sleep Sleep Sleep Ready	C1 c2 c3	State Ready Ready Ready Running Running Sleep Sleep Sleep Sleep Sleep	T <sub>p</sub> p1 p2 p4 p5	State Ready Ready Ready Ready Ready Ready Running Running Running Running			

d *p	roducer(void *arg) {		
int	i;		
for	(i = 0; i < loops; i++) {		
	<pre>Pthread_mutex_lock(&amp;mutex);</pre>	11	p1
	while (count == 1)	11	p2
	<pre>Pthread_cond_wait(&amp;cond, &amp;mutex);</pre>	11	pЗ
	put(i);	11	p4
	Pthread_cond_signal(&cond);	11	p5
	Pthread_mutex_unlock(&mutex);	11	p6
}			

$T_{c1}$	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	0 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	<b>p</b> 6	Running	1	
	Ready		Sleep	p1	Running	1	
л И	Ready		Sleep	p2	Running	1	
	Ready		Sleep	p3	Sleep	1	Must sleep (full)
c2	Running		Sleep		Sleep	1	Recheck condition

voi	d *c int	consumer	(void *arg) {		C1 runnin	Ig	<pre>void *produce int i;</pre>	er(void *a:	rg) {	
	for	(i = 0)	; i < loops;	i++) {			for $(i = 0; i < loops; i++)$ {			
		Pthread	d_mutex_lock	(&mutex)	;	// c1	Pthre	ead_mutex_	lock(&mutex);	
		while	(count == 0)			// c2	while	e (count ==	= 1)	
		Ptl	nread_cond_wa	it (&con	d, &mutex);	// c3	1	Pthread_com	nd_wait(&cond, &mutex);	
		int tm	p = get();			// c4	put (	i);		
		Pthread	d_cond_signal	(&cond)	;	// c5	Pthre	ead_cond_s:	Ignal (&cond);	
		Pthread	d_mutex_unloc	ck(&mute	x);	// c6	Pthre	ead_mutex_	INLOCK (& MULEX);	
	,	printf	("%d\n", tmp)	;			} l			
1	}						J	toward to the same		
5		$T_{c1}$	State	$T_{c2}$	State	$\mid T_p$	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	0 0	
			Sleep	c2	Running		Ready	0		
			Sleep	c3	Sleep		Ready	0	Nothing to get	
			Sleep		Sleep	p1	Running	0	0 0	
			Sleep		Sleep	p2	Running	0		
			Sleep		Sleep	p4	Running	1	Buffer now full	
			Ready		Sleep	p5	Running	1	T <sub>c1</sub> 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	

// p1

// p2

// p3 // p4 // p5

// p6

void	d *co int	onsume: i;	r(void *arg) {		C1 runni	ng	vo	id i
	for	(i = (	); i < loops;	i++) {				f
		Pthrea	ad_mutex_lock	(&mutex);		// c1		
		while	(count == 0)			// c2		
		Pt	thread_cond_wa	ait (&cond,	&mutex);	// c3		
		int tr	mp = get();			// c4		
		Pthrea	ad_cond_signal	L(&cond);		// c5		
		Pthrea	ad_mutex_unloc	ck(&mutex)	;	// c6		
		print	f("%d\n", tmp)	;				}
	}						}	
}		$T_{c1}$	State	$T_{c2}$	State	$ T_p $		S
	-	c1	Running		Ready			Re
		c2	Running		Ready			Re
		c3	Sleep		Ready			Re
			Sloop	c1	Running			R

\*producer(void \*arg) { .nt i; for (i = 0; i < loops; i++) {</pre> Pthread\_mutex\_lock(&mutex); // p1 while (count == 1)
 Pthread\_cond\_wait(&cond, &mutex); // p2 // p3 put(i); // p4 Pthread\_cond\_signal(&cond); // p5 Pthread\_mutex\_unlock(&mutex); // p6

$T_{c1}$	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 <sub>c2</sub>

<pre>void *consumer(void *arg) { C1 sleeping int i:</pre>					ng	<pre>void *producer(void *arg) {     int i;</pre>			
fo	r (i = 0)	; i < loops;	i++) {			for (i = 0; i < loops; i++) {			
	Pthrea	d_mutex_lock	(&mutex)	;	// cl	<pre>Pthread_mutex_lock(&amp;mutex);</pre>			
	while	(count == 0)			// c2	whil	e (count ==	= 1)	
	Pt	hread_cond_wa	ait(&con	d, &mutex);	// c3	put (	Pthread_com	nd_walt(&cond, &mutex);	
	int tmp = get();  // c4				Pthread cond signal (& cond).				
	Pthrea	d mutex unloc	rk (&mute	/ x):	// c6	Pthread mutex unlock (& mutex);			
	printf	("%d\n", tmp)	;	, /	// 00	}			
}	-					}			
}	$T_{c1}$	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	0 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	I.	Sleep	1	Recheck condition	
	c4	Running		Sleep		Sleep	0	T <sub>c1</sub> grabs data	
	c5	Running		Ready		Sleep	0	Oops! Woke Te2	
	c6	Running		Ready		Sleep	0		
	c1	Running		Ready		Sleep	Ő		
	c2	Running		Ready		Sleep	Ő		
	c3	Sleep		Ready		Sleep	0	Nothing to get	

Nothing to get

// p1

// p2

// p3 // p4 // p5

// p6

voi	d *c int for	onsumer i; (i = 0 Pthread while Pt	<pre>(void *arg) ; i &lt; loops; d_mutex_lock (count == 0) hread_cond_wa</pre>	<pre>{     i++) {     (&amp;mutex); ait(&amp;cond); }</pre>	C2 runnin	ng // c1 // c2 // c3	<pre>void *product int i; for (i = Pthrewhile put(</pre>	er(void *a) 0; i < loc ead_mutex_1 e (count == Pthread_cor	cg) { pps; i++) { lock(&mutex); = 1) nd_wait(&cond, &mutex);	// p1 // p2 // p3
		Pthrea	p = get(); d cond signal	l(&cond):		$// c_{4}$	Pthr	ead_cond_s:	ignal(&cond);	// p5
		Pthrea	d_mutex_unloc	ck (&mutex	:);	// c6	Pthread_mutex_unlock(&mutex); // p6			// p6
		printf	("%d\n", tmp)	);			}			
}	}	-							<b>C</b> (	
,		1 <sub>c1</sub>	State	$1_{c2}$	State	$T_p$	State	Count	Comment	
		cl	Running		Ready		Ready	0		
		c2	Running		Ready		Ready	0	NT (1.1	
		c3	Sleep	1	Ready		Ready	0	Nothing to get	
			Sleep	CI	Running		Ready	0		
			Sleep	c2	Kunning		Ready	0	NT (1.1	
			Sleep	C3	Sleep	. 1	Ready	0	Nothing to get	
			Sleep		Sleep	pl	Running	0		
			Sleep		Sleep	p2	Running	0	D (( 11	
			Sleep		Sleep	p4	Running	1	Buffer now full	
			Ready		Sleep	p5	Running	1	$I_{c1}$ awoken	
			Ready		Sleep	p6	Running	1		
			Ready		Sleep	pl	Running	1		
			Ready		Sleep	p2	Running	1		
		0	Ready		Sleep	p3	Sleep	1	Must sleep (full)	
		c2	Running		Sleep		Sleep		Recheck condition	
		c4	Running		Sleep		Sleep	0	$I_{c1}$ grabs data	
		C5	Running		Ready		Sleep	0	Oops! Woke $I_{c2}$	
		C6	Running		Ready		Sleep	0		
		cl	Running		Ready		Sleep	0		
		c2	Kunning		Ready		Sleep	0	NT (1.1	
		C3	Sleep		Ready		Sleep		Nothing to get	
			Sleep	c2	Kunning		Sleep			
			Sleep	C3	Sleep		Sleep		Everyone asleep	

```
void *consumer(void *arg) {
                                      C2 sleeping
    int i;
    for (i = 0; i < loops; i++) {</pre>
        Pthread_mutex_lock(&mutex);
                                                // c1
                                                // c2
        while (count == 0)
            Pthread_cond_wait(&cond, &mutex); // c3
        int tmp = get();
                                                // c4
        Pthread_cond_signal(&cond);
                                               // c5
                                                // c6
        Pthread_mutex_unlock(&mutex);
        printf("%d\n", tmp);
```

}

$T_{c1}$	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	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	
<b>c</b> 1	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

```
cond_t Cempty, fill;
1
    mutex_t mutex;
2
3
                                                  Two CVs and while
    void *producer(void *arg) {
4
        int i;
5
        for (i = 0; i < loops; i++) {
6
             Pthread_mutex_lock(&mutex);
7
             while (count == 1)
8
                 Pthread_cond_wait(&empty, &mutex);
9
             put(i);
10
             Pthread_cond_signal(&fill);
11
             Pthread mutex unlock (&mutex);
12
13
         }
14
    }
15
    void *consumer(void *arg) {
16
        int i;
17
        for (i = 0; i < loops; i++) \{
18
             Pthread_mutex_lock(&mutex);
19
             while (count == 0)
20
                 Pthread_cond_wait(&fill, &mutex);
21
             int tmp = qet();
22
             Pthread_cond_signal(&empty);
23
             Pthread_mutex_unlock(&mutex);
24
             printf("%d\n", tmp);
25
         }
26
    1
27
```

```
cond_t Cempty, fill;
1
    mutex t mutex;
2
3
                                                  Two CVs and while
    void *producer(void *arg) {
4
        int i;
5
        for (i = 0; i < loops; i++) {
6
             Pthread_mutex_lock(&mutex);
7
             while (count == 1)
8
                 Pthread_cond_wait (&empty)
                                              &mutex);
9
             put(i);
10
             Pthread_cond_signal(&fill);
11
             Pthread mutex unlock (&mutex);
12
13
                                              Using two CVs to distinguish two
14
    }
                                           types of threads; in order to properly
15
                                            signal which thread should wake up
    void *consumer(void *arg) {
16
         int i;
17
                                                      Producer waits on empty
        for (i = 0; i < loops; i++)
18
                                                         Consumer waits on fill
             Pthread_mutex_lock(&mutex);
19
             while (count == 0)
20
                 Pthread_cond_wait(&fill, &mutex);
21
             int tmp = qet();
22
             Pthread_cond_signal & empty);
23
             Pthread_mutex_unlock(&mutex);
24
             printf("%d\n", tmp);
25
26
                                                                         62
27
```

# **Dining-Philosophers Problem**

# **Dining-Philosophers Problem**



Shared data



while(food available) {pick up 2 adj. forks; eat; put down forks; think awhile;

Philosopher 1

- 5 philosophers share a common circular table. There are 5 forks (or chopsticks) and food (in the middle). When a philosopher gets hungry, he tries to pick up the closest forks
- A philosopher may pick up only one fork at a time, and cannot pick up a fork already in use. When done, he puts down both of his forks, one after the other

GMU CS571 Spring 2022

# **Dining-Philosophers Problem**

• The basic loop of a philosopher



# The Helper Functions

```
int left(int p) { return p; }
int right(int p) { return (p + 1) % 5; }
```

```
sem t forks[5]
```

Each fork initialized to 1

```
1 void getforks() {
2 sem_wait(forks[left(p)]);
3 sem_wait(forks[right(p)]);
5
                                                                      Is this solution correct?
6 void putforks() {
7 sem_post(forks[left(p)]);
8 sem_post(forks[right(p)]);
                                              GMU CS571 Spring 2022
   Y. Cheng
```

66



sem\_wait(fork[0])
sem\_wait(fork[1])
sem\_signal(fork[0])
sem\_signal(fork[1])

sem\_wait(fork[1])
sem\_wait(fork[0])
sem\_signal(fork[1])
sem\_signal(fork[0])

W/ only two philosophers and two forks

Thread 0	Interleaving	Thread 1
<pre>sem_wait(fork[0]) sem_wait(fork[1]) sem_signal(fork[0]) sem_signal(fork[1])</pre>	<pre>sem_wait(fork[0])</pre>	<pre>sem_wait(fork[1]) sem_wait(fork[0]) sem_signal(fork[1]) sem_signal(fork[0])</pre>

W/ only two philosophers and two forks

Thread 0	Interleaving	Thread 1
<pre>sem_wait(fork[0]) sem_wait(fork[1]) sem_signal(fork[0]) sem_signal(fork[1])</pre>	<pre>sem_wait(fork[0]) sem_wait(fork[1])</pre>	<pre>sem_wait(fork[1]) sem_wait(fork[0]) sem_signal(fork[1]) sem_signal(fork[0])</pre>

W/ only two philosophers and two forks

Thread 0

Interleaving

Thread 1

sem wait(fork[1])

sem wait(fork[0])

sem signal(fork[1])

sem signal(fork[0])

```
sem_wait(fork[0])
sem_wait(fork[1])
sem_signal(fork[0])
sem_signal(fork[1])
```

sem wait(fork[0])

```
sem wait(fork[1])
```

sem wait(fork[0])

W/ only two philosophers and two forks

Thread 0

Interleaving

Thread 1

```
sem_wait(fork[0])
sem_wait(fork[1])
sem_signal(fork[0])
sem_signal(fork[1])
```

sem\_wait(fork[0])

```
sem_wait(fork[1])
sem_wait(fork[0])
sem_signal(fork[1])
sem_signal(fork[0])
```

```
sem_wait(fork[0])
     wait...
```

sem wait(fork[1])

```
sem_wait(fork[1])
```

W/ only two philosophers and two forks

Thread 0

Interleaving

Thread 1

sem wait(fork[1])

sem wait(fork[0])

sem signal(fork[1])

sem signal(fork[0])

```
sem_wait(fork[0])
sem_wait(fork[1])
sem_signal(fork[0])
sem_signal(fork[1])
```

sem\_wait(fork[0])

```
sem wait(fork[1])
```

sem\_wait(fork[0])
 wait...

```
sem_wait(fork[1])
     wait...
```
## **Review: Conditions for Deadlocks**

- Mutually exclusive access of shared resources
  - Binary semaphore fork[0] and fork[1]
- Circular waiting
  - Thread 0 waits for Thread 1 to signal(fork[1]) and
  - Thread 1 waits for Thread 0 to signal(fork[0])
- Hold and wait
  - Holding either fork[0] or fork[1] while waiting on the other
- No preemption
  - Neither fork[0] and fork[1] can be removed from their respective holding threads

## Why 5DP is Interesting?

- How to eat with your fellows without causing deadlocks
  - Circular arguments (the circular wait condition)
  - Not giving up on firmly held things (no preemption)
  - Infinite patience with half-baked schemes (hold some & wait for more)

## Why 5DP is Interesting?

- How to eat with your fellows without causing deadlocks
   How to mess with your fellows!
  - Circular arguments (the circular wait condition)
  - Not giving up on firmly held things (no preemption)
  - Infinite patience with half-baked schemes (hold some & wait for more)

## Dijkstra's Solution: Break the Circular Wait Condition

- Change how forks are acquired by at least one of the philosophers
- Assume P0 P4, 4 is the highest number

```
void getforks() {
1
      if (p == 4) {
2
        sem_wait(forks[right(p)]);
3
        sem_wait(forks[left(p)]);
4
      } else {
5
        sem_wait(forks[left(p)]);
6
        sem_wait(forks[right(p)]);
7
8
9
```