

30

condition

[join()]

30.1

```
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 }
```

30.1

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

30.2

CPU

```
1 volatile int done = 0;
2
3 void *child(void *arg) {
4     printf("child\n");
5     done = 1;
6     return NULL;
7 }
8
9 int main(int argc, char *argv[]) {
10    printf("parent: begin\n");
```

```

11     pthread_t c;
12     Pthread_create(&c, NULL, child, NULL); // create child
13     while (done == 0)
14         ; // spin
15     printf("parent: end\n");
16     return 0;
17 }

```

30.2

%" ~#

condition variable
condition

waiting

Dijkstra

[D01]

Hoare

[H74]

pthread_cond_t c;

c

wait()

signal()

wait()

signal()

POSIX

30.3

```

pthread_cond_wait(pthread_cond_t *c, pthread_mutex_t *m);
pthread_cond_signal(pthread_cond_t *c);

```

```

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);
10 }
11
12 void *child(void *arg) {
13     printf("child\n");
14     thr_exit();
15     return NULL;
16 }

```

```

17
18 void thr_join() {
19     Pthread_mutex_lock(&m);
20     while (done == 0)
21         Pthread_cond_wait(&c, &m);
22     Pthread_mutex_unlock(&m);
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 }

```

30.3

```

        wait()    signal()                wait()
        wait()    wait()

30.3    join

        thr_join()
        wait()

child    thr_exit()                                done

        wait()

parent:end

        done    1    signal
                thr_join()    done

1

        while    if

        thr_exit()    thr_join()
                    done

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  }
```

thr_exit()

done

wait

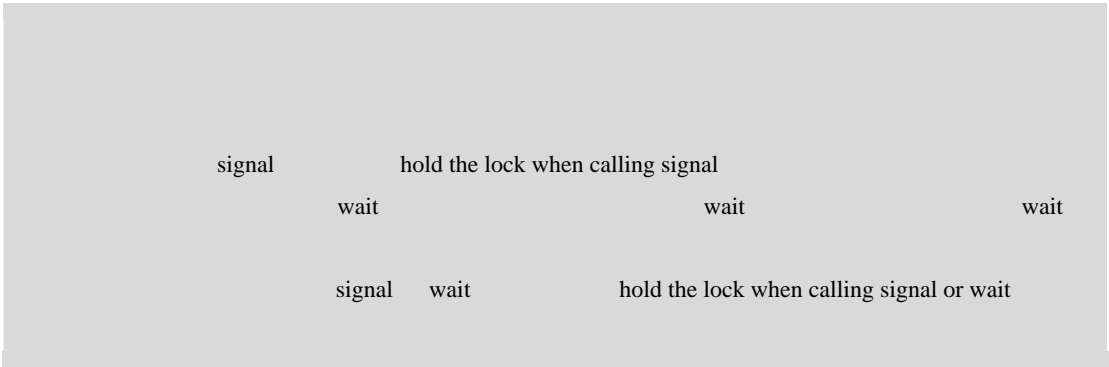
```
1  void thr_exit() {
2      done = 1;
3      Pthread_cond_signal(&c);
4  }
5
6  void thr_join() {
7      if (done == 0)
8          Pthread_cond_wait(&c);
9  }
```

thr_join()

done 0

done 1

wait



join

/

producer/consumer

bounded-buffer

%Z\$!

/

producer/consumer

bounded buffer

Dijkstra [D72]

Dijkstra [D01]

HTTP

```

graph LR
    A[wc] -- "file.txt | wc -l" --> B[wc]
    B -- "UNIX shell" --> C[pipe]
    C -- "grep" --> D[wc]
    D -- "grep foo" --> E[foo]
    E -- "file.txt" --> F[wc]
    F -- "grep" --> G[wc]

```

30.4

```

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     }

```

30.4	put	get	1
------	-----	-----	---

```

1      put()
      count
      get()
      count
      0

```

```
count    0
```

1

producer

consumer	30.5
----------	------

loops

```

1 void *producer(void *arg) {
2     int i;
3     int loops = (int) arg;
4     for (i = 0; i < loops; i++) {
5         put(i);
6     }
7 }
8
9 void *consumer(void *arg) {
10    int i;
11    while (1) {
12        int tmp = get();
13        printf("%d\n", tmp);
14    }
15 }

```

30.5 / 1

put() get()

put()

get()

30.6

cond mutex

```

1 cond_t cond;
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         if (count == 1)                       // p2
9             Pthread_cond_wait(&cond, &mutex); // p3
10        put(i);                               // p4
11        Pthread_cond_signal(&cond);           // 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         if (count == 0)                       // c2
21             Pthread_cond_wait(&cond, &mutex); // c3
22         int tmp = get();                      // c4
23         Pthread_cond_signal(&cond);           // c5
24         Pthread_mutex_unlock(&mutex);         // c6

```

```
25     printf("%d\n", tmp);
26     }
27 }
```

30.6 / if

p1 p3 c1 c3

30.6

if T_{c1} T_{c2}

T_p T_{c1} c1
c2 c3
T_p p1 p2
p4 p5
T_{c1} T_{c1}

p6,p1-p3

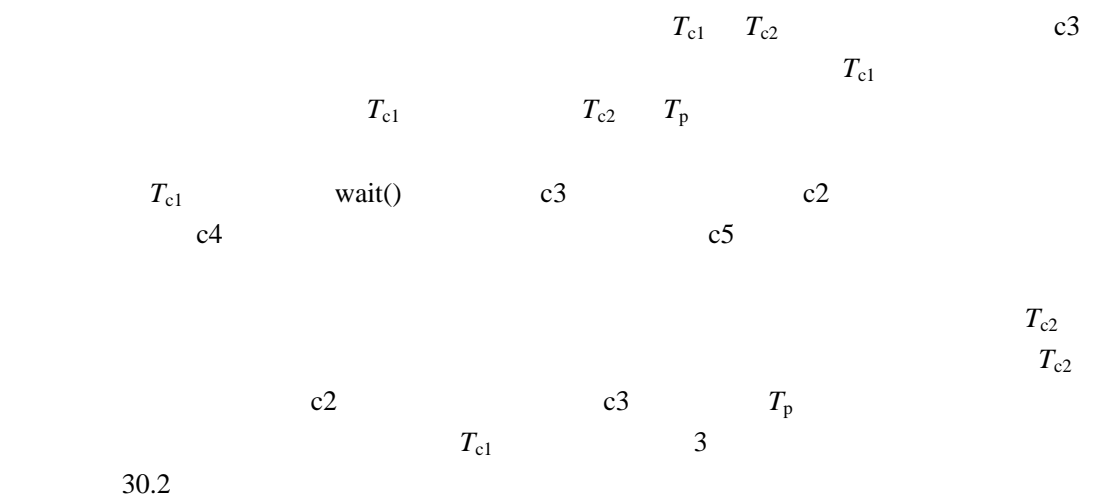
c3 T_{c2} c1,c2,c4,c5,c6
T_{c1} wait

get() (p4)

30.1

% z#				#			
T _{c1}		T _{c2}		T _p		count	
c1						0	
c2						0	
c3						0	
				p1		0	
				p2		0	
				p4		1	
				p5		1	T _{c1}
				p6		1	
				p1		1	
				p2		1	
				p3		1	
		c1				1	T _{c2}
		c2				1	
		c4				0	

Mesa while always use
while loop



T_{c1}	T_{c2}	T_p	count	
c1			0	
c2			0	
c3			0	
	c1		0	
	c2		0	
	c3		0	
		p1	0	
		p2	0	
		p4	1	
		p5	1	T_{c1}
		p6	1	
		p1	1	
		p2	1	
		p3	1	
c2			1	
c4			0	T_{c1}
c5			0	T_{c2}
c6			0	

T_{c1}		T_{c2}		T_p		count	
c1						0	
c2						0	
c3						0	
		c2				0	
		c3				0	

!

30.8

```
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);
8          while (count == 1)
9              Pthread_cond_wait(&empty, &mutex);
10         put(i);
11         Pthread_cond_signal(&fill);
12         Pthread_mutex_unlock(&mutex);
13     }
14 }
15
16 void *consumer(void *arg) {
17     int i;
18     for (i = 0; i < loops; i++) {
19         Pthread_mutex_lock(&mutex);
20         while (count == 0)
21             Pthread_cond_wait(&fill, &mutex);
22         int tmp = get();
23         Pthread_cond_signal(&empty);
24         Pthread_mutex_unlock(&mutex);
25         printf("%d\n", tmp);
26     }
27 }
```

30.8 / while

empty fill

fill empty

!

/

put() get() 30.9

30.10

p2

c2 /

```

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

```

30.9

put() get()

```

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 }
```

30.10

while

while

while

if

spurious wakeup

[L11]

% 2%

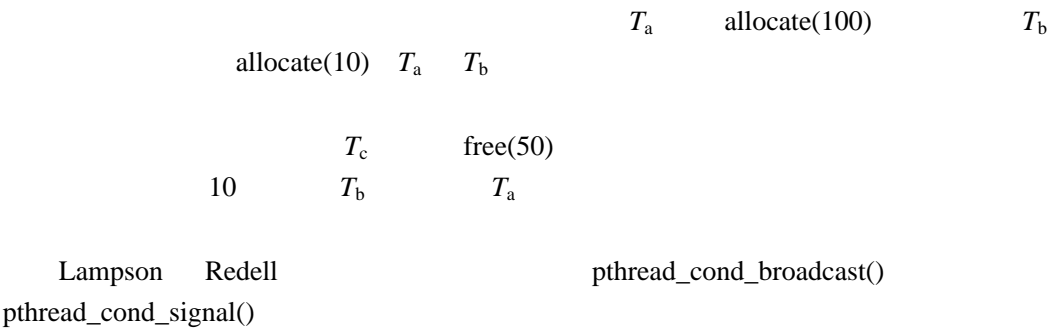
		Lampson	Redell
[LR80]	Mesa	Mesa semantic	Mesa

30.11

```
1  // how many bytes of the heap are free?
2  int bytesLeft = MAX_HEAP_SIZE;
3
4  // need lock and condition too
5  cond_t c;
6  mutex_t m;
7
8  void *
9  allocate(int size) {
10     Pthread_mutex_lock(&m);
11     while (bytesLeft < size)
12         Pthread_cond_wait(&c, &m);
13     void *ptr = ...; // get mem from heap
14     bytesLeft -= size;
15     Pthread_mutex_unlock(&m);
16     return ptr;
17 }
18
```

```
19 void free(void *ptr, int size) {
20     pthread_mutex_lock(&m);
21     bytesLeft += size;
22     pthread_cond_signal(&c); // whom to signal??
23     pthread_mutex_unlock(&m);
24 }
```

30.11



Lampson Redell covering condition

/

%&

/

/

[D01] My recollections of operating system design

E.W. Dijkstra April, 2001

[H74] Monitors: An Operating System Structuring Concept

C.A.R. Hoare

Communications of the ACM, 17:10, pages 549-567, October 1974

Hoare

[L11] Pthread cond signal Man Page

Linux

/

[LR80] Experience with Processes and Monitors in Mesa

B.W. Lampson, D.R. Redell

Communications of the ACM. 23:2, pages 105-117, February 1980

Mesa

Tony Hoare [H74]

Hoare