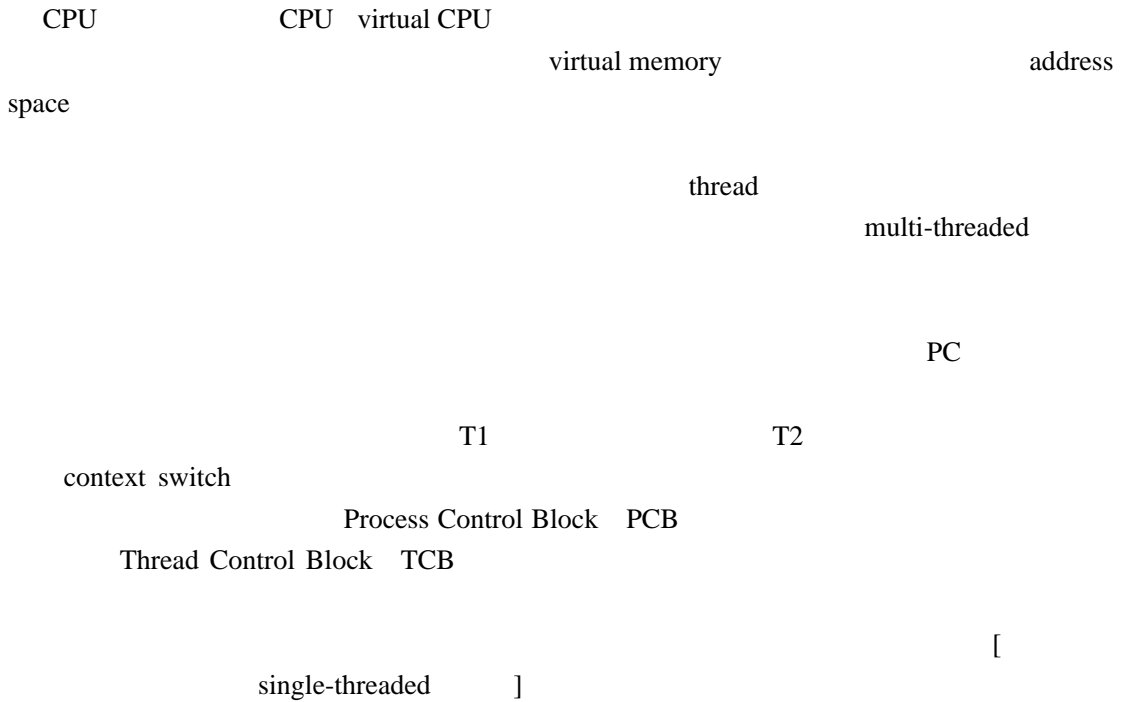
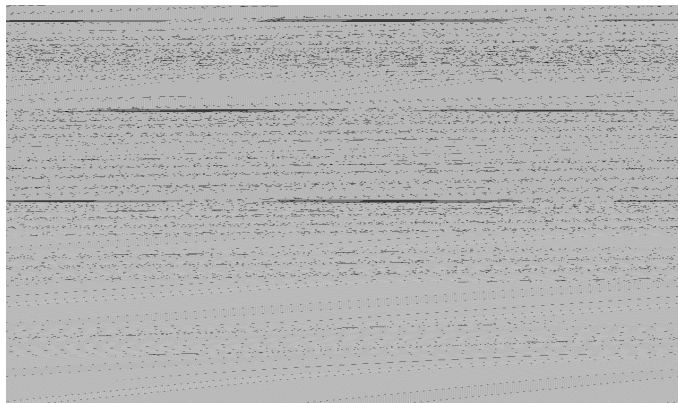


26



26.1



26.1

26.1

26.1

thread-local

\$(~~z~~#

A B

26.2

mythread()

A B

T1 T2

pthread_join()

```

1  #include <stdio.h>
2  #include <assert.h>
3  #include <pthread.h>
4
5  void *mythread(void *arg) {
6      printf("%s\n", (char *) arg);
7      return NULL;
8  }
9
10 int
11 main(int argc, char *argv[]) {
12     pthread_t p1, p2;
13     int rc;
14     printf("main: begin\n");
15     rc = pthread_create(&p1, NULL, mythread, "A"); assert(rc == 0);
16     rc = pthread_create(&p2, NULL, mythread, "B"); assert(rc == 0);
17     // join waits for the threads to finish
18     rc = pthread_join(p1, NULL); assert(rc == 0);
19     rc = pthread_join(p2, NULL); assert(rc == 0);
20     printf("main: end\n");
21     return 0;
22 }

```

26.2

t0.c

26.1

1

2

\$(#		
#		
	1	2
main:begin		
1		
2		
1		
	A	
2		
		B
main:end		

26.2

\$(\$		
\$		
	1	2
main:begin		
1		
	A	
2		
		B
1		
1		
2		
2		
main:end		

1

\$(2%	%	
	1	2
main:begin		
1		
2		
		B
1		
	A	
2		
2		
main:end		

\$(2%

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include "mythreads.h"
4
5  static volatile int counter = 0;
6
7  //
8  // mythread()
9  //
```

```

10 // Simply adds 1 to counter repeatedly, in a loop
11 // No, this is not how you would add 10,000,000 to
12 // a counter, but it shows the problem nicely.
13 //
14 void *
15 mythread(void *arg)
16 {
17     printf("%s: begin\n", (char *) arg);
18     int i;
19     for (i = 0; i < 1e7; i++) {
20         counter = counter + 1;
21     }
22     printf("%s: done\n", (char *) arg);
23     return NULL;
24 }
25
26 //
27 // main()
28 //
29 // Just launches two threads (pthread_create)
30 // and then waits for them (pthread_join)
31 //
32 int
33 main(int argc, char *argv[])
34 {
35     pthread_t p1, p2;
36     printf("main: begin (counter = %d)\n", counter);
37     Pthread_create(&p1, NULL, mythread, "A");
38     Pthread_create(&p2, NULL, mythread, "B");
39
40     // join waits for the threads to finish
41     Pthread_join(p1, NULL);
42     Pthread_join(p2, NULL);
43     printf("main: done with both (counter = %d)\n", counter);
44     return 0;
45 }

```

26.3

t1.c

Stevens

[SR05]

Pthread_create()

pthread_create()

0

Pthread_create()

1000

 10^7

20000000

```

prompt> gcc -o main main.c -Wall -pthread
prompt> ./main
main: begin (counter = 0)
A: begin
B: begin
A: done
B: done
main: done with both (counter = 20000000)

```

```

prompt> ./main
main: begin (counter = 0)
A: begin
B: begin
A: done
B: done
main: done with both (counter = 19345221)

```

deterministic

```

prompt> ./main
main: begin (counter = 0)
A: begin
B: begin
A: done
B: done
main: done with both (counter = 19221041)

```



\$(Z%

counter

1

x86

mov 0x8049a1c, %eax

add \$0x1, %eax

mov %eax, 0x8049a1c

counter

eax

0x8049a1c

eax

3

1

0x1

x86

mov

eax

1

counter

50

1

eax = 50

1

eax = 51

1

eax

TCB

2

eax [

virtualized

]

counter

2

eax = 50

2

eax

1

eax

50

= 51

51

counter

0x8049a1c

counter

1

mov

add

mov

mov

counter

51

50

51

counter

52

100

x86

mov

5

RISC

add

3

100 mov 0x8049a1c, %eax

105 add \$0x1, %eax

108 mov %eax, 0x8049a1c

26.4

counter

50

race condition

indeterminate

deterministic

critical section

\$ (Z&					
OS	1	2			
			PC	%eax	counter
	mov 0x8049a1c, %eax add \$0x1, %eax		100	0	50
			105	50	50
			108	51	50
T1 T2			100	0	50
		mov 0x8049a1c, %eax add \$0x1, %eax mov %eax, 0x8049a1c	105	50	50
			108	51	50
			113	51	51
T2 T1			108	51	51
	mov %eax, 0x8049a1c		113	51	51

mutual exclusion

Edsger Dijkstra
1968 Cooperating Sequential Processes

[D68]
Dijkstra

\$ (Z&

memory-add 0x8049a1c, \$0x1

atomically

3

mov 0x8049a1c, %eax
add \$0x1, %eax
mov %eax, 0x8049a1c

B

B

synchronization primitive

4		Dijkstra
[D65	D68]	
1	critical section	
1	race condition	
1	indeterminate	
		deterministic
1		mutual exclusion

\$(ž

I/O

I/O

/

condition variable

\$(\checkmark

OS

write()

inode

3

inode

OS

[L+93]

atomic

transaction

[GR92]

[D65] Solution of a Problem in Concurrent Programming Control

E. W. Dijkstra

Communications of the ACM, 8(9):569, September 1965

Dijkstra

[D68] Cooperating Sequential Processes Edsger W. Dijkstra, 1968

Dijkstra

Technische Hochschule of Eindhoven THE

Cooperating Sequential Processes

Dijkstra

THE

THE

THE

the

[GR92] Transaction Processing: Concepts and Techniques Jim Gray and Andreas Reuter
Morgan Kaufmann, September 1992

Jim Gray

Jim Gray

Gray

Gray

[L+93] Atomic Transactions

Nancy Lynch, Michael Merritt, William Weihl, Alan Fekete Morgan Kaufmann, August 1993

[SR05] Advanced Programming in the UNIX Environment

UNIX

x86.py

README

1 loop.s

cat loop.s

./x86.py -p loop.s -t 1 -i 100 -R dx

100

%dx

%dx

-c

2

./x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx

%dx

3 %dx

-c

3

```
./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx
```

-s

4

looping-race-nolock.s

2000

x

```
./x86.py -p looping-race-nolock.s -t 1 -M 2000
```

x

2000

-c

5

```
./x86.py -p looping-race-nolock.s -t 2 -a bx=3 -M 2000
```

3

x

6

```
./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 0
```

-s 1

-s 2

x

7

```
./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 1
```

x

-i 2 -i 3

8

set -a bx = 100

-i

9

wait-for-me.s

```
./x86.py -p wait-for-me.s -a ax=1,ax=0 -R ax -M 2000
```

0 %ax

1

1

0

%ax

2000

2000

10

```
./x86.py -p wait-for-me.s -a ax=0,ax=1 -R ax -M 2000
```

0

-i 1000

CPU