Parallel and Distributed Computing

POSIX Threads(pthread) Programming

Kun Suo

Computer Science, Kennesaw State University

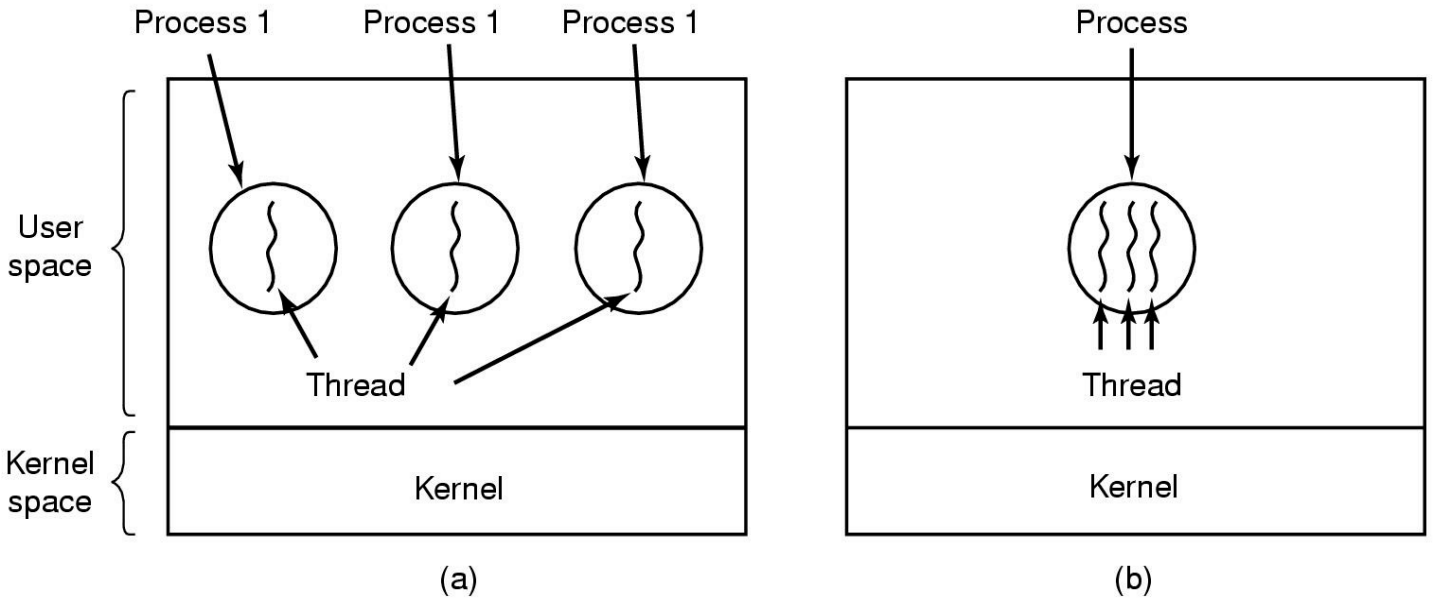
https://kevinsuo.github.io/

Outline

- What is pthread?
  - Thread model
  - Pthread overview
- Pthread management
  - Creation and termination
  - Identification
  - Join and detach
- Pthread data sharing
  - Mutex
  - Condition variable
  - Barrier
  - Semaphore

The Thread Model

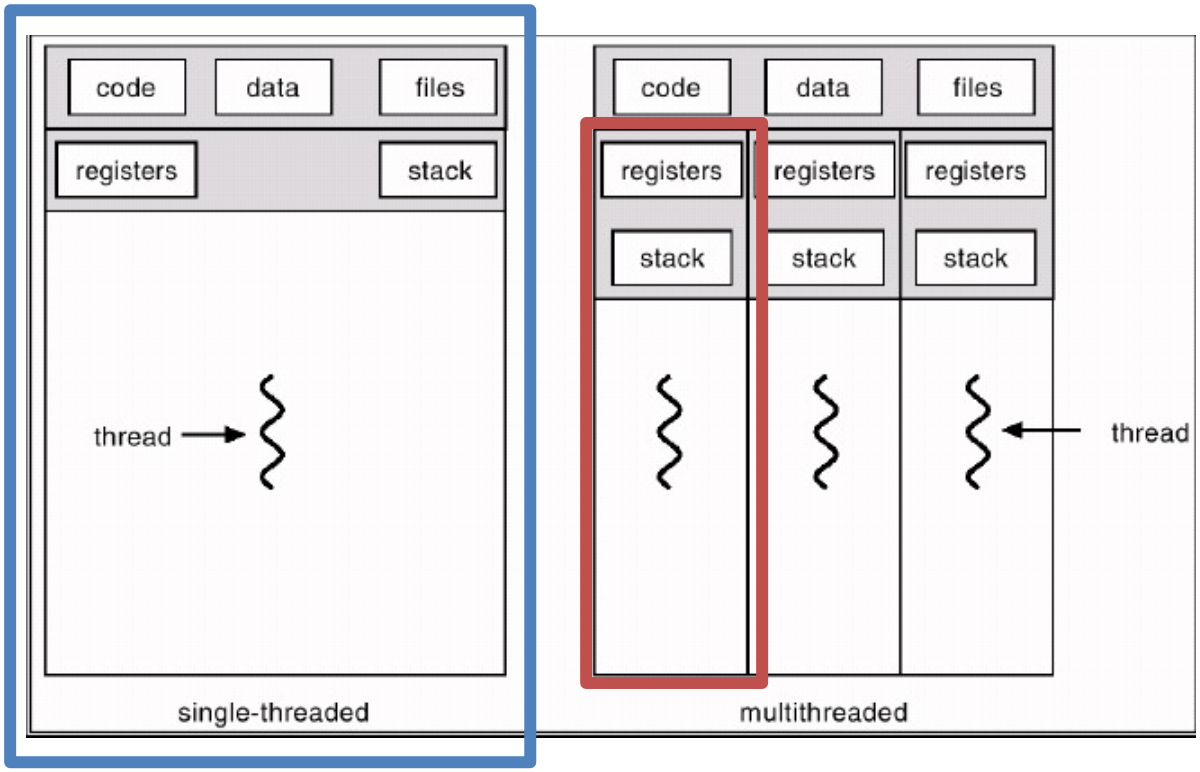
- Process: for resource grouping and execution
- Thread: a finer-granularity entity for execution and parallelism



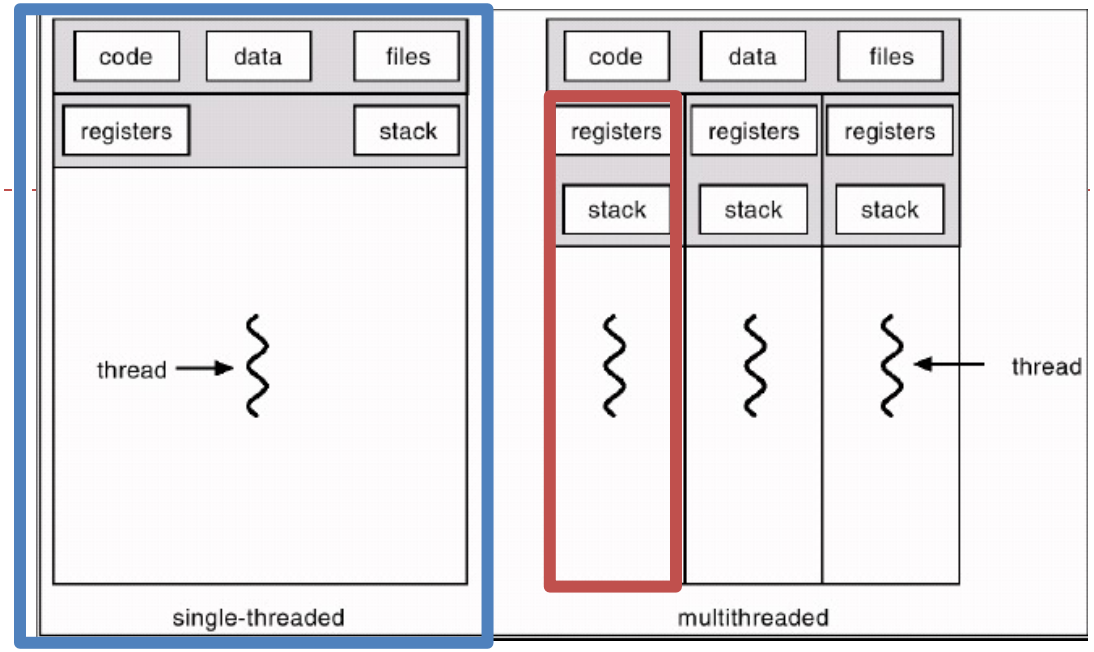
(a) Three processes each with one thread, but different address spaces

(b) One process with three threads, sharing the address space

The Thread Model



Process	Thread
Have data/code/heap	Share code, data, heap
Include at lease one thread	Multiple can coexist in a process
Have own address space, isolated from other processes	Only have own stack and registers

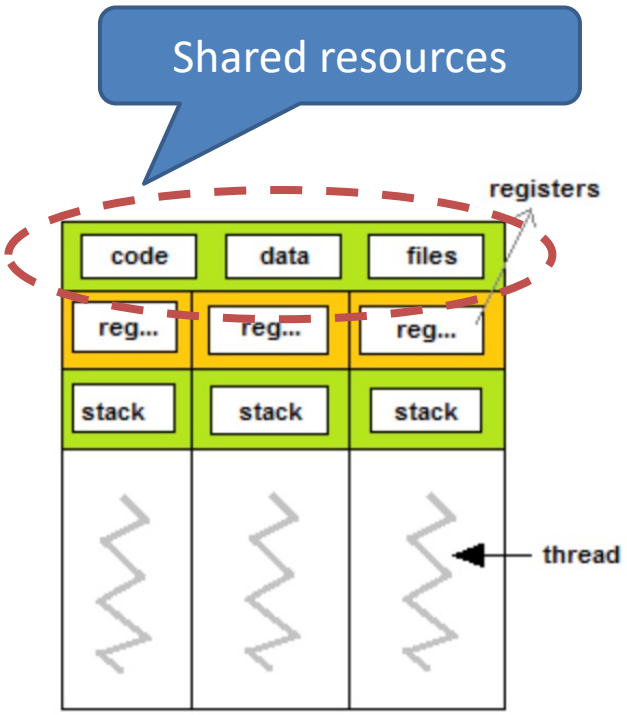


Per process items	Per thread items
Address space Global variables Open files Child processes Pending alarms Signals and signal handlers Accounting information	Program counter Registers Stack State

Items shared by all threads in a process

Items private to each thread

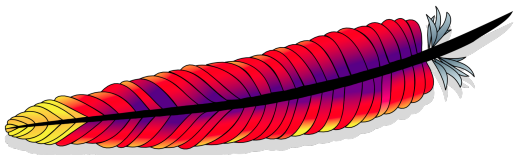
The Thread Model



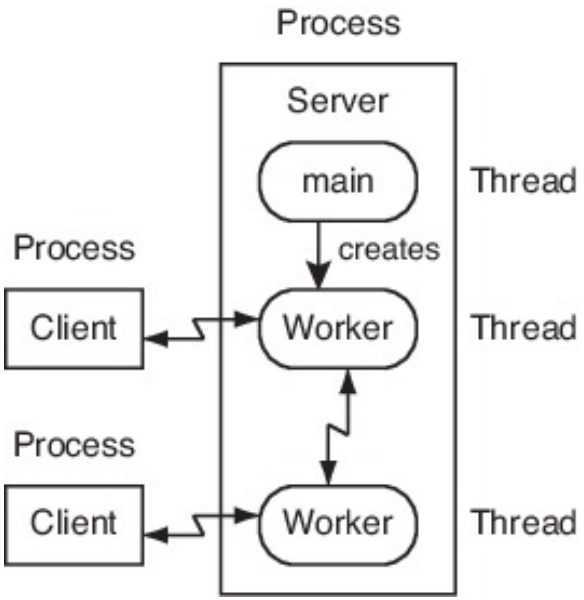
Three threads

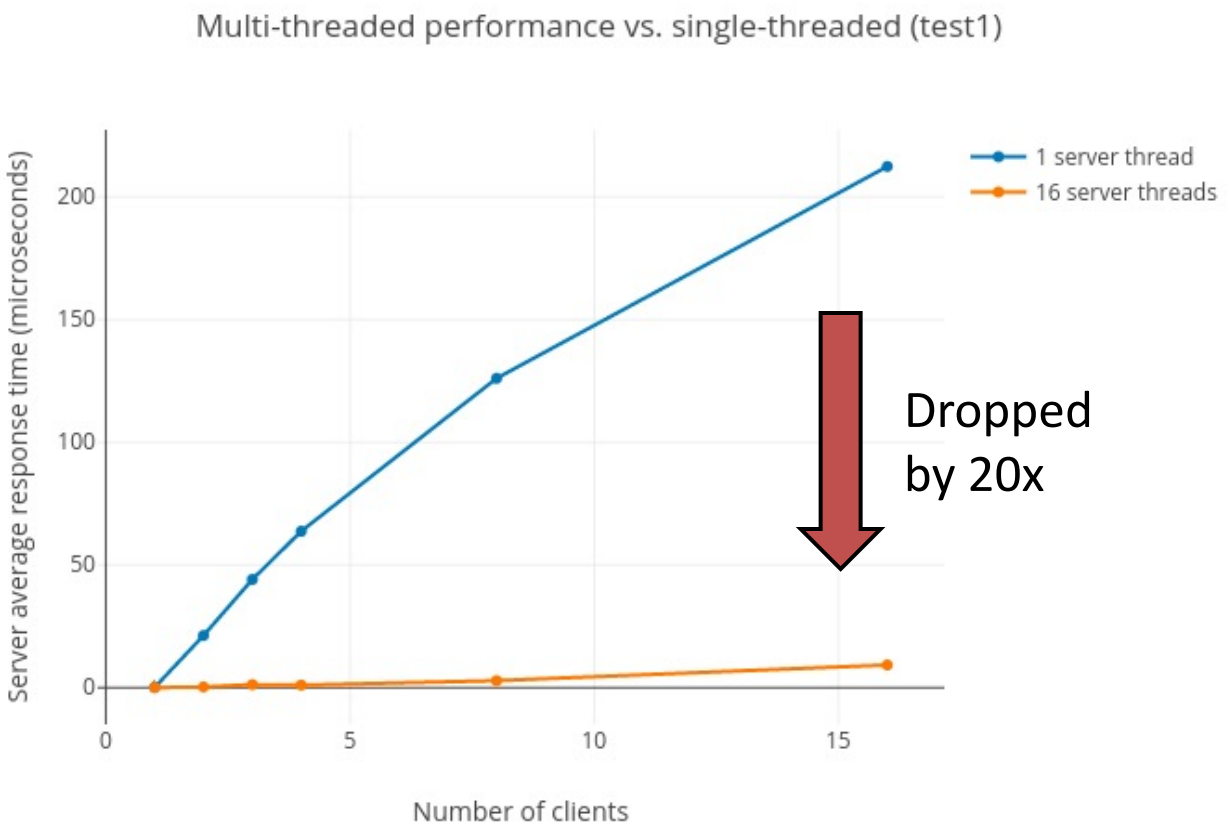
- Because threads within the same process share resources
  - Changes made by one thread to shared system resources (closing a file) will be seen by all other threads
  - Two pointers having the same value point to the same data
  - Reading and writing to the same memory location is possible, and therefore requires explicit synchronization by the programmer!

Multi-threads for application performance



Apache Web Server

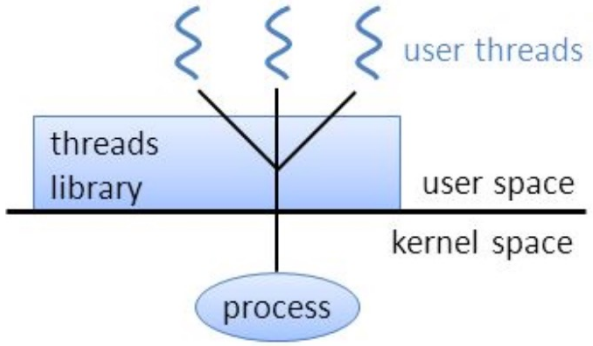




Threading Models

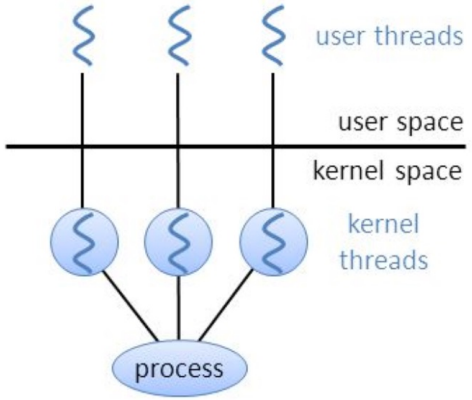
- N:1 (User-level threading)
  - GNU Portable Threads

    https://www.gnu.org/software/pth/

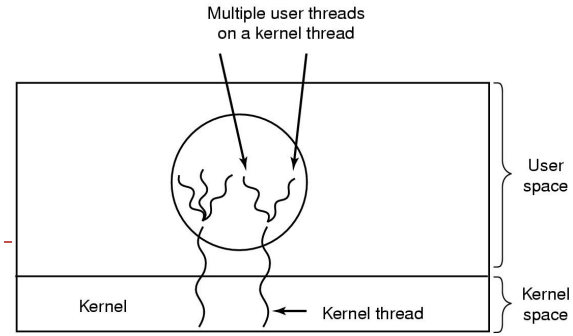


- 1:1 (Kernel-level threading)
  - Native POSIX Thread Library (Pthread)

    https://en.wikipedia.org/wiki/Native_POSIX_Thread_Library
    Linux/MacOS/Windows



- M:N (Hybrid threading)
  - Solaris https://www.oracle.com/solaris/solaris11/



Pthreads Overview

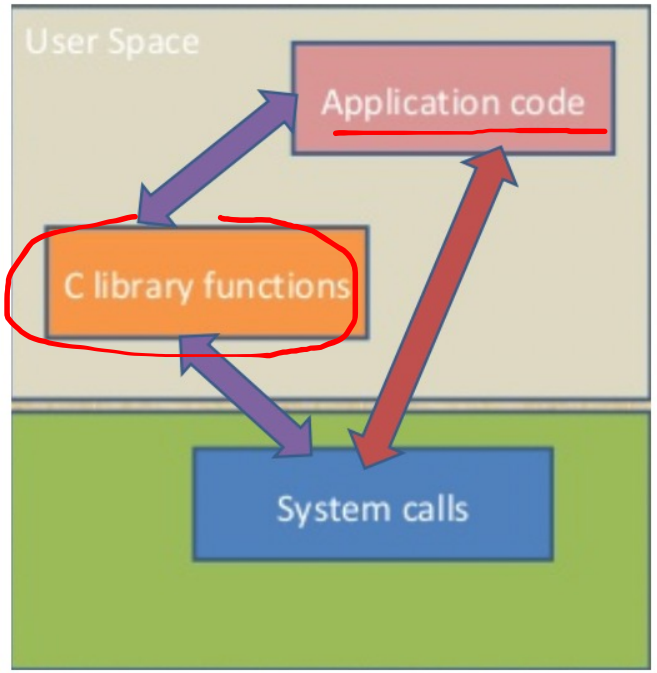
- What are Pthreads?
  - An IEEE standardized thread programming interface (IEEE POSIX 1003.1c)
  - http://www.open-std.org/jtc1/sc22/WG15/
  - POSIX (Portable Operating System Interface) threads

Pthreads Overview

- What are Pthreads?
  - Defined as a set of C language programming types and procedure calls, implemented with a pthread.h header/include file and a thread library

Pthread_create()
in the user space

Clone()
in the kernel



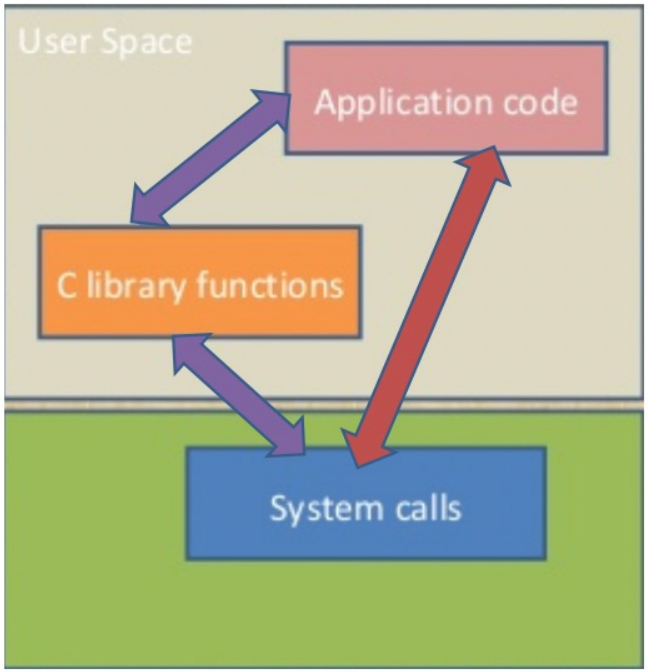
Why Pthreads ?

- Performance!
  - Lightweight
  - Communication
  - Fine granularity of concurrency
  - Easy to use
  - …

Why Pthreads ?

Pthread_create()
in the user space

Clone()
in the kernel



Time in second with running 50000 fork() or Pthread_create()

10~50 times faster

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
IBM 1.9 GHz POWER5 p5-575	50.66	3.32	42.75	1.13	0.54	0.75
INTEL 2.4 GHz Xeon	23.81	3.12	8.97	1.70	0.53	0.30
INTEL 1.4 GHz Itanium 2	23.61	0.12	3.42	2.10	0.04	0.01

Outline

- What is pthread?
  - Thread model
  - Pthread overview
- Pthread management
  - Creation and termination
  - Identification
  - Join and detach
- Pthread data sharing
  - Mutex
  - Condition variable
  - Barrier
  - Semaphore

Hello World!

https://github.com/kevinsuo/CS7172/blob/master/hello-pthread.c

```c
#include <stdio.h>

int main(void) {
    printf("hello, world\n");

    return 0;
}
```

helloworld-pthread.c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
        long tid;
        tid = (long)threadid;
        printf("Hello World! It's me, thread #%ld!\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        pthread_t threads[NUM_THREADS];
        int rc;
        long t;
        for(t=0;t<NUM_THREADS;t++){
                printf("In main: creating thread %ld\n", t);
                rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
                if (rc){
                        printf("ERROR; return code from pthread_create() is %d\n", rc);
                        exit(-1);
                }
        }

        pthread_exit(NULL);
}
```

Header file

What the thread will do?

Create new threads

Pthread exit

The Pthreads API

- The API is defined in the ANSI/IEEE POSIX 1003.1 – 1995
  - Naming conventions: all identifiers in the library begins with pthread_
  - Three major classes of subroutines
    - Thread management, mutexes, condition variables

Routine Prefix	Functional Group
pthread_	Threads themselves and miscellaneous subroutines
pthread_attr_	Thread attributes objects
pthread_mutex_	Mutexes
pthread_mutexattr_	Mutex attributes objects.
pthread_cond_	Condition variables
pthread_condattr_	Condition attributes objects
pthread_key_	Thread-specific data keys

Compile

https://github.com/kevinsuo/CS7172/blob/master/hello-pthread.c

Compile command

With pthread library

```
gcc  hello-pthread.c  -o  hello-pthread.o  -pthread
```

source file

create this executable file name

(as opposed to default a.out)

Execution

https://github.com/kevinsuo/CS7172/blob/master/hello-pthread.c

./hello-pthread.o

Run the output file

```
ksuo@LinuxKernel2 ~> gcc hello-pthread.c -o hello-pthread.o -pthread
ksuo@LinuxKernel2 ~> ./hello-pthread.o
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
In main: creating thread 3
Hello World! It's me, thread #0!
In main: creating thread 4
Hello World! It's me, thread #1!
Hello World! It's me, thread #2!
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

Example

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
        long tid;
        tid = (long)threadid;
        printf("Hello World! It's me, thread #%ld!\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        pthread_t threads[NUM_THREADS];
        int rc;
        long t;
        for(t=0;t<NUM_THREADS;t++){
                printf("In main: creating thread %ld\n", t);
                rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
                if (rc){
                        printf("ERROR; return code from pthread_create() is %d\n", rc);
                        exit(-1);
                }
        }

        pthread_exit(NULL);
}
```

https://github.com/kevinsuo/CS7172/blob/master/hello-pthread.c

```
ksuo@LinuxKernel2 ~> gcc hello-pthread.c -o hello-pthread.o -pthread
ksuo@LinuxKernel2 ~> ./hello-pthread.o
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
In main: creating thread 3
Hello World! It's me, thread #0!
In main: creating thread 4
Hello World! It's me, thread #1!
Hello World! It's me, thread #2!
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

1, Creation & Termination

pthread_create (thread, attr, thread_function, arg)

- **1st parameter: pthread in the array**
- **3rd parameter: the function for the thread to run**
- ***4th parameter: arg* is passed by reference as a pointer cast of type void**

Initially, your main() program comprises a single, default thread.

e.g.,

```
pthread_t threads[NUM_THREADS];
int t;
for(t=0;t<NUM_THREADS;t++){
    rc = pthread_create(&threads[t], NULL, thread_function, (void *)t);
}
```

1, Creation example

https://github.com/kevinsuo/CS3502/blob/master/pthread_create_example.c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *doSomeThing(void *threadid)
{
        long tid = (long)threadid;
        printf("I am thread #%ld!\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        pthread_t threads[NUM_THREADS];
        int t;
        for(t=0;t<NUM_THREADS;t++){
                pthread_create(&threads[t], NULL, doSomeThing, (void *)t);
        }
        pthread_exit(NULL);
}
```

gcc pthread_create_example.c -o pthread_create_example.o -pthread

1, Creation example

$ gcc pthread_create_example.c -o create_example.o   -pthread

```
ksuo@LinuxKernel2 ~> ./create-example.o
I am thread #0!
I am thread #1!
I am thread #3!
I am thread #2!
I am thread #4!
```

NUM_THREADS=5

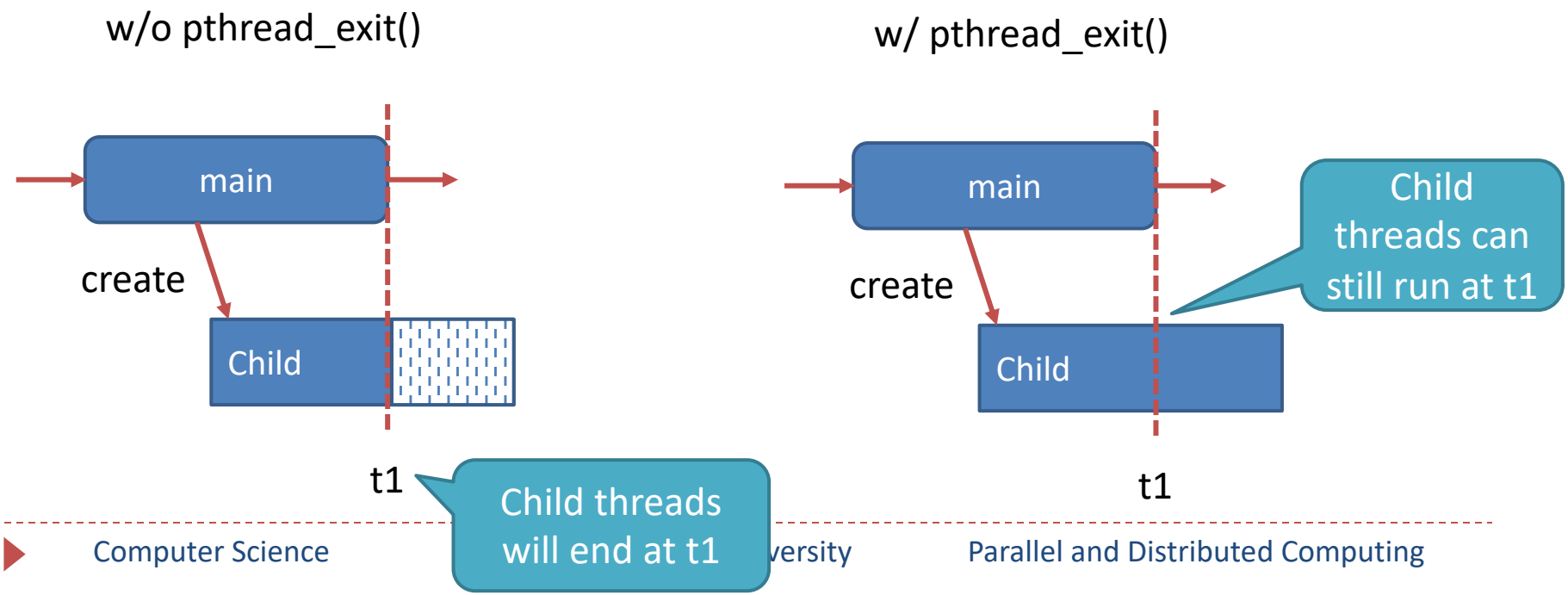
```
ksuo@LinuxKernel2 ~> ./create-example.o
I am thread #0!
I am thread #1!
I am thread #3!
I am thread #2!
I am thread #4!
I am thread #5!
I am thread #8!
I am thread #6!
I am thread #7!
I am thread #9!
```

NUM_THREADS=10

1, Creation & Termination

pthread_exit (status)

• **If** main() **finishes before the threads it has created, and exists with the** pthread_exit(), **the other threads will continue to execute. Otherwise, they will be automatically terminated when** main() **finishes**



1, Termination example

https://github.com/kevinsuo/CS3502/blob/master/pthread_create.c

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void *myThread1(void)
{
    int i;
    for (i=0; i<10; i++)
    {
        printf("This is the 1st pthread.\n");
        sleep(1);
    }
}

int main()
{
    int ret=0;
    pthread_t id1;

    printf("This is main thread!\n");

    ret = pthread_create(&id1, NULL, (void*)myThread1, NULL)
    if (ret)
    {
        printf("Create pthread error!/n");
        return 1;
    }

    pthread_exit(NULL);

    return 0;
}
```

gcc pthread_create.c -o pthread_create.o -pthread

```
./pthread_create.o  /home/
ksuo@LinuxKernel2 ~> ./pthread_create.o
This is main thread!
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
```

With it: Child threads will continue to run when the main thread ends.

1, Termination example

https://github.com/kevinsuo/CS3502/blob/master/pthread_create.c

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void *myThread1(void)
{
    int i;
    for (i=0; i<10; i++)
    {
        printf("This is the 1st pthread.\n");
        sleep(1);
    }
}

int main()
{
    int ret=0;
    pthread_t id1;

    printf("This is main thread!\n");

    ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
    if (ret)
    {
        printf("Create pthread error!/n");
        return 1;
    }

//  pthread_exit(NULL);

    return 0;
}
```

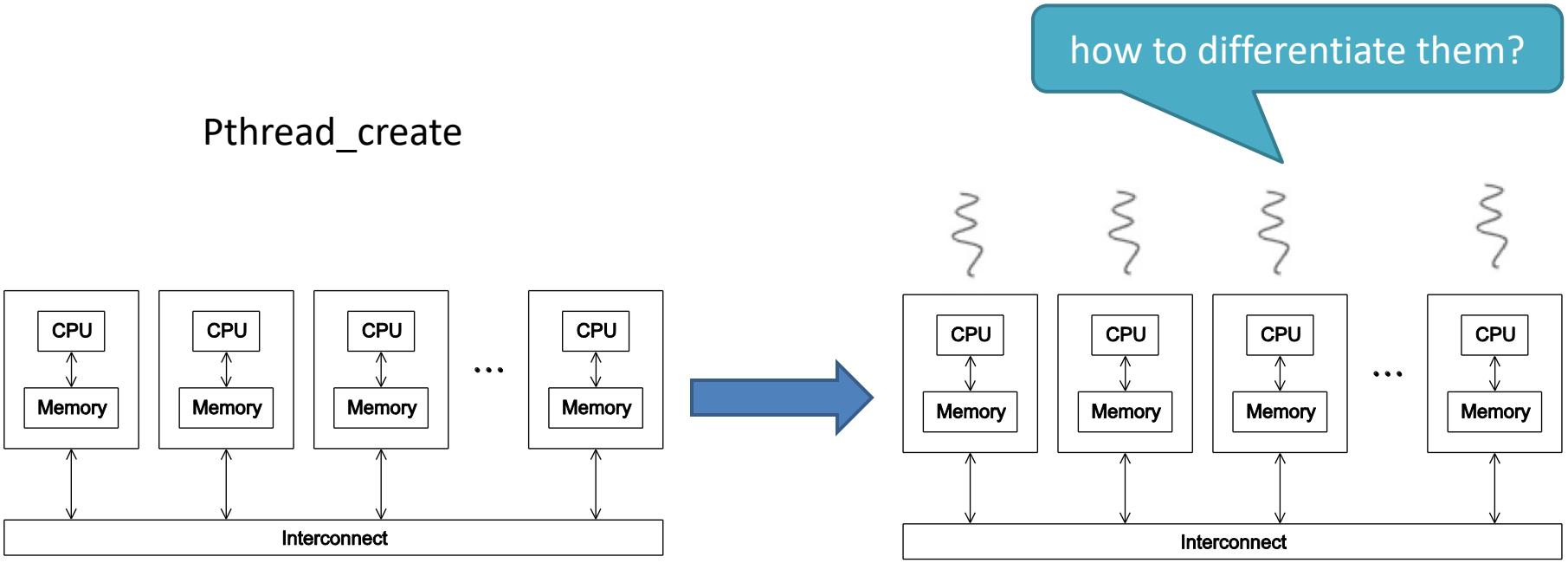
gcc pthread_create.c -o pthread_create.o -pthread

```
ksuo@LinuxKernel2 ~> ./pthread_create.o
This is main thread!
ksuo@LinuxKernel2 ~>
```

Without it: Child threads will end if the main thread ends.

2, identification

Pthread_create



2, identification

pthread_self (void)

- **returns the ID of the thread in which it is invoked.**

- A thread can obtain its own thread ID by calling the pthread_self() function

2, identification

https://github.com/kevinsuo/CS7172/blob/master/pthread_id.c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *doSomeThing(void *threadid)
{
        long tid = (long)pthread_self();
        printf("id is %ld\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        pthread_t threads[NUM_THREADS];
        int t;
        for(t=0;t<NUM_THREADS;t++){
                pthread_create(&threads[t], NULL, doSomeThing, (void *)t);
        }
        pthread_exit(NULL);
}
```

gcc pthread_id.c -o self.o -pthread

```
ksuo@LinuxKernel2 ~> ./self.o
id is 140603160495872
id is 140603135317760
id is 140603143710464
id is 140603152103168
id is 140603126925056
```

2, identification example

https://github.com/kevinsuo/CS7172/blob/master/pthread_id_2.c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *doSomeThing(void *t)
{
        long tid = (long)t;
        printf("id is %ld\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        pthread_t threads[NUM_THREADS];
        int t;
        for(t=0;t<NUM_THREADS;t++){
                pthread_create(&threads[t], NULL, doSomeThing, (void *)t);
        }
        pthread_exit(NULL);
}
```

gcc pthread_id_2.c –o pthread_id_2.o –pthread

arg can be used to differentiate threads

2, identification example

https://github.com/kevinsuo/CS7172/blob/master/pthread_id_2.c

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS     5

void *doSomeThing(void *t)
{
        long tid = (long)t;
        printf("id is %ld\n", tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        pthread_t threads[NUM_THREADS];
        int t;
        for(t=0;t<NUM_THREADS;t++){
                pthread_create(&threads[t], NULL, doSomeThing, (void *)t);
        }
        pthread_exit(NULL);
}
```

arg can be used to differentiate threads

Compile:

gcc pthread_id_2.c –o pthread_id_2.o –pthread

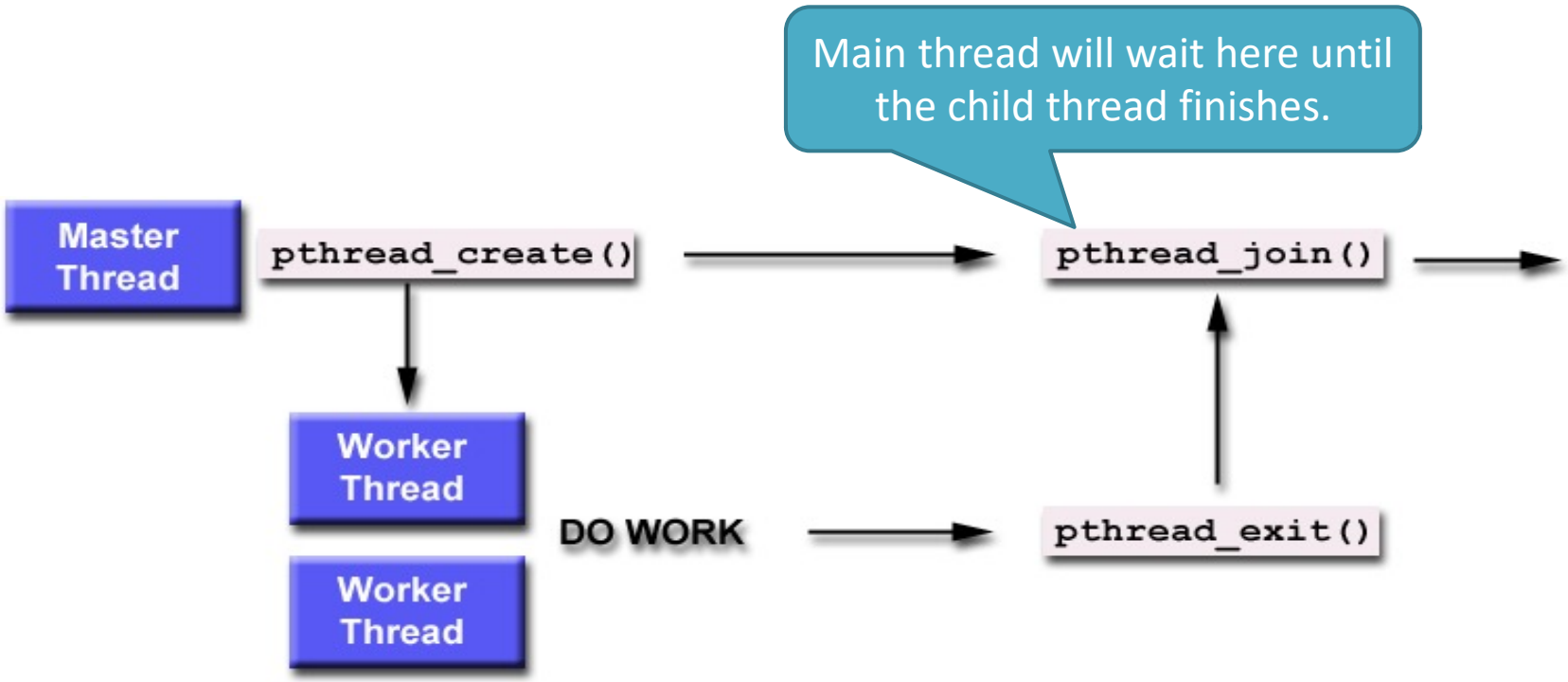
Run:

./pthread_id_2.o

```
ksuo@LinuxKernel2 ~> ./pthread_id_2.o
id is 0
id is 1
id is 3
id is 4
id is 2
```

3, join

pthread_join (threadid, status) * Joining is one way to accomplish synchronization between threads: The pthread_join() subroutine blocks the calling thread until the specified threadid thread terminates.
The main thread will be blocked here to wait the child thread to finish



3, join example

https://github.com/kevinsuo/CS7172/blob/master/pthread_join.c

```c
int count = 0;

void *myThread1(void)
{
        int i;
        for (i=0; i<3; i++)
        {
                printf("This is the 1st pthread.\n");
                count++;
                sleep(1);
        }
}


int main()
{
        int ret=0;
        pthread_t id1;

        printf("This is main thread!\n");

        ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
        if (ret)
        {
                printf("Create pthread error!/n");
                return 1;
        }

//      pthread_join(id1, (void *)ret);

        printf("count:%d\n", count);
        pthread_exit(NULL);

        return 0;
}
```

Compile:

gcc pthread_join.c –o pthread_join.o –pthread

Run:

./pthread_join.o

```
pi@raspberrypi ~/Downloads> ./test.o
This is main thread!
This is the 1st pthread.
count:0
This is the 1st pthread.
This is the 1st pthread.
```

Count is 0 as the main thread keeps executing.

3, join example

https://github.com/kevinsuo/CS7172/blob/master/pthread_join.c

```c
int count = 0;

void *myThread1(void)
{
        int i;
        for (i=0; i<3; i++)
        {
                printf("This is the 1st pthread.\n");
                count++;
                sleep(1);
        }
}


int main()
{
        int ret=0;
        pthread_t id1;

        printf("This is main thread!\n");

        ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
        if (ret)
        {
                printf("Create pthread error!/n");
                return 1;
        }

        pthread_join(id1, (void *)ret);

        printf("count:%d\n", count);
        pthread_exit(NULL);

        return 0;
}
```

Compile:

gcc pthread_join.c –o pthread_join.o –pthread

Run:

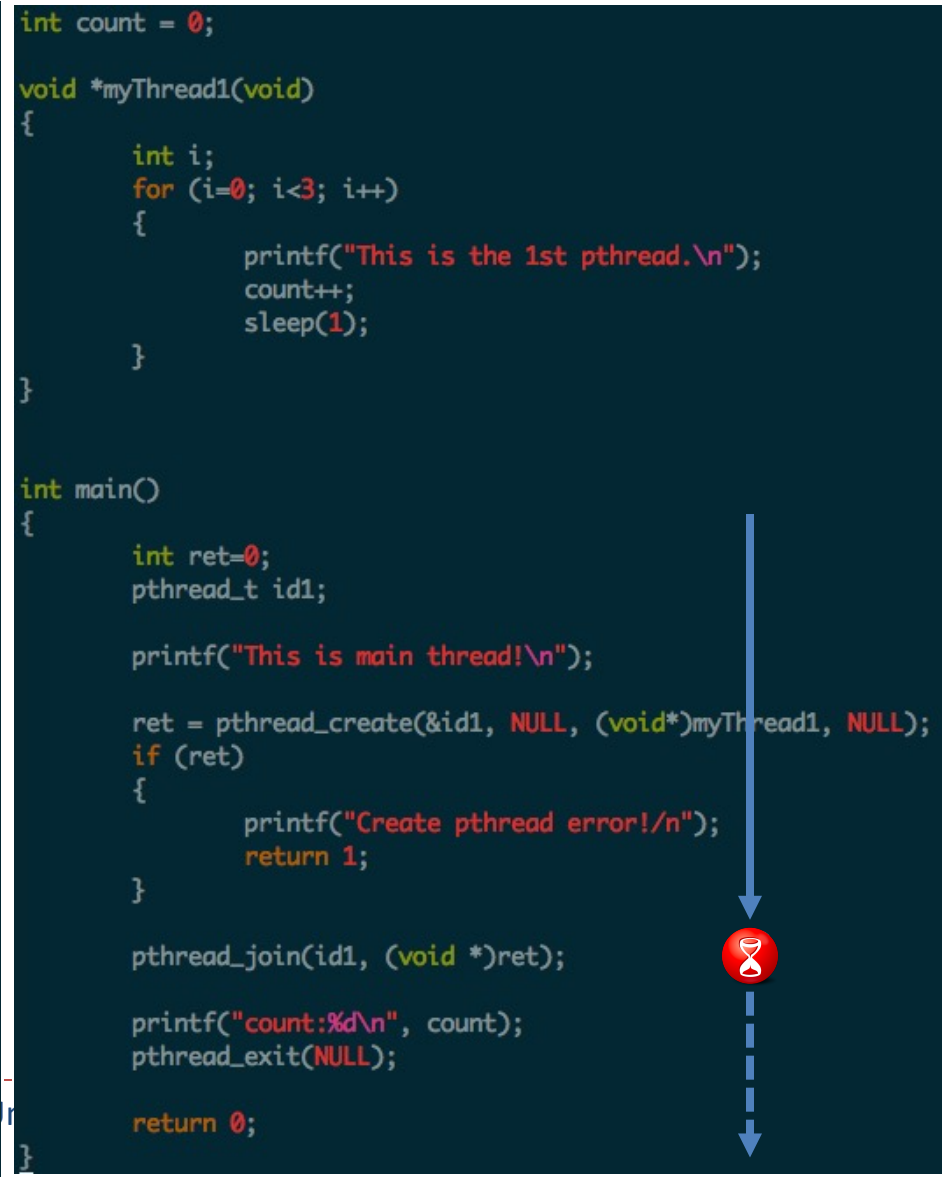
./pthread_join.o

```
pi@raspberrypi ~/Downloads> ./test.o
This is main thread!
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
count:3
```

Count is 3 as the main thread is blocked here until child thread finished.

3, join example

```c
int count = 0;

void *myThread1(void)
{
        int i;
        for (i=0; i<3; i++)
        {
                printf("This is the 1st pthread.\n");
                count++;
                sleep(1);
        }
}


int main()
{
        int ret=0;
        pthread_t id1;

        printf("This is main thread!\n");

        ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
        if (ret)
        {
                printf("Create pthread error!/n");
                return 1;
        }

//      pthread_join(id1, (void *)ret);

        printf("count:%d\n", count);
        pthread_exit(NULL);

        return 0;
}
```

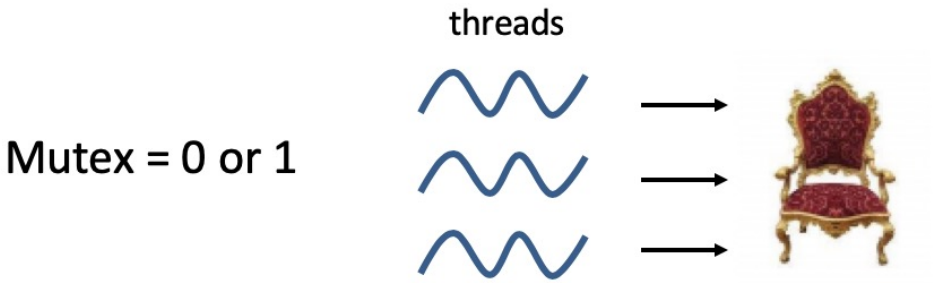


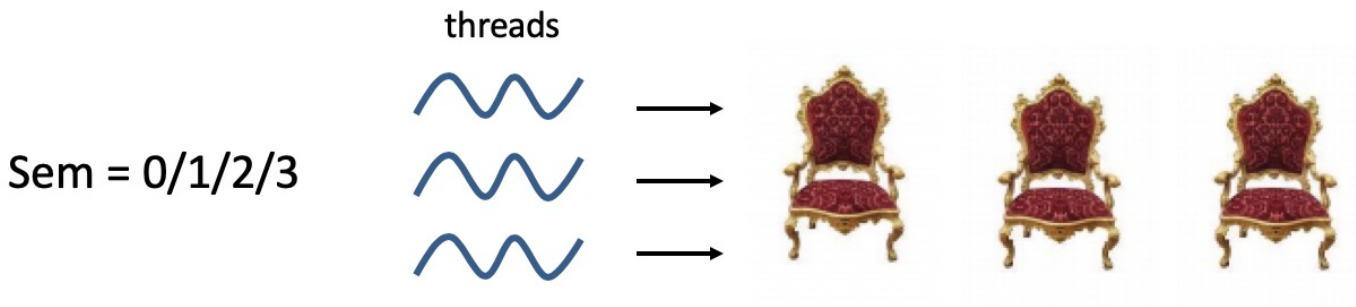
Outline

- What is pthread?
  - Thread model
  - Pthread overview
- Pthread management
  - Creation and termination
  - Identification
  - Join and detach
- Pthread data sharing
  - Mutex
  - Condition variable
  - Barrier
  - Semaphore

4, mutex lock

- Mutex: a simplified version of the semaphores
  - A variable that can be in one of two states: unlocked or locked
  - Supports synchronization by controlling access to shared data





4, mutex – Creating and Destroying Mutexes

pthread_mutex_init (mutex, attr)

pthread_mutex_destroy (mutex)

1. Mutex variables must be declared with type pthread_mutex_t, and must be initialized before can be used. The mutex is initially not locked.

   Statically: **pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER**
   Dynamically, with **pthread_mutex_init (mutex, attr)**

2. The attr object must be declared with type pthread_mutexattr_t

3. Programmers should free a mutex object that is no longer used

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int counter = 0;
static pthread_mutex_t mlock;

void *compute()
{
    int i = 0;
//    pthread_mutex_lock(&mlock)
    while (i < 10000) {
        counter = counter + 1;
        i++;
    }
    printf("Counter value: %d\n", counter);
//    pthread_mutex_unlock(&mlock);
}

int main()
{
    pthread_t thread1, thread2, thread3, thread4, thread5;
//    if (pthread_mutex_init(&mlock, NULL) != 0)
//    {
//        printf("mutex init failed\n");
//        return 1;
//    }

    pthread_create(&thread1, NULL, compute, (void *)&thread1);
    pthread_create(&thread2, NULL, compute, (void *)&thread2);
    pthread_create(&thread3, NULL, compute, (void *)&thread3);
    pthread_create(&thread4, NULL, compute, (void *)&thread4);
    pthread_create(&thread5, NULL, compute, (void *)&thread5);

    pthread_exit(NULL);
//    pthread_mutex_destroy(&mlock);
    exit(0);
}
```

https://github.com/kevinsuo/CS7172/blob/master/mutexlock.c

Compile:

gcc mutexlock.c –o mutexlock.o –pthread

Run:

./mutexlock.o

```
ksuo@LinuxKernel2 ~> ./mutexlock.o
Counter value: 10000
Counter value: 40358
Counter value: 20000
Counter value: 30358
Counter value: 47442
ksuo@LinuxKernel2 ~> ./mutexlock.o
Counter value: 10244
Counter value: 12739
Counter value: 25027
Counter value: 31032
Counter value: 40015
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int counter = 0;
static pthread_mutex_t mlock;

void *compute()
{
    int i = 0;
    pthread_mutex_lock(&mlock);
    while (i < 10000) {
        counter = counter + 1;
        i++;
    }
    printf("Counter value: %d\n", counter);
    pthread_mutex_unlock(&mlock);
}

int main()
{
    pthread_t thread1, thread2, thread3, thread4, thread5;
    if (pthread_mutex_init(&mlock, NULL) != 0)
    {
        printf("mutex init failed\n");
        return 1;
    }

    pthread_create(&thread1, NULL, compute, (void *)&thread1);
    pthread_create(&thread2, NULL, compute, (void *)&thread2);
    pthread_create(&thread3, NULL, compute, (void *)&thread3);
    pthread_create(&thread4, NULL, compute, (void *)&thread4);
    pthread_create(&thread5, NULL, compute, (void *)&thread5);

    pthread_exit(NULL);
    pthread_mutex_destroy(&mlock);
    exit(0);
}
```

https://github.com/kevinsuo/CS7172/blob/master/mutexlock.c

Compile:

gcc mutexlock.c –o mutexlock.o –pthread

Run:

./mutexlock.o

Init the mutex lock

```
pi@raspberrypi ~/Downloads> ./mutexlock.o
Counter value: 10000
Counter value: 20000
Counter value: 30000
Counter value: 40000
Counter value: 50000
```

destroy the mutex lock

5, mutex – Locking and Unlocking of Mutexes

pthread_mutex_lock (mutex) ; **P(down)** **pthread_mutex_trylock (mutex)** **pthread_mutexattr_unlock (mutex)** ; **V(up)**
1. **pthread_mutex_lock (mutex)** is a blocking call. 2. **pthread_mutex_trylock (mutex)** is a non-blocking call, useful in preventing the deadlock conditions (priority-inversion problem) 3. If you use multiple mutexes, the order is important;

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int counter = 0;
static pthread_mutex_t mlock;

void *compute()
{
    int i = 0;
    pthread_mutex_lock(&mlock);
    while (i < 10000) {
        counter = counter + 1;
        i++;
    }
    printf("Counter value: %d\n", counter);
    pthread_mutex_unlock(&mlock);
}

int main()
{
    pthread_t thread1, thread2, thread3, thread4, thread5;
    if (pthread_mutex_init(&mlock, NULL) != 0)
    {
        printf("mutex init failed\n");
        return 1;
    }

    pthread_create(&thread1, NULL, compute, (void *)&thread1);
    pthread_create(&thread2, NULL, compute, (void *)&thread2);
    pthread_create(&thread3, NULL, compute, (void *)&thread3);
    pthread_create(&thread4, NULL, compute, (void *)&thread4);
    pthread_create(&thread5, NULL, compute, (void *)&thread5);

    pthread_exit(NULL);
    pthread_mutex_destroy(&mlock);
    exit(0);
}
```

Lock the “counter”

Unlock the “counter”

https://github.com/kevinsuo/CS7172/blob/master/mutexlock.c

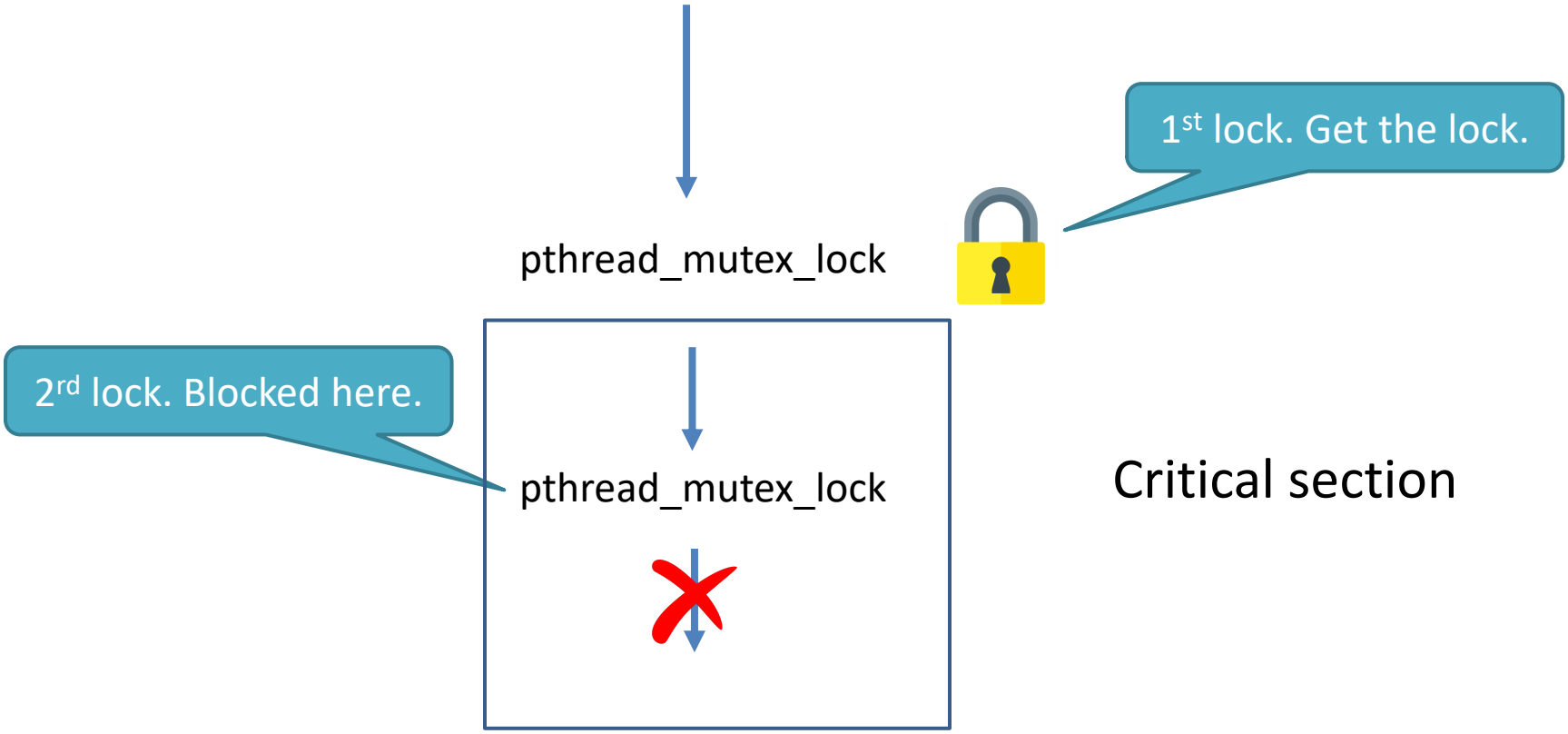
Critical section

```
pi@raspberrypi ~/Downloads> ./mutexlock.o
Counter value: 10000
Counter value: 20000
Counter value: 30000
Counter value: 40000
Counter value: 50000
```

pthread_mutex_lock vs. pthread_mutex_trylock

https://github.com/kevinsuo/CS7172/blob/master/lock.c

- pthread_mutex_lock (mutex) is a blocking call.



pthread_mutex_lock vs. pthread_mutex_trylock

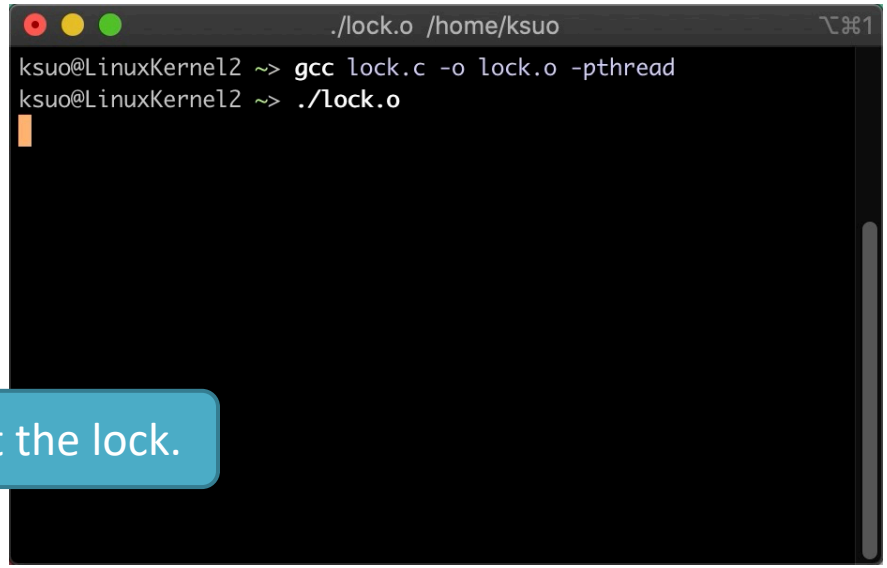
https://github.com/kevinsuo/CS7172/blob/master/lock.c

- pthread_mutex_lock (mutex) is a blocking call.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int main(void)
{
        int i = 0;
        int res;

        pthread_mutex_t lock;
        pthread_mutex_init(&lock, NULL);

        pthread_mutex_lock(&lock);
        if((res = pthread_mutex_lock(&lock) != 0)) {
                printf("don't lock\n");
        } else {
                printf("locked\n");
        }

        return 0;
}
```

1st lock. Get the lock.

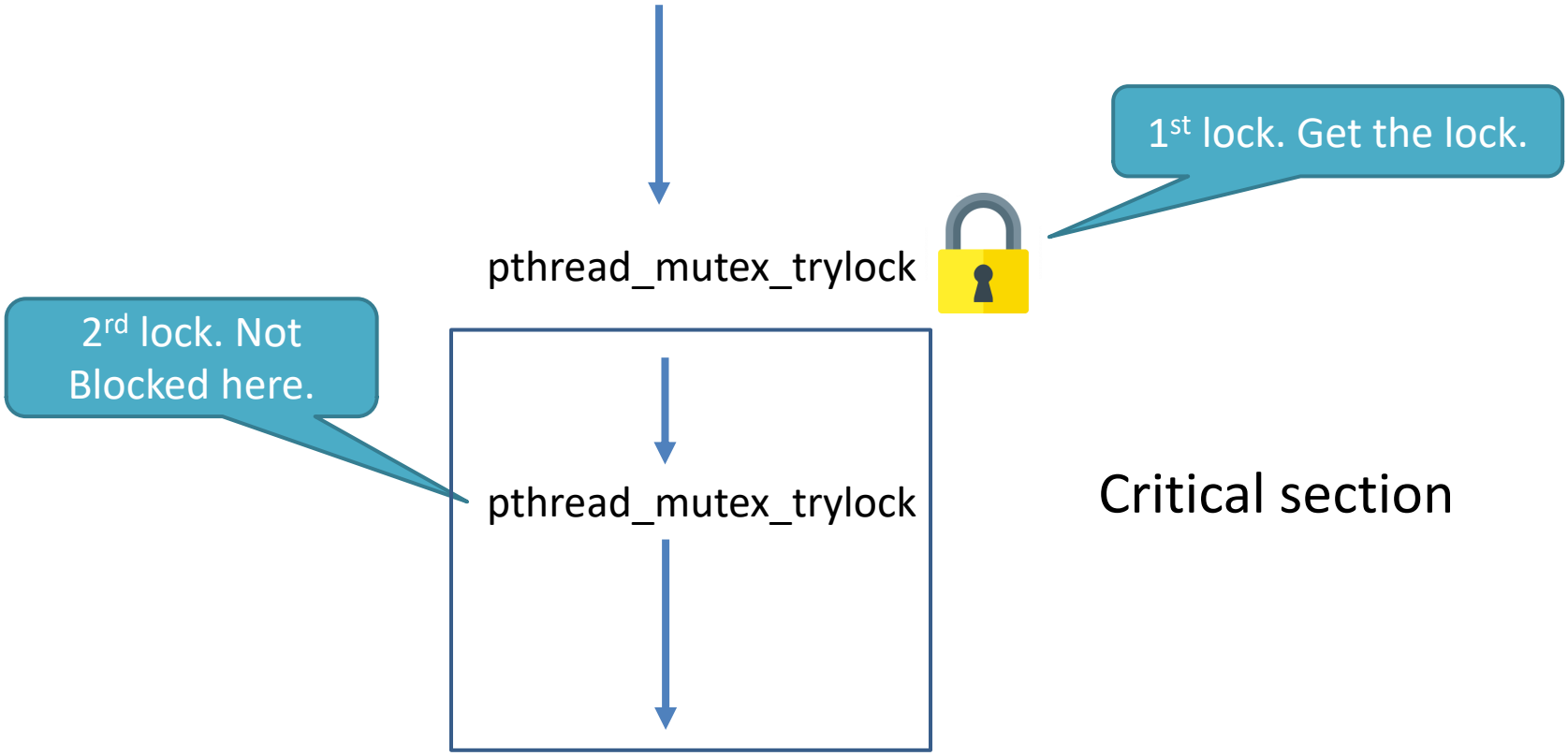
2rd lock. Blocked here.



pthread_mutex_lock vs. pthread_mutex_trylock

https://github.com/kevinsuo/CS7172/blob/master/lock.c

- pthread_mutex_lock (mutex) is a blocking call.



pthread_mutex_lock vs. pthread_mutex_trylock

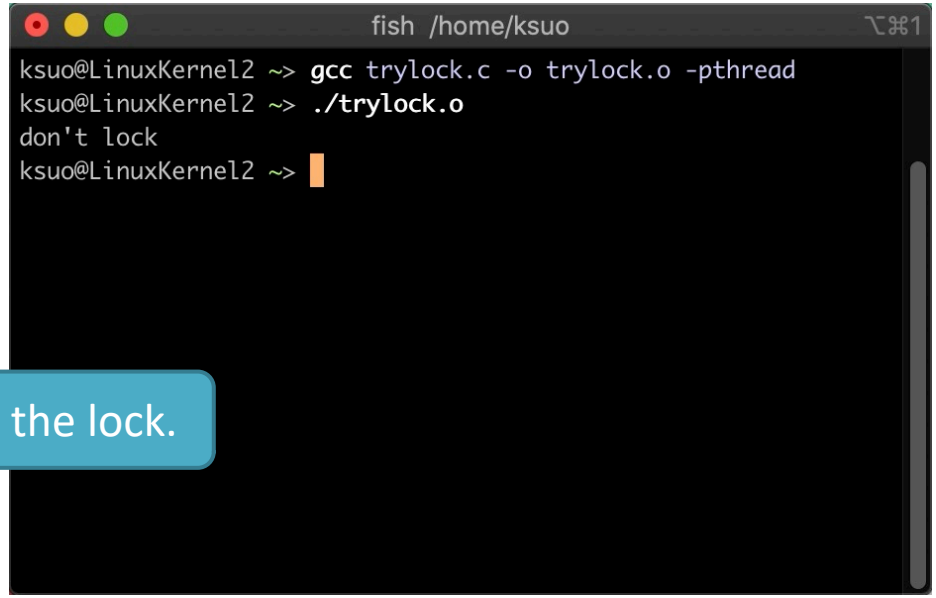
https://github.com/kevinsuo/CS7172/blob/master/trylock.c

- pthread_mutex_trylock (mutex) is a non-blocking call.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int main(void)
{
        int i = 0;
        int res;

        pthread_mutex_t lock;
        pthread_mutex_init(&lock, NULL);

        pthread_mutex_trylock(&lock);
        if((res = pthread_mutex_trylock(&lock) != 0)) {
                printf("don't lock\n");
        } else {
                printf("locked\n");
        }

        return 0;
}
```

1st lock. Get the lock.

2rd lock. Not blocked here.



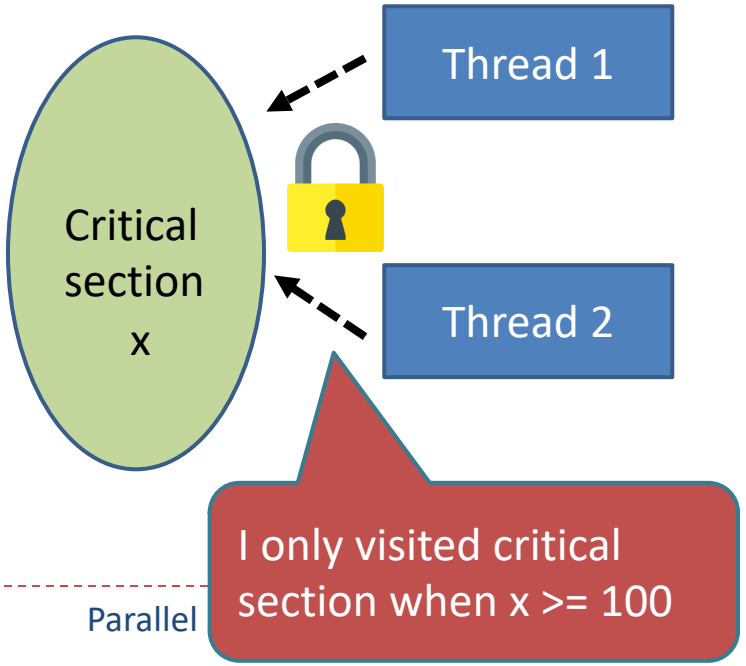
6, Condition variable

- Mutexes: support synchronization by controlling thread access to data (without conditions)

- Condition variables: another way for threads to synchronize (with conditions)

- Example:
  - T1: increase x every time;
  - T2: when x is larger than 99,

  then set x to 0;



6, condition variable example

https://github.com/kevinsuo/CS7172/blob/master/lock_wo_condition.c

- Mutex without condition
  - Example: T1: increase x every time;
  - T2: when x is larger than 99, then set x to 0;

T2 needs to:
lock;
determine;
unlock;
every time to check

```
//thread 1:

while(true)
{
    pthread_mutex_lock(&mutex);
    iCount++;
    pthread_mutex_unlock(&mutex);
}
```

```
//thread 2:
while(true)
{
    pthread_mutex_lock(&mutex);
    if(iCount >= 100)
    {
        iCount = 0;
    }
    pthread_mutex_unlock(&mutex);

}
```

```c
int iCount = 0;
static pthread_mutex_t mlock;

void *thread1_work(void *id) {
        long tid = (long)id;
        while (1) {
                pthread_mutex_lock(&mlock);
                iCount++;
                printf("thread: %ld iCount: %d\n", tid, iCount);
                pthread_mutex_unlock(&mlock);
                sleep(1);
        }
}

void *thread2_work(void *id) {
        long tid = (long)id;
        while (1) {
                pthread_mutex_lock(&mlock);
                if (iCount >= 100)
                        iCount = 0;
                printf("thread: %ld iCount: %d\n", tid, iCount);
                pthread_mutex_unlock(&mlock);
                sleep(1);
        }
}



int main() {
        pthread_t thread1, thread2;
        int id1=1, id2=2;
        if (pthread_mutex_init(&mlock, NULL) != 0) {
                printf("mutex init failed\n");
                return 1;
        }

        pthread_create(&thread1, NULL, thread1_work, (void *)(intptr_t)id1);
        pthread_create(&thread2, NULL, thread2_work, (void *)(intptr_t)id2);

        pthread_exit(NULL);
        pthread_mutex_destroy(&mlock);
        exit(0);
}
```

https://github.com/kevinsuo/CS7172/blob/master/lock_wo_condition.c

```c
//thread 1:

while(true)
{
    pthread_mutex_lock(&mutex);
    iCount++;
    pthread_mutex_unlock(&mutex);
}
```

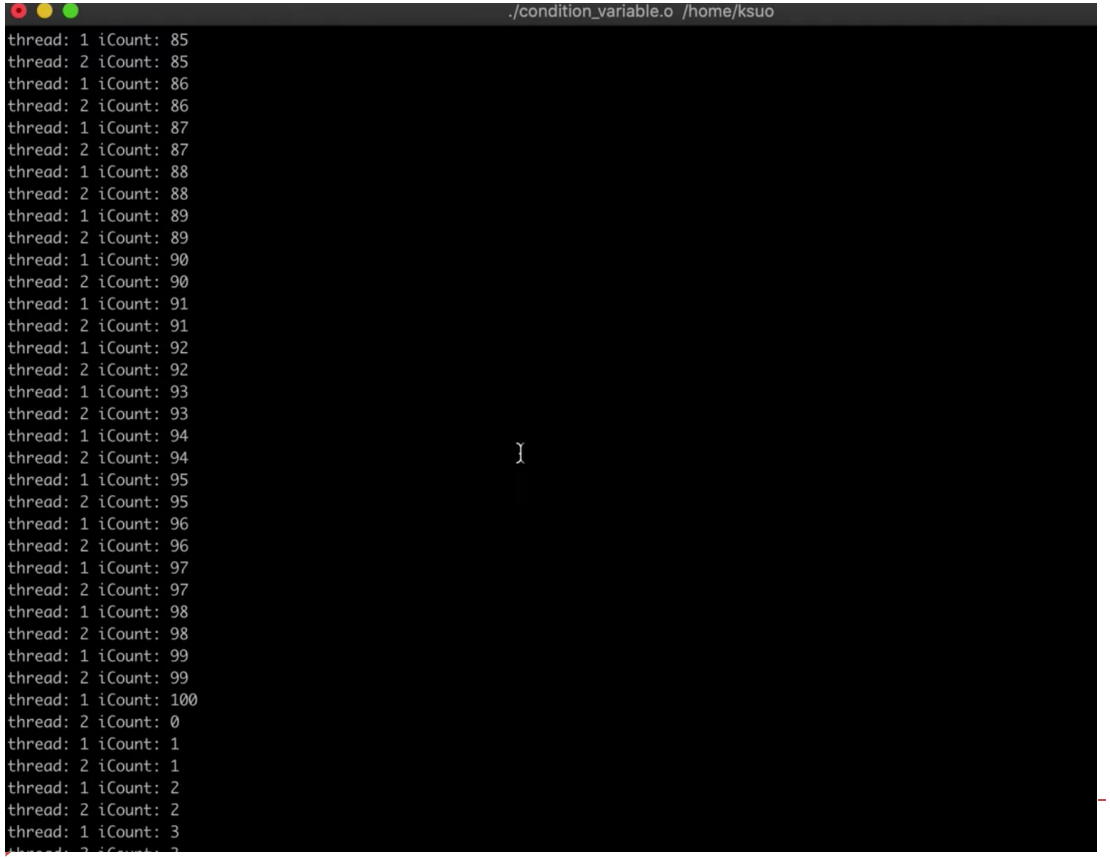
```c
//thread 2:
while(true)
{
    pthread_mutex_lock(&mutex);
    if(iCount >= 100)
    {
        iCount = 0;
    }
    pthread_mutex_unlock(&mutex);

}
```

6, condition variable example

https://github.com/kevinsuo/CS7172/blob/master/lock_wo_condition.c

gcc lock_wo_condition.c -o lock_wo_condition.o -pthread

./lock_wo_condition.o



https://www.youtube.com/watch?v=GADpqSLHcPM

6, condition variable – Waiting and Signaling Con. Variables

pthread_cond_wait (condition, mutex)

pthread_cond_signal (condition)

pthread_cond_broadcast (condition)

1. wait() blocks the calling thread until the specified condition is signaled. It should be called while mutex is locked

2. signal() is used to signal (or wake up) another thread which is waiting on the condition variable. It should be called after mutex is locked, and programmers must unlock mutex in order for wait() routine to complete

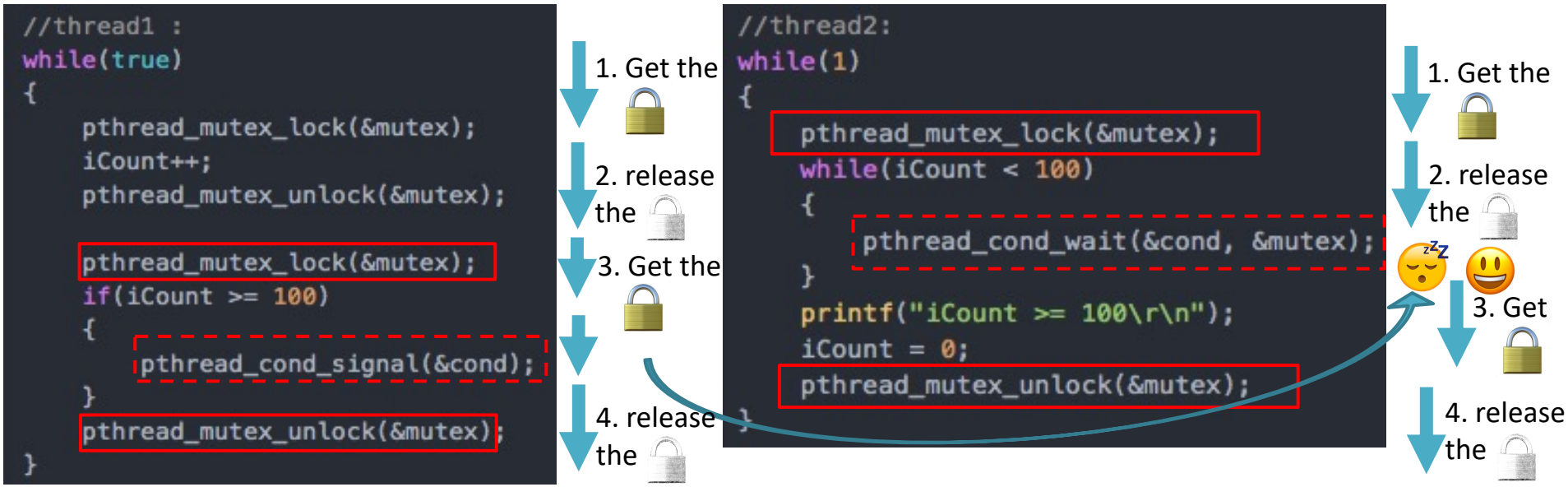
3. broadcast() is useful when multiple threads are in blocked waiting

4. wait() should come before signal() – conditions are not counters, signal would be lost if not waiting

6, condition variable example

https://github.com/kevinsuo/CS7172/blob/master/lock_w_condition.c

- Mutex with condition
  - Example: T1: increase x every time;
  - T2: when x is larger than 99, then set x to 0;



6, condition variable example

https://github.com/kevinsuo/CS7172/blob/master/lock_w_condition.c

```c
int iCount = 0;
static pthread_mutex_t mlock;
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *thread1_work(void *id) {
        long tid = (long)id;
        while (1) {
                pthread_mutex_lock(&mlock);
                iCount++;
                printf("thread: %ld iCount: %d\n", tid, iCount);
                pthread_mutex_unlock(&mlock);

                pthread_mutex_lock(&mlock);
                if (iCount >= 100) {
                        pthread_cond_signal(&cond);
                        printf("thread: %ld iCount: %d\n", tid, iCount);
                }
                pthread_mutex_unlock(&mlock);
                sleep(1);
        }
}

void *thread2_work(void *id) {
        long tid = (long)id;
        while (1) {
                pthread_mutex_lock(&mlock);
                if (iCount < 100) {
                        pthread_cond_wait(&cond, &mlock);
                }
                iCount = 0;
                printf("thread: %ld iCount: %d\n", tid, iCount);
                pthread_mutex_unlock(&mlock);
                sleep(1);
        }
}
```

```c
//thread1 :
while(true)
{
    pthread_mutex_lock(&mutex);
    iCount++;
    pthread_mutex_unlock(&mutex);

    pthread_mutex_lock(&mutex);
    if(iCount >= 100)
    {
        pthread_cond_signal(&cond);
    }
    pthread_mutex_unlock(&mutex);
}
```

```c
//thread2:
while(1)
{
    pthread_mutex_lock(&mutex);
    while(iCount < 100)
    {
        pthread_cond_wait(&cond, &mutex);
    }
    printf("iCount >= 100\r\n");
    iCount = 0;
    pthread_mutex_unlock(&mutex);
}
```

6, condition variable example

https://github.com/kevinsuo/CS7172/blob/master/lock_w_condition.c

```c
int iCount = 0;
static pthread_mutex_t mlock;
static pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

void *thread1_work(void *id) {
        long tid = (long)id;
        while (1) {
                pthread_mutex_lock(&mlock);
                iCount++;
                printf("thread: %ld iCount: %d\n", tid, iCount);
                pthread_mutex_unlock(&mlock);

                pthread_mutex_lock(&mlock);
                if (iCount >= 100) {
                        pthread_cond_signal(&cond);
                        printf("thread: %ld iCount: %d\n", tid, iCount);
                }
                pthread_mutex_unlock(&mlock);
                sleep(1);
        }
}

void *thread2_work(void *id) {
        long tid = (long)id;
        while (1) {
                pthread_mutex_lock(&mlock);
                if (iCount < 100) {
                        pthread_cond_wait(&cond, &mlock);
                }
                iCount = 0;
                printf("thread: %ld iCount: %d\n", tid, iCount);
                pthread_mutex_unlock(&mlock);
                sleep(1);
        }
}
```

gcc lock_w_condition.c -o lock_w_condition.o –pthread

./lock_w_condition.o

6, condition variable example

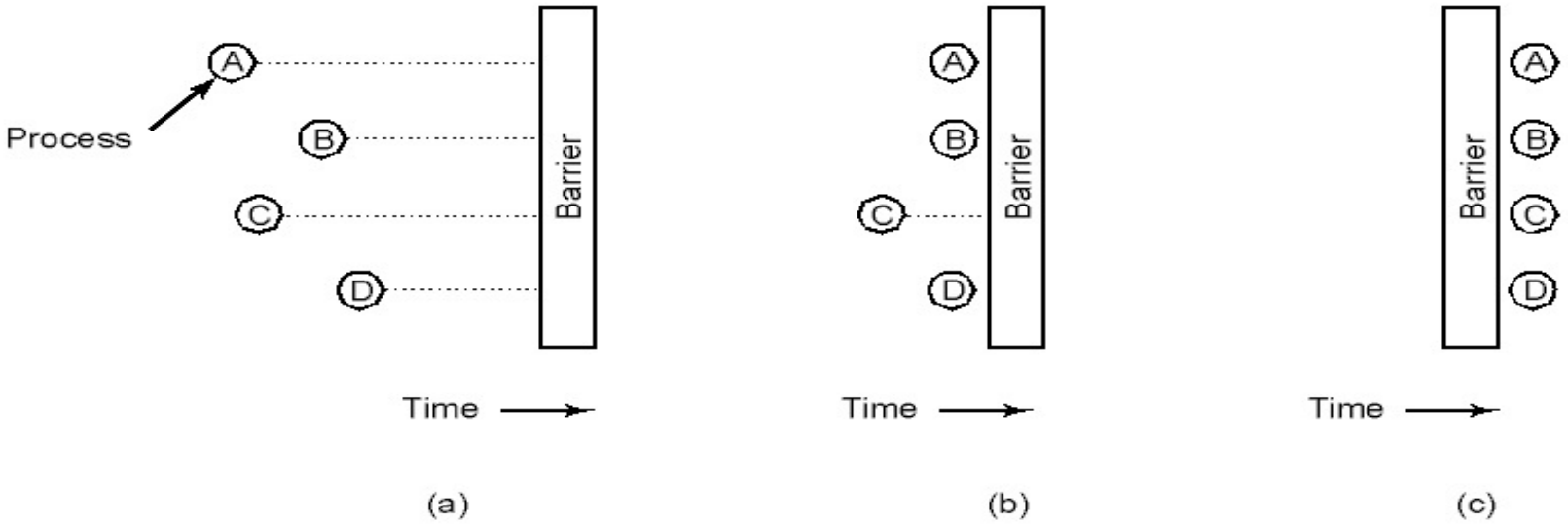
https://github.com/kevinsuo/CS7172/blob/master/lock_w_condition.c

gcc lock_w_condition.c -o lock_w_condition.o –pthread

./lock_w_condition.o

https://www.youtube.com/watch?v=81CFeuKRscw

7, barriers



- Use of a barrier (for programs operate in *phases,* neither enters the next phase until all are finished with the current phase) for groups of processes to do synchronization
  - (a) processes approaching a barrier
  - (b) all processes but one blocked at barrier
  - (c) last process arrives, all are let through

7, barriers – Initializing and waiting on barriers

pthread_barrier_init (barrier, attr, count)

pthread_barrier_wait (barrier)

pthread_barrier_destroy (barrier)

1. init() function shall allocate any resources required to use the barrier referenced by **barrier** and shall initialize the barrier with **attr**. If attr is NULL, default settings will be applied. The **count** argument specifies the number of threads that must call wait() before any of them successfully return from the call.

2. wait() function shall synchronize participating threads at the barrier.

3. destroy() function shall destroy the barrier and release any resources used by the barrier.

7, without barriers example

https://github.com/kevinsuo/CS7172/blob/master/pthread_barrier.c

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

pthread_barrier_t b;

void* task(void* param) {
    long id = (long) param;
    printf("before the barrier %ld\n", id);
//    pthread_barrier_wait(&b);
    printf("after the barrier %ld\n", id);
}

int main() {
    int nThread = 5;
    int i;

    pthread_t thread[nThread];
//    pthread_barrier_init(&b, 0, nThread);
    for (i = 0; i < nThread; i++)
        pthread_create(&thread[i], 0, task, (void*)(intptr_t)i);
    for (i = 0; i < nThread; i++)
        pthread_join(thread[i], 0);
//    pthread_barrier_destroy(&b);
    return 0;
}
```

Some “after” appears ahead of “before”

```
ksuo@LinuxKern      ~> ./pthread_barrier.o
before the barrier 0
before the barrier 4
after the barrier 4
before the barrier 1
after the barrier 1
before the barrier 2
after the barrier 0
before the barrier 3
after the barrier 3
after the barrier 2
```

7, barriers example

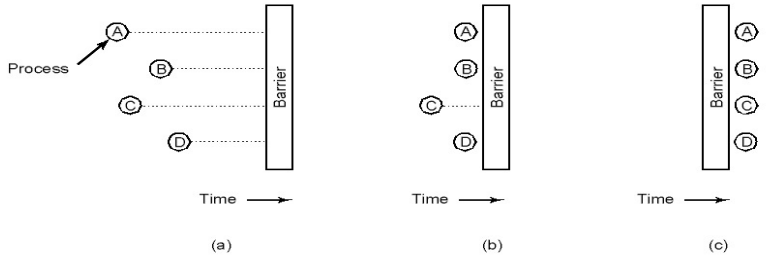
https://github.com/kevinsuo/CS7172/blob/master/pthread_barrier.c

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

pthread_barrier_t b;

void* task(void* param) {
    long id = (long) param;
    printf("before the barrier %ld\n", id);
    pthread_barrier_wait(&b);
    printf("after the barrier %ld\n", id);
}

int main() {
    int nThread = 5;
    int i;

    pthread_t thread[nThread];
    pthread_barrier_init(&b, 0, nThread);
    for (i = 0; i < nThread; i++)
        pthread_create(&thread[i], 0, task, (void*)(intptr_t)i);
    for (i = 0; i < nThread; i++)
        pthread_join(thread[i], 0);
    pthread_barrier_destroy(&b);
    return 0;
}
```

All threads are waiting here.

Create barrier

Destroy barrier




```
pi@raspberrypi ~/Downloads> ./test.o
before the barrier 1
before the barrier 0
before the barrier 2
before the barrier 3
before the barrier 4
after the barrier 0
after the barrier 2
after the barrier 3
after the barrier 1
after the barrier 4
```

Question

https://github.com/kevinsuo/CS7172/blob/master/pthread_barrier_test.c

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void* task(void* param) {
        long id = (long) param;
        printf("step 1   %ld\n", id);
        printf("step 2   %ld\n", id);
        printf("step 3   %ld\n", id);
}

int main() {
        int nThread = 5;
        int i;

        pthread_t thread[nThread];
        printf("step Num ID\n");

        for (i = 0; i < nThread; i++)
                pthread_create(&thread[i], 0, task, (void *)(intptr_t)i);
        for (i = 0; i < nThread; i++)
                pthread_join(thread[i], 0);

        return 0;
}
```

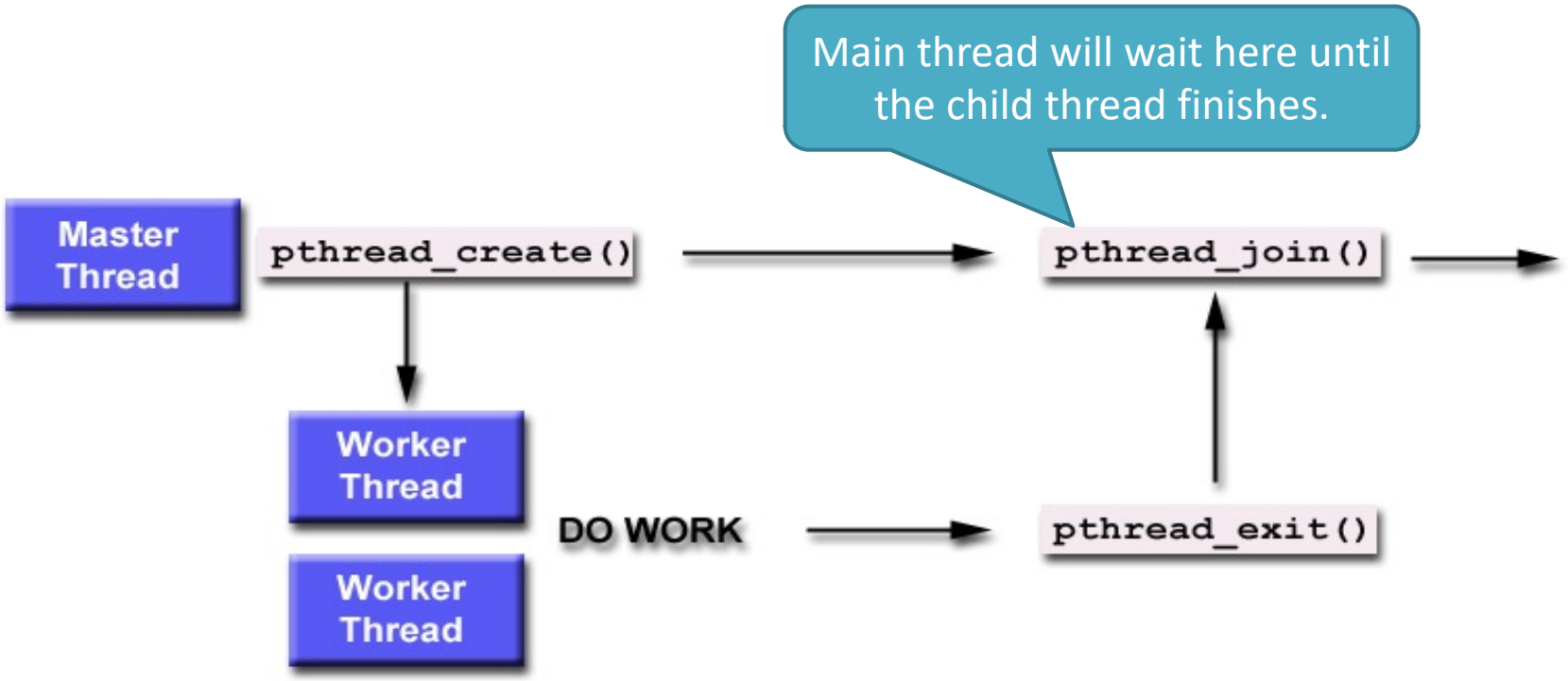

```
ksuo@LinuxKernel2 ~> ./pthread_barrier.o
step Num ID
step 1   1
step 2   1
step 3   1
step 1   0
step 2   0
step 3   0
step 1   2
step 2   2
step 3   2
step 1   3
step 2   3
step 3   3
step 1   4
step 2   4
step 3   4
```

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

pthread_barrier_t b;
pthread_barrier_t c;

void* task(void* param) {
        long id = (long) param;
        printf("step 1 %ld\n", id);
        pthread_barrier_wait(&b);
        printf("step 2 %ld\n", id);
        pthread_barrier_wait(&c);
        printf("step 3 %ld\n", id);
}

int main() {
        int nThread = 5;
        int i;

        pthread_t thread[nThread];
        pthread_barrier_init(&b, 0, nThread);
        pthread_barrier_init(&c, 0, nThread);

        for (i = 0; i < nThread; i++)
                pthread_create(&thread[i], 0, task, (void *)(intptr_t)i);
        for (i = 0; i < nThread; i++)
                pthread_join(thread[i], 0);

        pthread_barrier_destroy(&b);
        pthread_barrier_destroy(&c);

        return 0;
}
```


```
ksuo@LinuxKernel2 ~> ./pthread_barrier.o
step 1 0
step 1 2
step 1 4
step 1 3
step 1 1
step 2 1
step 2 2
step 2 4
step 2 0
step 2 3
step 3 2
step 3 0
step 3 4
step 3 3
step 3 1
```

Question

- Can you use pthread_barriers to replace the pthread_join?



https://github.com/kevinsuo/CS7172/blob/master/pthread_join.c

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int count = 0;

void *myThread1(void)
{
    int i;
    for (i=0; i<3; i++)
    {
        printf("This is the 1st pthread.\n");
        count++;
        sleep(1);
    }
}


int main()
{
    int ret=0;
    pthread_t id1;

    printf("This is main thread!\n");

    ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
    if (ret)
    {
        printf("Create pthread error!/n");
        return 1;
    }

    pthread_join(id1, (void *)ret);

    printf("count:%d\n", count);
    pthread_exit(NULL);

    return 0;
}
```

https://github.com/kevinsuo/CS7172/blob/master/pthread_join.c

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int count = 0;

void *myThread1(void)
{
    int i;
    for (i=0; i<3; i++)
    {
        printf("This is the 1st pthread.\n");
        count++;
        sleep(1);
    }
}


int main()
{
    int ret=0;
    pthread_t id1;

    printf("This is main thread!\n");

    ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
    if (ret)
    {
        printf("Create pthread error!/n");
        return 1;
    }

    pthread_join(id1, (void *)ret);

    printf("count:%d\n", count);
    pthread_exit(NULL);

    return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

int count = 0;
pthread_barrier_t b;

void *myThread1(void)
{
    int i;
    for (i=0; i<3; i++)
    {
        printf("This is the 1st pthread.\n");
        count++;
        sleep(1);
    }
    pthread_barrier_wait(&b);
}


int main()
{
    int ret=0;
    pthread_t id1;
    pthread_barrier_init(&b, 0, 2);  //2 threads here

    printf("This is main thread!\n");

    ret = pthread_create(&id1, NULL, (void*)myThread1, NULL);
    if (ret)
    {
        printf("Create pthread error!/n");
        return 1;
    }
//    pthread_join(id1, (void *)ret);
    pthread_barrier_wait(&b);

    printf("count:%d\n", count);
    pthread_barrier_destroy(&b);
    pthread_exit(NULL);

    return 0;
}
```

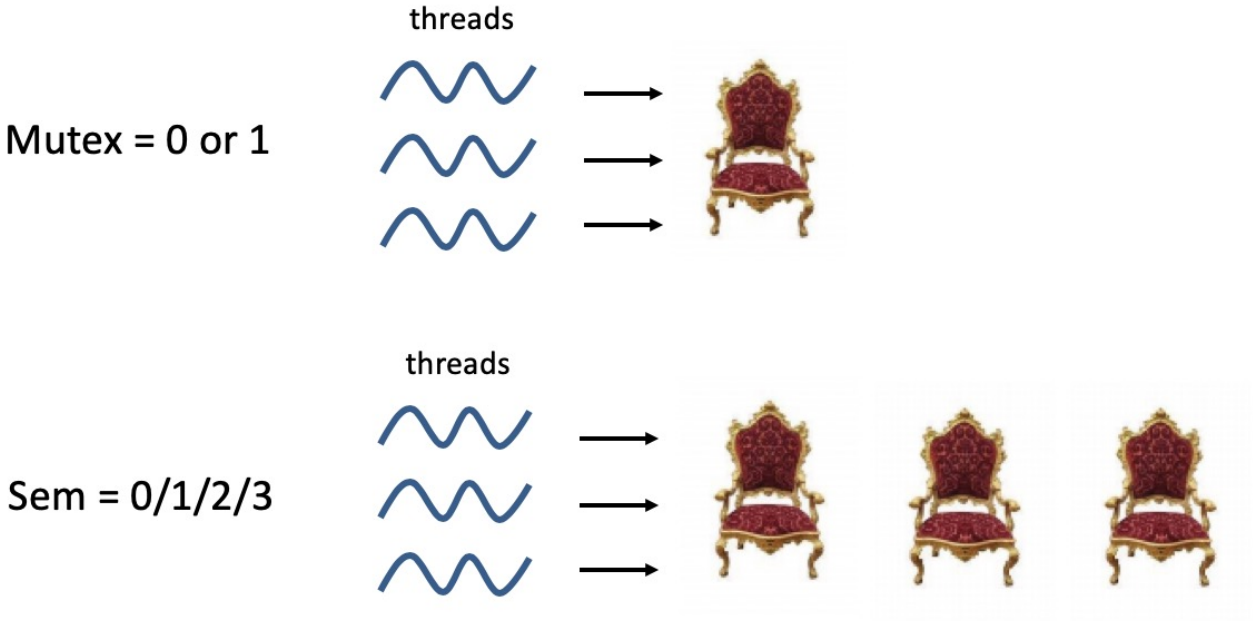
Question

- Can you use pthread_barriers to replace the pthread_join?

```
ksuo@LinuxKernel2 ~> ./pthread_join_2.o
This is main thread!
This is the 1st pthread.
This is the 1st pthread.
This is the 1st pthread.
count:3
```

8, semaphore

- Semaphores are **counters** for resources shared between threads. The basic operations on semaphores are: **increment** the counter atomically, and wait until the counter is non-null and **decrement** it atomically.



Semaphore

- Down operation (P; request):
  - Checks if a semaphore is > 0, sem--
    - Request one unit resource and one process enters

- Up operation (V; release)
  - sem++
    - Release one unit resource and one process leaves

8, semaphore

```
sem_init(&remain, 0, 3);
sem_init(&apple, 0, 0);
sem_init(&pear, 0, 0);
sem_init(&mutex, 0, 1);
```

int sem_init (sem_t *sem, int pshared, unsigned int value)
//Init a sem. The **pshared** argument indicates whether this semaphore is to be shared between the threads of a process (pshared=0), or between processes (pshared≠0). The **value** argument specifies the initial value for the semaphore (how many resources).

int sem_destroy (sem_t * sem)
//Destroy a sem

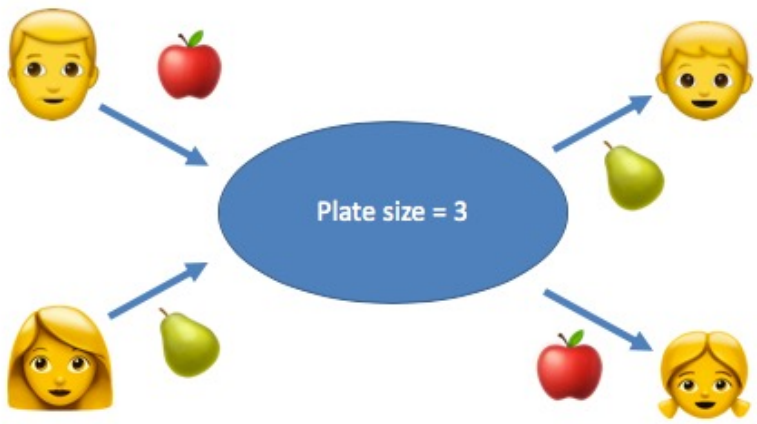
int sem_wait (sem_t * sem)
//Request one unit resource, blocking call. If successful, sem--.

int sem_trywait (sem_t * sem)
//Request one unit resource, non-blocking call. If successful, sem--.

int sem_post (sem_t * sem)
//Release one unit resource. If successful, sem++.

Semaphore example

https://github.com/kevinsuo/CS7172/blob/master/semaphore.c



- Semaphore:
  - Son: whether there is pear, s1
  - Daughter: whether there is apple, s2
  - Father/Mother: whether there is space, s3

Father thread:
peel apple
P(s3)
put apple
V(s2)

Mother thread:
peel pear
P(s3)
put apple
V(s1)

Son thread:
P(s1)
get pear
V(s3)
eat pear

Daughter thread:
P(s2)
get apple
V(s3)
eat apple

Semaphore example

https://github.com/kevinsuo/CS7172/blob/master/semaphore.c

- Semaphore:
  - Son: whether there is pear, s1
  - Daughter: whether there is apple, s2
  - Father/Mother: whether there is space, s3

Father thread:
peel apple
P(s3)
put apple
V(s2)

Daughter thread:
P(s2)
get apple
V(s3)
eat apple

```
void *father(void *arg) {
        while(1) {
                sleep(5);  //simulate peel apple
P(s3)           sem_wait(&remain);
                sem_wait(&mutex);
                nremain--;
                napple++;
                sem_post(&mutex);
V(s2)           sem_post(&apple);
        }
}
```

```
void *daughter(void *arg) {
        while(1) {
P(s2)           sem_wait(&apple);
                sem_wait(&mutex);
                nremain++;
                napple--;
                sem_post(&mutex);
V(s3)           sem_post(&remain);
                sleep(10); //simulate eat apple
        }
}
```

Semaphore example

https://github.com/kevinsuo/CS7172/blob/master/semaphore.c

- Semaphore:
  - Son: whether there is pear, s1
  - Daughter: whether there is apple, s2
  - Father/Mother: whether there is space, s3

Mother thread:
peel pear
P(s3)
put apple
V(s1)

Son thread:
P(s1)
get pear
V(s3)
eat pear

```
void *mother(void *arg) {
    while(1) {
        sleep(7);   //simulate peel pear
P(s3)   sem_wait(&remain);
        sem_wait(&mutex);
        nremain--;
        npear++;
        sem_post(&mutex);
V(s1)   sem_post(&pear);
    }
}
```

```
void *son(void *arg) {
    while(1) {
P(s1)   sem_wait(&pear);
        sem_wait(&mutex);
        nremain++;
        npear--;
        sem_post(&mutex);
V(s3)   sem_post(&remain);
        sleep(10); //simulate eat pear
    }
}
```

- Semaphore:
  - Son: whether there is pear, s1
  - Daughter: whether there is apple, s2
  - Father/Mother: whether there is space, s3

https://github.com/kevinsuo/CS7172/blob/master/semaphore.c

```
void *father(void *arg) {
        while(1) {
                sleep(5);  //simulate peel apple
                sem_wait(&remain);
                sem_wait(&mutex);
                nremain--;
                napple++;
                sem_post(&mutex);
                sem_post(&apple);
        }
}
```

```
void *daughter(void *arg) {
        while(1) {
                sem_wait(&apple);
                sem_wait(&mutex);
                nremain++;
                napple--;
                sem_post(&mutex);
                sem_post(&remain);
                sleep(10); //simulate eat apple
        }
}
```

```
void *mother(void *arg) {
    while(1) {
        sleep(7);   //simulate peel pear
        sem_wait(&remain);
        sem_wait(&mutex);
        nremain--;
        npear++;
        sem_post(&mutex);
        sem_post(&pear);
    }
}
```

remain is semaphore
nremain is an integer

```
void *son(void *arg) {
    while(1) {
        sem_wait(&pear);
        sem_wait(&mutex);
        nremain++;
        npear--;
        sem_post(&mutex);
        sem_post(&remain);
        sleep(10); //simulate eat pear
    }
}
```

Semaphore example

https://github.com/kevinsuo/CS7172/blob/master/semaphore.c

```c
sem_t remain, apple, pear, mutex;
static unsigned int nremain = 3, napple = 0, npear = 0;
```

```c
int main() {
    pthread_t fa, ma, so, da;
    setlocale(LC_ALL, "en_US.utf8");

    sem_init(&remain, 0, 3);
    sem_init(&apple, 0, 0);
    sem_init(&pear, 0, 0);
    sem_init(&mutex, 0, 1);

    pthread_create(&fa, NULL, &father, NULL);
    pthread_create(&ma, NULL, &mather, NULL);
    pthread_create(&so, NULL, &son, NULL);
    pthread_create(&da, NULL, &daughter, NULL);

    for(;;);
}
```

```c
void *mother(void *arg) {
    while(1) {
        sleep(7);   //simulate peel pear
        sem_wait(&remain);
        sem_wait(&mutex);
        nremain--;
        npear++;
        sem_post(&mutex);
        sem_post(&pear);
    }
}
```

```c
void *son(void *arg) {
    while(1) {
        sem_wait(&pear);
        sem_wait(&mutex);
        nremain++;
        npear--;
        sem_post(&mutex);
        sem_post(&remain);
        sleep(10); //simulate eat pear
    }
}
```

Semaphore example

https://github.com/kevinsuo/CS7172/blob/master/semaphore.c

```
pi@raspberrypi ~/Downloads> ./semaphore.o
father 👨 before put apple, remain=3, apple🍎=0, pear🍐=0
father 👨 after  put apple, remain=2, apple🍎=1, pear🍐=0

daughter👧 before eat apple, remain=2, apple🍎=1, pear🍐=0
daughter👧 after  eat apple, remain=3, apple🍎=0, pear🍐=0

mother 👩 before put pear , remain=3, apple🍎=0, pear🍐=0
mother 👩 after  put pear , remain=2, apple🍎=0, pear🍐=1

son    👦 before eat pear , remain=2, apple🍎=0, pear🍐=1
son    👦 after  eat pear , remain=3, apple🍎=0, pear🍐=0

father 👨 before put apple, remain=3, apple🍎=0, pear🍐=0
father 👨 after  put apple, remain=2, apple🍎=1, pear🍐=0

mother 👩 before put pear , remain=2, apple🍎=1, pear🍐=0
mother 👩 after  put pear , remain=1, apple🍎=1, pear🍐=1

daughter👧 before eat apple, remain=1, apple🍎=1, pear🍐=1
daughter👧 after  eat apple, remain=2, apple🍎=0, pear🍐=1

father 👨 before put apple, remain=2, apple🍎=0, pear🍐=1
father 👨 after  put apple, remain=1, apple🍎=1, pear🍐=1

son    👦 before eat pear , remain=1, apple🍎=1, pear🍐=1
son    👦 after  eat pear , remain=2, apple🍎=1, pear🍐=0

mother 👩 before put pear , remain=2, apple🍎=1, pear🍐=0
mother 👩 after  put pear , remain=1, apple🍎=1, pear🍐=1

father 👨 before put apple, remain=1, apple🍎=1, pear🍐=1
father 👨 after  put apple, remain=0, apple🍎=2, pear🍐=1

daughter👧 before eat apple, remain=0, apple🍎=2, pear🍐=1
daughter👧 after  eat apple, remain=1, apple🍎=1, pear🍐=1
```

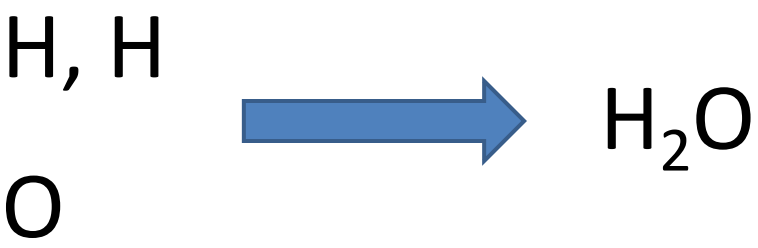
gcc -pthread semaphore.c -o semaphore.o

https://youtu.be/ZIWwvcuROME

Conclusion

- What is pthread?
  - Thread model
  - Pthread overview
- Pthread management
  - Creation and termination
  - Identification
  - Join and detach
- Pthread data sharing
  - Mutex
  - Condition variable
  - Barrier
  - Semaphore

Semaphore example



Semaphore h = 0;
Semaphore o = 0;

```
//executed when H is available
hReady()
{
   //h++
   V(h);
   //request O; if succeed, O--; otherwise, sleep&wait
   P(o);
   return;
}
```

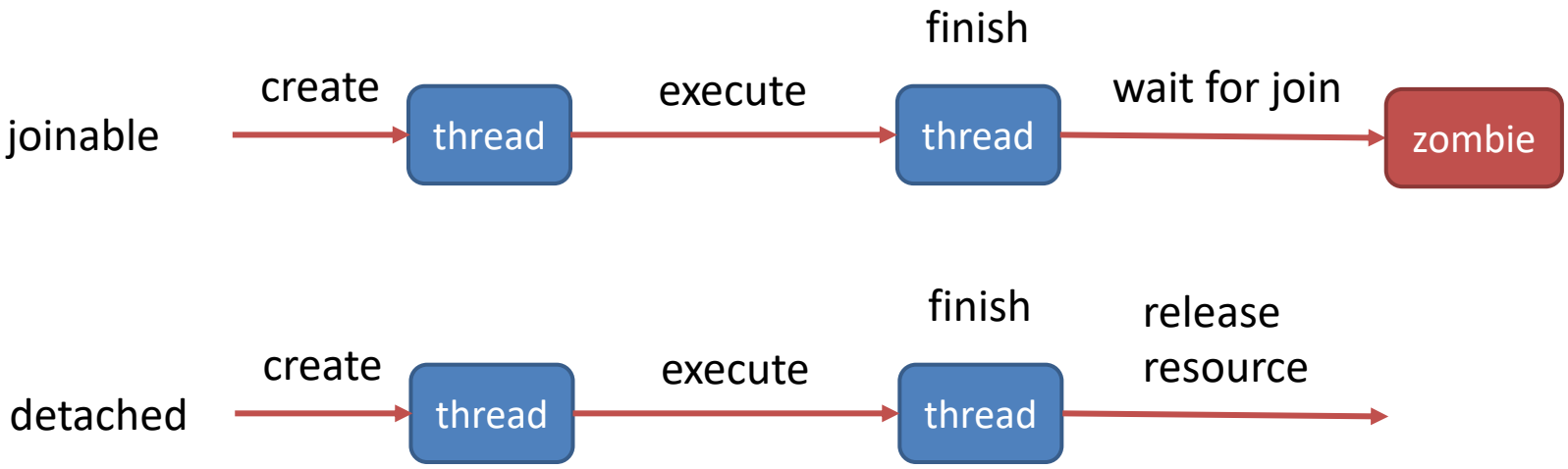
```
//executed when O is available
oReady()
{
   //o++
   V(o);
   //request h; if succeed, h--; otherwise, sleep&wait
   P(h);
   P(h);
   //make H2O
   makeWater();
   return;
}
```

Compiling Pthreads Programs

Platform	Compiler Command	Description
IBM AIX	xlc_r / cc_r	C (ANSI / non-ANSI)
	xlC_r	C++
	xlf_r -qnosave xlf90_r -qnosave	Fortran - using IBM's Pthreads API (non-portable)
INTEL LINUX	icc -pthread	C
	icpc -pthread	C++
COMPAQ Tru64	cc -pthread	C
	cxx -pthread	C++
All Above Platforms	gcc -lpthread/pthread	GNU C
	g++ -lpthread/pthread	GNU C++
	guidec -pthread	KAI C (if installed)
	KCC -pthread	KAI C++ (if installed)

4, detach

- The default state of a thread is joinable. If a thread finishes but is not joined, its state is similar to the zombie process, which is, some resources are not recycled.



4, detach example

https://github.com/kevinsuo/CS7172/blob/master/pthread_detach.c

Resources will not be released after execution.

```c
void *myThread(void)
{
//    pthread_detach(pthread_self());
    printf("This is child thread tid %u!\n", (unsigned int)pthread_self());
}
```

```c
int main()
{
        int ret=0;
        pthread_t id1, id2;

        printf("This is main thread pid %u!\n", (unsigned int)getpid());

        ret = pthread_create(&id1, NULL, (void*)myThread, NULL);
        ret = pthread_create(&id2, NULL, (void*)myThread, NULL);
        if (ret)
        {
                printf("Create pthread error!/n");
                return 1;
        }

        pthread_join(id1, (void *)ret);
        pthread_join(id2, (void *)ret);

        pthread_exit(NULL);

        return 0;
}
```

```
pi@raspberrypi ~/Downloads> ./test.o
This is main thread pid 2356!
This is child thread tid 1994404976!
This is child thread tid 1986012272!
```

4, detach example

https://github.com/kevinsuo/CS7172/blob/master/pthread_detach.c

Set itself not joinable
Resources will be released.

```c
void *myThread(void)
{
        pthread_detach(pthread_self());
        printf("This is child thread tid %u!\n", (unsigned int)pthread_self());
}
```

```c
int main()
{
        int ret=0;
        pthread_t id1, id2;

        printf("This is main thread pid %u!\n", (unsigned int)getpid());

        ret = pthread_create(&id1, NULL, (void*)myThread, NULL);
        ret = pthread_create(&id2, NULL, (void*)myThread, NULL);
        if (ret)
        {
                printf("Create pthread error!/n");
                return 1;
        }

        pthread_join(id1, (void *)ret);
        pthread_join(id2, (void *)ret);

        pthread_exit(NULL);

        return 0;
}
```

```
pi@raspberrypi ~/Downloads> ./test.o
This is main thread pid 2356!
This is child thread tid 1994404976!
This is child thread tid 1986012272!
```

4, mutex

- A typical sequence in the use of a mutex
  1. Create and initialize a mutex variable
  2. Several threads attempt to lock the mutex
     - Only one succeeds and that thread owns the mutex
     - The owner thread performs some set of actions
  3. The owner unlocks the mutex
  4. Another thread acquires the mutex and repeats the process
  5. Finally the mutex is destroyed

```
int counter = 0;
static pthread_mutex_t mlock;   1

void *compute()
{
        int i = 0;
    2   pthread_mutex_lock(&mlock);
        while (i < 10000) {
                counter = counter + 1;
                i++;
        }
    3   pthread_mutex_unlock(&mlock);
        printf("Counter value: %d\n", counter);
}
```

```
pthread_t thread1, thread2, thread3, thread4, thread5;

pthread_create(&thread1, NULL, compute, (void *)&thread1);
pthread_create(&thread2, NULL, compute, (void *)&thread2);
pthread_create(&thread3, NULL, compute, (void *)&thread3);
pthread_create(&thread4, NULL, compute, (void *)&thread4);
pthread_create(&thread5, NULL, compute, (void *)&thread5);
```