CS 3502
Operating Systems

Midterm Review

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

Topics

- OS Overview
- Process
- Thread
- System call
- IPC
- Interrupt
- Lock
- Pthread
- Scheduling

Topics

- OS Overview
- Process
- Thread

Basic concepts

- System call
- IPC
- Interrupt
- Lock
- Pthread

Communication

- App <-> OS
- Process <-> Process
- Outside <-> OS
- Thread <-> Thread
- Implementation

- Scheduling

Execution Order

Lec2 overview

- What is an OS?
  - Virtualization
  - Concurrency
  - Persistence
- History of OS
  - Monolithic
  - Microkernel
  - Hybrid

Lec3 process

- What is process?
  - Process vs Program
  - Linux Process Control Block
- Process related System calls
  - Fork
  - Exec
  - Wait
- Orphan and Zombie process

Lec4 thread

- What is thread?
  - Multiple thread application
  - Thread vs Process
  - Advantage and disadvantage of thread
- Thread in Linux
- Thread design
  - Kernel space vs User space
  - Local thread vs Global thread scheduling

Lec5 system call

- What is system call?
  - Kernel space vs user space
  - System call vs library call
  - What service can system call provide?
  - System call naming, input, output
- How to design a system call
  - Example
  - Project 1

Not included

Lec6 IPC

- Inter-process communication
  - What is IPC
  - Why we need IPC?
- Main types of IPC
  - Pipe
  - Shared memory
  - Signal
  - Semaphore
  - Message queue
  - Socket

Lec7 interrupt

- What is interrupt?
  - Interrupt vs Polling
  - Advanced programmable interrupt controller
  - Interrupt processing
- Interrupt types and affinity
  - Hardware and software interrupt
  - Interrupt affinity

Lec8/9 lock and pthread

- Concurrency and synchronization
  - Execution models
  - Race condition
  - Critical section
- Mutual exclusion
  - Spinlock
  - Mutex lock
  - Semaphore
  - Deadlock and priority inversion
- Pthread implementation (Code)

Lec10 scheduling

- Introduction to CPU scheduling
  - What is CPU scheduling
  - Why we need CPU scheduling
  - When scheduling happens
- Scheduling policies
  - FCFS, SJF, RR, Priority
  - Scheduling on multiple CPUs
  - Scheduling in Linux

Midterm Format

- 7 short answer questions
  - 6 points for each
  - Totally 42 points

Same as HW1

- 4 code reading questions
  - Totally 58 points
  - Read the code, select the correct answer and explain why

Part 1: Short answer question example

- What is system call used for?

Answer:

- A system call is a way for programs to interact with the operating system
- A computer program makes a system call when it makes a request to the operating system's kernel
- System call provides the services of the operating system to the user programs via Application Program Interface (API)

Part 1: Short answer question example

- List three different ways to avoid race conditions.

Answer:

- Spin lock (busy waiting lock)
- Mutex lock (sleep and wakeup lock)
- Semaphore
- …

Part 1: Short answer question coverage

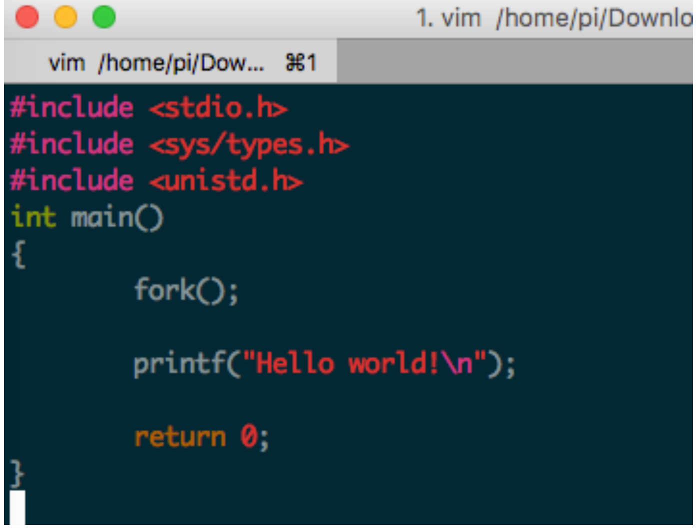
- Lec2: OS Overview
- Lec3: Process
- Lec4: Thread
- Lec5: System call
- Lec6: IPC
- Lec7: Interrupt
- Lec8: Lock
- Lec9: Pthread
- Lec10: Scheduling

Part 2: Code reading question coverage

- Lec3/4: Process & thread
- Lec8/9: Lock & pthread
- Lec8/9: Semaphore & pthread
- Lec10: Scheduling (not code, calculation for turnaround time and response time)

Part 2: Code reading question example

- how many times will the message “Hello world!” be displayed? And explain.

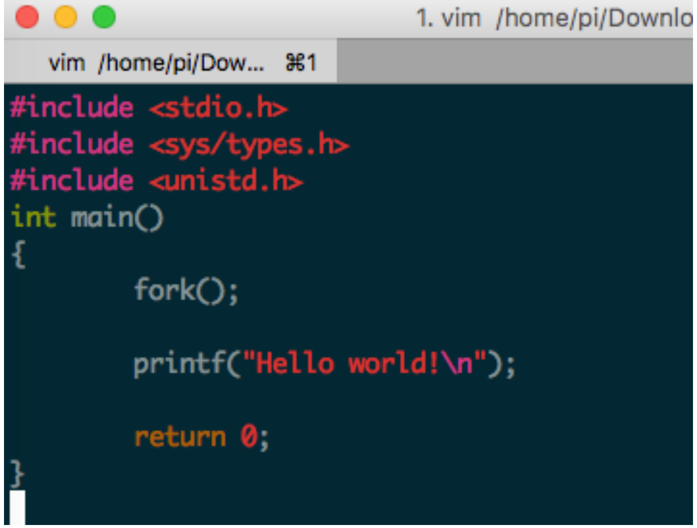


```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();

        printf("Hello world!\n");

        return 0;
}
```

a) 1
b) 2
c) 3
d) 4
e) None of the above

Part 2: Code reading question example

- how many times will the message “Hello world!” be displayed? And explain.



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();

        printf("Hello world!\n");

        return 0;
}
```

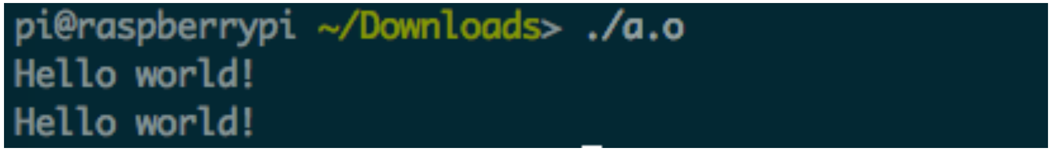
a) 1

b) 2

c) 3

d) 4

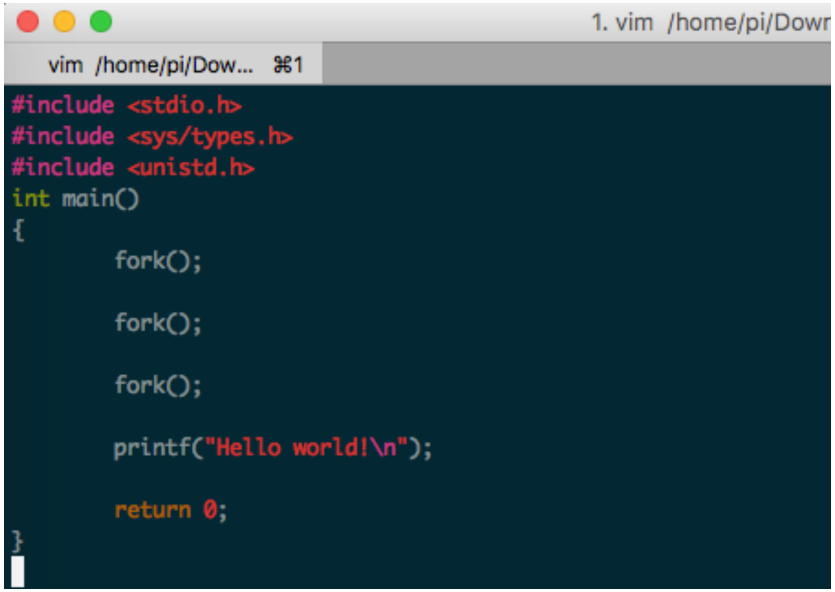
e) None of the above



```
pi@raspberrypi ~/Downloads> ./a.o
Hello world!
Hello world!
```

Part 2: Code reading question example

- how many times will the message “Hello world!” be displayed? And explain.

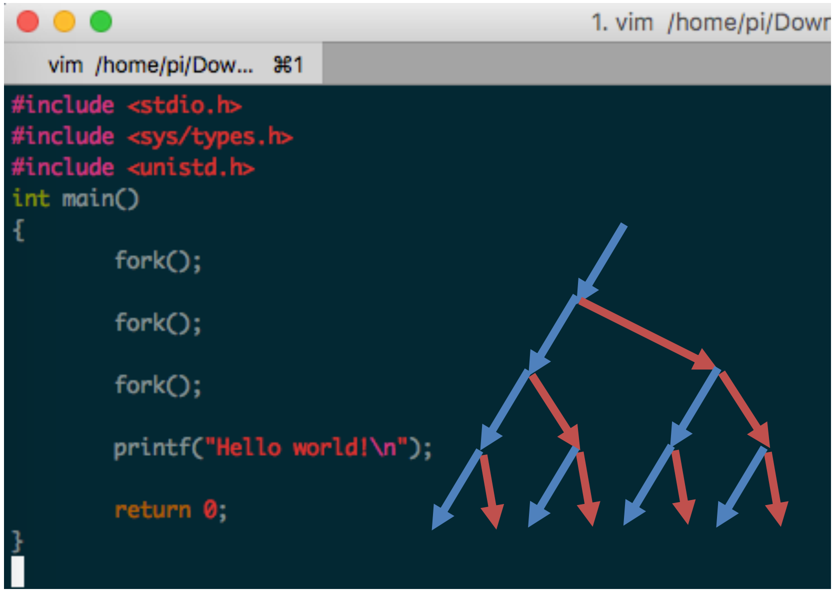


```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();

        fork();

        fork();

        printf("Hello world!\n");

        return 0;
}
```

a) 3
b) 6
c) 8
d) 10
e) None of the above

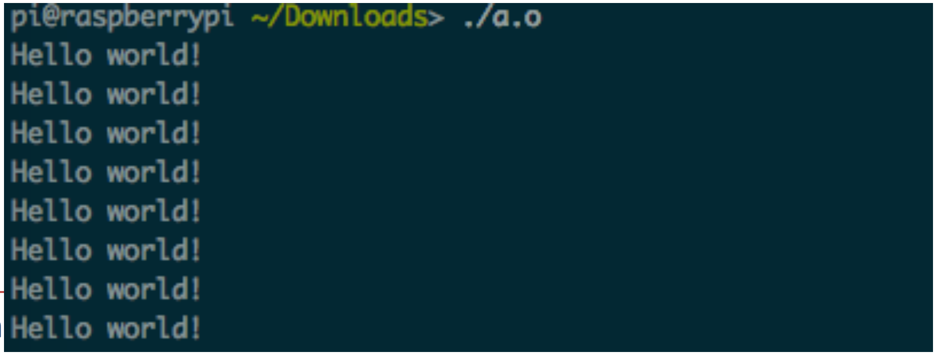
Part 2: Code reading question example

- how many times will the message “Hello world!” be displayed? And explain.



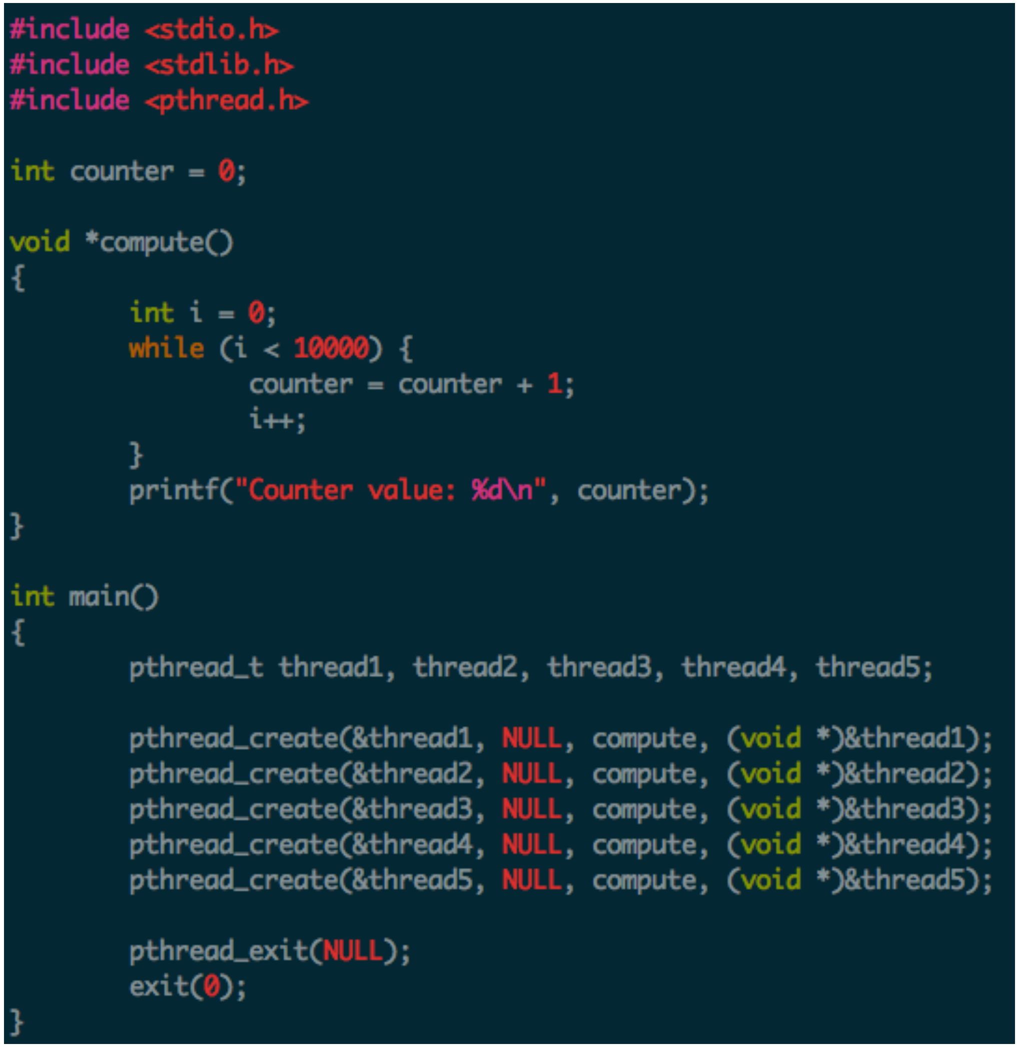
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        fork();

        fork();

        fork();

        printf("Hello world!\n");

        return 0;
}
```

a) 3
b) 6
c) 8
d) 10
e) None of the above



```
pi@raspberrypi ~/Downloads> ./a.o
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

Part 2: Code reading question example

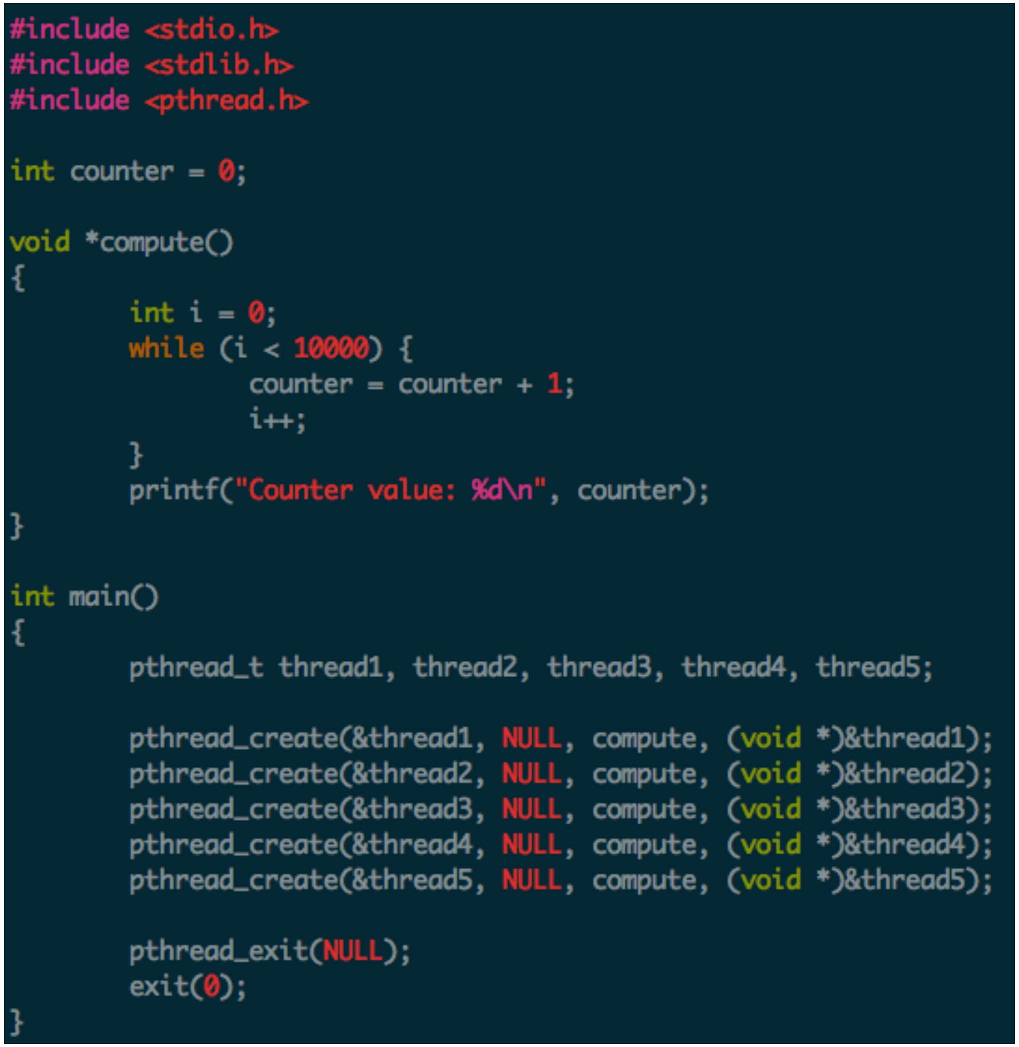


```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int counter = 0;

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

int main()
{
        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);

        pthread_exit(NULL);
        exit(0);
}
```

- What will the counter be after the last thread finishes?

a) Due to race conditions, “counter” may have different values on different runs of the program.
b) 40000
c) 50000
d) 60000
e) None of the above

Part 2: Code reading question example

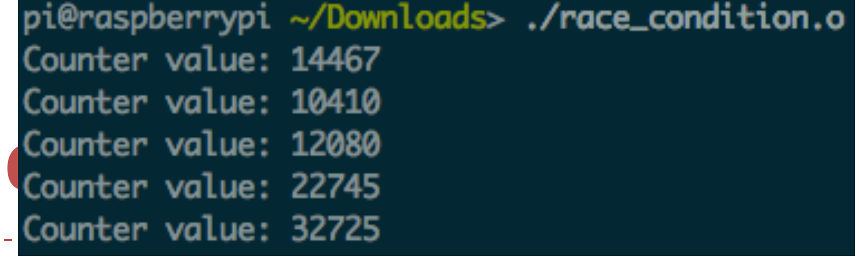


```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int counter = 0;

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

int main()
{
        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);

        pthread_exit(NULL);
        exit(0);
}
```

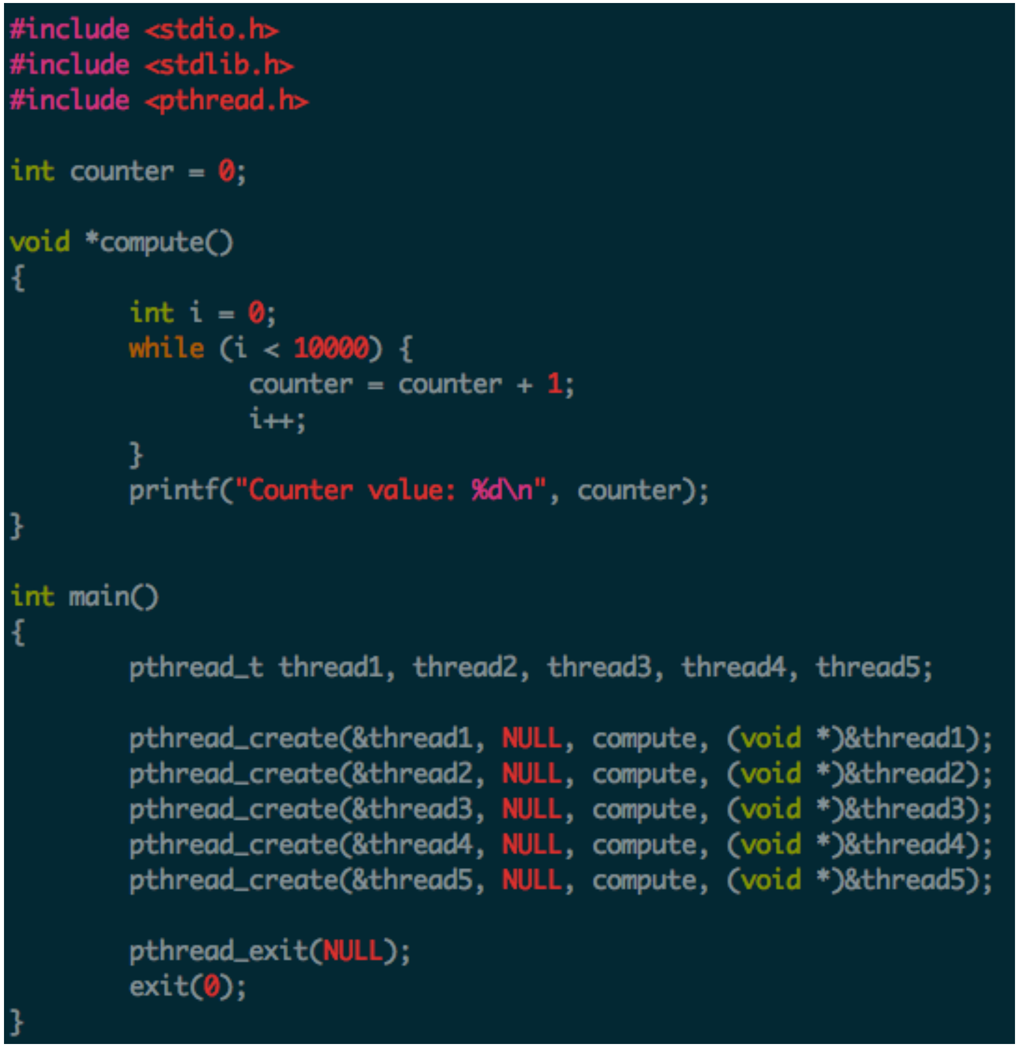
- What will the counter be after the last thread finishes?

a) Due to race conditions, “counter” may have different values on different runs of the program.
b) 40000
c) 50000
d) 60000
e) None of the above

Part 2: Code reading qu



```
pi@raspberrypi ~/Downloads> ./race_condition.o
Counter value: 14467
Counter value: 10410
Counter value: 12080
Counter value: 22745
Counter value: 32725
```

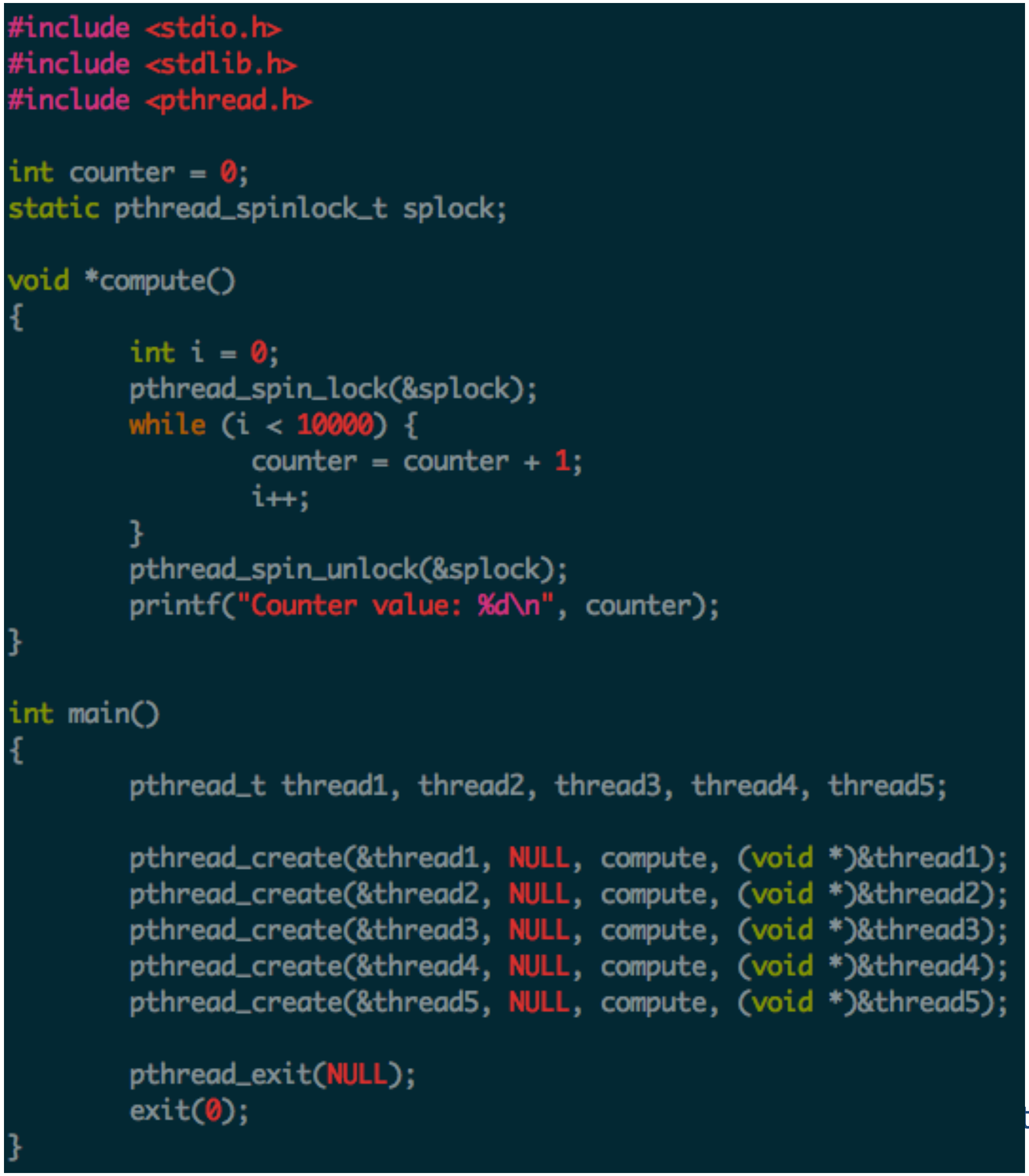


```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int counter = 0;

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

int main()
{
        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);

        pthread_exit(NULL);
        exit(0);
}
```

- What will the counter be after the last thread finishes?

a) Due to race conditions, “counter” may have different values on different runs of the program.
b) 40000
c) 50000
d) 60000
e) None of the above

Part 2: Code reading question example

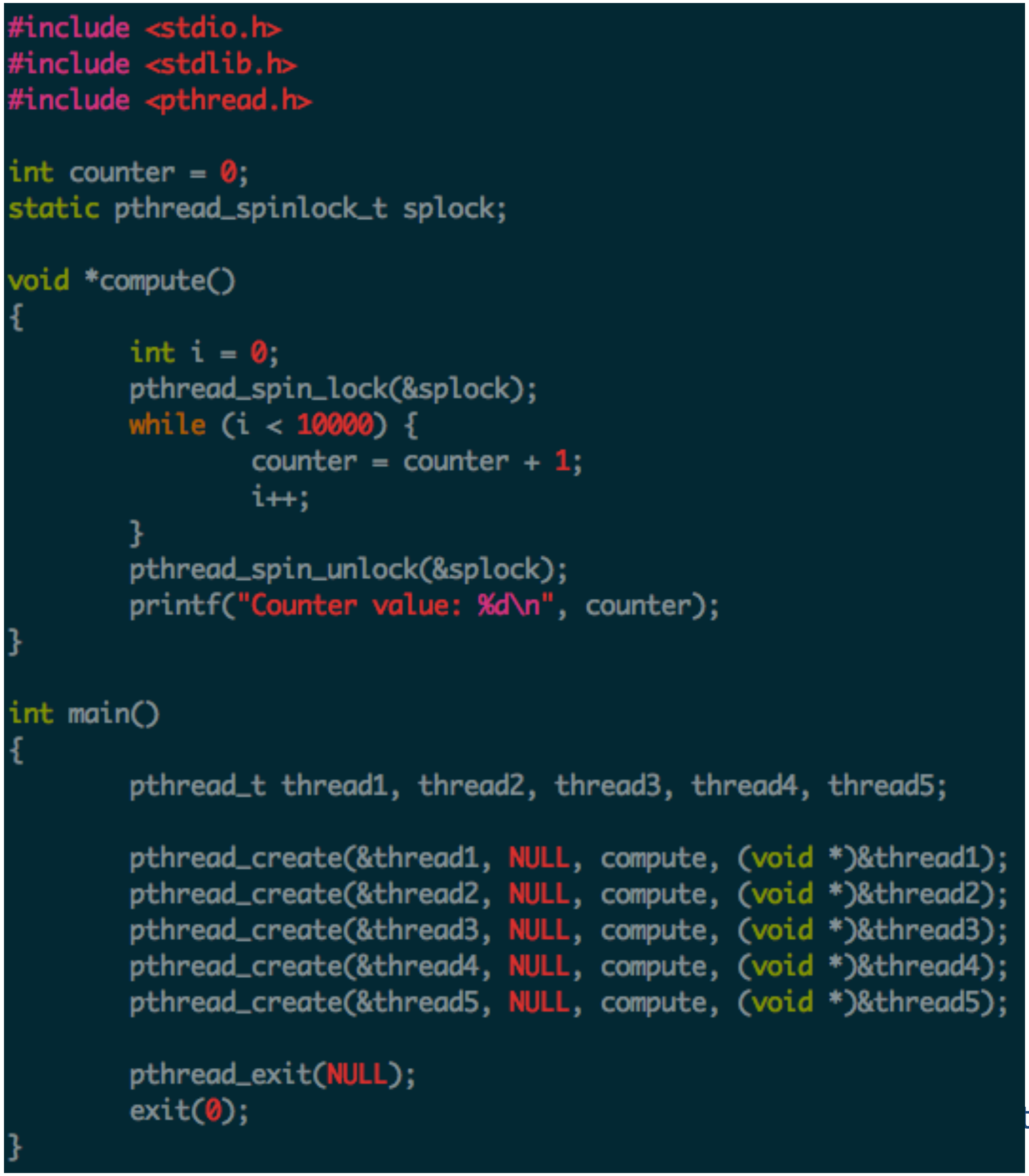


```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int counter = 0;
static pthread_spinlock_t splock;

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

int main()
{
        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);

        pthread_exit(NULL);
        exit(0);
}
```

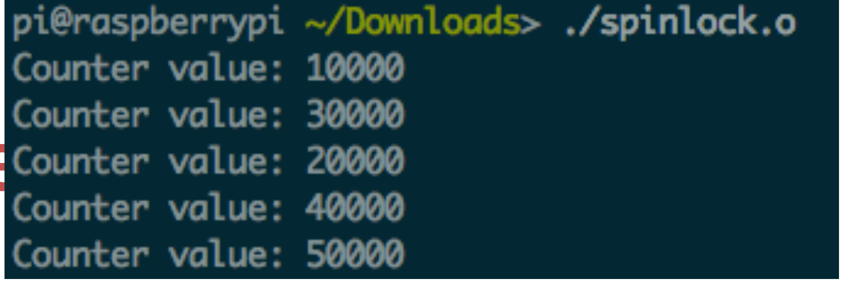
- What will the counter be after the last thread finishes?

a) Due to race conditions, “counter” may have different values on different runs of the program.
b) 40000
c) 50000
d) 60000
e) None of the above

Part 2: Code reading que



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

int counter = 0;
static pthread_spinlock_t splock;

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

int main()
{
        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);

        pthread_exit(NULL);
        exit(0);
}
```



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

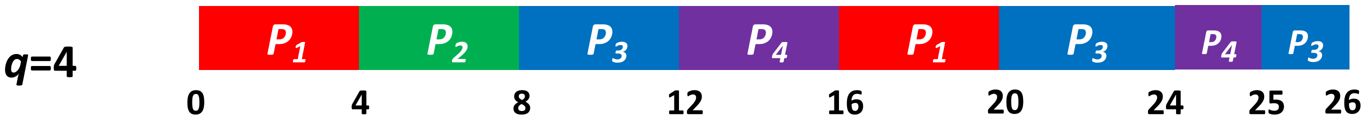
- What will the counter be after the last thread finishes?

a) Due to race conditions, “counter” may have different values on different runs of the program.
b) 40000
c) 50000
d) 60000
e) None of the above

Part 2: Code reading question example

- One scheduling question
  - Determine the process scheduling order
  - Calculate the turnaround time and response time

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



Average turnaround time = ((20-0)+(8-1)+(26-2)+(25-3)) / 4 = 18.25

Midterm Format

- 7 short answer questions
  - 6 points for each
  - Totally 42 points

Same as HW1

- 4 code reading questions
  - Totally 58 points
  - Read the code, select the correct answer and explain why

Midterm exam

- Time: Oct 14, 3:30pm to 4:45pm
- Open book, open note exam
- Not allowed using laptop/smartphone