Lab 2:

Pthreads (POSIX Threads Programming)

In this lab:

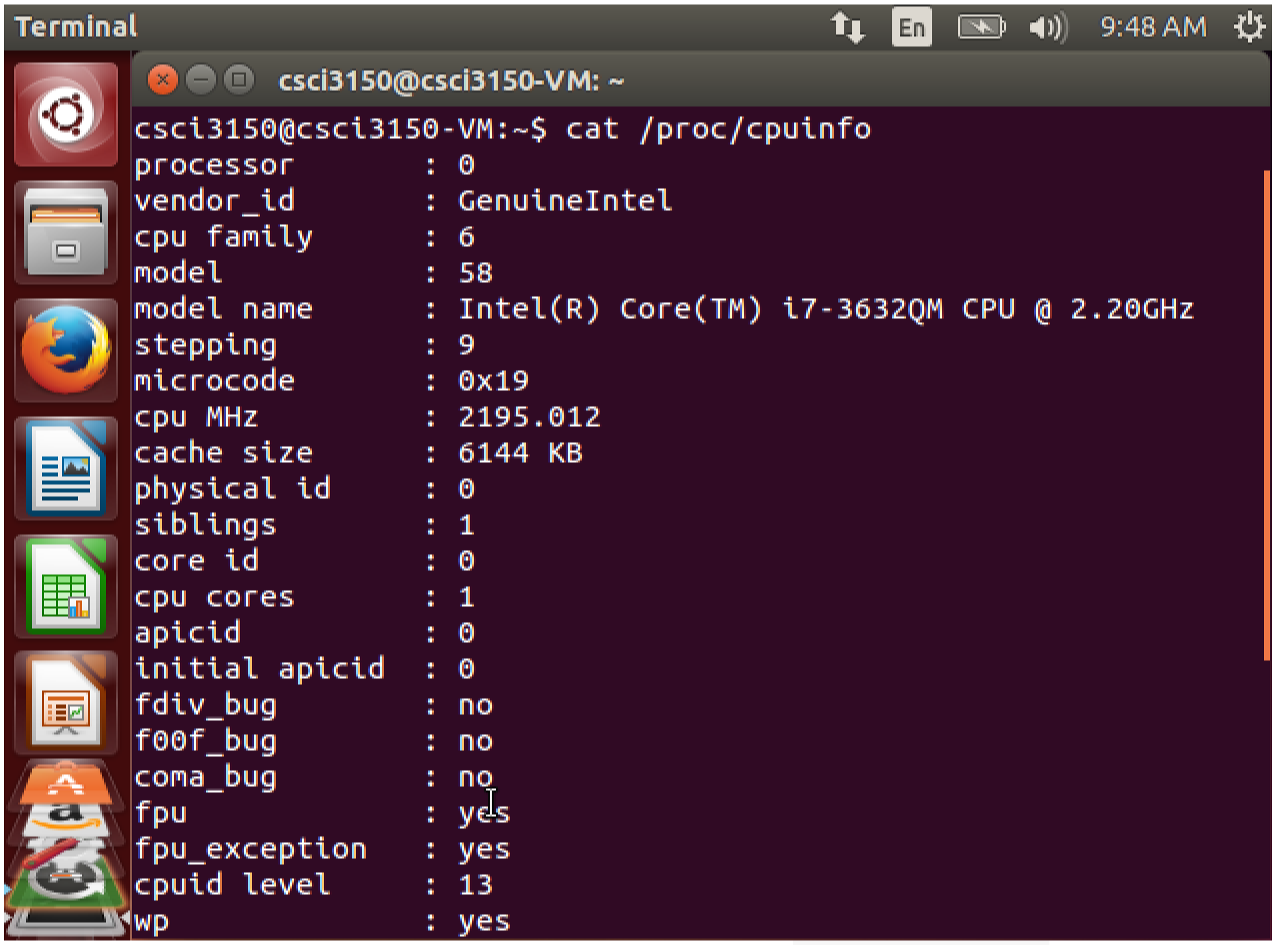
- A simple Introduction to pThread
- Configure your virtual machine to multi-cores
- Try some programs you learn in lecture
- You will get some programs which will show you about some problems like starvation problems.
- Try to solve it, I believe you can find many solutions to these problems

Introduction to pThread

- Pthreads are defined as a set of C language programming types and procedure calls, implemented with a pthread.h header/include file and a thread library - though this library may be part of another library, such as libc, in some implementations.
- To implement a multi-threads program using pThread library:
    - `#include <pthread.h>`
    - pthread_t: to define a thread id
    - pthread_create: to create a thread
    - pthread_join: join with a terminated thread
    - pthread_mutex_t: to create a mutex in pthread
    - pthread_mutex_lock: to lock a mutex in pthread
    - pthread_mutex_unlock: unlock a mutex in pthead
- We will have some example programs later
- There are many functions you can use in pthread, you can refer to materials in google. Also, we provide some materials in useful links.

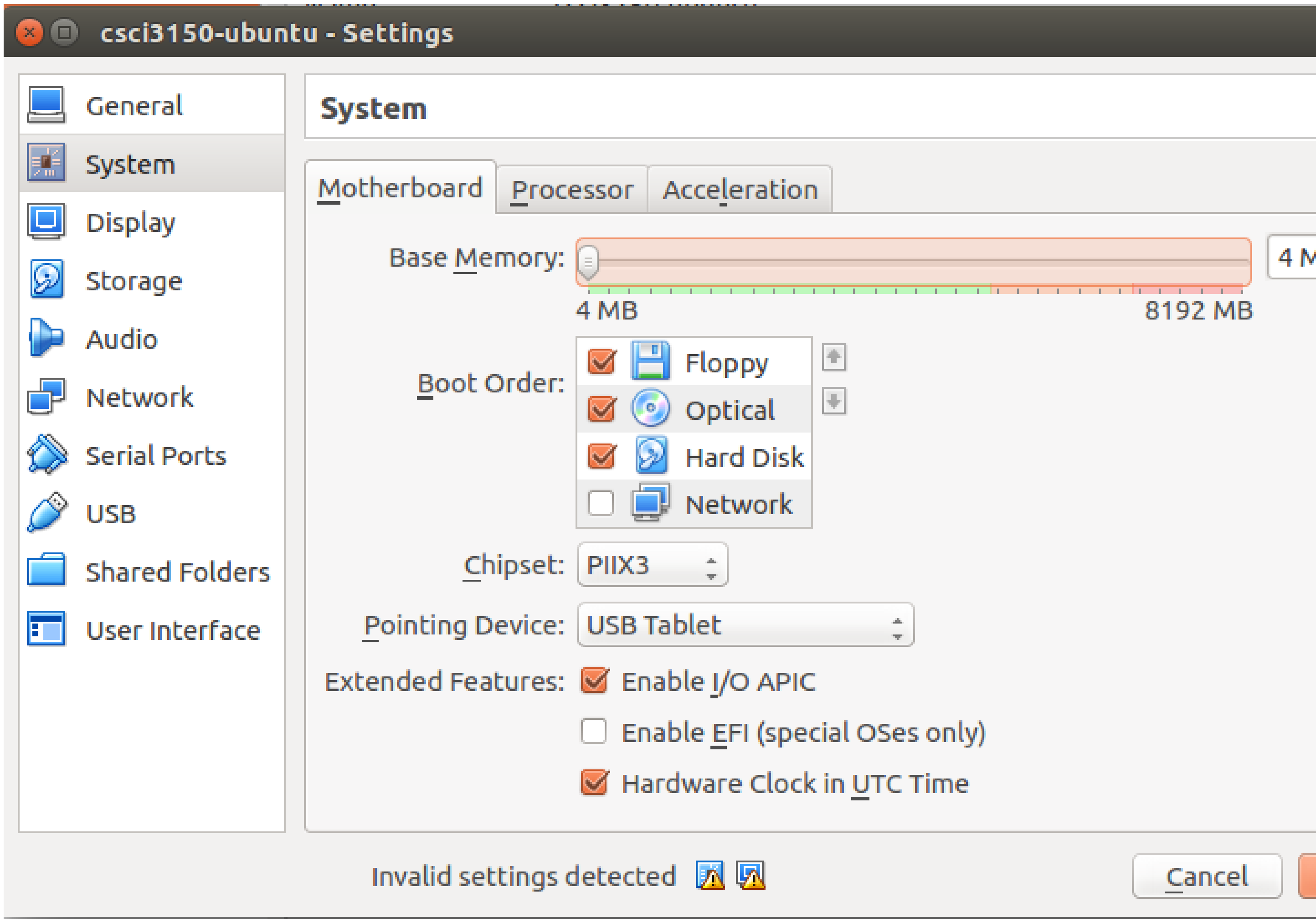
Configuration

- Check number of cpu cores in your virtual machine
    - `cat /proc/cpuinfo`



```
csci3150@csci3150-VM:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 2195.012
cache size      : 6144 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fdiv_bug        : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
```

- You can find "cpu cores: 1" in the output after you run `cat /proc/cpuinfo`, which denotes that in your ubuntu virtual machine, you only have one cpu core.
- Check the memory capacity in your virtual machine
  - right click your virtual machine -> "Settings" -> "System" -> "Motherboard"



csci3150-ubuntu - Settings

System

Motherboard | Processor | Acceleration

Base Memory: 4 MB (range 4 MB – 8192 MB)

Boot Order: [✓] Floppy, [✓] Optical, [✓] Hard Disk, [ ] Network

Chipset: PIIX3

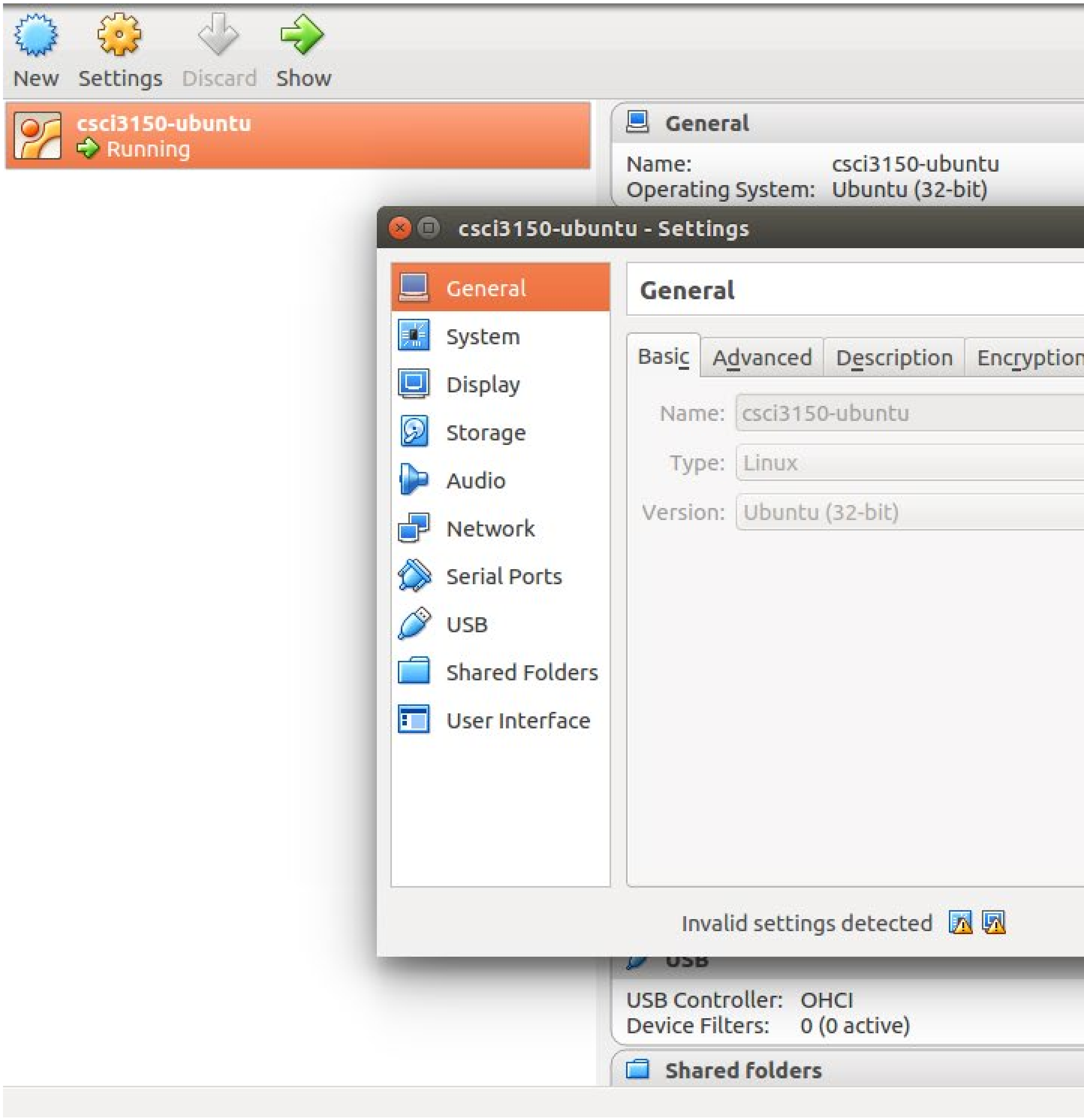
Pointing Device: USB Tablet

Extended Features: [✓] Enable I/O APIC; [ ] Enable EFI (special OSes only); [✓] Hardware Clock in UTC Time

Invalid settings detected

- We can see the default memory capacity is 4MB
- **Log out** and re-configure virtual machine

- right click your virtual machine -> "Settings" -> "System" -> "Prosessor"



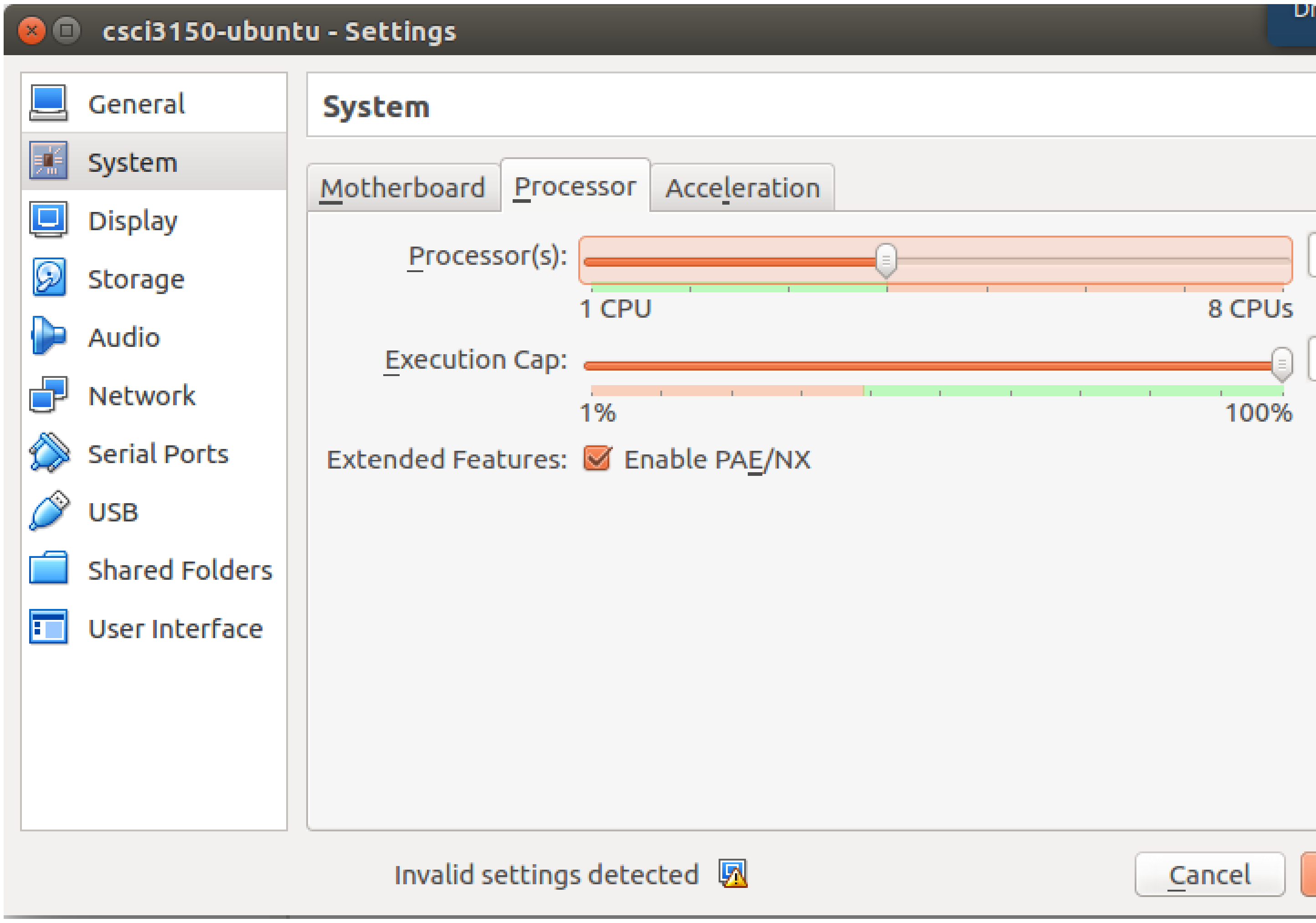
Name: csci3150-ubuntu
Operating System: Ubuntu (32-bit)

Name: csci3150-ubuntu
Type: Linux
Version: Ubuntu (32-bit)

Invalid settings detected

USB Controller: OHCI
Device Filters: 0 (0 active)

- modify "Processor(s)" to 4, or any number you want


csci3150-ubuntu - Settings

System

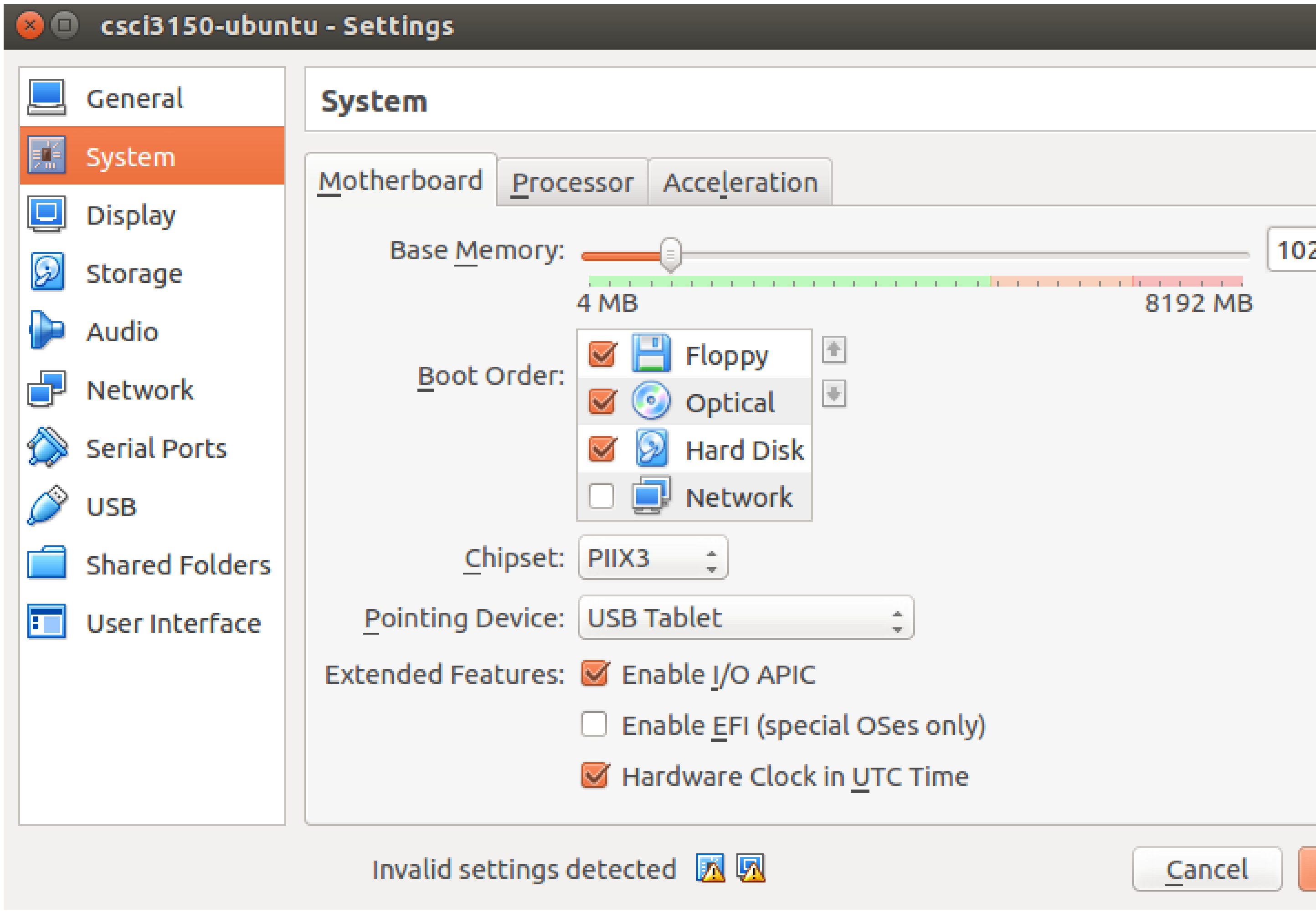
Processor(s): 1 CPU – 8 CPUs

Execution Cap: 1% – 100%

Extended Features: ☑ Enable PAE/NX

Invalid settings detected

- "Settings" -> "System" -> "Motherboard"



csci3150-ubuntu - Settings

System

Motherboard | Processor | Acceleration

Base Memory: 102
4 MB 8192 MB

Boot Order:
- [✓] Floppy
- [✓] Optical
- [✓] Hard Disk
- [ ] Network

Chipset: PIIX3

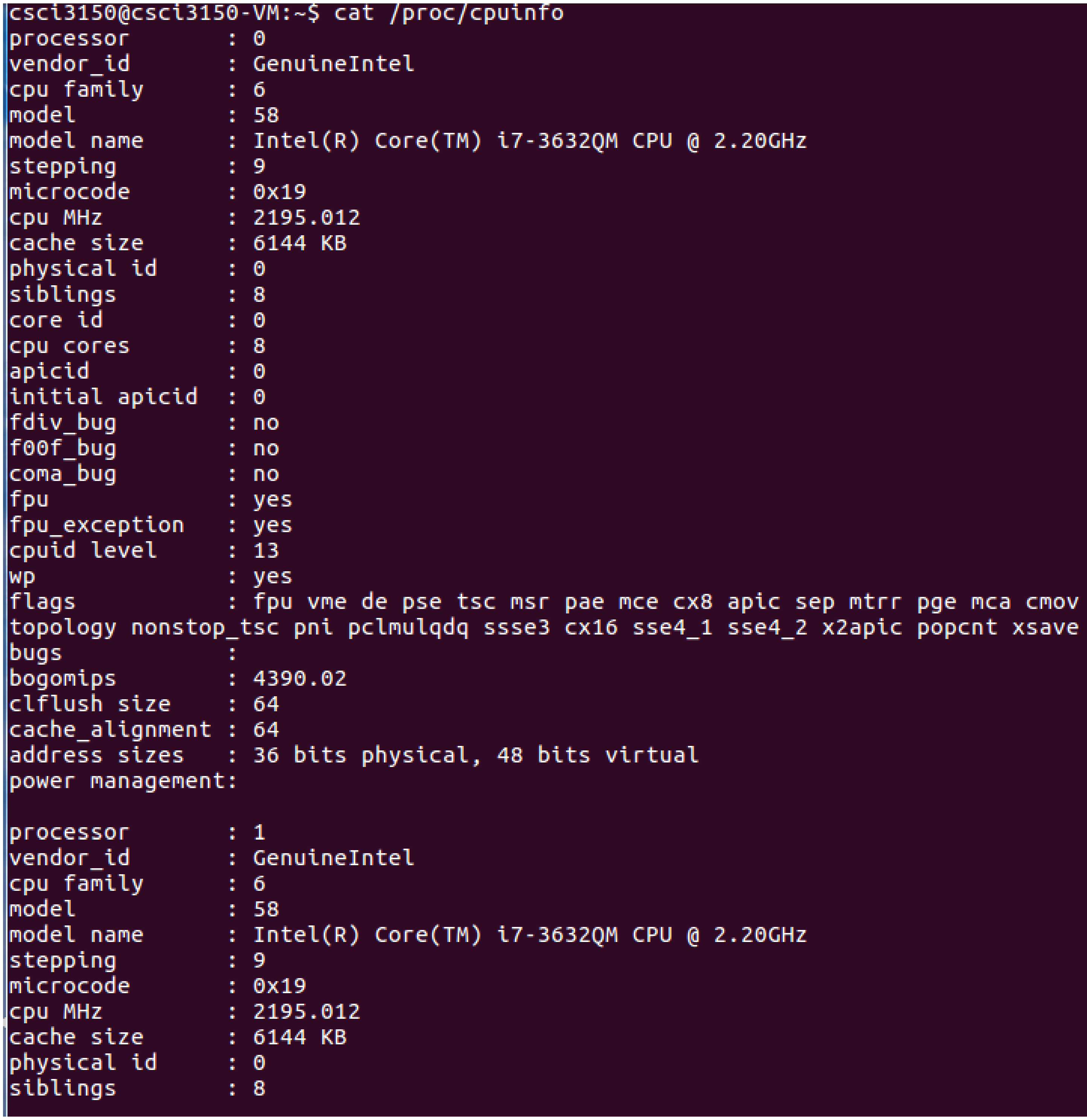
Pointing Device: USB Tablet

Extended Features:
- [✓] Enable I/O APIC
- [ ] Enable EFI (special OSes only)
- [✓] Hardware Clock in UTC Time

Invalid settings detected

- click "ok" ant start your virtual machine and now you find the number of cpu cores is 4. And the base memory is

1024MB



```
csci3150@csci3150-VM:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 2195.012
cache size      : 6144 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 8
apicid          : 0
initial apicid  : 0
fdiv_bug        : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
topology nonstop_tsc pni pclmulqdq ssse3 cx16 sse4_1 sse4_2 x2apic popcnt xsave
bugs            :
bogomips        : 4390.02
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz
stepping        : 9
microcode       : 0x19
cpu MHz         : 2195.012
cache size      : 6144 KB
physical id     : 0
siblings        : 8
```

Compile

GOOD NEWS: You don't need to install any other libraries because pthread library is pre-installed in your ubuntu version

Here we have a simple helloword.c program for you to try compiling programs using pthread library:

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

void * hello(void *input) {
    printf("%s\n", (char *)input);
    pthread_exit(NULL);
}

int main(void) {
    pthread_t tid;
    pthread_create(&tid, NULL, hello, "hello world");
    pthread_join(tid, NULL);
    return 0;
}
```

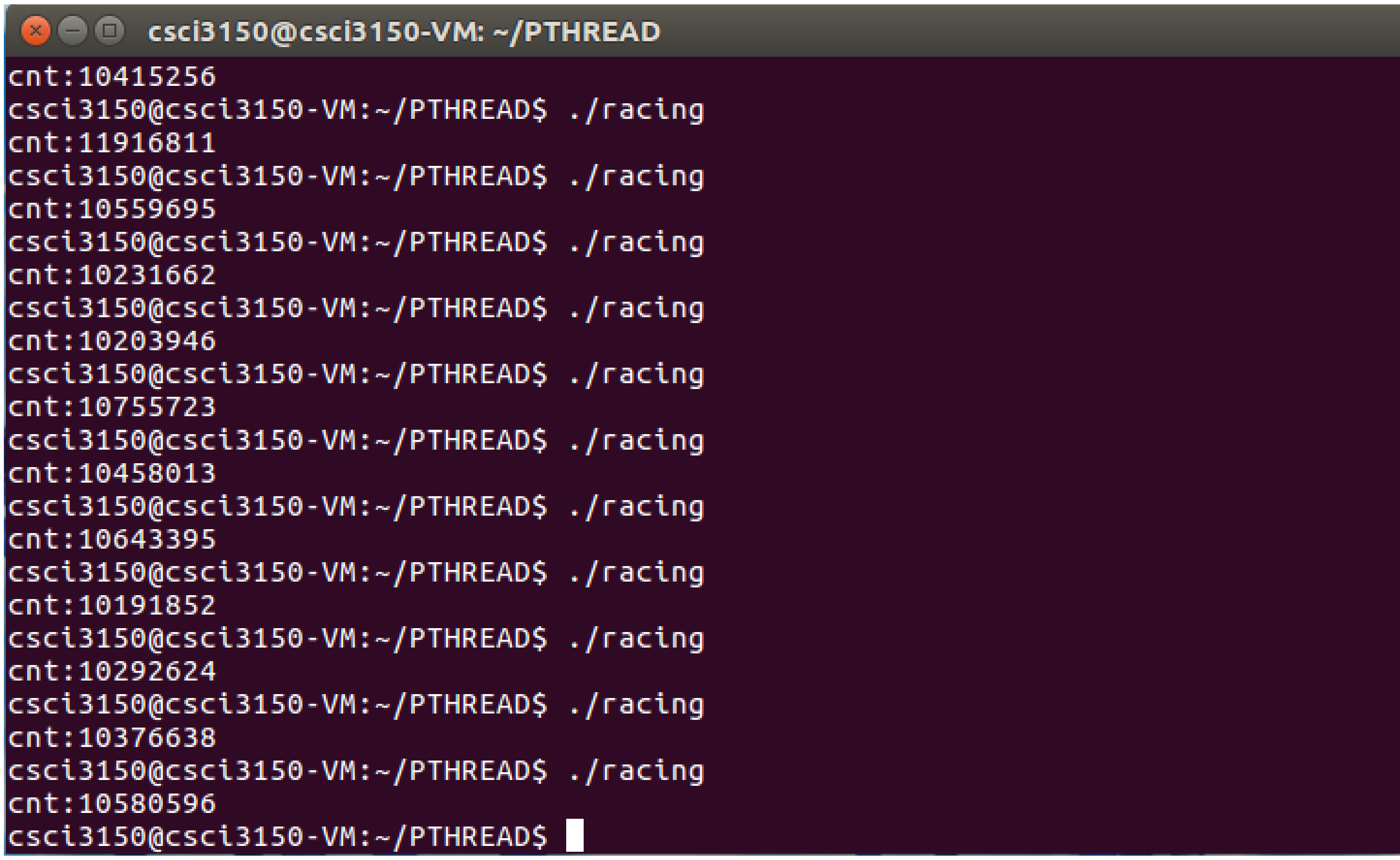
- Be sure to add `#include <pthread.h>` if you want to use pthread functions you learned in lecture
- To compile programs: `gcc -o helloworld helloworld.c -lpthread`
- To test your program: `./helloworld`
- Now you can get your hands dirty: why not try programs you learn in the lecture?

Race Condition

Here we have a simple program `racing.c`:
We have one global variable with 0 at the beginning, two threads all wants to perform `+ 1` to this variable 1000000 times, what is the result after these two threads end? Will the result be 20000000?

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

#define NITERS 10000000

void *count (void *arg);

volatile unsigned int cnt = 0;

int main () {

    pthread_t tid1, tid2;
    pthread_create (&tid1, NULL, count, NULL);
    pthread_create (&tid2, NULL, count, NULL);
    pthread_join (tid1, NULL);
    pthread_join (tid2, NULL);
    printf ("cnt:%d\n", cnt);
    exit (0);

}

void *count (void *arg) {

    volatile int i = 0;

    for (; i < NITERS; i++) {
        cnt++;
    }

    return NULL;
}
```

Let's try this program now:



```
cnt:10415256
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:11916811
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10559695
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10231662
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10203946
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10755723
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10458013
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10643395
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10191852
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10292624
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10376638
csci3150@csci3150-VM:~/PTHREAD$ ./racing
cnt:10580596
csci3150@csci3150-VM:~/PTHREAD$ 
```

Deadlock

- Book1 and Book2 are needed to do assignment, Bob and Allen all need to borrow these two books.
- Allen borrows Book1 first, after he borrowed Book1, he starts to find Book2.
- Bob borrows Book2 first, anfter he borrowed Book2, he starts to find Book1.

- No one can borrow these two books.

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


void *Allen(void *arg);
void *Bob(void *arg);

pthread_mutex_t book1=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t book2=PTHREAD_MUTEX_INITIALIZER;

int main() {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, Allen, NULL);
    pthread_create(&tid2, NULL, Bob, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return 0;
}

void *Allen(void *arg) {
    pthread_mutex_lock(&book1);
    sleep(10);
    pthread_mutex_lock(&book2);
    printf("Allen has collected all books he need, he is going to do homework!");
    pthread_mutex_unlock(&book2);
    pthread_mutex_unlock(&book1);
}

void *Bob(void *arg) {
    pthread_mutex_lock(&book2);
    sleep(10);
    pthread_mutex_lock(&book1);
    printf("Bob has collected all books he need, he is going to do homework!");
    pthread_mutex_unlock(&book1);
    pthread_mutex_unlock(&book2);
}
```

How can you solve this problem? You can find many ways to solve this deadlock problem: timeout, require for resources in a sequence, changing priority level time to time......

Starvation problem

Bob and Allen also are borrowing books from library, but now after Allen borrows a book, he doesn't want to return it to library. So Bob can't borrow this book.

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

pthread_mutex_t book=PTHREAD_MUTEX_INITIALIZER;

void *Allen(void *arg);
void *Bob(void *arg);

int main() {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, *Allen, NULL);
    pthread_create(&tid2, NULL, *Bob, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return 0;
}

void *Allen(void *arg) {
    printf("I'm Allen, I borrow this book and I won't return it!\n");
    pthread_mutex_lock(&book);
}

void *Bob(void *arg) {
    pthread_mutex_lock(&book);
    printf("I'm Bob, I can borrow it now\n");
}
```

I think you can solve this problem using pthread mutex, just try it!

Passing arguments to pthread function

- Recall the helloworld program you compile in the "Compile" section:

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

void * hello(void *input) {
    printf("%s\n", (char *)input);
    pthread_exit(NULL);
}

int main(void) {
    pthread_t tid;
    pthread_create(&tid, NULL, hello, "hello world");
    pthread_join(tid, NULL);
    return 0;
}
```

- We use "pthread_create" to create a thread, thread id is the first argument, NULL is the second argument(which should be some attribute, but we may not use it), the third argument is the function, then the last argument is what we want to pass to the new thread.
- When we define the function `hello(void *)`, this function can accept a pointer as an argument.
- In this example we just pass one string to the function, what if we want to pass more than one argument? We can use `structure` in c.

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

struct args {
    char* name;
    int age;
};

void *hello(void *input) {
    printf("name: %s\n", ((struct args*)input)->name);
    printf("age: %d\n", ((struct args*)input)->age);
}

int main() {
    struct args *Allen = (struct args *)malloc(sizeof(struct args));
    char allen[] = "Allen";
    Allen->name = allen;
    Allen->age = 20;

    pthread_t tid;
    pthread_create(&tid, NULL, hello, (void *)Allen);
    pthread_join(tid, NULL);
    return 0;
}
```

- In this example, we want to pass more than one argument to the function, so we create a pointer point to a struct we have created, transfer it into `(void *)` and pass to function.
- In function, we transfer the type of pointer to the real type, so that we can properly use it.

Answers to the exercises

- Here is only one possible answer of those problems. It's ok you solve those problem in a different way.
- avoid_race.c

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

#define NITERS 10000000

void *count (void *arg);

volatile unsigned int cnt = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int main () {

    pthread_t tid1, tid2;
    pthread_create (&tid1, NULL, count, NULL);
    pthread_create (&tid2, NULL, count, NULL);

    pthread_join (tid1, NULL);
    pthread_join (tid2, NULL);
    printf ("cnt:%d\n", cnt);
    exit (0);

}

void *count (void *arg) {

    volatile int i = 0;

    for (; i < NITERS; i++) {
        pthread_mutex_lock(&mutex);
        cnt++;
        pthread_mutex_unlock(&mutex);
    }

    return NULL;
}
```

- avoid_deadlock.c

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

void *Allen(void *arg);
void *Bob(void *arg);

pthread_mutex_t book1;
pthread_mutex_t book2;
pthread_cond_t Allenfinish;

int main() {
    pthread_t tid1, tid2;
    pthread_create(&tid1, NULL, &Allen, NULL);
    pthread_create(&tid2, NULL, &Bob, NULL);

    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return 0;
}

void *Allen(void *arg) {
    pthread_mutex_lock(&book1);
    sleep(5);
    pthread_mutex_lock(&book2);
    printf("Allen has collected all books he need, he is going to do 
homework!\n");
    sleep(5);
    printf("Allen has finished his homework, he has returned all books he 
borrowed!\n");
    pthread_mutex_unlock(&book2);
    pthread_mutex_unlock(&book1);
    pthread_cond_signal(&Allenfinish);
}

void *Bob(void *arg) {
    pthread_mutex_lock(&book2);
    pthread_cond_wait(&Allenfinish, &book2);

    printf("Bob knows he can borrow those two books now!\n");
    pthread_mutex_lock(&book1);
    sleep(5);
    printf("Bob has finished his homework now!\n");
    pthread_mutex_unlock(&book1);
    pthread_mutex_unlock(&book2);
}
```

- No answers to Starvation to problem. It's really a easy one and you can solve it!