CS444/544
Operating Systems II

Lecture 15
Deadlock (cont.)
Prep. for Quiz 3
11/30/2023

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Odds and Ends

• Meme Competition due: Sunday midnight via Canvas (12/3 11:59 pm)
Part-A Result

• You should get this OK before start exercise 8

• FAQ
  • What if dumbfork halts?
    • Check if your sched_yield()/env_run() is implemented correctly
    • curenv must set as ENV_RUNNABLE state if it is scheduled out...
  • What if I have a syscall error?
    • Check if your implementation returns the return value of the syscall correctly
    • Check syscall arguments and orders
    • There always be syscalls to SYS_getenvid and SYS_cputs
CAUTION:
You Will See LOTS of Page Faults in Part B

• What should I do if I see a page fault?

• Check information related to the fault
  • Check tf_eip (the origin of the fault)
  • Check fault_va (read cr2, rcr2())
    • You can reason a lot from this address, e.g., 0xcabeffe?
    • If it is 0, a null pointer dereference, check your impl!!!
  • Check error code (user/kernel, read/write, present?)

• Think about why this fault happens???
How Can I Get the Code for User Exec?

- Read `obj/user/xxxx.asm`
- E.g., `dumbfork`:
  - You can match `eip` and the source code

```assembly
void duppage(envid_t dstenv, void *addr)
{
    800040: 55           push %ebp
    800041: 89 e5        mov %esp,%ebp
    800043: 56           push %esi
    800044: 53           push %ebx
    800045: 83 ec 20     sub $0x20,%esp
    800048: 8b 75 08     mov 0x8(%ebp),%esi
    80004b: 8b 5d 0c     mov 0xc(%ebp),%ebx
    int r;

    // This is NOT what you should do in your fork.
    if ((r = sys_page_alloc(dstenv, addr, PTE_P|PTE_U|PTE_W)) < 0)
        if ((r = sys_page_map(dstenv, addr, 0, UTEMP, PTE_P|PTE_U|PTE_W)) < 0)
            panic("sys_page_map: %e", r);
    panic("sys_page_alloc: %e", r);
    80006a: c7 44 24 0b   mov %eax,%esi
    80006d: 89 44 24 0c   mov %eax,%esi
    800070: 00
    800073: c7 44 24 0b   mov %eax,%esi
    800076: 89 44 24 0c   mov %eax,%esi
    800079: 00
    80007c: c7 44 24 0b   mov %eax,%esi
    80007f: 89 44 24 0c   mov %eax,%esi
    800082: 00
    800085: c7 44 24 0b   mov %eax,%esi
    800088: 89 44 24 0c   mov %eax,%esi
    80008b: 00
    80008e: c7 44 24 0b   mov %eax,%esi
    800091: 89 44 24 0c   mov %eax,%esi
    800094: 00
    800097: c7 44 24 0b   mov %eax,%esi
    80009a: 89 44 24 0c   mov %eax,%esi
    80009d: 00
    8000a0: c7 44 24 0b   mov %eax,%esi
    8000a3: 89 44 24 0c   mov %eax,%esi
    8000a6: 00
    8000a9: c7 44 24 0b   mov %eax,%esi
    8000ab: 89 44 24 0c   mov %eax,%esi
    8000ad: 00
    8000ae: c7 44 24 0b   mov %eax,%esi
    8000b1: 89 44 24 0c   mov %eax,%esi
    8000b4: 00
    8000b7: c7 44 24 0b   mov %eax,%esi
    8000ba: 89 44 24 0c   mov %eax,%esi
    8000bd: 00
    8000c0: c7 44 24 0b   mov %eax,%esi
    8000c3: 89 44 24 0c   mov %eax,%esi
    8000c6: 00
    8000c9: c7 44 24 0b   mov %eax,%esi
    8000cb: 89 44 24 0c   mov %eax,%esi
    8000cd: 00
    8000ce: c7 44 24 0b   mov %eax,%esi
    8000d1: 89 44 24 0c   mov %eax,%esi
    8000d4: 00
    8000d7: c7 44 24 0b   mov %eax,%esi
    8000da: 89 44 24 0c   mov %eax,%esi
    8000dd: 00
    8000e0: c7 44 24 0b   mov %eax,%esi
    8000e3: 89 44 24 0c   mov %eax,%esi
    8000e6: 00
    8000e9: c7 44 24 0b   mov %eax,%esi
    8000eb: 89 44 24 0c   mov %eax,%esi
    8000ed: 00
    8000f0: c7 44 24 0b   mov %eax,%esi
    8000f3: 89 44 24 0c   mov %eax,%esi
    8000f6: 00
    8000f9: c7 44 24 0b   mov %eax,%esi
    8000fa: 89 44 24 0c   mov %eax,%esi
    8000fd: 00
    8000fe: 00
    8000ff: 00
}
```
Debugging Tips

• Check your traps. Recommend to print out some trap information whenever you got a trap...

```c
static void
trap_dispatch(struct Trapframe *tf) {
    // Handle processor exceptions.
    // LAB 3: Your code here.

    uint32_t evid;
    if (curenv == NULL) evid = 0;
    else evid = curenv->env_id;
    if (tf->tf_trapno == T_SYSCALL) {
        cprintf("Syscall from \%s(%p, %p, %p, %p, %p) from "
            "eip %p\n",
            "stringtbl[tf->tf_regs.reg_eax],
            tf->tf_regs.reg_edx,
            tf->tf_regs.reg_ecx,
            tf->tf_regs.reg_ebx,
            tf->tf_regs.reg_edi,
            tf->tf_regs.reg_esi,
            tf->tf_eip);
    } else if (tf->tf_trapno == T_PGFLT) {
        cprintf("Page fault from %p from va %p eip %p\n",
            envid,
            rcr2(), tf->tf_eip);
    } else {
        cprintf("Trap from %p number %d from eip %p\n",
            envid,
            tf->tf_trapno, tf->tf_eip);
    }
```
Debugging Tips

- Check your traps. Recommend to print out some trap information whenever you got a trap...
Recap: Concurrency Bugs

• Code does not have a bug when it runs with single thread could have a bug when it runs with multiple threads
  • Multiple cores, etc.

• What are the types of concurrency bugs?
  • Atomicity
  • Ordering
  • Deadlock
Recap: Atomicity: Use Lock

```c
1. pthread_mutex_t proc_info_lock = PTHREAD_MUTEX_INITIALIZER;
2. 
3. Thread 1::
4. pthread_mutex_lock(&proc_info_lock);
5. if (thd->proc_info) {
   6.     ...
   7.     fputs(thd->proc_info, ...);
   8.     ...
   9. }
10. pthread_mutex_unlock(&proc_info_lock);
11. 
12. Thread 2::
13. pthread_mutex_lock(&proc_info_lock);
14. thd->proc_info = NULL;
15. pthread_mutex_unlock(&proc_info_lock);
```

- **Time of check**
- **Time of use**
- **Update!**

In critical section, NO UPDATE
Do not have TOCTTOU!

This will also block other threads that run line 5 while thread 2 updates thd->proc_info.
Recap: How Can We Resolve the Ordering Issue?

• Use locks and conditional variables to force a specific ordering...

```c
Thread 1::
void init() {
    ...
    mThread = PR_CreateThread(mMain, ...);
    // signal that the thread has been created...
    pthread_mutex_lock(&mtLock);
    mtInit = 1;
    pthread_cond_signal(&mtCond); // Sends Signal..
    pthread_mutex_unlock(&mtLock);
    ...
}

Thread 2::
void mMain(...) {
    ...
    // wait for the thread to be initialized...
    pthread_mutex_lock(&mtLock);
    while (mtInit == 0) // Waits condition..
        pthread_cond_wait(&mtCond, &mtLock);
    pthread_mutex_unlock(&mtLock);
    mState = mThread->State;
    ...
}
```
Recap: Deadlock

- Two or more threads are waiting for the other to take some actions thus neither makes any progress.

Thread 1:
```
pthread_mutex_lock(L1);
pthread_mutex_lock(L2);
```

Thread 2:
```
pthread_mutex_lock(L2);
pthread_mutex_lock(L1);
```
Recap: Circular Dependency

Thread 1:
 pthread_mutex_lock(L1);
 pthread_mutex_lock(L2);

Thread 2:
 pthread_mutex_lock(L2);
 pthread_mutex_lock(L1);
Recap: Non-Circular Dependency

Thread 1:
`pthread_mutex_lock(L1);`
`pthread_mutex_lock(L2);`

Thread 2:
`pthread_mutex_lock(L1);`
`pthread_mutex_lock(L2);`
Thread-safe Data structure

```c
set_t *set_intersection (set_t *s1, set_t *s2) {
    set_t *rv = new set_t();
    Mutex_lock(&s1->lock);
    Mutex_lock(&s2->lock);
    for(int i=0; i<s1->len; i++) {
        if(set_contains(s2, s1->items[i])
            set_add(rv, s1->items[i]);
    }
    Mutex_unlock(&s2->lock);
    Mutex_unlock(&s1->lock);
    return rv;
}
```
Thread-safe Data structure

Thread 1:
rv = set_intersection(setA, setB);

set_t *set_intersection (set_t *s1, set_t *s2) {

...  
Mutex_lock(&s1->lock);  
Mutex_lock(&s2->lock);  
...
}

Thread 2:
rv = set_intersection(setA, setB);
Thread-safe Datastructure

Thread 1:

```c
rv = set_intersection(setA, setB);
Mutex_lock(&setA->lock);
Mutex_lock(&setB->lock);
...
Mutex_unlock(&setB->lock);
Mutex_unlock(&setA->lock);
```

Thread 2:

```c
rv = set_intersection(setA, setB);
Mutex_lock(&setA->lock);
Mutex_lock(&setB->lock);
...
Mutex_unlock(&setB->lock);
Mutex_unlock(&setA->lock);
```
Is This a Thread-safe Datastructure?

```c
set_t *set_intersection (set_t *s1, set_t *s2) {
    set_t *rv = new set_t();
    Mutex_lock(&s1->lock);
    Mutex_lock(&s2->lock);
    for(int i=0; i<s1->len; i++) {
        if(set_contains(s2, s1->items[i])
            set_add(rv, s1->items[i]);
    }
    Mutex_unlock(&s2->lock);
    Mutex_unlock(&s1->lock);
    return rv;
}
```
Find a Problem..

Thread 1:

rv = set_intersection(setA, setB);

Thread 2:

rv = set_intersection(setB, setA);

set_t *set_intersection(set_t *s1, set_t *s2) {
    ...
    Mutex_lock(&s1->lock);
    Mutex_lock(&s2->lock);
    ...
}
Find a Problem..

Thread 1:

rv = set_intersection(setA, setB);

✓ Mutex_lock(&setA->lock);
 Mutex_lock(&setB->lock);

Thread 2:

rv = set_intersection(setB, setA);

✓ Mutex_lock(&setB->lock);
 Mutex_lock(&setA->lock);

Deadlock!
Deadlock Theory

• Deadlocks can only happen if threads are having
  • Mutual exclusion
  • Hold-and-wait
  • No preemption
  • Circular wait

• We can eliminate deadlock by removing such conditions...
Mutual Exclusion

• Definition
  • Threads claims an exclusive control of a resource
  • E.g., Threads grabs a lock
How to Remove Mutual Exclusion

• Do not use lock
  • What???

• Replace locks with atomic primitives
  • compare_and_swap(uint64_t *addr, uint64_t prev, uint64_t value);
  • if *addr == prev, then update *addr = value;
  • lock cmpxchg in x86..

```c
void add (int *val, int amt) {
    Mutex_lock(&m);
    *val += amt;
    Mutex_unlock(&m);
}
```

```c
void add (int *val, int amt) {
    do {
        int old = *val;
    } while(!CompAndSwap(val, ??, old+amt);
}
```
Hold-and-Wait

• Definition
  • Threads hold resources allocated to them (e.g., locks they have already acquired) while waiting for additional resources (e.g., locks they wish to acquire).

```c
Mutex_lock(&setA->lock);
Mutex_lock(&setB->lock);
```
How to Remove Hold-and-Wait

• Strategy: Acquire all locks atomically once
  • Can release lock over time, but cannot acquire again until all have been released

• How to do this? Use a meta lock, like this:

```cpp
lock(&meta);
lock(&L1);
lock(&L2);
...
unlock(&meta);

// Critical section code
unlock(...);
```
Remove Hold-and-Wait

```c
set_t *set_intersection (set_t *s1, set_t *s2) {
    Mutex_lock(&meta_lock)
    Mutex_lock(&s1->lock);
    Mutex_lock(&s2->lock);
    ...
    Mutex_unlock(&s2->lock);
    Mutex_unlock(&s1->lock);
    Mutex_unlock(&meta_lock);
}
```
Remove Hold-and-Wait

Thread 1:

\[ rv = \text{set\_intersection}(\text{setA}, \text{setB}); \]

Mutex_lock(&meta_lock);
Mutex_lock(&setA->lock);
Mutex_lock(&setB->lock);
...
Mutex_unlock(&setB->lock);
Mutex_unlock(&setA->lock);
Mutex_unlock(&meta_lock);

Thread 2:

\[ rv = \text{set\_intersection}(\text{setB}, \text{setA}); \]

Mutex_lock(&meta_lock);
Mutex_lock(&setB->lock);
Mutex_lock(&setA->lock);

\[ \text{Will wait until Thread 1 finishes (release meta\_lock)!} \]
No Preemption

• Definition
  • Resources (e.g., locks) cannot be forcibly removed from threads that are holding them.

```
lock(A);
lock(B);
...  // In case if B is acquired by other thread
...  // All other threads must wait for acquiring A
```
How to Remove No Preemption

Release the lock if obtaining a resource fails...

top:

lock(A);
if (trylock(B) == -1) {
    unlock(A);
    goto top;
}
...
Circular Wait

• Definition
  • There exists a circular chain of threads such that each thread holds a resource (e.g., lock) being requested by next thread in the chain.
How to Remove Circular Wait

Thread 1:
pthread_mutex_lock(L1);
pthread_mutex_lock(L2);

Thread 2:
pthread_mutex_lock(L2);
pthread_mutex_lock(L1);

Thread 1:
pthread_mutex_lock(L1);
pthread_mutex_lock(L2);

Thread 2:
pthread_mutex_lock(L1);
pthread_mutex_lock(L2);
How to Remove Circular Wait

Lock variable is mostly a pointer, then provide a correct order of having a lock e.g.,
if(l1 > l2) {
    Mutex_lock(l1);
    Mutex_lock(l2);
} else {
    Mutex_lock(l2);
    Mutex_lock(l1);
}

Thread 1:
pthread_mutex_lock(L1);
pthread_mutex_lock(L2);

Thread 2:
pthread_mutex_lock(L2);
pthread_mutex_lock(L1);
Deadlock Theory

- Deadlocks can only happen if threads are having
  - Mutual exclusion
  - Hold-and-wait
  - No preemption
  - Circular wait

- We can eliminate deadlock by removing such conditions...
Quiz 3

• Next Tuesday (12/5 from 8:00 am to 11:59 pm)
  • Open materials (slides, videos, code, and textbook)

• You will have 2 attempts for the quiz
Quiz 3 Coverage

• Lab 3 (User/Kernel, System Call and Interrupt Handling)
• Lab 4 (Preemptive Multitasking & Copy-on-write Fork)
• Lecture 12: Multithreading and Synchronization
• Lecture 13-14: Lock and Thread Synchronization
• Lecture 14-15: Concurrency Bugs and Deadlock
Sample Questions

• In x86, which of the following instruction runs atomically?
  • cmpxchg
  • popa
  • lea
  • xchg
  • mov
Sample Questions

• In x86, which of the following instruction runs atomically?
  • cmpxchg
  • popa
  • lea
  • xchg
  • mov
Sample Questions

• In x86, which of the following instruction runs atomical test and test-and-set?
  • cmpxchg
  • int $0x30
  • lock cmpxchg
  • lock
  • xchg
Sample Questions

• In x86, which of the following instruction runs atomical test and test-and-set?
  • cmpxchg
  • int $0x30
  • lock cmpxchg
  • lock
  • xchg

cmpxchg in x86 is not a hardware atomic instruction. However, when used with the lock prefix, the instruction will be an atomic test and test-and-set instruction.
Sample Questions

• In x86, which register is being used for storing “compare” value when running the cmpxchg instruction?
  • CR3
  • EAX
  • EBX
  • ESP
  • EIP
Sample Questions

- In x86, which register is being used for storing “compare” value when running the cmpxchg instruction?
  - CR3
  - EAX
  - EBX
  - ESP
  - EIP
Sample Questions

• T/F: Page table is not relevant to data racing / thread synchronization.
Sample Questions

• T/F: Page table is not relevant to data racing / thread synchronization.

  True. Page table is for virtual memory, and thus is not relevant to thread sync.
Sample Questions

• In JOS lab, which value will the fork() returns to the child environment if the function has been executed successfully?
  • 0
  • 1
  • The envid of the parent env
  • The envid of the child env
  • The address of the page table of the child env
Sample Questions

• In JOS lab, which value will the fork() returns to the child environment if the function has been executed successfully?
  • 0
  • 1
  • The envid of the parent env
  • The envid of the child env
  • The address of the page table of the child env

Fork returns:
Parent: child envid
Child: 0
Sample Questions

• Which of the following stores the information about the reason of a page fault?
  • EAX
  • CR2
  • CR3
  • eflags
  • Trapframe
Sample Questions

• Which of the following stores the information about the reason of a page fault?
  • EAX
  • CR2
  • CR3
  • eflags
  • Trapframe

Error code in trapframe
Sample Questions

• Will this implementation cause deadlock (assuming no infinite loop in the critical section)?

Thread 1:

\[ \text{pthread_mutex_lock}(	ext{L1}); \]
\[ \text{pthread_mutex_lock}(	ext{L2}); \]

Thread 2:

\[ \text{pthread_mutex_lock}(	ext{L2}); \]
\[ \text{pthread_mutex_lock}(	ext{L1}); \]

Yes
Sample Questions

• Will this implementation cause deadlock (assuming no infinite loop in the critical section)?

Thread 1:
spin_lock(&meta);
spin_lock(&l1);
spin_lock(&l2);
spin_unlock(&meta);
...
spin_unlock(&l2);
spin_unlock(&l1);

Thread 2:
spin_lock(&meta);
spin_lock(&l2);
spin_lock(&l1);
spin_unlock(&meta);
...
spin_unlock(&l1);
spin_unlock(&l2);