CS444/544
Operating Systems II

Lecture 15
Lock and Synchronization (cont.)
Concurrency Bugs and Deadlock
5/22/2024

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Odds and Ends

• No lecture next Monday (5/27) – Memorial Day

• It is recommended to complete Part A by next lecture
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_getenv 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., 0xcafefbe?
    • 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

```c
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) )
    80004e:  c7 44 24 00 00 00 00 movl %eax,%esi
    800053:  89 5c 24 04 mov %ebx,0x4(%esi)
    800056:  89 34 24 mov %esi,(%esp)
    800059:  e8 81 0d 00 00 call 800de3 <sys_page_alloc>
    800062:  85 c0 test %eax,%eax
    800064:  79 20 jns 800086 <duppage+0x46>
    panic("sys_page_alloc: %e", r);
    800066:  89 44 24 0c mov %eax,0xc(%esp)
    80006a:  c7 44 24 08 a0 12 80 movl $0x8012a0,0x8(%esp)
    800071:  00
    800072:  c7 44 24 04 20 00 00 movl $0x20,0x4(%esp)
    800079:  00
    80007c:  c7 44 24 b3 12 80 00 movl $0x8012b3,0x4(%esp)
    800081:  e8 24 02 00 00 call 8002a9 <panic>
    if ( ((r = sys_page_map(dstenv, addr, 0, UTEMP, PTE_P | PTE_U | PTE_W)) < 0) )
    800085:  c7 44 24 10 07 00 00 movl $0x7,0x10(%esp)
    800089:  00
    80008e:  c7 44 24 0c 00 00 40 movl $0x400000,0xc(%esp)
    800095:  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) from "
                "eip \%p\n", 
                evid, 
                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", 
                evid, 
                rcr2(), tf->tf_eip);
    } else {
        cprintf("Trap from \%p number \%d from eip \%p\n", 
                evid, 
                tf->tf_trapno, tf->tf_eip);
    }
```
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 envid;
    if (curenv == NULL) envid = 0;
    else envid = curenv->env_id;
    if (tf->tf_trapno == T_SYSCALL) {
        cprintf("Syscall from \%p %s(%p, %p, %p, %p, %p) from "
                "eip \%p\n",
            envid,
            stringtbl[tf->tf_regs.reg_eax],
            tf->tf_regs.reg_edx,
            tf->tf_regs.reg_ecx,
            tf->tf_regs.reg_rsp,
            tf->tf_regs.reg_rdi,
            tf->tf_regs.reg_rsi,
            tf->tf_regs.reg_rdx);
    }

    [00000000] new env 00001000
    Syscall from 0x1000 SYS_getenv(0x0, 0x0, 0x0, 0x0, 0x0) from eip 0x800bdf
    Syscall from 0x1000 SYS_puts(0xeeebe88, 0x27, 0x0, 0x0, 0x0) from eip 0x800b4f
    I am the parent. Forking the child...
    Syscall from 0x1000 SYS_page_alloc(0x1000, 0xeef000, 0x7, 0x0, 0x0) from eip 0x800c23
    Syscall from 0x1000 SYS_env_set_pgfault_upcall(0x0, 0x8012b9, 0x0, 0x0, 0x0) from eip 0x800d6f
    Syscall from 0x1000 SYS_exofork(0x0, 0x8012b9, 0x0, 0x0, 0x0) from eip 0x800f77
    [00001000] new env 00001001
    Syscall from 0x1000 SYS_page_map(0x0, 0x200000, 0x1001, 0x200000, 0x805) from eip 0x800c76
    Syscall from 0x1000 SYS_page_map(0x0, 0x200000, 0x0, 0x200000, 0x805) from eip 0x800c76
    Trap from 0x1000 number 32 from eip 0x800c76
```
Recap: Data Race Example

- counter += value
  - edx = value;
  - eax = counter;
  - eax = edx + eax;
  - counter = eax;

- Assume counter = 0 at start

Counter = 1, not 2!!!
Recap: Mutex

• Lock
  • Prevent others enter the critical section
• Unlock
  • Release the lock, let others acquire the lock
• counter += value
  • `lock()`
  • `edx = value;`
  • `eax = counter;`
  • `eax = edx + eax;`
  • `counter = eax;`
  • `unlock()`
Spinlock Examples

- unzip lock-example-master.zip
- Run 30 threads, each count upto 10000
- Build code
  - $ make

```bash
os2 ~/cs444/s21/lock-example-master 146% make
gcc -o lock lock.c -std=c99 -g -Wno-implicit-function-declaration -O2 -lpthread
```
Summary

• 5 Lock implementations
  • Naïve lock (bad_lock, not working)
  • xchg lock (test-and-set, slow)
  • cmpxchg lock (a fake test and test-and-set, still slow)
  • Software test and hardware test-and-set (fast!)
  • Hardware test-and-set with exponential backoff (faster!)

• Performance check
  • Total execution time
  • L1-dcache-load-misses
  • Compare the performance to pthread_mutex
If the lock variable is not 0

```c
if (LLL_MUTEX.TryLock (mutex) != 0)
{
    int cnt = 0;
    int max_cnt = MIN (max_adaptive_count (),
        mutex->__data.__spins * 2 + 10);
    do
    {
        if (cnt++ >= max_cnt)
        {
            LLL_MUTEX_LOCK (mutex);
            break;
        }
        atomic_spin_nop ();
    }
    while (LLL_MUTEX.TryLock (mutex) != 0);
```

Spins * 2 + 10... exp backoff!

Check if the lock variable is 0...

```c
#define atomic_spin_nop() __asm ("pause")
```
Lock is Slow

• Run While() internally

• Can block other threads

• We need to determine when and where to use lock
When Do We Need to Use a Lock?

- Write must be finished before the next load
- Many writers and one reader
  - Yes... many writers..
- Two writers and two readers
  - Yes, two writers...
- One writer and many readers
  - Not always if there is only one writer
Where Do We Need to Put a Lock?

• What will happen if a critical section is too big?

Four independent variables
Require a lock on updating these..

Wasting time for waiting for all
Small Critical Sections

Fast, but developer must take care of splitting critical sections..
General Practice

• Use lock only if it is required
  • Determine the case when you do not need a lock
    • Atomic read
    • Only one writer

• Use a small critical section
  • Critical section prohibits concurrent execution
  • Determine where do we share a variable
  • Wrap only the code that updates the shared variable

• Looks simple, but sometimes it’s difficult
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
Atomicity

Read

1 Thread 1::
2 if (thd->proc_info) { Time of check
3     ...
4     fputs(thd->proc_info, ...); Time of use
5     ...
6 }

7

8 Thread 2::
9 thd->proc_info = NULL; TOCTTOU

Write!
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**: In critical section, NO UPDATE
- **Time of use**: Do not have TOCTTOU!
- **Update!**: This will also block other threads that run line 5 while thread 2 updates thd->proc_info.
Ordering: Mozilla – Order 1

1   Thread 1::
2     void init() {
3         ...
4         mThread = PR_CreateThread(mMain, ...);
5         ...
6     }
7
8   Thread 2::
9     void mMain(...) {
10        ...
11        mState = mThread->State;
12        ...
13     }
Thread 2::
void mMain(...) {
    ...
    mState = mThread->State;  // Not Initialized...
    ...
}

Thread 1::
void init() {
    ...
    mThread = PR_CreateThread(mMain, ...);
    ...
}
How Can We Resolve the Ordering Issue?

• Use locks and conditional variables to force a specific ordering...
  • `pthread_cond_wait(cond, lock)`
    • Set `cond = 0`
    • You will release the lock
    • Wait until `cond == 1`
    • Acquire the lock again
  • `pthread_cond_signal(cond)`
    • `cond = 1`

```
5 Thread 1::
6    void init() {
7        ...
8            mThread = PR_CreateThread(mMain, ...);
9        // signal that the thread has been created...
10           pthread_mutex_lock(&mtLock);
11            mtInit = 1;
12           pthread_cond_signal(&mtCond);  // Sends Signal..
13           pthread_mutex_unlock(&mtLock);
14        ...
15    }

18 Thread 2::
19    void mMain(...) {
20        ...
21        // wait for the thread to be initialized...
22           pthread_mutex_lock(&mtLock);
23            while (mtInit == 0)
24                pthread_cond_wait(&mtCond, &mtLock);  // Waits condition..
25           pthread_mutex_unlock(&mtLock);
26        mState = mThread->State;
27        ...
28    }
```
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);`
Deadlock!
Deadlock: Example

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

Thread 2:
```c
pthread_mutex_lock(L2);
pthread_mutex_lock(L1);
```
How Can We Resolve Circular Dependency

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);
Circular Dependency

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

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

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

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