xv6ⓒ-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)
Locks

Carmi Merimovich

Tel-Aviv Academic College

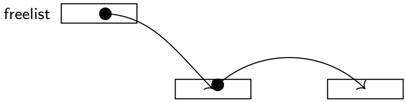
November 30, 2017

What is a uniprocessing risk with kalloc?

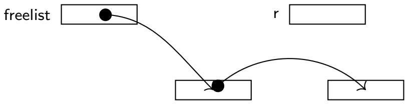
```
char *kalloc(void) {
 struct run *r;

 r = kmem.freelist;
 if (r)
  kmem.freelist = r->next;

 return (char*)r;
}
```

What might happen if kalloc is called within an hardware interrupt routine?

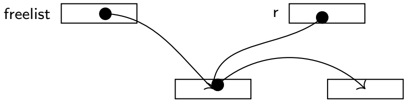
Uniprocessor **kalloc()** risks



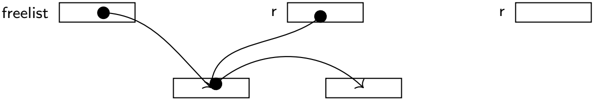
Uniprocessor **kalloc()** risks



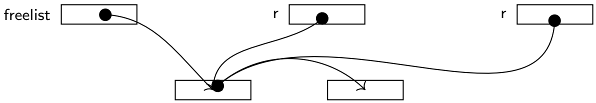
Uniprocessor **kalloc()** risks



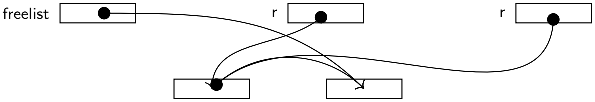
Uniprocessor **kalloc()** risks



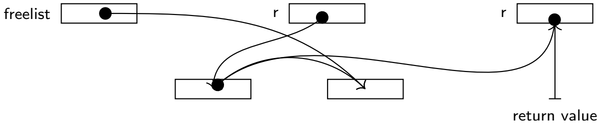
Uniprocessor **kalloc()** risks



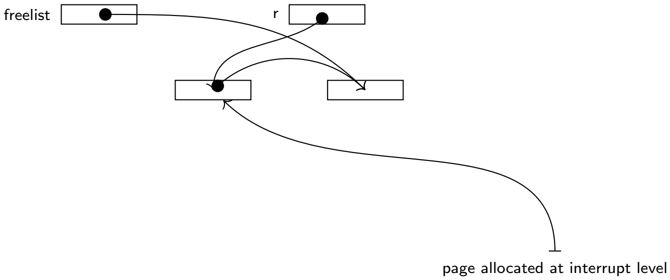
Uniprocessor **kalloc()** risks



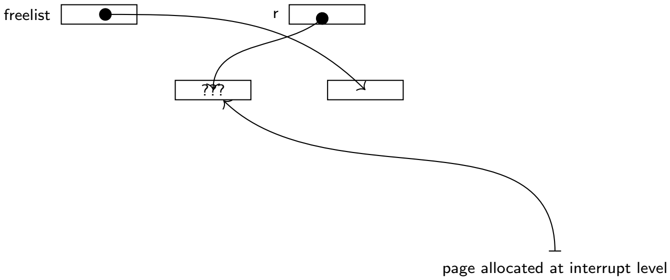
Uniprocessor **kalloc()** risks



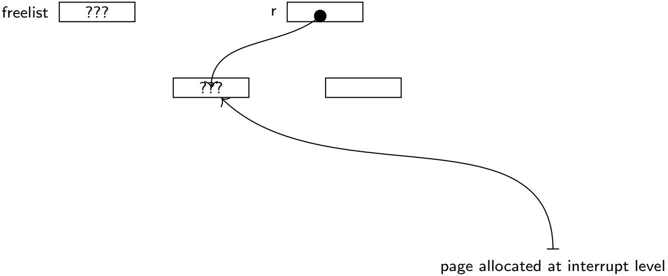
Uniprocessor **kalloc()** risks



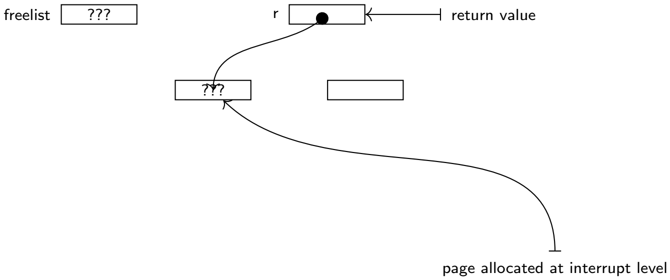
Uniprocessor **kalloc()** risks



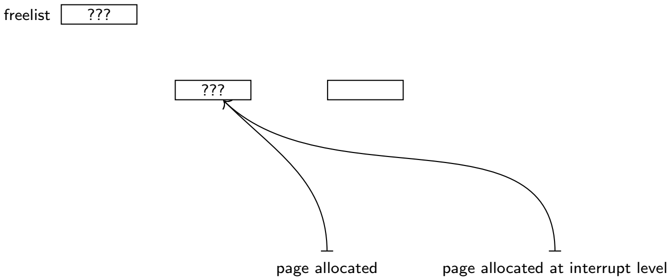
Uniprocessor **kalloc()** risks



Uniprocessor **kalloc()** risks



Uniprocessor **kalloc()** risks



Race condition

- When several execution contexts change common resource the behavior might be unpredictable.
- Simple solution:
  - Serialize changes to the common resource.
  - This decreases the amount of parallelisem.
  - Hence, performance is decreased.
- Complicated solution:
  - Discover parallel algorithm ...

Race Condition

Uniprocessor

Race condition demonstration

```
extern x=0,y=5;

x += y;
```

Figure: main code

```
x -= y;
```

Figure: interrupt service code

The compiled code is something like the following:

```
movl x,%eax
addl y,%eax
movl %eax,x
```

```
movl x,%eax
subl y,%eax
movl %eax,x
```

x value

- Assume the main code executes once.
- Assume the interrupt level code executes once.
- What will be the value of **x** at the end of execution?

Scenario 1

x	0
y	5

eax	?

```
.                      .
.                      .
.                      .
movl x,%eax            movl x,%eax
addl y,%eax            subl y,%eax
movl %eax,x            movl %eax,x
.                      .
.                      .
.                      .
```

Scenario 1

x	0
y	5

eax	?

```
.                     .
.                     .
.                     .
movl x,%eax           movl x,%eax
addl y,%eax           subl y,%eax
movl %eax,x           movl %eax,x
.                     .
.                     .
.                     .
```

Scenario 1

x	0
y	5

eax	0

```
.                  .
.                  .
.                  .
movl x,%eax        movl x,%eax
addl y,%eax        subl y,%eax
movl %eax,x        movl %eax,x
.                  .
.                  .
.                  .
```

Scenario 1

x	0	eax	5
y	5		

```
.                    .
.                    .
.                    .
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
.                    .
.                    .
.                    .
```

Scenario 1

x	5	eax	5
y	5		

```
.                  .
.                  .
.                  .
movl x,%eax        movl x,%eax
addl y,%eax        subl y,%eax
movl %eax,x        movl %eax,x
.                  .
.                  .
.                  .
```

Scenario 1

x	5	eax	5
y	5		

```
⋮                  ⋮
movl x,%eax        movl x,%eax
addl y,%eax        subl y,%eax
movl %eax,x        movl %eax,x
⋮                  ⋮
```

Scenario 1

x 5   eax 5

y 5

```
.                      .
.                      .
.                      .
movl x,%eax            movl x,%eax
addl y,%eax            subl y,%eax
movl %eax,x            movl %eax,x
.                      .
.                      .
.                      .
```

Scenario 1

x 5
y 5

eax 0

```
.                   .
.                   .
.                   .
movl x,%eax         movl x,%eax
addl y,%eax         subl y,%eax
movl %eax,x         movl %eax,x
.                   .
.                   .
.                   .
```

Scenario 1

x	0
y	5

eax	0

```
⋮                  ⋮
movl x,%eax        movl x,%eax
addl y,%eax        subl y,%eax
movl %eax,x        movl %eax,x
⋮                  ⋮
```

Scenario 1

x	0
y	5

eax	5

```
.                   .
.                   .
.                   .
movl x,%eax         movl x,%eax
addl y,%eax         subl y,%eax
movl %eax,x         movl %eax,x
.                   .
.                   .
.                   .
```

Scenario 1

x	0	eax	?
y	5		

```
.                    .
.                    .
.                    .
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
.                    .
.                    .
.                    .
```

x=0

Scenario 2

x 0   eax ?

y 5

```
.                     .
.                     .
.                     .
movl x,%eax           movl x,%eax
addl y,%eax           subl y,%eax
movl %eax,x           movl %eax,x
.                     .
.                     .
.                     .
```

Scenario 2

x	0	eax	?
y	5		

```
.                    .
.                    .
.                    .
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
.                    .
.                    .
.                    .
```

Scenario 2

x	0	eax	?
y	5		

```
.                   .
.                   .
.                   .
movl x,%eax         movl x,%eax
addl y,%eax         subl y,%eax
movl %eax,x         movl %eax,x
.                   .
.                   .
.                   .
```

Scenario 2

x	0	eax	0
y	5		

```
.                      .
.                      .
.                      .
movl x,%eax            movl x,%eax
addl y,%eax            subl y,%eax
movl %eax,x            movl %eax,x
.                      .
.                      .
.                      .
```

Scenario 2

x	0	eax	-5
y	5		

```
.                    .
.                    .
.                    .
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
.                    .
.                    .
.                    .
```

Scenario 2

x	-5
y	5

eax	-5

```
.                  .
.                  .
.                  .
movl x,%eax        movl x,%eax
addl y,%eax        subl y,%eax
movl %eax,x        movl %eax,x
.                  .
.                  .
.                  .
```

Scenario 2

x	-5	eax	?
y	5		

```
.                   .
.                   .
.                   .
movl x,%eax         movl x,%eax
addl y,%eax         subl y,%eax
movl %eax,x         movl %eax,x
.                   .
.                   .
.                   .
```

Scenario 2

x	-5	eax	-5
y	5		

```
.                 .
.                 .
.                 .
movl x,%eax       movl x,%eax
addl y,%eax       subl y,%eax
movl %eax,x       movl %eax,x
.                 .
.                 .
.                 .
```

Scenario 2

x	-5	eax	0
y	5		

```
.                      .
.                      .
.                      .
movl x,%eax            movl x,%eax
addl y,%eax            subl y,%eax
movl %eax,x            movl %eax,x
.                      .
.                      .
.                      .
```

Scenario 2

x	0		eax	0
y	5			

```
.                       .
.                       .
.                       .
movl x,%eax             movl x,%eax
addl y,%eax             subl y,%eax
movl %eax,x             movl %eax,x
.                       .
.                       .
.                       .
```

Scenario 2

x	0	eax	0
y	5		

```
⋮                    ⋮
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
⋮                    ⋮
```

x=0

Scenario 3

x	0	eax	?
y	5		

```
.                   .
.                   .
.                   .
movl x,%eax         movl x,%eax
addl y,%eax         subl y,%eax
movl %eax,x         movl %eax,x
.                   .
.                   .
.                   .
```

Scenario 3

x	0
y	5

eax	?

```
⋮                   ⋮
movl x,%eax         movl x,%eax
addl y,%eax         subl y,%eax
movl %eax,x         movl %eax,x
⋮                   ⋮
```

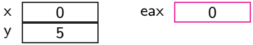
Scenario 3

x	0
y	5

eax 0

```
   .                  .
   .                  .
   .                  .
movl x,%eax        movl x,%eax
addl y,%eax        subl y,%eax
movl %eax,x        movl %eax,x
   .                  .
   .                  .
   .                  .
```

Scenario 3



```
.                      .
.                      .
.                      .
movl x,%eax            movl x,%eax
addl y,%eax            subl y,%eax
movl %eax,x            movl %eax,x
.                      .
.                      .
.                      .
```

Scenario 3

x	0	eax	0
y	5		

```
.                    .
.                    .
.                    .
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
.                    .
.                    .
.                    .
```

Scenario 3

x	0		eax	-5
y	5			

```
.                      .
.                      .
.                      .
movl x,%eax            movl x,%eax
addl y,%eax            subl y,%eax
movl %eax,x            movl %eax,x
.                      .
.                      .
.                      .
```

Scenario 3

```
x   -5        eax   -5
y    5
```

```
.                     .
.                     .
.                     .
movl x,%eax           movl x,%eax
addl y,%eax           subl y,%eax
movl %eax,x           movl %eax,x
.                     .
.                     .
.                     .
```

Scenario 3

x	-5
y	5

eax	0

```
⋮                  ⋮
movl x,%eax        movl x,%eax
addl y,%eax        subl y,%eax
movl %eax,x        movl %eax,x
⋮                  ⋮
```

Scenario 3

x	-5		eax	5
y	5			

```
.                      .
.                      .
.                      .
movl x,%eax            movl x,%eax
addl y,%eax            subl y,%eax
movl %eax,x            movl %eax,x
.                      .
.                      .
.                      .
```

Scenario 3

x	5
y	5

eax	5

```
.                    .
.                    .
.                    .
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
.                    .
.                    .
.                    .
```

Scenario 3

x	5	eax	5
y	5		

```
.                    .
.                    .
.                    .
movl x,%eax          movl x,%eax
addl y,%eax          subl y,%eax
movl %eax,x          movl %eax,x
.                    .
.                    .
.                    .
```

x=5

Hence:

- The above code is NONDETERMINSTIC.
- A simple solution is to (properly) SERIALIZE.

Solution: Masking external interrupts

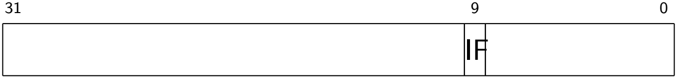


Figure: eflags

- If `eflags.IF==0` then external interrupts are ignored.
- if `eflags.IF==1` then external interrupts are servuced.

`eflags.IF` can be controlled with the following privileged instrutions:

- `cli`: `eflags.IF` ← 0.
- `sti`: `eflags.IF` ← 1.

Reading whole of `eflags` into (say) `%eax` can be done using:

```
pushfl
popl %eax
```

xv6 routines for controlling eflags.IF

```
static inline void cli(void) {
 asm volatile("cli");
}

static inline void sti(void) {
 asm volatile("sti");
}

static inline uint readeflags(void) {
 uint eflags;
 asm volatile("pushfl; popl %0" : "=r" (eflags));
 return eflags;
}
```

Serialization

```
extern x=0,y=5;

cli();
x += y;
sti();
```

Figure: main code

```
x -= y;
```

Figure: interrupt service code

- If the interrupt service is not interruptible, the solution is OK.

Serialize at interrupt level

```
extern x=0,y=5;

cli();
x += y;
sti();
```

Figure: main code

```
cli();
x -= y;
sti();
```

Figure: interrupt service code

- If the interrupt service is not interruptible, this solution is not OK.
- Using naked **sti** and **cli** is error prone.

Using cli and sti directly

Problem:

```
func b() {
 cli();
 ⋮
 sti();
}

func a() {
 cli();
 ⋮
 b();
 ⋮  // We are in trouble here
 sti();
}
```

Solution

We will not call `cli()` and `sti()` directly. Instead:

- We add a global `uint ncli`.
- On each `cli()` needed we call `pushcli()` which will:
  - `cli()`.
  - Increase `ncli`.
- On each `sti()` needed we call `popcli()` which will:
  - Decrease `ncli`.
  - On transition of `ncli` from 1 to 0, call `sti()`.

Solution code

```
int ncli=0; // Multiprocessors problem!

void pushcli(void) {
 cli();
 ncli++;
}

void popcli(void) {
 if (--ncli == 0)
  sti();
}
```

Uniprocessor interrupt service masking

Problem:

```
func b() {
 pushcli();
 ⋮
 popcli();
}

func a() {
 pushcli();
 ⋮
 b();
 ⋮
 popcli();
}
```

eflags

- In the above pushcli()/popcli() implementation it was implicit before the outermost pushcli() interrupts were enabled.
- This is not necessarily true (e.g., xv6 initialization vs. rest of the code).
- In order to handle this we add the following.
- On transition of ncli from 0 to 1 we save eflags.IF.
- On transition of ncli from 1 to 0 we restore eflags.IF.

Uniprocessor solution code

```
#define FL_IF 0x00000200
int ncli=0, intena; // Multiprocessors problem!

void pushcli(void) {
 int eflags = readeflags();
 cli();
 if (ncli++ == 0)
  intena = eflags & FL_IF;
}

void popcli(void) {
 if (--ncli == 0 && intena)
  sti();
}
```

Multiprocessor solution code

```
#define FL_IF 0x00000200

1655 void pushcli(void) {
      int eflags = readeflags();
      cli();
      if (mycpu()->ncli == 0)
       mycpu()->intena = eflags & FL_IF;
     mycpu()->ncli += 1;
     }

1667 void popcli(void) {
      if (--mycpu()->ncli == 0 && cpu->intena)
       sti();
     }
```

Race condition

Multiprocessors

What is the multiprocessing risk with `scheduler`?

```
for (p = ptable.proc; p < &ptable.proc[NPROC]; p++)
 if (p->state != RUNNABLE)
  continue;
```

Race condition demonstration

```
extern x=0;

x += 5;
```

Figure: code on processor 0

```
x -= 5;
```

Figure: code on processor 1

The compiled code is something like the following:

```
addl $5,x
```

```
subl $5,x
```

However, the RAM is passive so in reality the CPU microsteps are:

```
%tmp ← x;
%tmp ← %tmp + 5;
x ← %tmp;
```

```
%tmp ← x;
%tmp ← %tmp − 5;
x ← %tmp;
```

Atomic instruction on MP systems

- x86:

```
lock; xchgl mem, reg
```

- xv6 function:

```
static inline uint xchg(volatile uint *addr,
                        uint newval) {
  uint result;

  asm volatile("lock; xchgl %0, %1" :
               "+m" (*addr), "=a" (result) :
               "1" (newval) :
               "cc");
  return result;
}
```

Effect of the xchg function

- The call

  ```
  old = xchg(&lock, 1);
  ```

  achieves

  ```
  old  ← lock;
  lock ← 1;
  ```

  with no other processor executing other `locked` instructions in parallel.
- If we have recursive interrupts using `lock` then `pushcli()/popcli()` should be added.

spinlock structure

```
struct spinlock {
 uint locked; // Is the lock held?

 // For debugging:
 char *name; // Name of lock.
 struct cpu *cpu; // The cpu holding the lock.
 uint pcs[10]; // The call stack (an array of progra
 // that locked the lock.
};

void initlock(struct spinlock *lk, char *name) {
 lk->name = name;
 lk->locked = 0;
 lk->cpu = 0;
}
```

```
1574 acquire(struct spinlock *lk) {
  pushcli(); // disable interrupts to avoid deadlock.

  while(xchg(&lk->locked, 1) != 0);
}

1601 void release(struct spinlock *lk) {
  xchg(&lk->locked, 0);
  popcli();
}
```

Race condition reason and solution

- Resources shared by recursive interrupts serivce routines:
  - `pushcli()`.
  - `popcli()`.
- Resources shared by different processors:
  - `acquire()`.
  - `release()`.

Solving potential race condition in in `scheduler`.

scheduler

```
2758 void scheduler(void) {
      struct proc *p;
      for(;;) { sti();

        acquire(&ptable.lock);
        for(p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
          if (p->state != RUNNABLE) continue;

          proc = p;
          switchuvm(p);
          p->state = RUNNING;
          swtch(&cpu->scheduler, proc->context);
          switchkvm();

          proc = 0;
        }
        release(&ptable.lock);
      }
    }
```