CS 3650 – Computer Systems
Spring 2024
Peter Desnoyers

Lecture 15, Tue Feb 27 2024

Midterm exam logistics

conflicts: Wed 2:50-4:30 Cargill 097

proctor will have exam w/ your name

→ no notes or other material

section 3 exam - regular time / place

your own handwritten notes

(iPad etc? print them)

Exam review

* C programming language

- variable scope -
- arguments passed by value

```
int a = 5;
f(a)  ← copies 'a'
        (passes 5)

                       f(int x)
g(&a)                  :
                       g(int *x) {
                       *x += 1;
```

function can take address arg and change things that arg points to

argument:

```
f(int a) {
    'a' range of validity
}
```

local:

```
f(..) {
    int a;
    int b;
    'b' range of validity
}
    if () {
        int x
    }
}
```

* shell - simple commands & redirection

cd, pwd, echo, cat, ls, mkdir, grep <pattern>

read "input" - stdin or named files

"find" or "search"

same input style as cat - e.g.

```
grep pattern file
ls | grep pattern
```

$$\underbrace{> \quad < \quad |}_{\text{redirection}}$$

wildcards: *

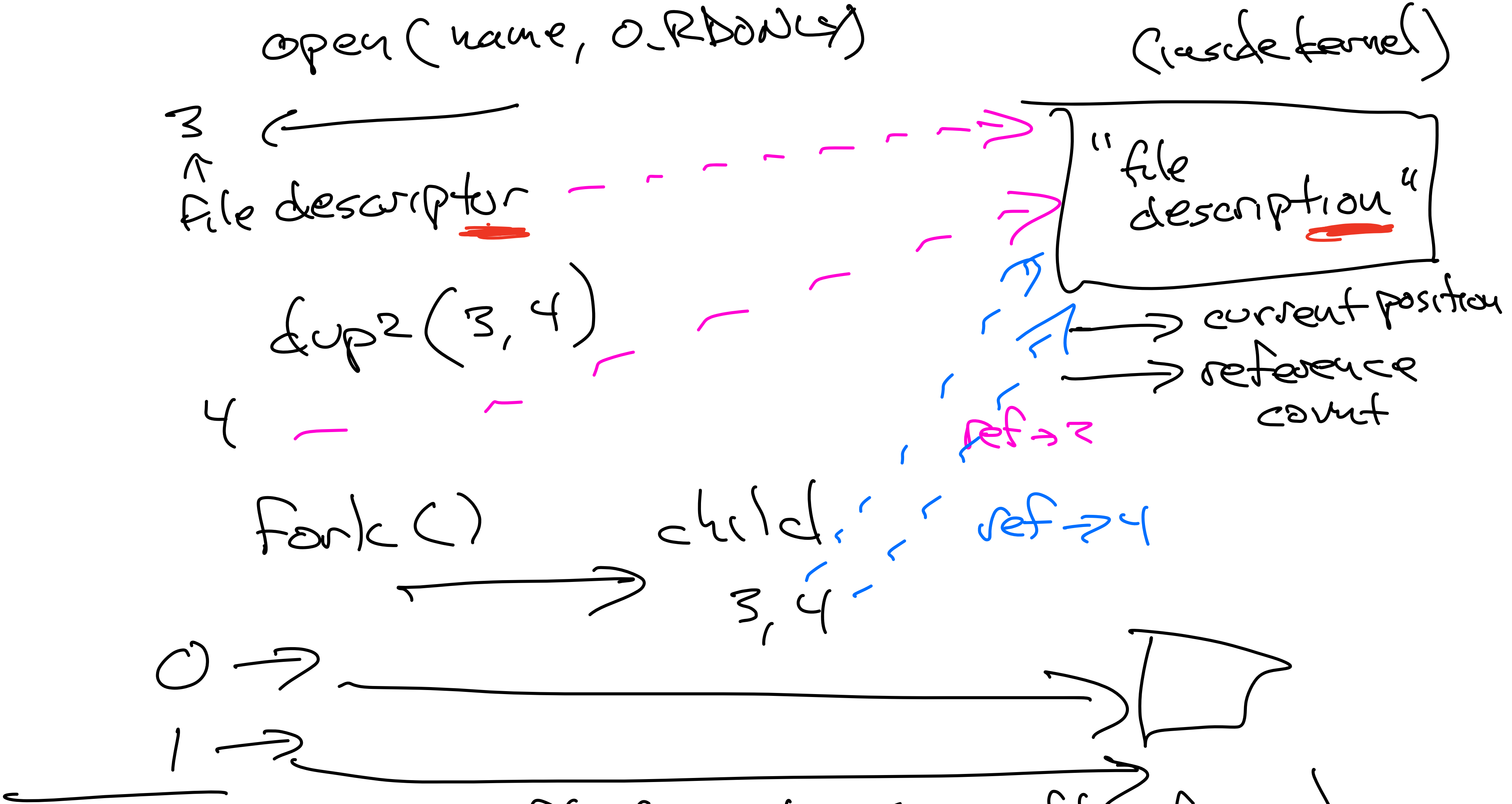
$\boxed{\texttt{ls *.c}}$ abc*

expands to all files matching <zero-or-more anything>.c

$$\underbrace{\langle\text{expression}\rangle *}_{0,\,1,\,\ldots\ \text{of}\ \langle\text{expression}\rangle}$$

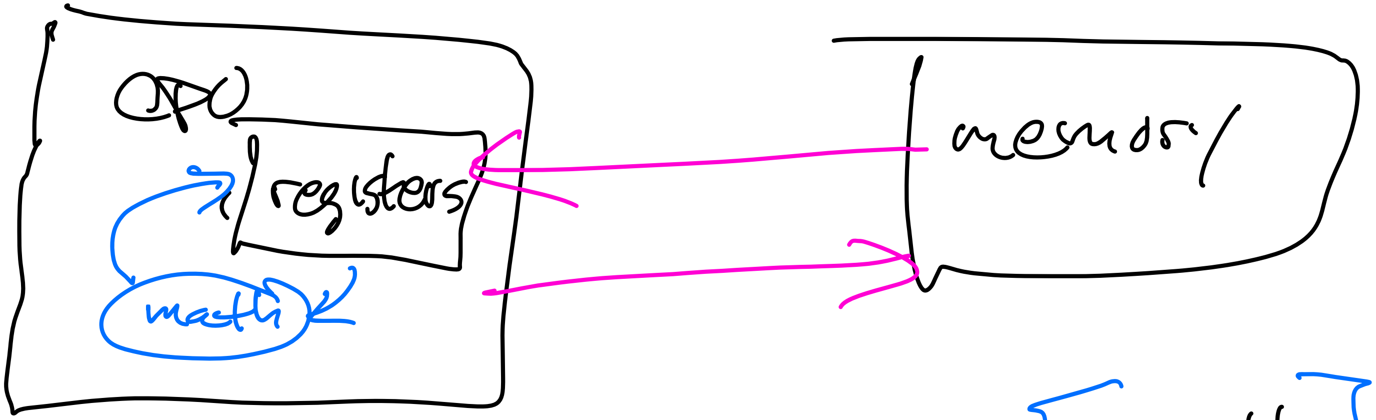
* file descriptors, dup2, etc

open(name, O_RDONLY)



2 ways to copy file descriptor (i.e. add reference)
① fork ② dup

* CPU & registers



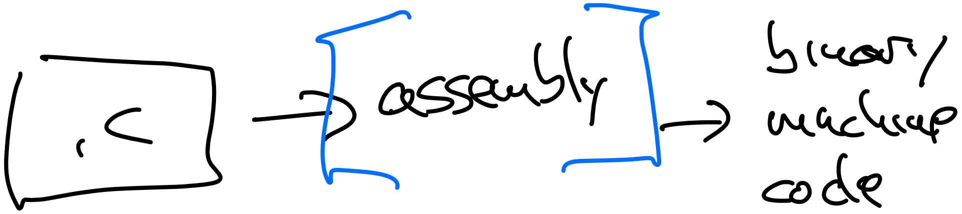
read mem → reg

reg → reg operations

write reg → Mem

CALL <addr>

RET



assembly ⇒ binary/
machine
code

* race conditions

10

deposit (sum) {

1) tmp=0 int tmp = balance

3) tmp=10 tmp += sum

5) bal = 10 balance = tmp

A

0

~~10~~ 30

30

dep (~~20~~)

2) int tmp = balance

tmp = 0

4) tmp = ~~20~~ tmp += sum

30

balance = tmp

6) bal = ~~20~~ 30

B

⇒ reason about 2 concurrent executions of same code

* monitors
(mutex + CV)

"Condition" = queue not empty

atomic

(mutex)

```
mutex_lock(m)
while (condition not true)
    wait(cv, m)
— do something
mutex_unlock(m)
```

locks mutex before returning

queue not empty

```
mutex_lock(m)
make condition true
signal(cv)
unlock(m)
```

avoiding TOCTTOU

time of check to time of use

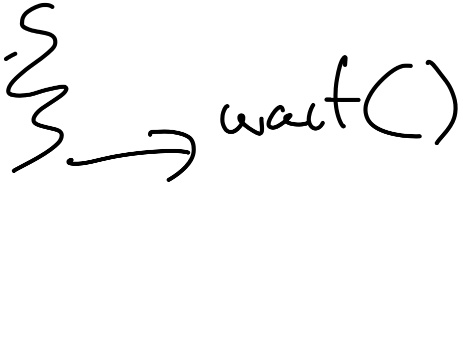
non - monitor

```
lock
while ~~if~~ condition not true {
    unlock
    wait
    lock
}
unlock
```

time of check

make condition T
signal

time of use



"give the sequence of ..."

```
deposit (sum) {
① int tmp = balance
② tmp += sum
③ balance = tmp
```

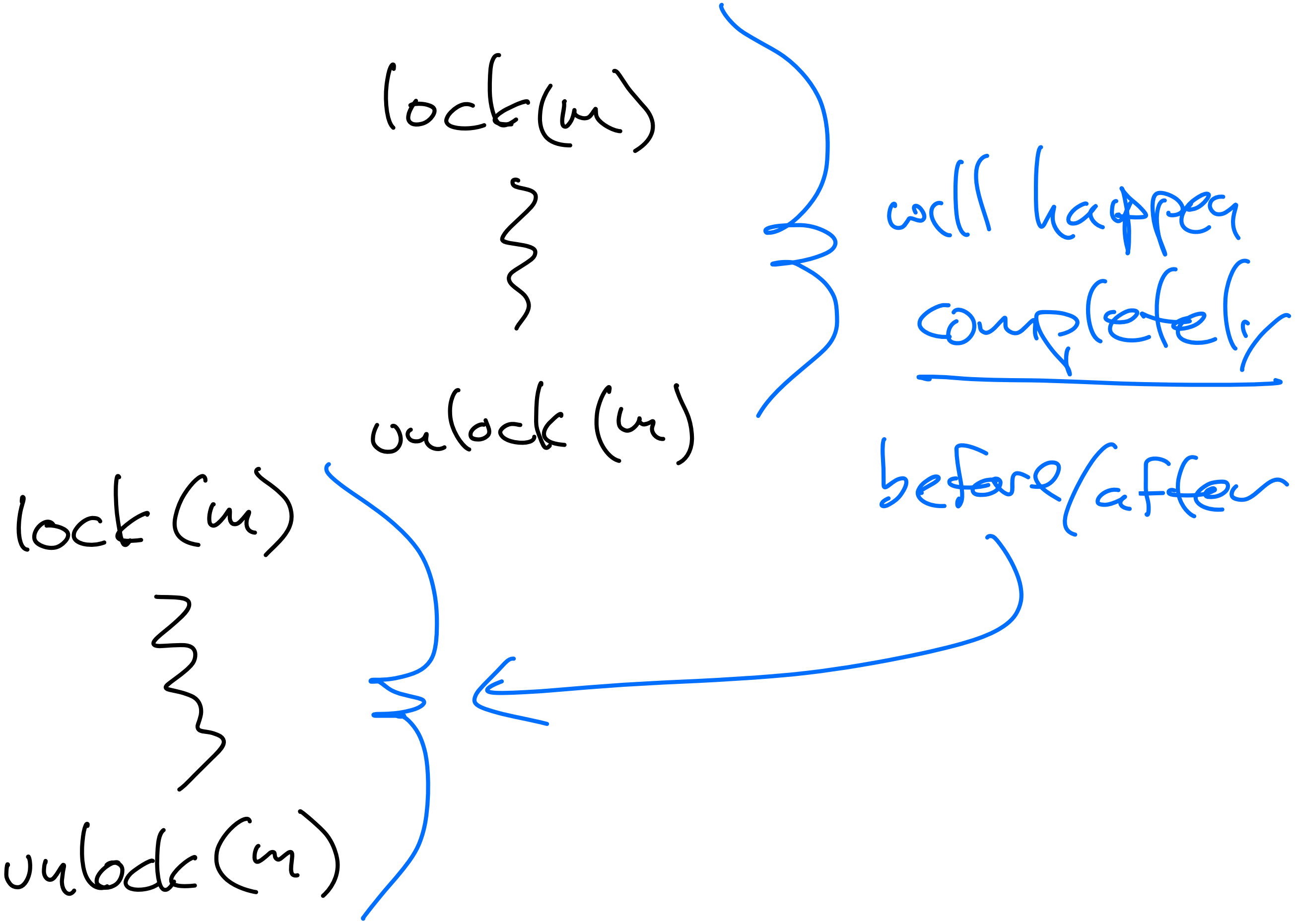
threads A, B

A1 ← thread A executes line 1

B2 ← B 2

A1 A2, A3, B1, B3, B2,

6 questions 16 points each + 4 free points



lock(m)

unlock(m)

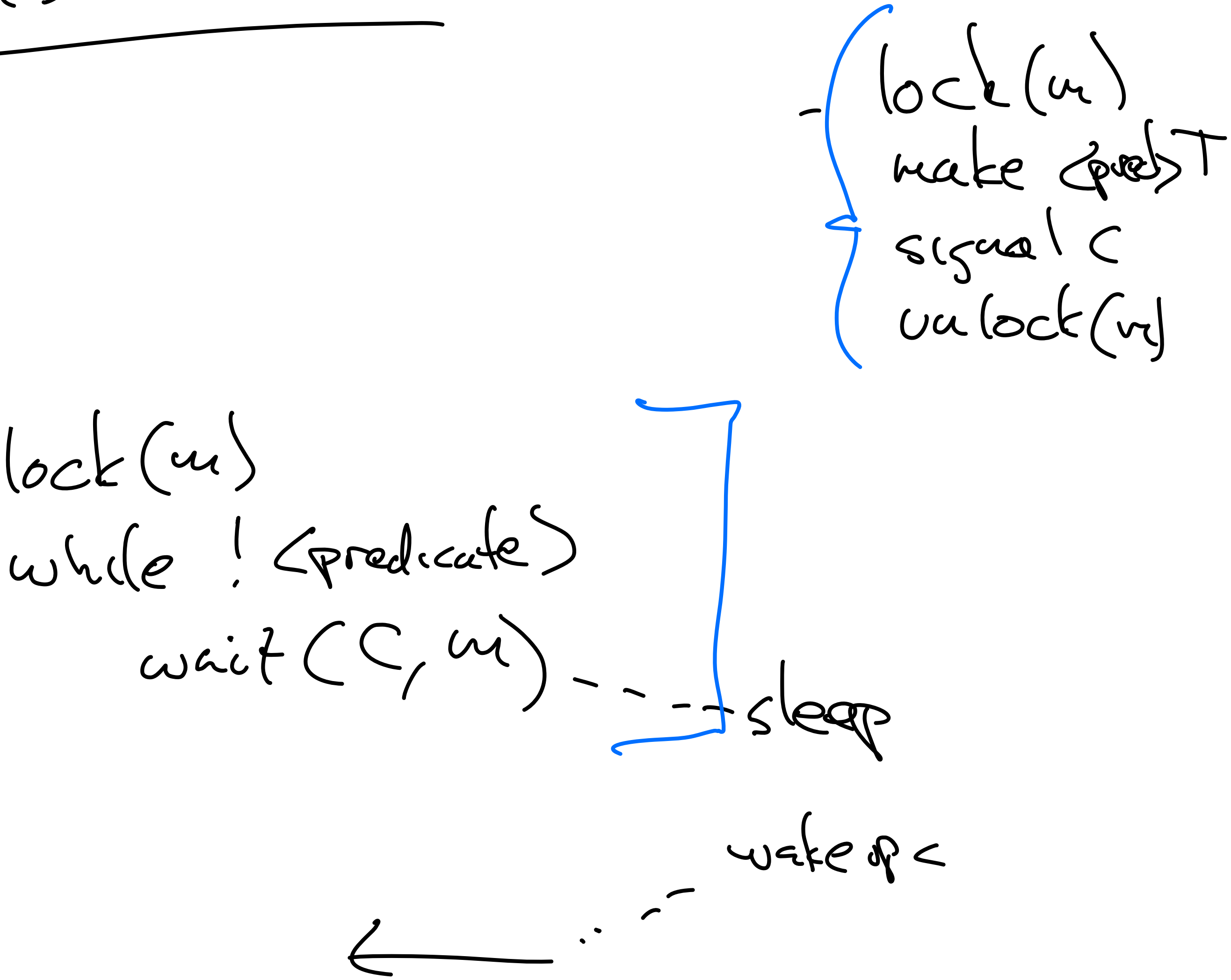
lock(m)

unlock(m)

will happen completely

before/after

it's not a monitor if there's no CV



```
lock(m)
make <pred> T
signal c
unlock(m)
```

```
lock(m)
while !<predicate>
    wait(C, m) - - - sleep
```

wakeup <