CSE 330: Operating Systems

Class: 11 Date: 9/22

Fall 2016

Note Title

Semaphores

- definition (by Dijkstra)
  - Busy wait
  - assumes atomicity
- Some usage

Implementation

- no busy waiting
- fair, no starvation
- atomicity needs...

```
type semaphore
        — struct
            [ int count;
              Queue of TCBs — Q;
            ]

Init(Sem,          ;    P(Sem) ;   V(Sem)
        value);
```

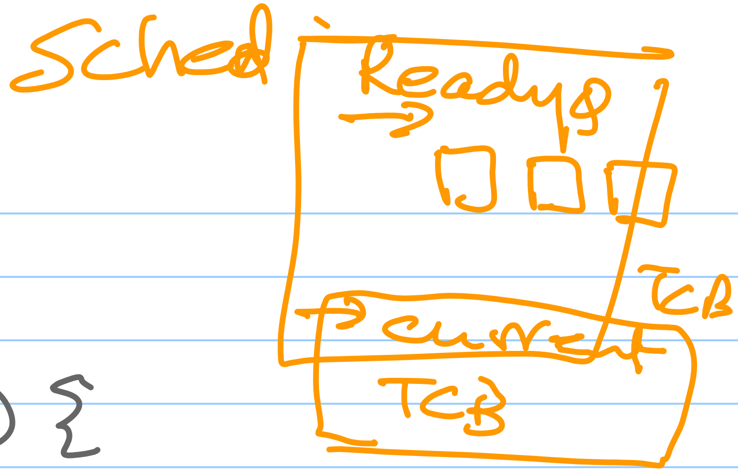
```
Init(Sem, v)
        { Sem.count = v }
```

```
P(Sem)
  { Sem.count --;
      if(Sem.count < 0) {
            - put currentTCB into
              Sem.Q
            - delete 1 TCB from ReadyQ
            - select as currentTCB
            - context switch }
  else ~ end atomic
```

start atomic

block current ~~process~~ thread

end atomic



①

start_atomic → DISABLE

end_atomic - ENABLE

②

test&set(semlock)

semlock = 1

```
V(Sem) {  Startatomic
          Sem.count++
          if (Sem.count <= 0)
             { • get one TCB from
                 Sem.Q (delete it)
unblock
               • add it to the
                 Ready Q
             }
          end atomic
}
```

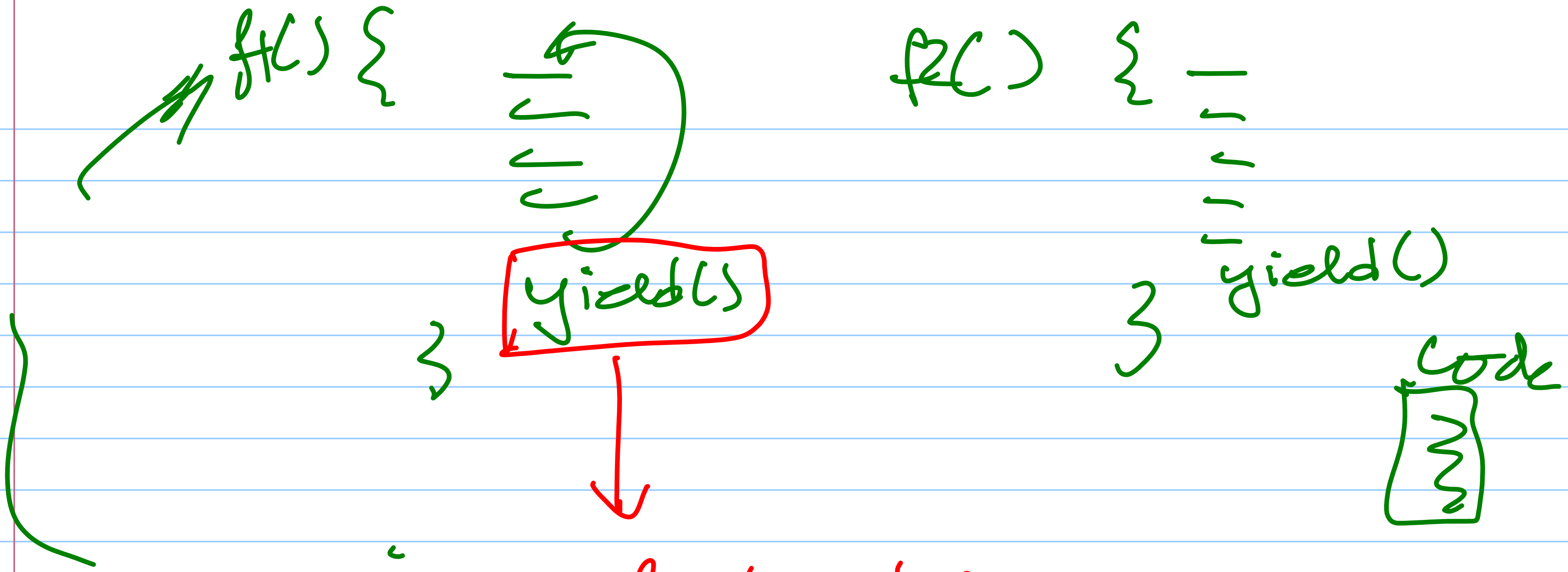
```
int global=0;
f1() { int local = 0;
          ∞ loop
                    global++; local++
                    print(global, local);

f2()
   →        { int local = 0;
                  global, local ++
                  print }

    main() { runs both f1 & f2 }
```

We need

TCBs and a queue of TCBs

functions to

- initialize TCBs & Queues
  and add/delete TCBs from/to Qs
- context switcher

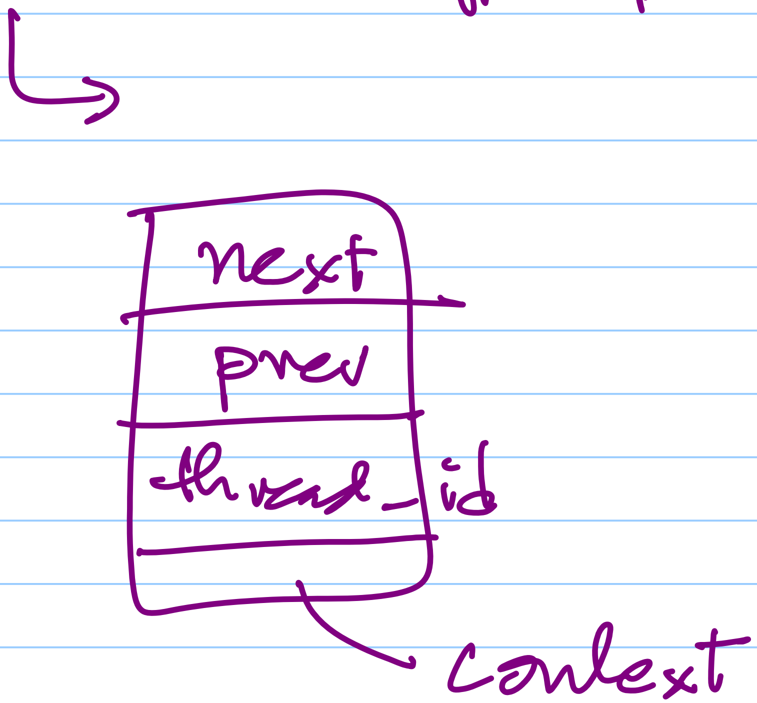


- Context switcher or scheduler

```
main()
  {
     startthread(f1);
        "       (f2);

       Run();
  }
```

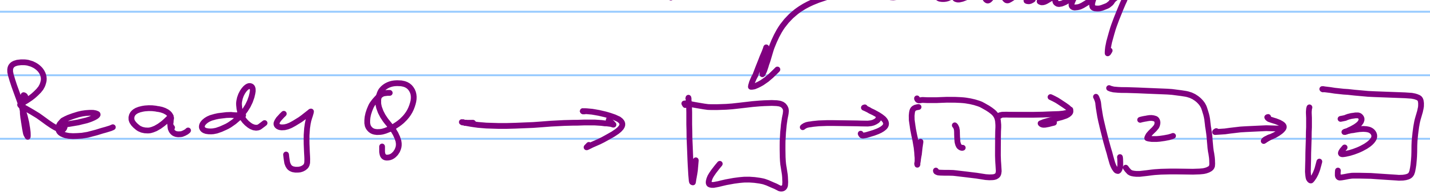
runs the threads
& never returns

tcb_t (typedef)



ucontext.h

- type ucontext()
- getcontext()
- swapcontext()
- makecontext()



```
startthread(function)
    ↳ { malloc a TCB → *tcb
        malloc a stack → *int
        initialize the TCB
            ↳ [ some code ] ← sets up stack in context
                makecontext(tcb.context,
                            function)
        ADD the TCB to ReadyQ
      }
```

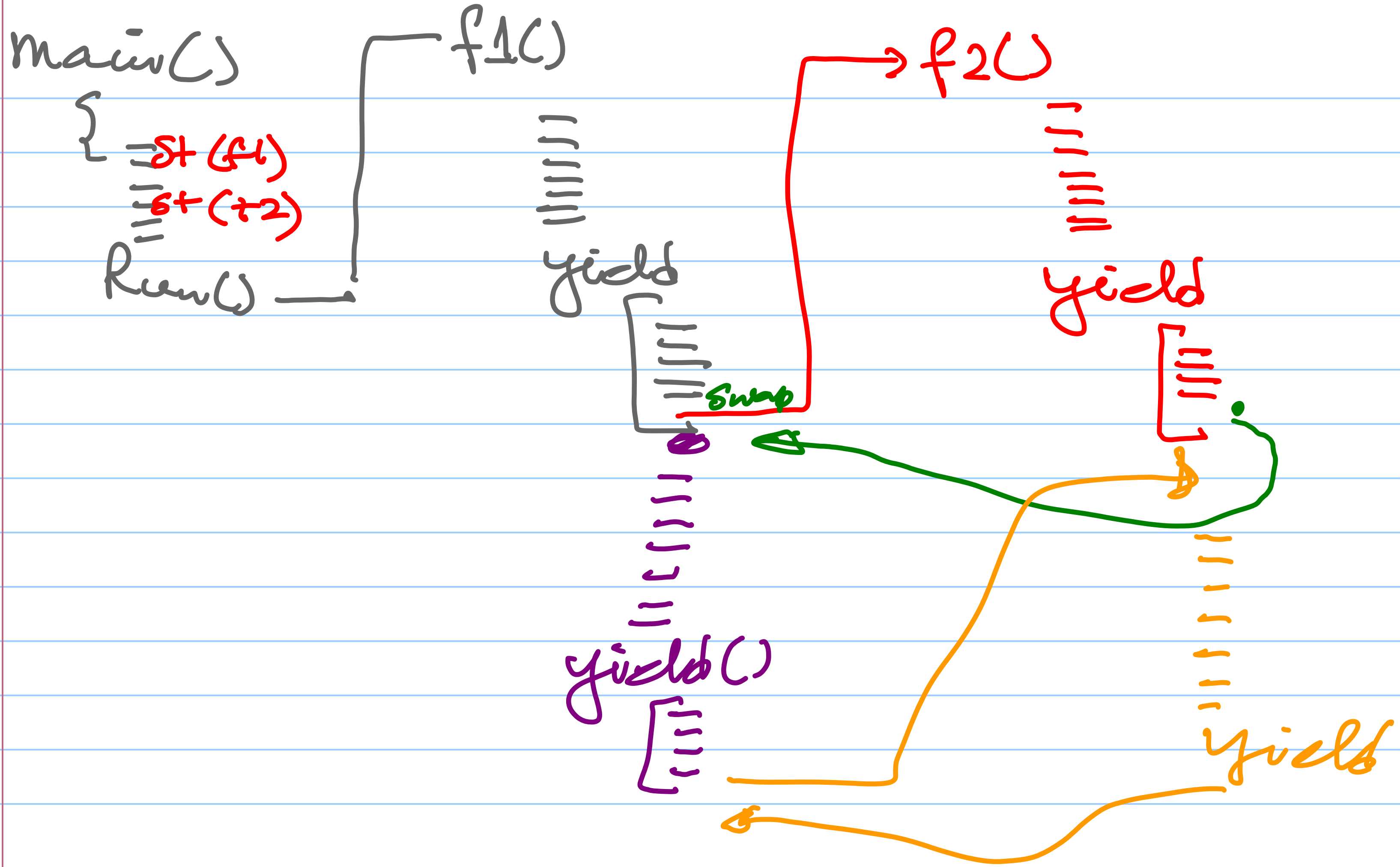
```
yield()
  { tcb * prev, next;                          global
      prev = current_thread
      next = deleteQ(ReadyQ)
      current_thread = next
AddQ  swapcontext
(ReadyQ,   (prev.context, next.context)
  prev)
      }
```

```
Run ()
  { u_context parent;
    getcontext (&parent)
    swapcontext (parent,
                          current_thread
                                 ->context);
```

dummy context

current_thread
    = DelQ(ReadyQ)

runs this


main()
{
st(f1)
st(f2)
Run()
f1()
yield
f2()
yield
swap
yield()
yield