Week 5.a
CS5600
02/06 2023
https://naizhengtan.github.io/23spring/

1. Last time
2. More about scheduling
3. Scheduling problem today
4. Scheduling lessons and conclusions
5. Threads
6. Intro to concurrency
7. Memory consistency model

---

- Votes from last time:
  (with candidate >=5 votes)

  "Best Turnaround Time": STCF (46)

  "Best Response Time": RR (31), MLFQ (11), STCF (9)

  "Best Fairness": MLFQ (22), lottery (19), RR (7)

  "Most popular algorithm": MLFQ (24), lottery (10), STCF (9)

projector didn't work at the moment.

- Incorporating I/O

P1, P2: both CPU bound, run for a week
P3: I/O bound, loop
(1 ms of CPU, 10 ms of disk I/O)

MLFQ

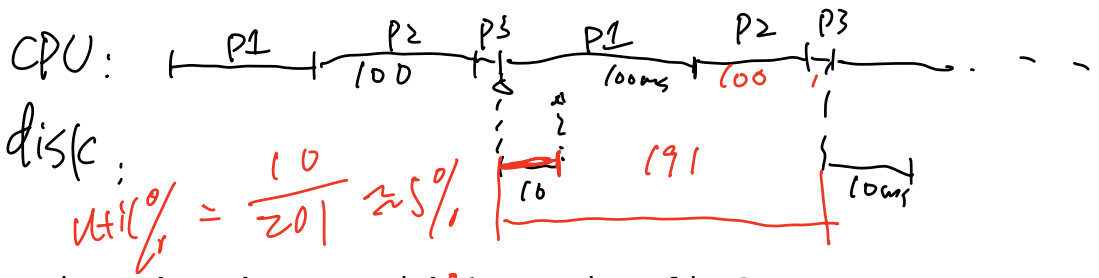
| process | arrival | running |
|---|---|---|
| P1 | 0 | 1 week |
| P2 | 0 | 1 week |
| P3 | 0 | 30 sec (with 300sec I/O) |


by itself, P3 uses ~90% of disk
By itself, P1 or P2 uses 100% of CPU

Question: what happens if we use FIFO? (arrival: P1, P2, P3)

A: you will get your handout in 2 weeks.

Question: what about RR with 100msec time slice?




Question: what about RR with 1msec time slice?

context switch 1000 times/sec

Question: what about STCF?

P2 starves.

Linux CFS

lottery ⇒ Stride algo

P1 ($t_1 = 20$): $S_1 = \frac{100}{t_1} = +5$

P2 ($t_2 = 10$): $S_2 = \frac{100}{t_2} = +10$

P1 P2 $P_1$ $P_1$ $P_2$ $P_1$ $P_1$ $P_2$

$P_1$ $P_1$: +2x5 = +10

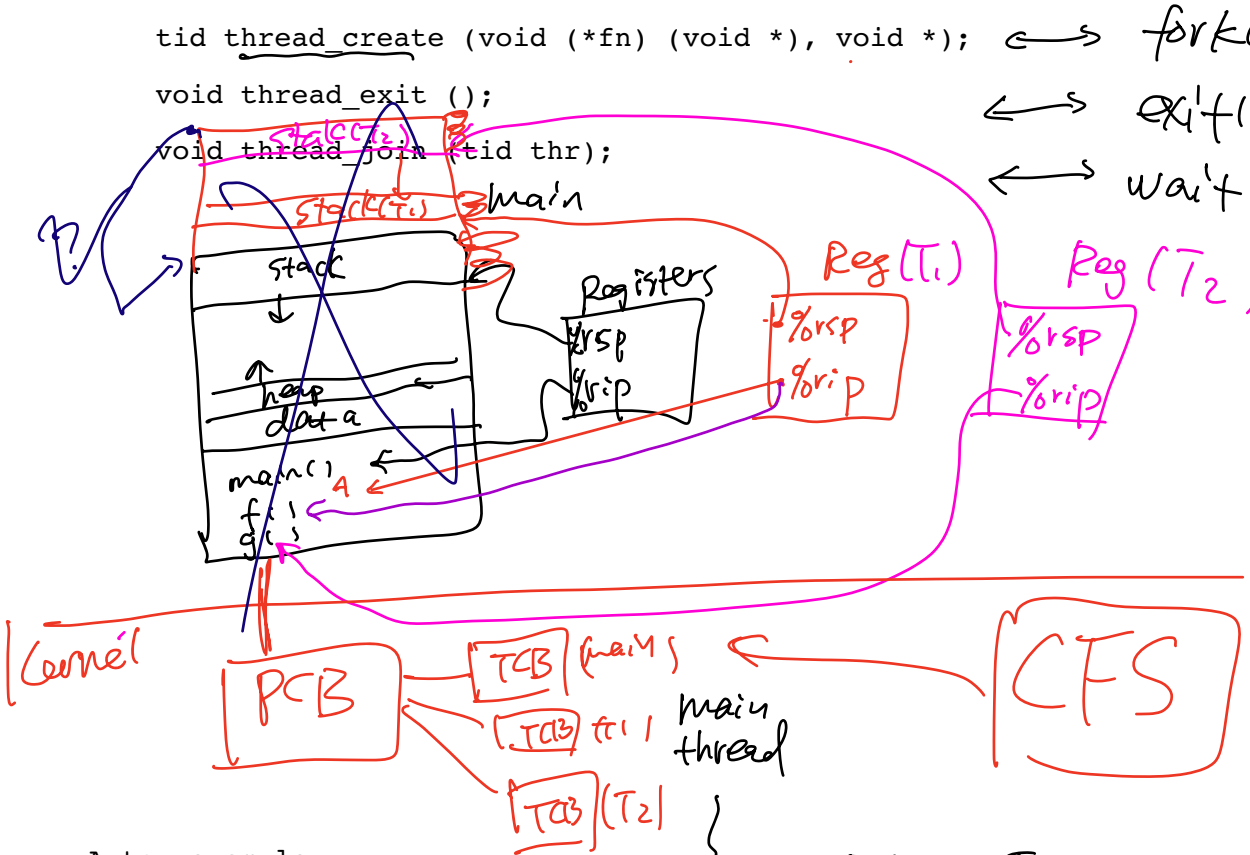
$P_2$: +10

---

Threads.

Interface to threads:

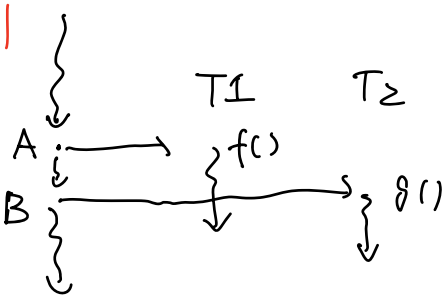
```
tid thread_create (void (*fn) (void *), void *);
void thread_exit ();
void thread_join (tid thr);
```




A toy example:

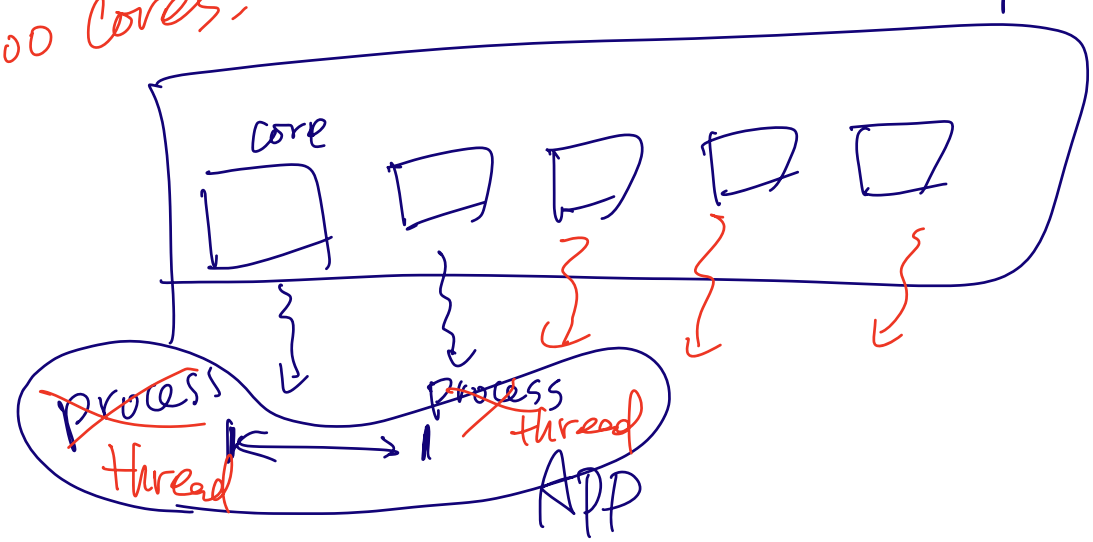
```
  void f() {...}
  void g() {...}

  int main() {
    thread_create(f, NULL)
    thread_create(g, NULL)
    ...
  }
```

//A
//B



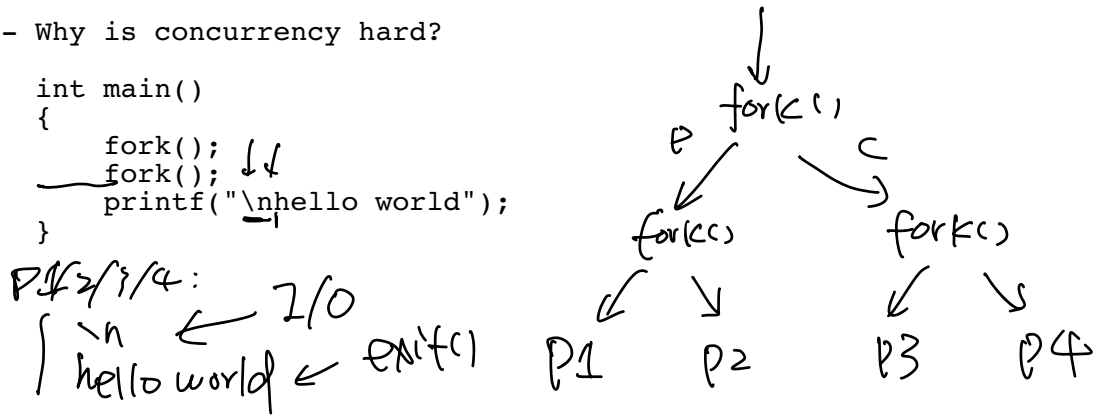

Chip

100 cores.

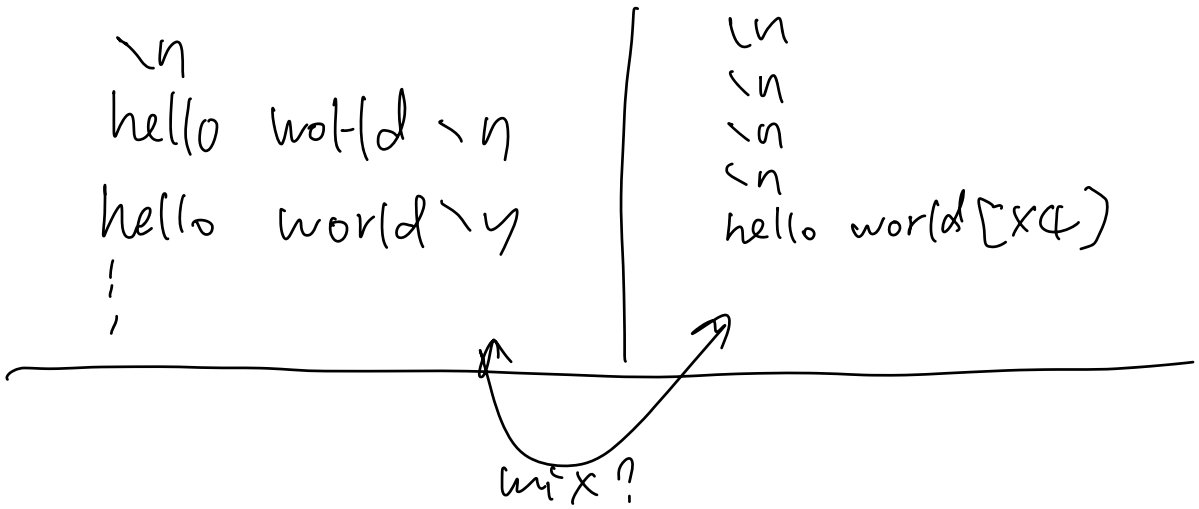


- Why is concurrency hard?

```
int main()
{
    fork();
    fork();
    printf("\nhello world");
}
```




Q: What's on your screen?




\n
helloworld

\n
helloworld

\n
helloworld

\n
helloworld

1. Example to illustrate interleavings: say that thread A executes f() and thread B executes g(). (Here, we are using the term "thread" abstractly. This example applies to any of the approaches that fall under the word "thread".)

a. [this is pseudocode]

```
int x;

int main(int argc, char** argv) {

    tid tid1 = thread_create(f, NULL);
    tid tid2 = thread_create(g, NULL);

    thread_join(tid1);
    thread_join(tid2);

    printf("%d\n", x);

}

void f() {
    x = 1;
    thread_exit();
}

void g() {
    x = 2;
    thread_exit();
}
```

What are possible values of x after A has executed f() and B has executed g()? In other words, what are possible outputs of the program above?

b. Same question as above, but f() and g() are now defined as follows

```
int y = 12;

f() { x = y + 1; }
g() { y = y * 2; }
```

What are the possible values of x?

c. Same question as above, but f() and g() are now defined as follows:

```
int x = 0;

f() { x = x + 1; }
g() { x = x + 2; }
```

What are the possible values of x?

A. x=3

B. x=1 or 2

C. x=1 or 2 or 3





2. Linked list example

```
struct List_elem {
    int data;
    struct List_elem* next;
};

List_elem* head = 0;

insert(int data) {
    List_elem* l = new List_elem;
    l->data = data;
    l->next = head;
    head = l;
}
```

What happens if two threads execute insert() at once and we get the following interleaving?

```
thread 1: l->next = head
thread 2: l->next = head
thread 2: head = l;
thread 1: head = l;
```