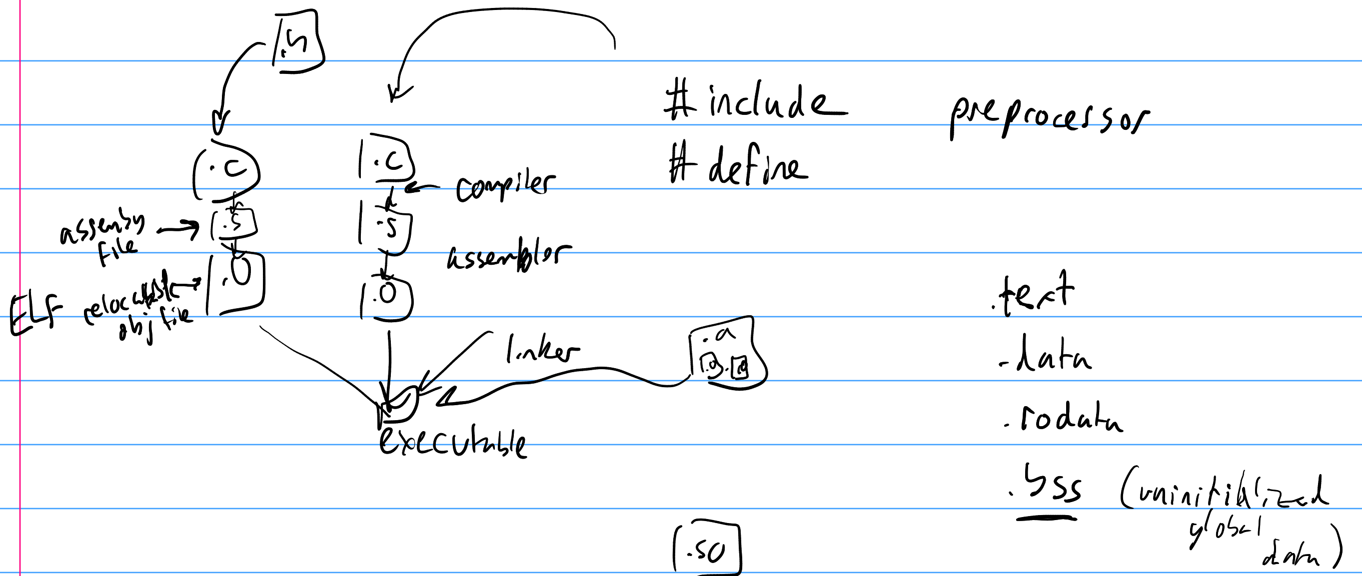
CS 3214 "multi threading" lecture #12



#include preprocessor
#define

.text
.data
.rodata
.bss (uninitialized global data)

```
int x;                          bss
const int x = 100;              .rodata
int x() { return 0; }           .text
void foo() { int x; ... }       stack / reg.
 static int x[100];             .bss
 extern void x(void);           nowhere
```

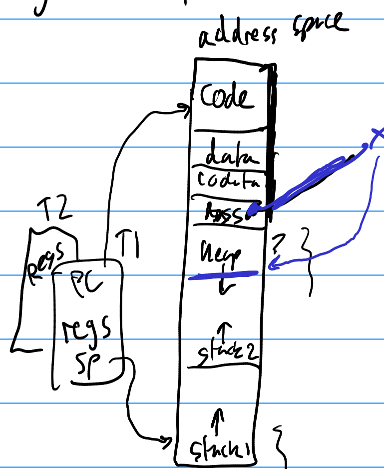
[Multi-threading]

→ Concurrent

"thread of execution"
logical unit

- memory
- registers PC, SP
- stack

in the same address space



[int *x;]
int main() {
x = malloc(8);

Shared:
- code
- data
- heap
- open fds

not shared
- regs
- stack

vdso

Processes don't share
- regs
- stack
- address space

CAN share
- shmem (shared memory)
- inherit fds
- mapped files

Do the kernel need to help to implement threads? NO

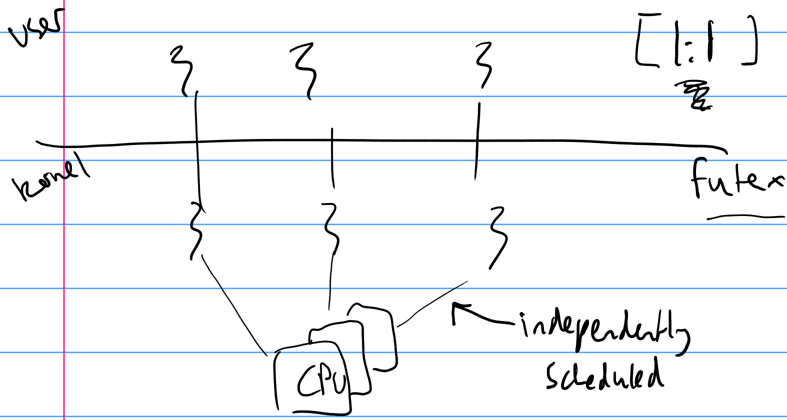
Cooperative multithreading
- "co-routines" or "user-level threads" or "green threads"
- non-preemptive
- yield
  - directed "yield to"
  - undirected "whoever"
- save/restore register state

Higher-level language
- yield temp result
- yield to a promise object for async I/O async/await

Pros
+ no syscall overhead
+ OS support not req'd
+ async I/O

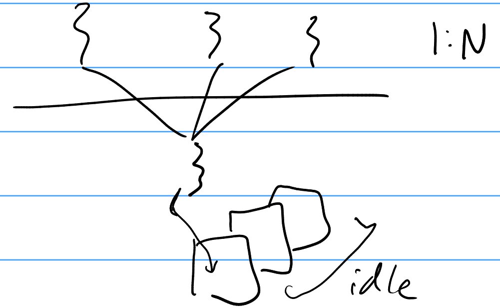
Cons
- no preemption for running thread
- multiple CPUs: others are idle
- block on I/O?

Kernel supported threads
+ multiple CPUs, OS can schedule
+ preempt running thread
+ blocking for I/O handled by kernel

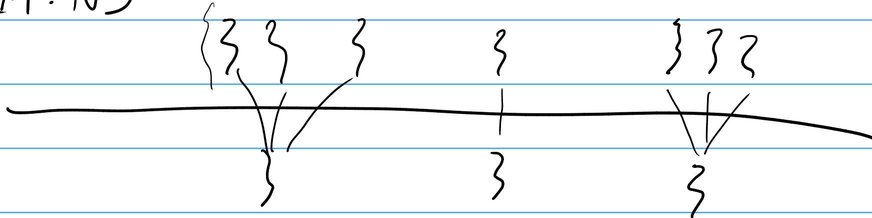


clone(SHARE_MM

"user-level threads"



[M:N]



Solaris

windows fibers

Posix threads pthreads

<u>why multiple threads?</u>

- use the cpus you have
- parallel computation
- UI
- I/O computation overlap