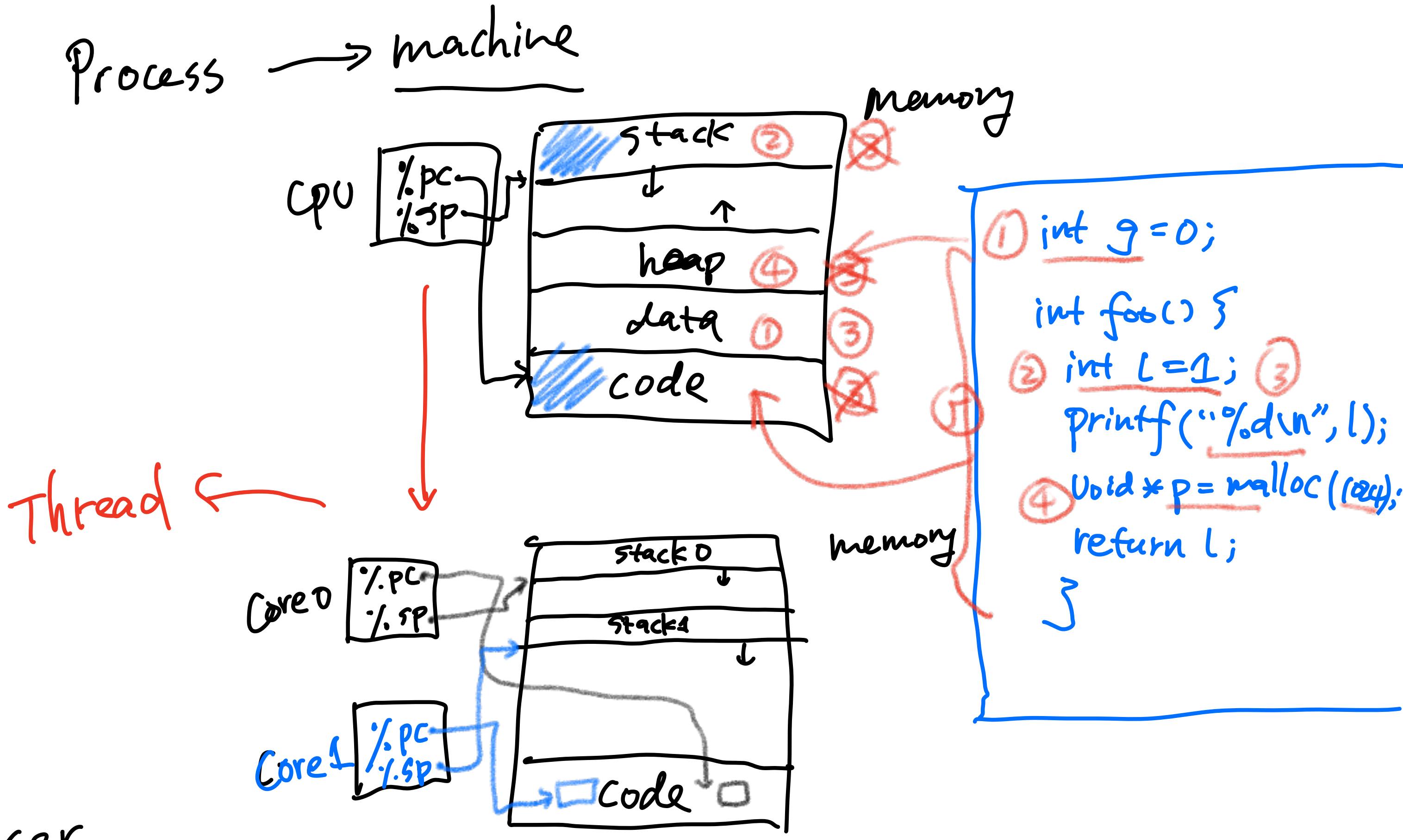
Week 2
CS4973/CS6640
01/13 2025
https://naizhengtan.github.io/25spring/

☑ 0. Admin, lab1, and lab2
☑ 1. Review: threading ←
☑ 2. Context switches in user-space
☐ 3. Cooperative vs. Preemptive multithreading in user-level } skip
☐ 4. Kernel debugging
☑ 5. Memory layout in egos ←
☑ 6. gdb ←

- office hour
- lab1

- PATH
- 0x2000 0000

- Review



```
int g=0;
int foo() {
  int l=1;
  printf("%d\n", l);
  void *p = malloc(1024);
  return l;
}
```

user

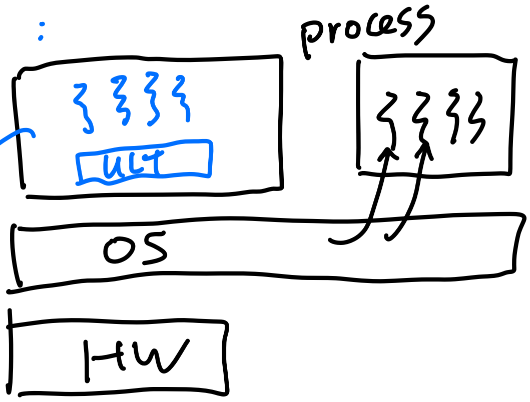
kernel

struct PCB {
- registers
- memory
- fd array
}

→

struct TCB {
sp
}
↳ one thread

ULT:



1. manage threads (TCBs)
2. create threads (allocate stack)
3. scheduling
4. context switch (!)

- Context switch in user space

↳ states process context

A. %PC
B. %SP
C. stack
D. heap
E. other process memory

thread context

- A, B, C,
- A, B, C, D
- A, B, D
- [✓ A, B]
- A, B. (all other registers)

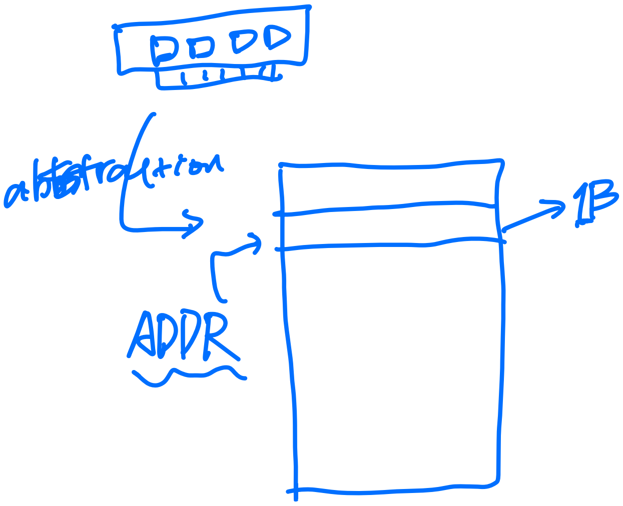
(ULT)

(ULT)

VM, Container, process, thread, fiber/coroutine

"heavy" ⟷ "light"

- Memory



sifive_e

OSI Handout Week02.a

1. Background: RISC-V assembly I

a) registers
   [see "RISC-V registers" in reference page]

b) addi rd, rs1, immediate   opcode [oprands]

   rd = rs1 + immediate

c) sw rs2, offset(rs1)

   Memory[rs1 + offset] = rs2

d) mv rd, rs1

   rd = rs1

e) call rd, symbol   %pc → call foo   %ra = pc+4

   rd = pc+4
   pc = &symbol

   If rd is omitted, ra is implied.   foo

f) ret

   pc = ra   pc → ret

2. Context switch in user-space:

a) void ctx_start(void** old_sp, void* new_sp);

This will be used when starting a new thread.
It will save registers on the old stack,
store current stack pointer to "old_sp",
switch stack to the "new_sp",
and
finally call ctx_entry().

b) void ctx_switch(void** old_sp, void* new_sp);   implementation

This will be used for context switch.
It will save registers on the old stack,
store current stack pointer to "old_sp",
switch stack to the "new_sp",
and
restore registers from the new stack,
finally return (to ra).

thread

.new_sp = SOMEWHERE

void * cur_sp;

ctx_switch(&cur_sp, newsp)

int a = 0;

3. grass/context.S

```
ctx_start:
    addi sp,sp,-64
    sw s0,4(sp)         /* Save callee-saved registers */
    sw s1,8(sp)
    sw s2,12(sp)
    sw s3,16(sp)
    sw s4,20(sp)
    sw s5,24(sp)
    sw s6,28(sp)
    sw s7,32(sp)
    sw s8,36(sp)
    sw s9,40(sp)
    sw s10,44(sp)
    sw s11,48(sp)
    sw ra,52(sp)        /* Save return address */
    sw sp,0(a0)         /* Save the current stack pointer */
    mv sp,a1            /* Switch the stack */
    call ctx_entry      /* Call ctx_entry() */

ctx_switch:
    addi sp,sp,-64
    sw s0,4(sp)         /* Save callee-saved registers */
    sw s1,8(sp)
    sw s2,12(sp)
    sw s3,16(sp)
    sw s4,20(sp)
    sw s5,24(sp)
    sw s6,28(sp)
    sw s7,32(sp)
    sw s8,36(sp)
    sw s9,40(sp)
    sw s10,44(sp)
    sw s11,48(sp)
    sw ra,52(sp)        /* Save return address */
    sw sp,0(a0)         /* Save the current stack pointer */
    mv sp,a1            /* Switch the stack */
    lw s0,4(sp)         /* Restore callee-saved registers */
    lw s1,8(sp)
    lw s2,12(sp)
    lw s3,16(sp)
    lw s4,20(sp)
    lw s5,24(sp)
    lw s6,28(sp)
    lw s7,32(sp)
    lw s8,36(sp)
    lw s9,40(sp)
    lw s10,44(sp)
    lw s11,48(sp)
    lw ra,52(sp)        /* Restore return address */
    addi sp,sp,64
    ret
```

sp = sp + (-64)

pc

a0

old_sp → %sp
void*

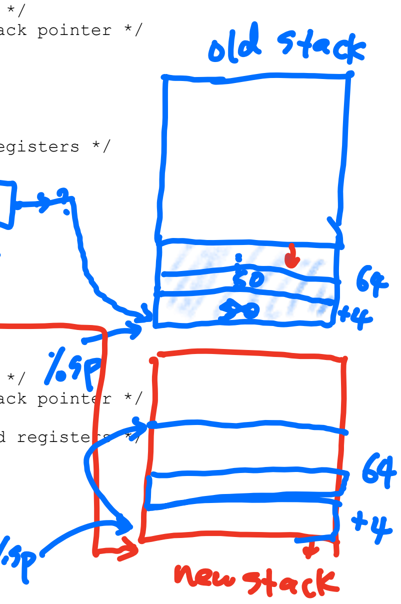
a1
new_sp

%sp = a1

50 ←

?

pc = ra



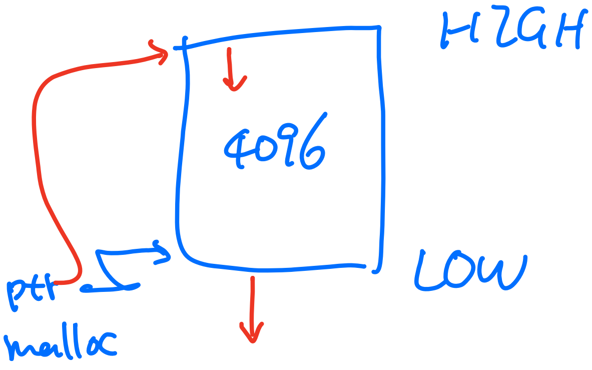
4. An example use of ctx_start+ctx_entry

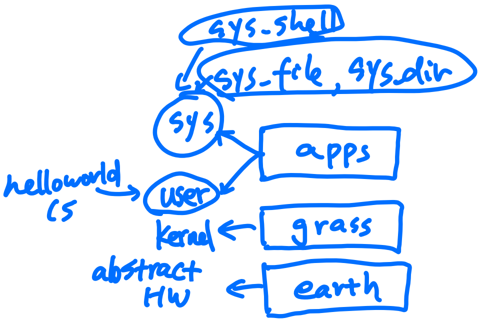
```
void thread_create(void (*f)(void *), void *arg, unsigned int stack_size) {

  tcb = create_thread_control_block();
  old_tcb = current_running_thread_control_block();

  … // do something necessary

  void **old_sp = …   // old stack pointer’s address in old_tcb
  void *new_sp =  …   // new stack pointer in tcb

  ctx_start(old_sp, new_sp);
}


void ctx_entry(void){

    … // do something useful

    (*f)(arg);  // run function “f” received by “thread_create”

    … // wrap up
}
```

egos gdb:

- break at main of ult.c → check pid
- exception → something is wrong

* "normal" C program
  + you do not need to understand hardware details (like CPU)
  + you have clear error messages
  + you do not have to worry about touching important memory
    (the program will be killed)
  + you do not use addresses directly
  + you have a nice address space containing your program only
  + you have a lot of tools (like IDE)





OSI Handout Week02.b

1. egos-2k+ memory layout (for the sifive_e CPU)

```
HIGH MEM ADDR
------- +----------------------+ <- 0x8040_0000
        |                      |    [FREE_MEM_END]
 DTIM   | free memory          |
memory  | (4MB - 16KB)         |
 (4MB)  +----------------------+ <- 0x8000_4000
        | earth interface      |    [FREE_MEM_START]
        | (128B)               |
        +----------------------+ <- 0x8000_3f80
        | earth/grass stack    |    [GRASS_STACK_TOP]
        | (~8KB)               |
        \/\/\/\/\/\/\/\/\/\/\/\/

        /\/\/\/\/\/\/\/\/\/\/\/\
        | grass interface      |
        +----------------------+ <- 0x8000_2000
        | app stack            |    [APPS_STACK_TOP]
        | (6KB)                |
        +----------------------+ <- 0x8000_0800
        | system call args     |
        | (1KB)                |
        +----------------------+ <- 0x8000_0400
        | app args             |    [SYSCALL_ARG]
        | (1KB)                |
------- +----------------------+ <- 0x8000_0000
                                    [APPS_ARG]

        ...
------- +----------------------+ <- 0x0a00_0000
        |                      |    [ITIM_END]
        \/\/\/\/\/\/\/\/\/\/\/\/

        /\/\/\/\/\/\/\/\/\/\/\/\
        |                      |
        +----------------------+ <- 0x0820_4000
 ITIM   | app code+data        |    [APPS_ENTRY+APPS_SIZE]
 (32MB) | (16KB)               |
        +----------------------+ <- 0x0820_0000
        | grass code+data      |    [APPS_ENTRY]
        | (1 MB)               |
        +----------------------+ <- 0x0810_0000
        | earth data           |    [GRASS_ENTRY]
        | (1 MB)               |
------- +----------------------+ <- 0x0800_0000
LOW MEM ADDR                        [ITIM_START]
```

app2 app1 ...

Where is running app's heap?

0x8201758

2. gdb cheat sheet

```
Breakpoints & watchpoints
(gdb) break main           set a breakpoint on a function
(gdb) break ult.c:10       set breakpoint at file and line (or function)
(gdb) info breakpoints     show breakpoints
(gdb) delete 1             delete a breakpoint by number
(gdb) watch expression     set software watchpoint on variable
(gdb) info watchpoints     show current watchpoints

Running the program
(gdb) c                continue the program
(gdb) s                a step in C; step into functions
(gdb) si               a step in asm; step into functions
(gdb) n                a step in C; step over functions
(gdb) ni               a step in asm; but step over functions
(gdb) CTRL-C           actually SIGINT, stop execution of current program
(gdb) finish           finish current function's execution

Stack backtrace
(gdb) bt                   print stack backtrace
(gdb) info locals          print automatic variables in frame
(gdb) info registers       print registers sans floats

Browsing Data
(gdb) p expr               print expression
(gdb) p/x expr             print in hex
(gdb) p/t expr             print in binary
(gdb) p/i expr             print as instructions

(gdb) x/FMT address        low-level examine command
(gdb) x/x 0x80001000       print memory in hex
(gdb) set var = expr       assign value

(gdb) display/FMT expr     display expression result at stop
(gdb) display/i $pc        print next instruction
(gdb) undisplay            delete displays

FMT (Format letters) are:
  o(octal), x(hex), d(decimal), u(unsigned decimal),
  t(binary), f(float), a(address), i(instruction), c(char), s(string)
  and z(hex, zero padded on the left).

Load a program's symbols
(gdb) add-symbol-file <elf>    load symbol table from <elf>

History Display
(gdb) show commands            print command history

[borrowed and customized from
 https://gist.github.com/rkubik/b96c23bd8ed58333de37f2b8cd052c30]
```