OSI Handout Week4

1. Timer interrupt handler

```
void handler() {
    CRITICAL("Got a timer interrupt!");
    // (4) reset timer

}

int main() {
    CRITICAL("This is a simple timer example");

    // (1) register handler() as interrupt handler


    // (2) set a timer


    // (3) enable timer interrupt


    while(1);
}
```


2. Background: RISC-V assembly II

Assembler instructions with C expression operands:

asm(Template : OutputOperands : InputOperands)

a) Template: a string that is the template for the assembler code.

```
asm("mret");
```

b) OutputOperands: the C variables modified by the instructions in the Template.

```
void *sp;
asm("mv %0, sp" : "=r"(sp));
```

c) InputOperands: C expressions read by the instructions in the Template.

```
int mie;
asm("csrr %0, mie" : "=r"(mie));
asm("csrw mie, %0" ::"r"(mie | 0x80));
```

3. Machine-mode exception CSRs

a) **mstatus**
Machine Status, holds the global interrupt enable, along with a plethora of other state.

b) **mie**
Machine Interrupt Enable, lists which interrupts the processor can take and which it must ignore

c) **mcause**
Machine Exception Cause, indicates which exception occurred

d) **mtvec**
Machine Trap Vector, holds the address the processor jumps to when an exception occurs

e) **mepc**
Machine Exception PC, points to the instruction where the exception occurred

f) **mtval**
Machine Trap Value, holds additional trap information: the faulting address for address exceptions, the instruction itself for illegal instruction exceptions, and zero for other exceptions

g) **mip**
Machine Interrupt Pending, lists the interrupts currently pending

4. egos process management (sifive_e)

a) process control block (PCB)

[grass/process.h]

```
struct process {
    int pid;
    int status;
    int receiver_pid; /* used when waiting to send a message */
    void *sp, *mepc;  /* process context = stack pointer (sp)
                       * + machine exception program counter (mepc) */
    // scheduling attributes
    union {
        unsigned char      chars[64];
        unsigned int       ints[16];
        float              floats[16];
        unsigned long long longlongs[8];
        double             doubles[8];
    } schd_attr;
};
```

b) global process data structures

[grass/kernel.c]

```
int proc_curr_idx;
struct process proc_set[MAX_NPROCESS];
```

[grass/process.h]

```
#define curr_pid      proc_set[proc_curr_idx].pid
#define curr_status   proc_set[proc_curr_idx].status
```

c) process life cycles

[grass/scheduler.c]

life-cycle functions:

* **proc_on_arrive**(int pid): when creating pid

* **proc_yield**(): when deciding next running process

* **proc_on_stop**(int pid): when destroying pid

a process's life cycle:

```
proc_on_arrive() ->
  proc_yield() -> [other proc] -> [ctx_switch to this proc] ->
  proc_yield() -> [other proc] -> [ctx_switch to this proc] ->
  ...
-> proc_on_stop()
```