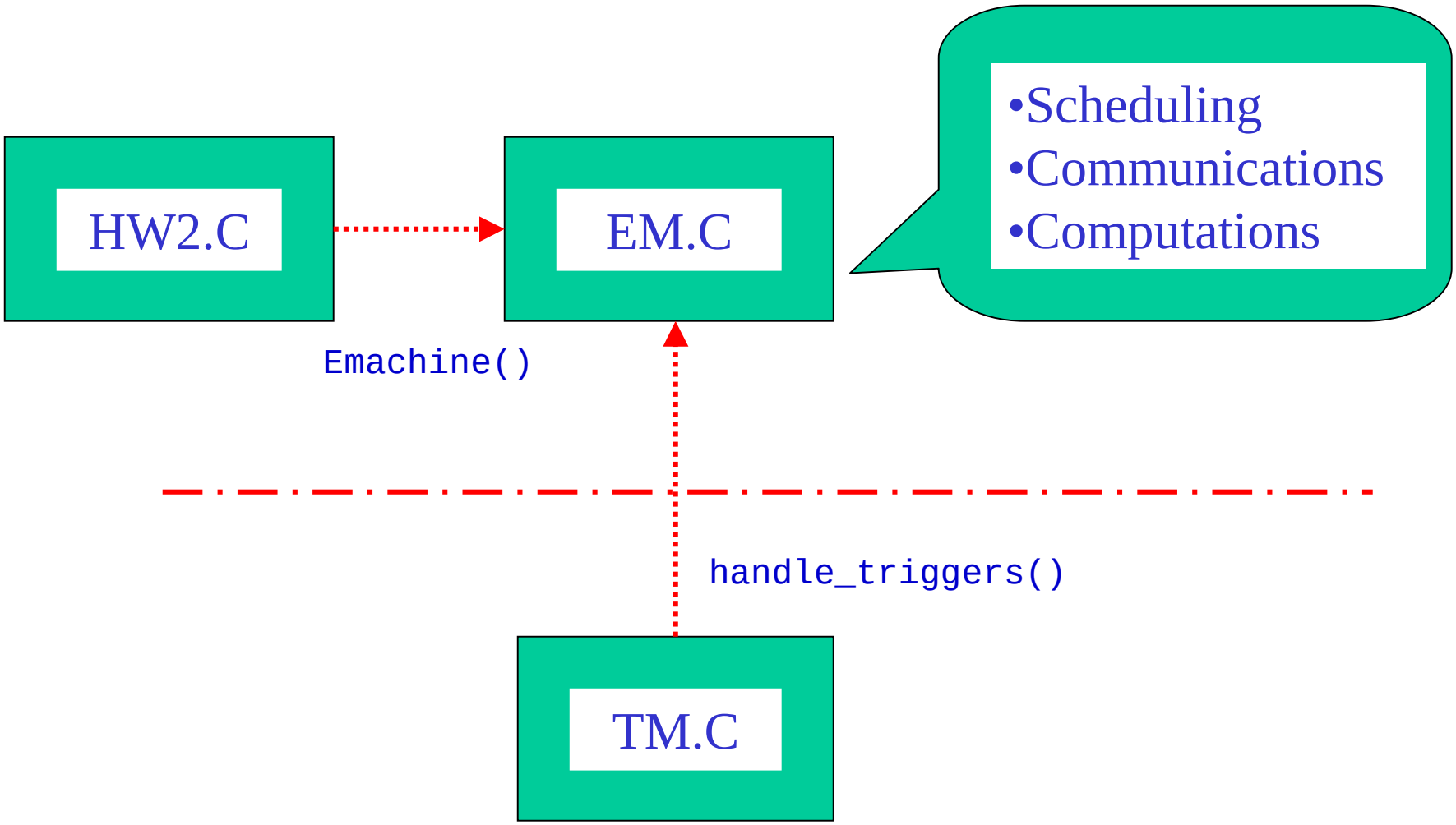
Code Review for Homework 2

Rahul Shah & Xuanming Dong

Feb 28, 2002

Architecture



EM.H

```
#include <config.h>

typedef enum { NOP, CALL, FUTURE, SCHEDULE } op_t;

typedef struct {
  op_t opcode;
  int arg1;
} inst_t;

typedef struct {
  inst_t* eco;
  int eco_size;
} emachine_t;

// EM SYSCALL (called by user program)
void Emachine(emachine_t*);
void Einterpreter();
void configure_em(emachine_t*);
void handle_triggers();

int fetch(inst_t**, int*);
int execute(int (*f) ());
```

EM.C -- Part 1

```
#include <em.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>
#include <dsensor.h>
#include <time.h>
#include <semaphore.h>

// #ifdef CONF_VIS
#include <sys/lcd.h>
#include <conio.h>
// #endif

const int eco_max_size = 55;
const int int_max_size = 20;
int ticks = 0;
int pc = 0;

emachine_t* em;

void handle_triggers() {
  ticks++;
  if ((ticks%50)==0)  {
      Einterpreter();
      ticks = 0;
  }
}

void Emachine(emachine_t* em_user) {
  configure_em(em_user);
}
```

```
void Einterpreter() {

  inst_t* inst;

  while(pc < em->eco_size) {
    if (!fetch(&inst, &pc)) {
      return;
    }
    switch(inst->opcode) {
    case NOP:
      break;
    case FUTURE:
      pc = inst->arg1;
      return;
      break;
    case CALL:
      execute((int (*)()) inst->arg1);
      break;
    case SCHEDULE: {
      int k = execi(((int (*)()) inst-
>arg1), 0, NULL, 1, DEFAULT_STACK_SIZE);
      if (k==-1) { cputs("err"); }
      break;
    }

    default:
      return;
    }
  }

  return;
}
```

EM.C -- Part 2

```
void configure_em(emachine_t* em_user) {
  int i;

  if (em!= 0) {
        free(em->eco);
        free(em);
  }
  em = (emachine_t*) malloc (sizeof(emachine_t));
  em->eco = (inst_t*) malloc (eco_max_size*sizeof(inst_t));

  em->eco_size = em_user->eco_size;

  for(i=0; i<em->eco_size; i++) {
      em->eco[i].opcode = em_user->eco[i].opcode;
      em->eco[i].arg1   = em_user->eco[i].arg1;
  }
}

int fetch(inst_t** inst, int* pc) {
  if (*pc >= em->eco_size) {
    return 0;
  }
  *inst = &(em->eco[*pc]);
  *pc = *pc + 1;
  return 1;
}

int execute(int (*pf) ()) {
  return pf();
}
```

HW2.C: NEW

```
#include <stdlib.h>

/* End of Kernel Space */
#include <sys/em.h>
#include <conio.h>
#include <unistd.h>
#include <time.h>
#include <dsound.h>
#include <dmotor.h>
#include <dsensor.h>

int count = 0;

int  start_sound()
{
    dsound_system(DSOUND_BEEP);
    return (1);
}

int   start_text()
{
    lcd_clear();
    lcd_int(count);
    return (1);
}

int   inc_count()
{
    count++;
    return (1);
}
```

```
int main() {
    inst_t eco[7];
    emachine_t* em;

    eco[0].opcode = CALL;
    eco[0].arg1 = (int)(&start_sound)

    eco[1].opcode = CALL;
    eco[1].arg1 = (int) (&start_text);

    eco[2].opcode = SCHEDULE;
    eco[2].arg1 = (int) (&inc_count);

    eco[3].opcode = FUTURE;
    eco[3].arg1 = 4;

    eco[4].opcode = CALL;
    eco[4].arg1 = (int ) (&start_text);

    eco[5].opcode = SCHEDULE;
    eco[5].arg1= (int) (&inc_count);

    eco[6].opcode = FUTURE;
    eco[6].arg1 = 0;

    em = (emachine_t*) malloc(sizeof(emachine_t));
    em->eco = eco;
    em->eco_size = 7;

    Emachine(em);
    return 0;
}
```

HW2.C: OLD

```
#include <conio.h>
#include <unistd.h>
#include <dsound.h>
#include <dbutton.h>
#include <dsensor.h>
#include <dmotor.h>
#include <sys/tm.h>
#include <rom/system.h>

pid_t pid1, pid2;

int  start_sound()
{
   while (1)
   {
        dsound_system(DSOUND_BEEP);
        sleep(2);
   }
}
```

```
int   start_text()
{
   int count = 0;

   while (1)
   {
        lcd_clear();
        lcd_int(count++);
        sleep(1);
   }
}

int main()
{
  pid1=execi(&start_sound, 0, NULL,
   1, DEFAULT_STACK_SIZE);
  pid2=execi(&start_text, 0, NULL, 2,
   DEFAULT_STACK_SIZE);
  return 0;
}
```

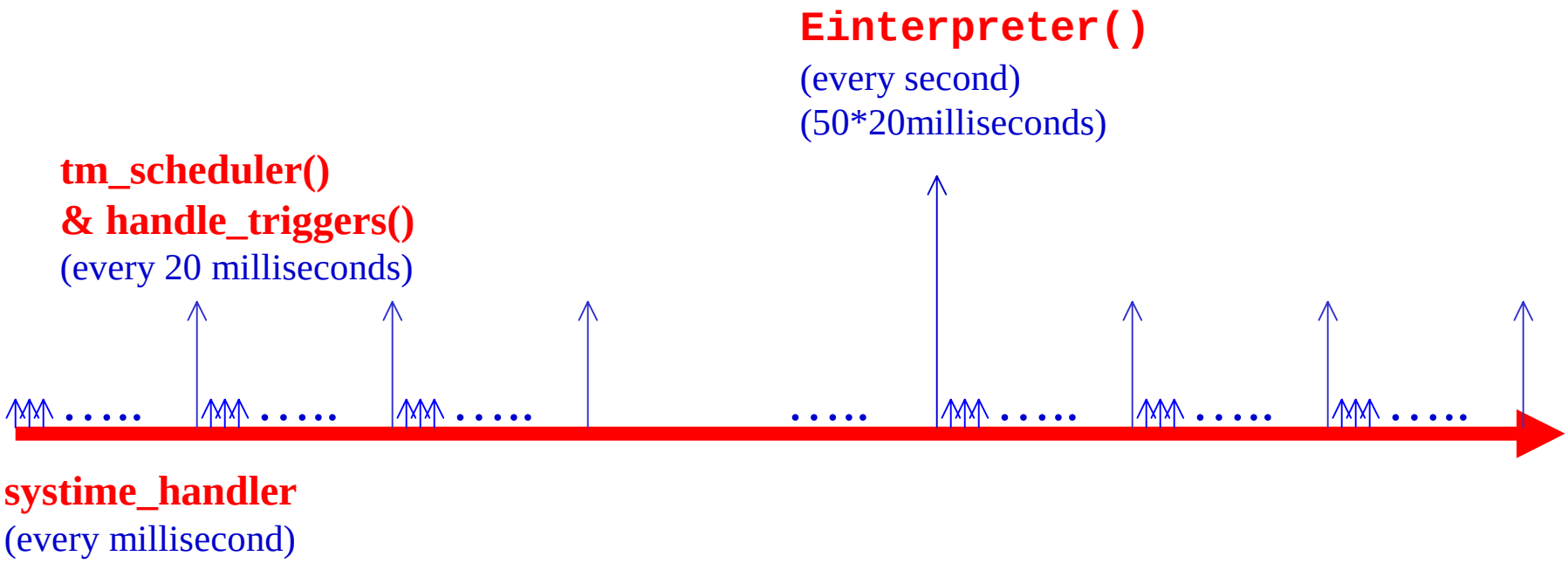
TM.C

```
size_t *tm_scheduler(size_t *old_sp) {
    pdata_t  *next;    // next process to execute
    pchain_t *priority;
    ……
    // for EE290O: 02/27/2002
    handle_triggers();
    …   …
    sem_post(&task_sem);
    return cpid->sp_save;
}
```

Timing

- LegOS is driven by interrupts from th 16 bit timer, which is configured to make an interrupt every millisecond.
- The timer interrupt is handled by a ROM function, which in turn calls the function pointed to by ocia_vector.
- This vector points to the systime_handler function, defined in kernel/systime.c.
- tm_scheduler() function (in kernel/tm.c) checks whether a task switch is needed by inspecting the timeslice counter of the current task.
- The default configuration of timeslices is 20 milliseconds.

Timing


Einterpreter()
(every second)
(50*20milliseconds)
tm_scheduler()
& handle_triggers()
(every 20 milliseconds)
systime_handler
(every millisecond)