Compiler-Assisted Thread Abstractions for Resource-Constrained Systems

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Alexander Bernauer[1], Kay Römer[12]
[1] Institute for Pervasive Computing, ETH Zurich
[2] Institute of Computer Engineering, University of Lübeck

MICS

Mote Programming Models

Threads
+ Convenient programming model
– Resource overhead at runtime

Code Generation

Combine the advantages

Events
+ Efficient at runtime
– Hard to maintain

Features

- Complete semantics of (static) cooperative threads
- C99 compliant
- Platform-agnostic
- Debugging support (planned)

Compared to native application (Contiki)

- Binary size: < +3%
- RAM usage: < +4%
- CPU cycles: +/- 0%
- Lines of code: < -25%

How It Works

Source Code

```
int dt = 200;

BLOCK void sleep(int ms);

THREAD void blinky() {
  uint8_t state;
  state = get_led();
  while(1) {

    wait();

    state ^= 0xff;
    set_leds(state);
  }
}
void wait() {
    int now;
    now = time();

    sleep(now + dt);
    return;
}
```

Target Code

```
int dt = 200;

typedef struct {
  void* cont; int ms;
frame_sleep_t;

typedef struct {
  void* cont;
  union {
    frame_sleep_t sleep;
  } f;
  int now;
} frame_wait_t;

typedef struct {
  union {
    frame_wait_t wait;
  } f;
  uint8_t state;
} frame_blinky_t;

frame_blinky_t bs; // blinky stack

void sleep(frame_sleep_t*);

void thread_blinky(void* cont) {
  if (cont) goto *cont;
  bs.state = get_leds();
  while (1) {
    bs.f.wait.cont = &&lbl_blinky_1;
    goto lbl_wait_0;
lbl_blinky_1:
    bs.state ^= 0xff;
    set_leds(bs.state);
  }
  return;
lbl_wait_0:
  bs.f.wait.now = time();
  bs.f.wait.f.sleep.ms = bs.f.wait.now + dt;
  bs.f.wait.f.sleep.cont = &&lbl_wait_1;
  sleep(&bs.f.wait.f.sleep);
  return;
lbl_wait_1:
  goto *bs.f.wait.cont;
}
```