

Concorrência



exercício 1

- Piscar o LED a cada 1 segundo
- Parar ao pressionar o botão, mantendo o LED aceso para sempre

```
void loop () {  
    digitalWrite(LED_PIN, HIGH);  
    delay(1000);  
    digitalWrite(LED_PIN, LOW);  
    delay(1000);  
  
    int but = digitalRead(BUT_PIN);  
    if (but) {  
        digitalWrite(LED_PIN, HIGH);  
        while(1);  
    }  
}
```

- Programa interativo!



versão sem bloqueio

- Guardar *timestamp* da última mudança
- Guardar estado atual do LED

```
int state = 1;
unsigned long old;
void setup () {
    old = millis();
    digitalWrite(LED_PIN, state);
}
```

```
void loop () {
    unsigned long now = millis();
    if (now >= old+1000) {
        old = now;
        state = !state;
        digitalWrite(LED_PIN, state);
    }
}
```

```
int but = digitalRead(BUT_PIN);
if (but) {
    digitalWrite(LED_PIN, HIGH);
    while(1);
}
```

```
}
```

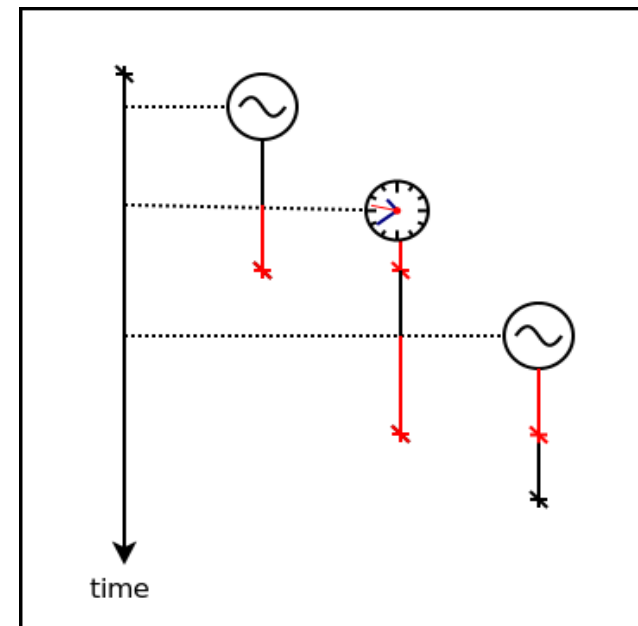
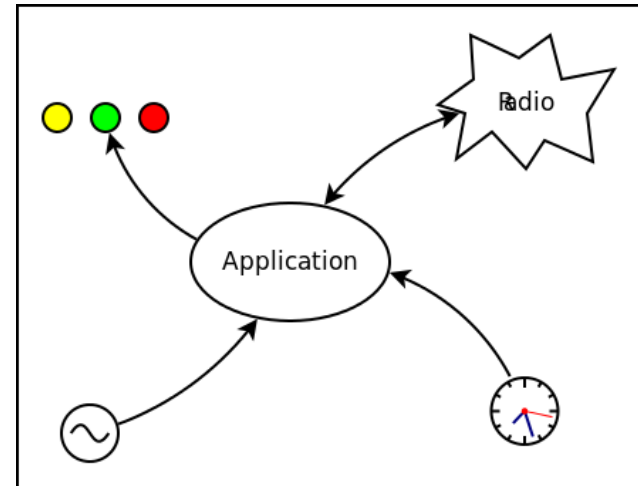
```
void loop () {
    unsigned long now = millis();
    if (now >= old+1000) {
        old = now;
        state = !state;
        digitalWrite(LED_PIN, state);
    }
}
```

```
void loop () {
    int but = digitalRead(BUT_PIN);
    if (but) {
        digitalWrite(LED_PIN, HIGH);
        exit();
    }
}
```



eventos concorrentes

- cada evento gera uma reação
 - reação tem duração
- cada evento pode ocorrer a qualquer momento



e daí?

- problemas quando reações concorrentes acessam o mesmo recurso.

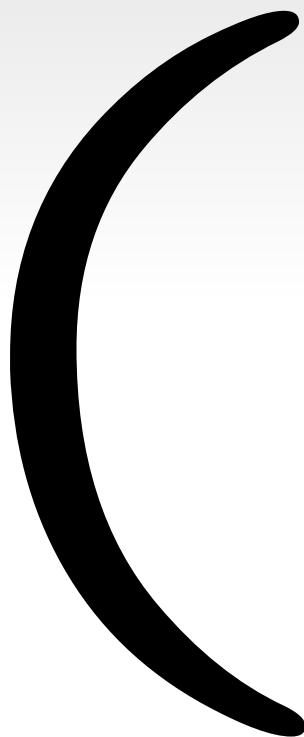
```
int state = 1;
unsigned long old;
void setup () {
    old = millis();
    digitalWrite(LED_PIN, state);
}
void loop () {
    unsigned long now = millis();
    if (now >= old+1000) {
        old = now;
        state = !state;
        digitalWrite(LED_PIN, state);
    }

    int but = digitalRead(BUT_PIN);
    if (but) {
        digitalWrite(LED_PIN, HIGH);
        while(1);
    }
}
```

```
void loop () {
    unsigned long now = millis();
    if (now >= old+1000) {
        old = now;
        state = !state;
        digitalWrite(LED_PIN, state);
    }
}
```

```
void loop () {
    int but = digitalRead(BUT_PIN);
    if (but) {
        digitalWrite(LED_PIN, HIGH);
        exit();
    }
}
```





concorrência

- várias atividades simultâneas
 - recursos em comum



concorrência

- uma ou mais linhas de execução



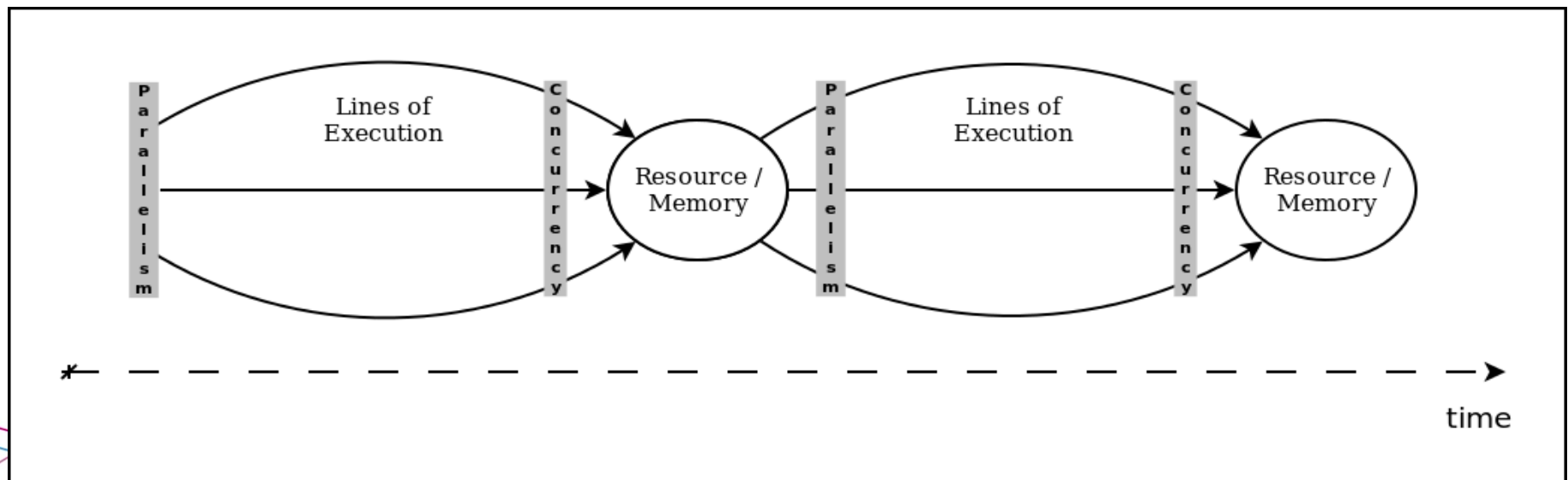
concorrência vs paralelismo

- concorrência: várias atividades simultâneas
 - recursos compartilhados
- paralelismo: várias linhas de execução



concorrência e paralelismo

- concorrência
 - acessos “simultâneos” a recursos
- paralelismo
 - execução simultânea



videos e textos - 2

- Concorrência e Paralelismo
 - Rob Pike - *“Concurrency Is Not Parallelism”*
 - Wikipedia - *“Embarrassingly Parallel”*





Modelos de Execução Concorrente

- Por quê?
 - Como descrever e entender as partes de um sistema concorrente (e.g., atividades, processos, atores, etc.).
 - Vocabulário e semântica
 - execução (escalonamento)
 - composição
 - compartilhamento
 - comunicação
 - sincronização



Modelos de Execução Concorrente

- Modelo Assíncrono
 - Execução independente / Sincronização explícita
 - *Threads + locks/mutexes (pthreads, Java-Threads)*
 - *Atores + troca de mensagens (Erlang, Go)*
- Modelo Síncrono
 - Execução dependente / Sincronização implícita
 - *Arduino-Loop, Game-Loop, Padrão Observer*



Modelo assíncrono

- Execução independente
 - Arduino: *ChibiOS*

```
void Thread1 (void) {  
    <...>  
}  
  
void Thread2 (void) {  
    <...>  
}  
  
void setup() {  
    chThdCreateStatic(..., Thread1);  
    chThdCreateStatic(..., Thread2);  
}
```



exercício

```
int state = 1;
unsigned long old;

void setup () {
    old = millis();
    digitalWrite(LED_PIN, state);
}

void loop () {
    unsigned long now = millis();
    if (now >= old+1000) {
        old = now;
        state = !state;
        digitalWrite(LED_PIN, state);
    }

    int but = digitalRead(BUT_PIN);
    if (but) {
        digitalWrite(LED_PIN, HIGH);
        while(1);
    }
}
```

```
void Thread1 (void) {
    while (TRUE) {
        digitalWrite(LED_PIN, HIGH);
        chThdSleepMilliseconds(1000);
        digitalWrite(LED_PIN, LOW);
        chThdSleepMilliseconds(1000);
    }
}

void Thread2 (void) {
    while (TRUE) {
        int but = digitalRead(BUT_PIN);
        if (but) {
            digitalWrite(LED_PIN, HIGH);
            break;
        }
    }
}

void setup () {
    chThdCreateStatic(..., Thread1);
    chThdCreateStatic(..., Thread2);
}
```


exercício 1 (assíncrono)

```
void Thread1 (void) {  
    while (TRUE) {  
        digitalWrite(LED_PIN, HIGH);  
        chThdSleepMilliseconds(1000);  
        digitalWrite(LED_PIN, LOW);  
        chThdSleepMilliseconds(1000);  
    }  
}  
  
void Thread2 (void) {  
    while (TRUE) {  
        int but = digitalRead(BUT_PIN);  
        if (but) {  
            digitalWrite(LED_PIN, HIGH);  
            break;  
        }  
    }  
}  
  
void setup () {  
    chThdCreateStatic(..., Thread1);  
    chThdCreateStatic(..., Thread2);  
}
```

mas e a thread 1 continua fazendo sua tarefa?



exercício 1 (assíncrono)

```
void Thread1 (void) {
    while (TRUE) {
        digitalWrite(LED_PIN, HIGH);
        chThdSleepMilliseconds(1000);
        digitalWrite(LED_PIN, LOW);
        chThdSleepMilliseconds(1000);
    }
}

void Thread2 (void) {
    while (TRUE) {
        int but = digitalRead(BUT_PIN);
        if (but) {
            digitalWrite(LED_PIN, HIGH);
            break;
        }
    }
}

void setup () {
    chThdCreateStatic(..., Thread1);
    chThdCreateStatic(..., Thread2);
}
```

Thread* t1;

```
void Thread1 (void) {
    while (TRUE) {
        digitalWrite(LED_PIN, HIGH);
        chThdSleepMilliseconds(1000);
        digitalWrite(LED_PIN, LOW);
        chThdSleepMilliseconds(1000);
    }
}

void Thread2 (void) {
    while (TRUE) {
        int but = digitalRead(BUT_PIN);
        if (but) {
            digitalWrite(LED_PIN, HIGH);
            chThdTerminate(t1);
            break;
        }
    }
}

void setup () {
    t1 = chThdCreateStatic(..., Thread1);
    chThdCreateStatic(..., Thread2);
}
```

exercício 1 (assíncrono)

```
void Thread1 (void) {
  while (TRUE) {
    digitalWrite(LED_PIN, HIGH);
    chThdSleepMilliseconds(1000);
    digitalWrite(LED_PIN, LOW);
    chThdSleepMilliseconds(1000);
  }
}

void Thread2 (void) {
  while (TRUE) {
    int but = digitalRead(BUT_PIN);
    if (but) {
      digitalWrite(LED_PIN, HIGH);
      break;
    }
  }
}

void setup () {
  chThdCreateStatic(..., Thread1);
  chThdCreateStatic(..., Thread2);
}
```

Thread* t1;

```
void Thread1 (void) {
  while (TRUE) {
    digitalWrite(LED_PIN, HIGH);
    chThdSleepMilliseconds(1000);
    digitalWrite(LED_PIN, LOW);
    chThdSleepMilliseconds(1000);
  }
}

void Thread2 (void) {
  while (TRUE) {
    int but = digitalRead(BUT_PIN);
    if (but) {
      digitalWrite(LED_PIN, HIGH);
      chThdTerminate(t1);
      break;
    }
  }
}

void setup () {
  t1 = chThdCreateStatic(..., Thread1);
  chThdCreateStatic(..., Thread2);
}
```

Quando?

exercício 1 (assíncrono)

```
void Thread1 (void) {
  while (TRUE) {
    digitalWrite(LED_PIN, HIGH);
    chThdSleepMilliseconds(1000);
    digitalWrite(LED_PIN, LOW);
    chThdSleepMilliseconds(1000);
  }
}

void Thread2 (void) {
  while (TRUE) {
    int but = digitalRead(BUT_PIN);
    if (but) {
      digitalWrite(LED_PIN, HIGH);
      break;
    }
  }
}

void setup () {
  chThdCreateStatic(..., Thread1);
  chThdCreateStatic(..., Thread2);
}
```

Thread* t1;

```
void Thread1 (void) {
  while (TRUE) {
    digitalWrite(LED_PIN, HIGH);
    chThdSleepMilliseconds(1000);
    digitalWrite(LED_PIN, LOW);
    chThdSleepMilliseconds(1000);
  }
}

void Thread2 (void) {
  while (TRUE) {
    int but = digitalRead(BUT_PIN);
    if (but) {
      digitalWrite(LED_PIN, HIGH);
      chThdTerminate(t1);
      break;
    }
  }
}

void setup () {
  t1 = chThdCreateStatic(..., Thread1);
  chThdCreateStatic(..., Thread2);
}
```

Quando?

artigos e videos

- Terminação de Threads
 - Java - “*Why Are Thread.stop, Thread.suspend, Thread.resume and Runtime.runFinalizersOnExit Deprecated?*”
 - pthreads – man pthreads_cancel
 - ChibiOS - “*How to cleanly stop the OS*”



Exercício 1 (async)

```
Thread* t1;
```

```
void Thread1(void) {  
    while (TRUE) {  
        digitalWrite(LED_PIN, HIGH);  
        chThdSleepMilliseconds(1000);  
        digitalWrite(LED_PIN, LOW);  
        chThdSleepMilliseconds(1000);  
    }  
}  
  
void Thread2(void) {  
    while (TRUE) {  
        int but = digitalRead(BUT_PIN);  
        if (but) {  
            digitalWrite(LED_PIN, HIGH);  
            chThdTerminate(t1);  
            break;  
        }  
    }  
}  
  
void setup () {  
    t1 = chThdCreateStatic(..., Thread1);  
    chThdCreateStatic(..., Thread2);  
}
```

```
Thread* t1;  
void Thread1(void) {  
    while (TRUE) {  
        digitalWrite(LED_PIN, HIGH);  
        chThdSleepMilliseconds(1000);  
        if (chThdShouldTerminateX())  
            break;  
        digitalWrite(LED_PIN, LOW);  
        chThdSleepMilliseconds(1000);  
        if (chThdShouldTerminateX())  
            break;  
    }  
}  
  
void Thread2(void) {  
    while (TRUE) {  
        int but = digitalRead(BUT_PIN);  
        if (but) {  
            digitalWrite(LED_PIN, HIGH);  
            chThdTerminate(t1);  
            break;  
        }  
    }  
}  
  
void setup () {  
    t1 = chThdCreateStatic(..., Thread1);  
    chThdCreateStatic(..., Thread2);  
}
```



escalonamento de threads

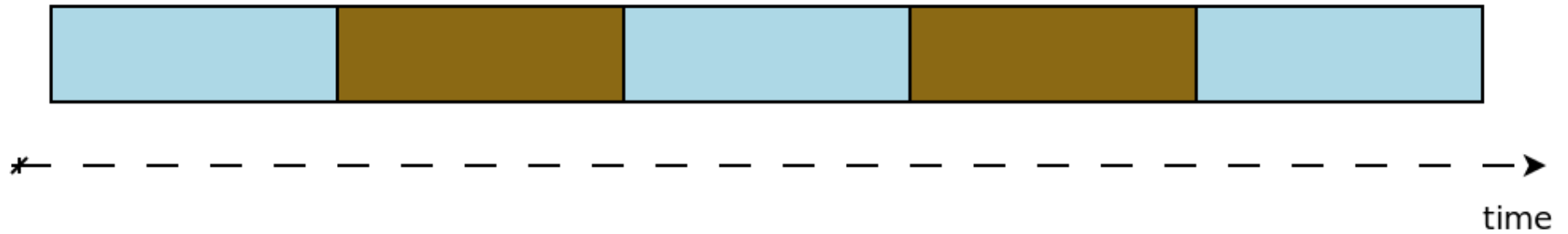
- como se compartilha o recurso CPU?
 - threads preemptivas e cooperativas



Síncrono /
Cooperativo

Activity 1

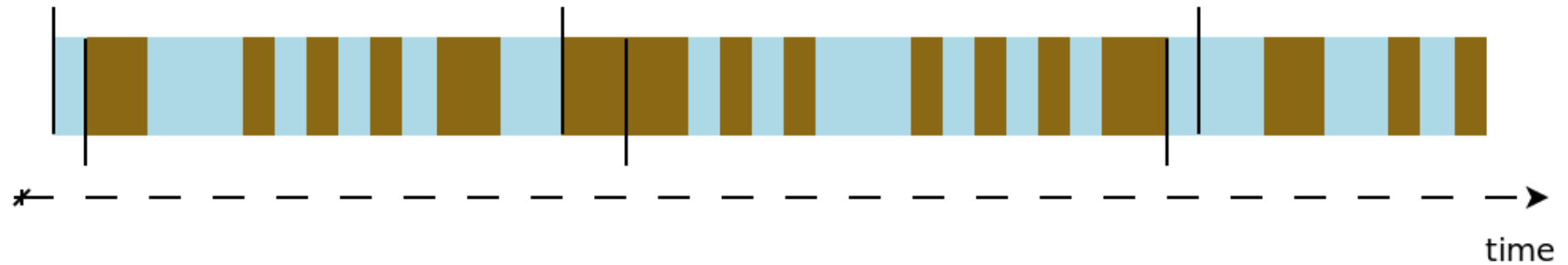
Activity 2



Assíncrono /
Preemptivo

Activity 1

Activity 2



exercício

```
Thread* t1;
void Thread1 (void) {
    while (TRUE) {
        digitalWrite(LED_PIN, HIGH);
        chThdSleepMilliseconds(1000);
        if (chThdShouldTerminateX())
            break;
        digitalWrite(LED_PIN, LOW);
        chThdSleepMilliseconds(1000);
        if (chThdShouldTerminateX())
            break;
    }
}
void Thread2 (void) {
    while (TRUE) {
        int but = digitalRead(BUT_PIN);
        if (but) {
            digitalWrite(LED_PIN, HIGH);
            chThdTerminate(t1);
            break;
        }
    }
}
void setup () {
    t1 = chThdCreateStatic(..., Thread1);
    chThdCreateStatic(..., Thread2);
}
```

```
MUTEX_DECL(mut);
Thread* t1;
void Thread1 (void) {
    while (TRUE) {
        digitalWrite(LED_PIN, HIGH);
        chThdSleepMilliseconds(1000);
        chMtxLock(&mut);
        if (chThdShouldTerminate())
            break;
        digitalWrite(LED_PIN, LOW);
        chMtxUnlock(&mut);
        chThdSleepMilliseconds(1000);
        if (chThdShouldTerminate())
            break;
    }
}
void Thread2 (void) {
    while (TRUE) {
        int but = digitalRead(BUT_PIN);
        if (but) {
            chMtxLock(&mut);
            digitalWrite(LED_PIN, HIGH);
            chThdTerminate(t1);
            chMtxUnlock(&mut);
            break;
        }
    }
}
void setup () {
    t1 = chThdCreateStatic(..., Thread1);
    chThdCreateStatic(..., Thread2);
}
```



```

MUTEX_DECL(mut);
Thread* t1;
void Thread1 (void) {
    while (TRUE) {
        digitalWrite(LED_PIN, HIGH);
        chThdSleepMilliseconds(1000);
        chMtxLock(&mut);
        if (chThdShouldTerminate())
            break;
        digitalWrite(LED_PIN, LOW);
        chMtxUnlock(&mut);
        chThdSleepMilliseconds(1000);
        if (chThdShouldTerminate())
            break;
    }
}
void Thread2 (void) {
    while (TRUE) {
        int but = digitalRead(BUT_PIN);
        if (but) {
            chMtxLock(&mut);
            digitalWrite(LED_PIN, HIGH);
            chThdTerminate(t1);
            chMtxUnlock(&mut);
            break;
        }
    }
}
void setup () {
    t1 = chThdCreateStatic(..., Thread1);
    chThdCreateStatic(..., Thread2);
}

```



exemplo - cálculo pesado

- *ordenação, criptografia, compressão, codificação/conversão*
- Piscar o LED a cada 1 segundo

```
void loop () {  
    digitalWrite(LED_PIN, HIGH);  
    f();  
    digitalWrite(LED_PIN, LOW);  
    f();  
}
```

```
void Thread1 (void) {  
    while (TRUE) {  
        f(); // operação longa  
    }  
}  
  
void Thread2 (void) {  
    while (TRUE) {  
        // pisca led!  
    }  
}
```



exemplo - cálculo pesado

- Criptografia XTEA
 - <https://en.wikipedia.org/wiki/XTEA>
- Criptografar / Decriptar uma string
- Calcular o tempo de execução das operações:
 - Medir o tempo de resposta para 10 execuções
- Variar o tamanho da string (podemos simular isso com um loop de chamadas a encipher) e avaliar a “responsividade”



- Usar Serial para debug/output
 - Remover output na hora de medir!
- Exemplo em “usaxtea”

```
void setup () {  
  Serial.begin(9600);  
}  
void loop () {  
  unsigned long t1 = millis();  
  Serial.print(<v>);  // antes  
  encipher(<v>);  
  Serial.print(<v>);  // durante  
  decipher(<v>);  
  Serial.print(<v>);  // depois  
  unsigned long t2 = millis();  
  Serial.print(t2-t1); // total  
}
```



```
#include "xtea.h"
#include "pindefs.h"

void setup () {
    pinMode(LED1, OUTPUT);
    Serial.begin(9600);
}

int led = 0;

uint32_t key[] = { 1, 2, 3, 4 };
uint32_t v[] = { 10, 20 };

void loop () {
    led = !led; digitalWrite(LED, led);

    unsigned long t1 = millis();

    Serial.print("antes: "); Serial.print(v[0]); Serial.print(" "); Serial.println(v[1]);

    encipher(32, v, key);

    Serial.print("durante: "); Serial.print(v[0]); Serial.print(" "); Serial.println(v[1]);

    decipher(32, v, key);

    Serial.print("depois: "); Serial.print(v[0]); Serial.print(" "); Serial.println(v[1]);

    unsigned long t2 = millis();
    Serial.println(t2-t1);
}
```



como tratar tarefas demoradas

- O que fazer se a execução demora demais?
 - sistema não mais reativo
- Inversão de controle
 - re-implementar o algoritmo!
- Usar *threads*
 - Praticamente não há concorrência



modelo síncrono

- Durante uma unidade de tempo lógico, o ambiente está invariante e não interrompe o programa
- Implementação:
 - *Sampling*: Arduino Loop
 - *Event-driven*: Padrão *Observer*

Hipótese de sincronismo: “*Reações executam infinitamente mais rápido do que a taxa de eventos.*”



padrão observador

```
wait ANY_EVENT_CHANGE do  
  react();  
end
```

- “Hollywood principle: don't call us, we'll call you.”
- Ocorrência de um evento executa uma “callback” no código
 - Botão => button_changed()
 - Timer => timer_expired()
 - Rede => packet_received()



padrão observador - exemplo

hello world

```
#include "pindefs.h"

void setup () {
    pinMode(LED_PIN, OUTPUT);
    pinMode(KEY1, INPUT_PULLUP);
    pinMode(KEY2, INPUT_PULLUP);
    pinMode(KEY3, INPUT_PULLUP);
}

void loop () {
    int but = digitalRead(KEY1);
    digitalWrite(LED1, but);
}
```

```
#include "event_driven.h"
#include "app.h"
#include "pindefs.h"

void appinit(){
    button_listen(KEY1);
}

void button_changed (int pin, int v) {
    digitalWrite(LED1, v);
}

void timer_expired () {
}
```



padrão observador - exemplo

hello world

```
#include "pindefs.h"

void setup () {
    pinMode(LED_PIN, OUTPUT);
    pinMode(KEY1, INPUT_PULLUP);
    pinMode(KEY2, INPUT_PULLUP);
    pinMode(KEY3, INPUT_PULLUP);
}

void loop () {
    int but = digitalRead(KEY1);
    digitalWrite(LED1, but);
}
```

```
#include "event_driven.h"
#include "app.h"
#include "pindefs.h"

void appinit(){
    button_listen(KEY1);
}

void button_changed (int pin, int v) {
    digitalWrite(LED1, v);
}

void timer_expired () {
}
```

```
#include "event_driven.h"
#include "app.h"
#include "pindefs.h"

void setup(){...}
void loop {...}
```

*“casca” criando
interface dirigida a eventos*



tarefa 3 – API (event_driven.ino)

```
/* Funções de registro: */

void button_listen (int pin) {
    <...>          // “pin” passado deve gerar notificações
}

void timer_set (int ms) {
    <...>          // timer deve expirar após “ms” milisegundos
}

/* Callbacks */

void button_changed (int pin, int v); // notifica que “pin” mudou para “v”
void timer_expired (void);           // notifica que o timer expirou

/* Programa principal: */

void setup () {
    <...>          // inicialização da API
    init();        // inicialização do usuário
}

void loop () {
    <...>          // detecta novos eventos
    button_changed(...); // notifica o usuário
    <...>          // detecta novos eventos
    timer_expired(...);  // notifica o usuário
}
```



tarefa

- Implementar “event_driven.ino”
 - Tratador para botões da placa
 - 2 timers
- Reimplementar os exemplos com orientação a eventos:
 - Hello World: Input
 - Tarefa 2



framework event_driven

event_driven.h

```
void button_listen (int pin);  
  
void timer_set (int ms);
```

app.h

```
void appinit(void);  
  
void button_changed ( int pin, int v);  
  
void timer_expired(void);
```

event_driven.ino

```
#include "event_driven.h"  
#include "app.h"  
#include "pindefs.h"  
  
void setup(){...}  
void loop {...}
```

app.ino

```
#include "event_driven.h"  
#include "app.h"  
#include "pindefs.h"  
  
void appinit(void) {...}  
  
void button_changed(int p, int v) { ...}  
  
void timer_expired(void) {...}
```



framework event-driven

event_driven.h

```
void button_listen (int pin);  
  
void timer_set (int ms);
```

app.h

```
void appinit(void);  
  
void button_changed ( int pin, int v);  
  
void timer_expired(void);
```

event_driven.ino

```
#include "event_driven.h"  
#include "app.h"  
#include "pindefs.h"  
  
void setup(){...}  
void loop {...}
```

app.ino

```
#include "event_driven.h"  
#include "app.h"  
#include "pindefs.h"  
  
void appinit(void) {...}  
  
void button_changed(int p, int v) { ...}  
  
void timer_expired(void) {...}
```

