

Linguagem de Programação



Francisco Sant'Anna
francisco@ime.uerj.br

“Hello world!” em Céu

■ Piscando LEDs

1. *on* ↔ *off* a cada 500ms
2. Parar após “press”
3. Repetir após 2s

■ Composições

- seq, loop, par (*trails*)
 - At any level of depth
- ~~variáveis de estado / comunicação~~

```
loop do
  par/or do
    loop do
      await 500ms;
      _leds_toggle();
    end
  with
    await PRESS;
  end
  await 2s;
end
```

Linhas de execução
=
Trails (em Céu)

Visão geral de Céu

- Reativa
 - Ambiente no controle: *eventos*
- Imperativa
 - Sequências, loops e atribuições
- Concorrente
 - Múltiplas linhas de execução: *trails (trilhas)*
- Síncrona
 - Trilhas são sincronizadas a cada evento
 - **Trilhas estão sempre esperando**
- Determinística
 - Mesmo comportamento para uma dada linha de tempo

Modelos de Concorrência

■ Assíncrono

- Atividades independentes
 - Requer primitivas de sincronização (*mutexes*, *monitors*, *channels*, *pipes*)
-
- Threads (Java, p-threads)
 - Processes (UNIX)
 - Actors (erlang)

■ Síncrono

- Execução em passos
 - Requer primitivas assíncronas (não bloqueantes)
-
- Loop do Arduino
 - Event-driven (callbacks)
 - *Céu*

Modelos de Concorrência

- Assíncrono

- programação estruturada
- sincronismo explícito
- implementação complexa
- execução paralela**

VS

- Síncrono

- programação não estruturada
- livre de sincronização**
- footprint* pequeno
- execução serializada

- programação estruturada
- livre de sincronização
- footprint* pequeno
- execução paralela

?

Exercício 1 (revisão-da-revisão)

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

```
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;  
        }  
    }  
}
```

Exercício 1 (revisão-da-revisão)

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

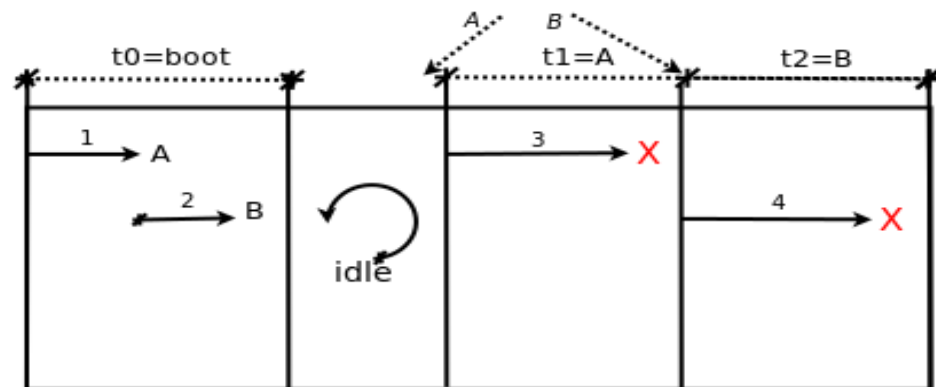
```
input  int PIN_02;
output int PIN_13;
par/or do
    loop do
        emit PIN_13(HIGH);
        await 1s;
        emit PIN_13(LOW);
        await 1s;
    end
with
    loop do
        var int on = await PIN_02;
        if on==LOW then
            break;
        end
    end
end
emit PIN13(HIGH);
await FOREVER;
```

```
input  int PIN_02;
output int PIN_13;
par/or do
    loop do
        emit PIN_13(HIGH);
        await 1s;
        emit PIN_13(LOW);
        await 1s;
    end
with
    var int on = await PIN_02
    until on==LOW;
end
emit PIN13(HIGH);
await FOREVER;
```


Céu: Modelo de Execução

1. O programa inicia na “reação de *boot*” em apenas uma trilha.
2. Trilhas ativas executam até esperarem ou terminarem. Esse passo é conhecido como “reação” e sempre executa em tempo *bounded*.
3. A ocorrência de um evento de entrada acorda **todas** as trilhas esperando aquele evento. Repete o “passo 2”.

```
par/and do
  <...>      // 1
  await A;
  <...>      // 3
with
  <...>      // 2
  await B;
  <...>      // 4
end
```



<...> são segmentos de trilha que não esperam
(e.g. atribuições, chamadas de função)

Céu: Modelo de Execução

- Hipótese de sincronismo: “Reações executam infinitamente mais rápido do que a taxa de eventos.”
- *Na teoria*: o programa não leva tempo no “passo 2” e está sempre ocioso no “passo 3”.
- *Na prática*: se um evento ocorre enquanto uma reação executa, ele é enfileirado para uma reação posterior.
- Reações a eventos nunca são sobrepostas (modelo síncrono)
- Quando múltiplas trilhas estão ativas (i.e., acordadas pelo mesmo evento), elas são executadas na ordem em que aparecem no código do programa.

Exercício 1 (revisão-da-revisão)

```
input  int PIN_02;
output int PIN_13;
par/or do
  loop do
    emit PIN_13(HIGH);
    await 1s;
    emit PIN_13(LOW);
    await 1s;
  end
with
  loop do
    var int on = await PIN_02;
    if on==LOW then
      break;
    end
  end
end
emit PIN_13(HIGH);
await FOREVER;
```


Tarefa-02 (revisão)

- Piscar o LED a cada 1 segundo
- Botão 1: Acelerar o pisca-pisca a cada pressionamento
- Botão 2: Desacelerar a cada pressionamento
- Botão 1+2 (em menos de 500ms): Parar

Tarefa-02 (revisão)

- Piscar o LED a cada 1 segundo
- Botão 1: Acelerar o pisca-pisca a cada pressionamento
- Botão 2: Desacelerar a cada pressionamento
- Botão 1+2 (em menos de 500ms): Parar

```
var int dt = 1000;
par/or do
  loop do
    emit PIN_13(HIGH);
    await (dt)ms;
    emit PIN_13(LOW);
    await (dt)ms;
  end
with
  loop do
    await PIN_02;
    await 50ms;
    dt = dt - 100;
  end
with
  loop do
    await PIN_03;
    await 50ms;
    dt = dt + 100;
  end
with
  loop do
    par/or do
      par/and do
        await PIN_02;
      with
        await PIN_03;
      end
      break;
    with
      par/or do
        await PIN_02;
      with
        await PIN_03;
      end
      await 500ms;
    end
  end
end
end
```

```

var int dt = 1000;
par/or do
    loop do
        emit PIN_13(HIGH);
        await (dt)ms;
        emit PIN_13(LOW);
        await (dt)ms;
    end
with
    loop do
        var int val = await PIN_02
                        until val==HIGH;
        await 50ms;
        dt = dt - 100;
    end
with
    loop do
        await PIN_03;
        await 50ms;
        dt = dt + 100;
    end
with
    loop do
        par/or do
            par/and do
                await PIN_02;
                with
                    await PIN_03;
                end
                break;
            with
                par/or do
                    await PIN_02;
                    with
                        await PIN_03;
                    end
                    await 500ms;
                end
            end
        end
    end
end

```

```

#define AWAIT_PIN_UNTIL(pin,val) \
    do \
        var int v = await pin \
                        until v==val; \
    end

var int dt = 1000;
par/or do
    loop do
        emit PIN_13(HIGH);
        await (dt)ms;
        emit PIN_13(LOW);
        await (dt)ms;
    end
with
    loop do
        AWAIT_PIN_UNTIL(PIN_02,1);
        await 50ms;
        dt = dt - 100;
    end
with
    loop do
        AWAIT_PIN_UNTIL(PIN_03,1);
        await 50ms;
        dt = dt + 100;
    end
with
    loop do
        par/or do
            par/and do
                AWAIT_PIN_UNTIL(PIN_02,HIGH);
                with
                    AWAIT_PIN_UNTIL(PIN_03,HIGH);
                end
                break;
            with
                <...>
            end
        end
    end
end

```


Arduino + Céu

- Substituir arquivo defeituoso:

```
$ wget http://www.ceu-lang.org/pins_inputs.c.h  
$ mv pins_inputs.c.h /opt/reativos/ceu-arduino/env/
```

- Configurar o ambiente (toda vez que abrir um terminal):

```
$ . /opt/reativos/login.sh
```

- Compilar e gravar:

```
$ cd /opt/reativos/ceu-arduino/  
$ make CEU_SRC=<caminho-arquivo-ceu>
```

- Testar exemplos do repositório:

- code/arduino/blink-00/blink-00.ceu
- code/arduino/button-01/button-01.ceu
- code/arduino/ex-01-c/ex-01.ceu
- code/arduino/ex-02/ex-02.ceu

Exercício - “Reiniciar”

- Após parada na *Exercício 1*, reiniciar o comportamento caso o botão seja pressionado por 5 segundos.