

Linguagem de Programação



LabLua – PUC-Rio

www.lua.inf.puc-rio.br

Francisco Sant'Anna

“Hello world!” em Céu

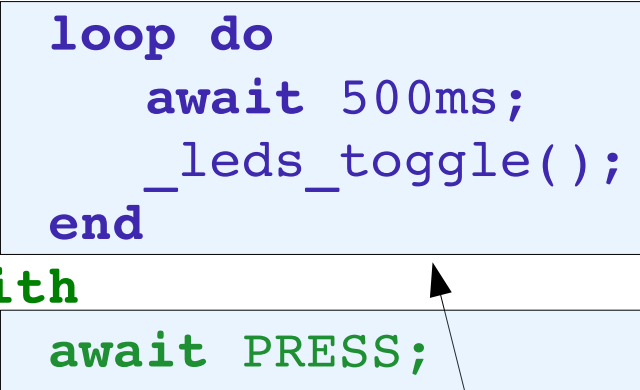
■ Piscando LEDs

1. *on* \leftrightarrow *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
  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

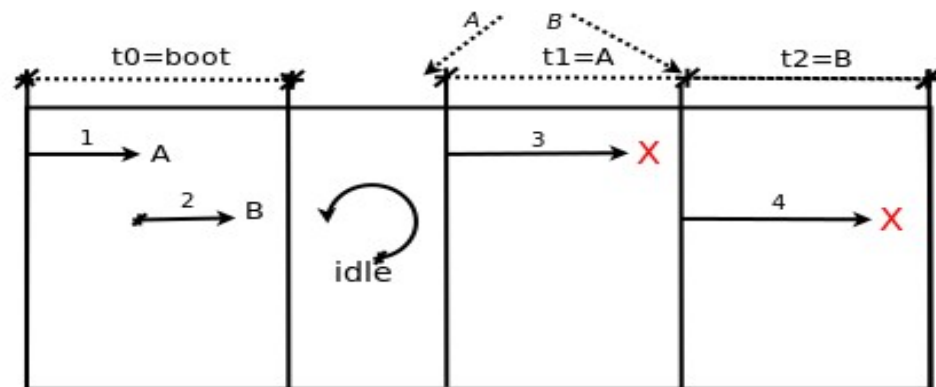
```
input  int PIN02;
output int PIN13;
par/or do
  loop do
    emit PIN13 => _HIGH;
    await 1s;
    emit PIN13 => _LOW;
    await 1s;
  end
with
  loop do
    var int on = await PIN02;
    if on then
      break;
    end
  end
end
emit PIN13 => _HIGH;
```

```
input  int PIN02;
output int PIN13;
par/or do
  loop do
    emit PIN13 => _HIGH;
    await 1s;
    emit PIN13 => _LOW;
    await 1s;
  end
with
  var int on = await PIN02
  until on;
end
emit PIN13 => _HIGH;
```


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 PIN02;
output int PIN13;
par/or do
  loop do
    emit PIN13 => _HIGH;
    await 1s;
    emit PIN13 => _LOW;
    await 1s;
  end
with
  loop do
    var int on? = await PIN02;
    if on? then
      break;
    end
  end
end
emit PIN13 => _HIGH;
```


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 PIN13 => _HIGH;
        await (dt)ms;
        emit PIN13 => _LOW;
        await (dt)ms;
    end
with
    loop do
        await PIN02;
        await 50ms;
        dt = dt - 100;
    end
with
    loop do
        await PIN03;
        await 50ms;
        dt = dt + 100;
    end
with
    loop do
        par/or do
            par/and do
                await PIN02;
            with
                await PIN03;
            end
            break;
        with
            par/or do
                await PIN02;
            with
                await PIN03;
            end
            await 500ms;
        end
    end
end
end
```

```

var int dt = 1000;
par/or do
  loop do
    emit PIN13 => _HIGH;
    await (dt)ms;
    emit PIN13 => _LOW;
    await (dt)ms;
  end
with
  loop do
    var int val = await PIN02
                      until val==1;
    await 50ms;
    dt = dt - 100;
  end
with
  loop do
    await PIN03;
    await 50ms;
    dt = dt + 100;
  end
with
  loop do
    par/or do
      par/and do
        await PIN02;
      with
        await PIN03;
      end
      break;
    with
      par/or do
        await PIN02;
      with
        await PIN03;
      end
      await 500ms;
    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 PIN13 => _HIGH;
    await (dt)ms;
    emit PIN13 => _LOW;
    await (dt)ms;
  end
with
  loop do
    AWAIT_PIN_UNTIL(PIN02,1);
    await 50ms;
    dt = dt - 100;
  end
with
  loop do
    AWAIT_PIN_UNTIL(PIN03,1);
    await 50ms;
    dt = dt + 100;
  end
with
  loop do
    par/or do
      par/and do
        AWAIT_PIN_UNTIL(PIN02,1);
      with
        AWAIT_PIN_UNTIL(PIN03,1);
      end
      break;
    with
      <...>
    end
  end
end
end

```

Exercício - “Reiniciar”

- Reiniciar o comportamento da “Tarefa 2” caso o Botão 1 seja pressionado por 5 segundos.

```
loop do
  <...>  // código anterior

  loop do
    AWAIT_PIN_UNTIL(PIN02,_HIGH);
    await 50ms;
    par/or do
      AWAIT_PIN_UNTIL(PIN02,_LOW);
    with
      await 5s;
      break;
    end
  end
end
end
```


Compilando / Executando

```
> cd /home/ceu/ceu-arduino  
> make CEUFILE=/home/ceu/reativos/code/arduino/ex_01.ceu  
> make CEUFILE=/home/ceu/reativos/code/arduino/ex_02.ceu
```


SDL: “Input” (Céu)

```
SDL_Event e;
while (1) {
    while (SDL_PollEvent(&e) == 0);
    if (e.type == SDL_QUIT) {
        break;
    } else if (e.type == SDL_KEYDOWN) {
        switch (e.key.keysym.sym) {
            case SDLK_UP:
                r.y -= 10;
            case SDLK_DOWN:
                r.y += 10;
            case SDLK_LEFT:
                r.x -= 10;
            case SDLK_RIGHT:
                r.x += 10;
        }
    }
    SDL_SetRenderDrawColor(renderer, ...);
    SDL_RenderFillRect(renderer, NULL);
    SDL_SetRenderDrawColor(renderer, ...);
    SDL_RenderFillRect(renderer, &r);
    SDL_RenderPresent(renderer);
}
```

```
input void                                SDL_QUIT;
input _SDL_KeyboardEvent* SDL_KEYDOWN;
input void                                SDL_REDRAW;

par/or do
    await SDL_QUIT;
with
    var _SDL_KeyboardEvent* key;
    every key in SDL_KEYDOWN do
        if key:keysym.sym == _SDLK_UP then
            r.y = r.y - 10;
        else/if key:keysym.sym == _SDLK_DOWN then
            r.y = r.y + 10;
        else/if key:keysym.sym == _SDLK_LEFT then
            r.x = r.x - 10;
        else/if key:keysym.sym == _SDLK_RIGHT then
            r.x = r.x + 10;
        end
    end
with
    every SDL_REDRAW do
        _SDL_SetRenderDrawColor(renderer, ...);
        _SDL_RenderFillRect(renderer, null);
        _SDL_SetRenderDrawColor(renderer, ...);
        _SDL_RenderFillRect(renderer, &r);
        _SDL_RenderPresent(renderer);
    end
end
```

SDL: “Animação” (Céu)

(code/sdl/ex_01.ceu)

```
par/or do
  loop do
    loop i in 100 do
      await 10ms;
      r1.x = r1.x + 1;
    end
    loop i in 100 do
      await 10ms;
      r1.y = r1.y + 1;
    end
    loop i in 100 do
      await 10ms;
      r1.x = r1.x - 1;
    end
    loop i in 100 do
      await 10ms;
      r1.y = r1.y - 1;
    end
  end
with
  var _SDL_MouseButtonEvent* but;
  but = await SDL_MOUSEBUTTONDOWN
  until _SDL_Rect_vs_Mouse(&r1, but);
end
```

Exercício - “Retomar”

- Retomar o movimento do retângulo após clicar novamente nele.

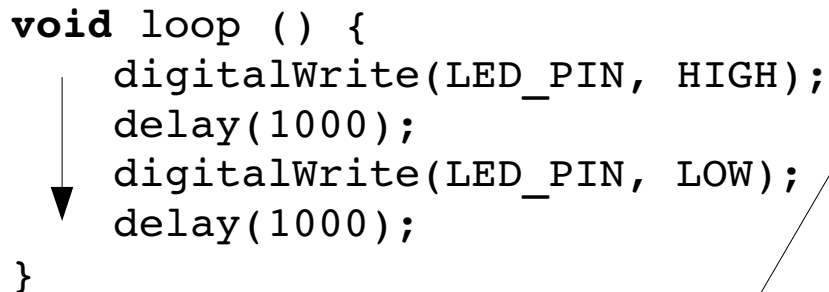
Compilando / Executando

```
> cd /home/ceu/ceu-sdl  
  
> make CEUFILE=/home/ceu/reativos/code/sdl/01_input.ceu  
> /home/ceu/reativos/code/sdl/01_input.exe  
  
> make CEUFILE=/home/ceu/reativos/code/sdl/ex_01.ceu  
> /home/ceu/reativos/code/sdl/ex_01.exe
```


Inversão de Controle

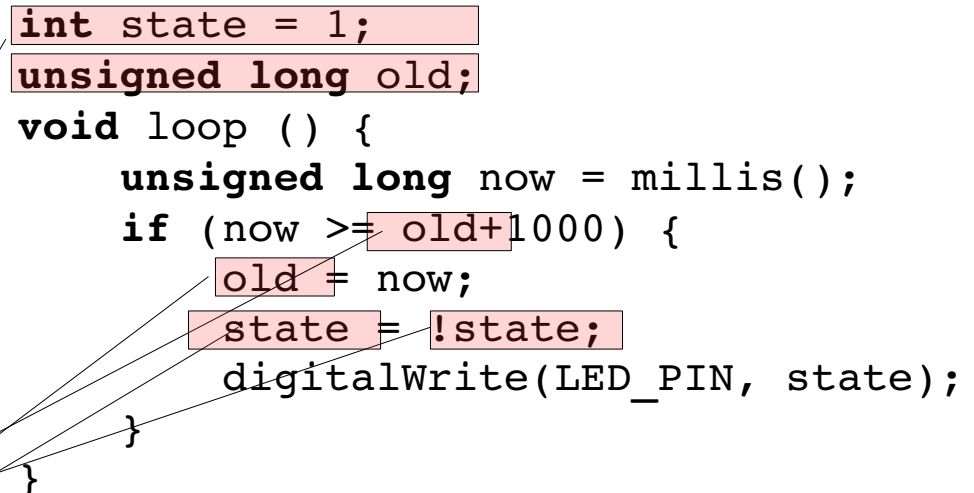
- Aplicação (programador) => Ambiente (dispositivos)
- Programação sequencial => Variáveis globais de estado

```
void loop () {  
    digitalWrite(LED_PIN, HIGH);  
    delay(1000);  
    digitalWrite(LED_PIN, LOW);  
    delay(1000);  
}
```



- inicialização
- decodificação
- codificação

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



Tradeoff

- Execução sequencial com chamadas bloqueantes
 - não reativo
- Inversão de controle e variáveis de estado
 - reativo

Máquinas de Estado

- Variáveis para controle de fluxo
 - booleanas (dois estados):
 - gerúndio: running, dying, isX, etc
 - particípio: pressed, visible, hasX, etc
 - inteiras (k estados):
 - time, state, direction, option, etc
- Exemplos encontrados:
 - `keep_going, old/newDirection, tNow/Old, entry/exitTime, clean, lost, isrepeat, last_moved, status, ultimot, mov, lPressed, reset`
- Olhar os ifs, variáveis globais/membros

Exemplo: Pingus

```
class ArmageddonButton {
  private:
    bool pressed = false;
    float press_time = 0;
    <...>
};

<...>

void ArmageddonButton::on_update (float delta) {
  if (this.pressed) {
    this.press_time += delta;
    if (this.press_time > 1.0f) {
      this.press_time = 0;
      this.pressed = false;
    }
  } else {
    this.pressed = false;
    this.press_time = 0;
  }
}

void ArmageddonButton::on_click (int x, int y) {
  if (this.pressed) {
    DESTROY_EVERYTHING();
  } else {
    pressed = true;
  }
}
}
```



```
class ArmageddonButton with
  <...>
do
  par do
    <...>
  with
    loop do
      par/or do
        await ON_CLICK;
        await ON_CLICK;
        _DESTROY_EVERYTHING();
      with
        await 1s;
      end
    end
  end
end
end
```


Tarefa-05

(24 de abril - próxima aula)

- Incluir uma opção de reinício no jogo
- Identificar todas as variáveis/máquinas de estado
 - `estado.md`

Mini-jogo

(08 de maio)

- Reimplementar o jogo em Céu
 - eliminando todas as máquinas de estado
- Adicionar alguma funcionalidade dinâmica no jogo

```
loop do
  par/or do
    <codigo-do-jogo>
  with
    <await-condicao-de-termino>
  end
end
end
```


Primeiros Passos

- Instalar Céu:
 - github (v0.9), ceu-windows (v0.9), CiB (v0.8)
 - <http://github.com/Tkachov/ceu-windows>
- Executar alguns exemplos:

```
> cd /home/ceu/ceu-sdl
> ls -l samples/*.ceu

> make CEUFILE=/home/ceu/reativos/code/sdl/01_input.ceu
> make CEUFILE=/home/ceu/reativos/code/sdl/ex_01.ceu
```


Documentação

- Tutorial online

- <http://www.ceu-lang.org/try.php>

- Manual de referência

- <https://github.com/fsantanna/ceu/blob/master/manual/manual-toc.md>

- Exemplos para Arduino e SDL

- <https://github.com/fsantanna/ceu-arduino/tree/master/samples>
 - <https://github.com/fsantanna/ceu-sdl/tree/master/samples>

- Grupo da Turma (!)

Artigos & Videos - 05

- Safe Concurrent Abstractions for Wireless Sensor Networks
 - http://ceu-lang.org/chico/ceu_sensys13_pre.pdf
- Structured Synchronous Reactive Programming with Céu
 - http://www.ceu-lang.org/chico/ceu_mod15_pre.pdf
 - <http://vimeo.com/110512582>