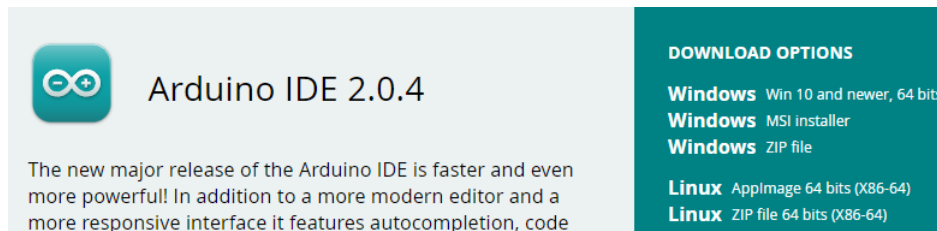


Material:

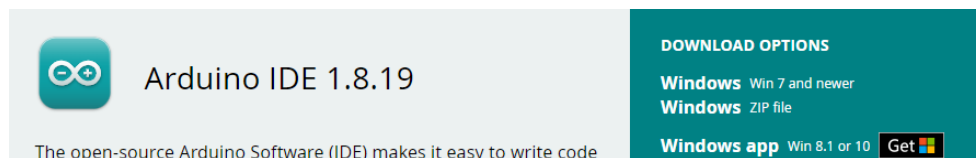
1. Arduino IDE (preferência pela 2.XX, mas pode ser a 1.8 também): [link](#)
2. Biblioteca FreeRTOS (instalada pela Arduino IDE)
3. Simulador SimulIDE 1.0: [link](#) (site oficial) ou [link](#) (disponível no material da disciplina).

Início com FreeRTOS para Arduino

1. (Caso já tenha baixado ou uso o computador da Univali, pule apara o passo 2) Baixe e instale o Arduino IDE no seu computador usando o link acima. Os computadores da Univali possuem uma a versão 1.8, porém você pode baixar a versão 2.XX já.

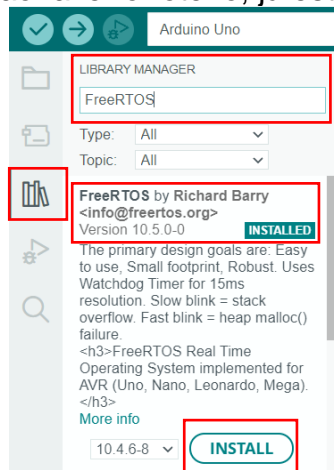


ou

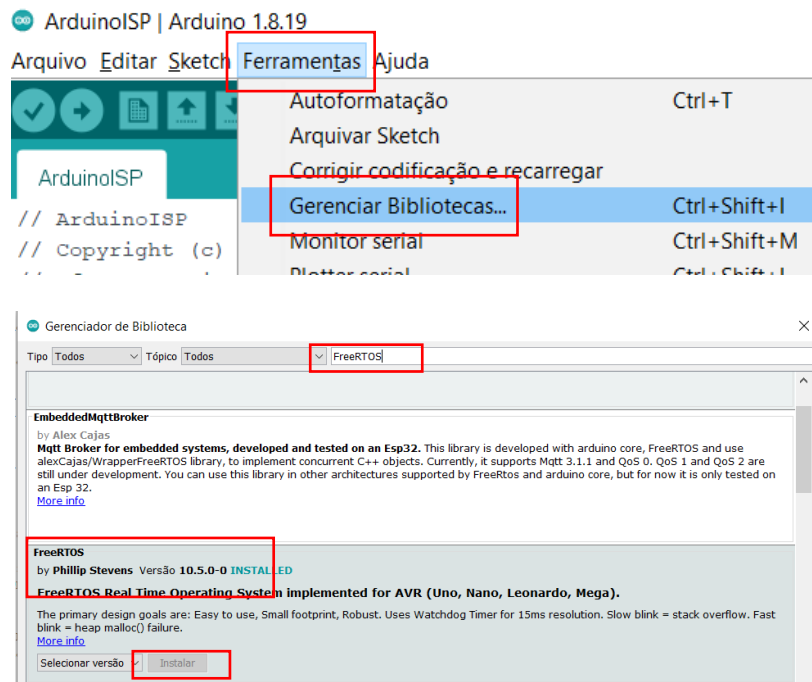


2. Instalando as bibliotecas Arduino do FreeRTOS

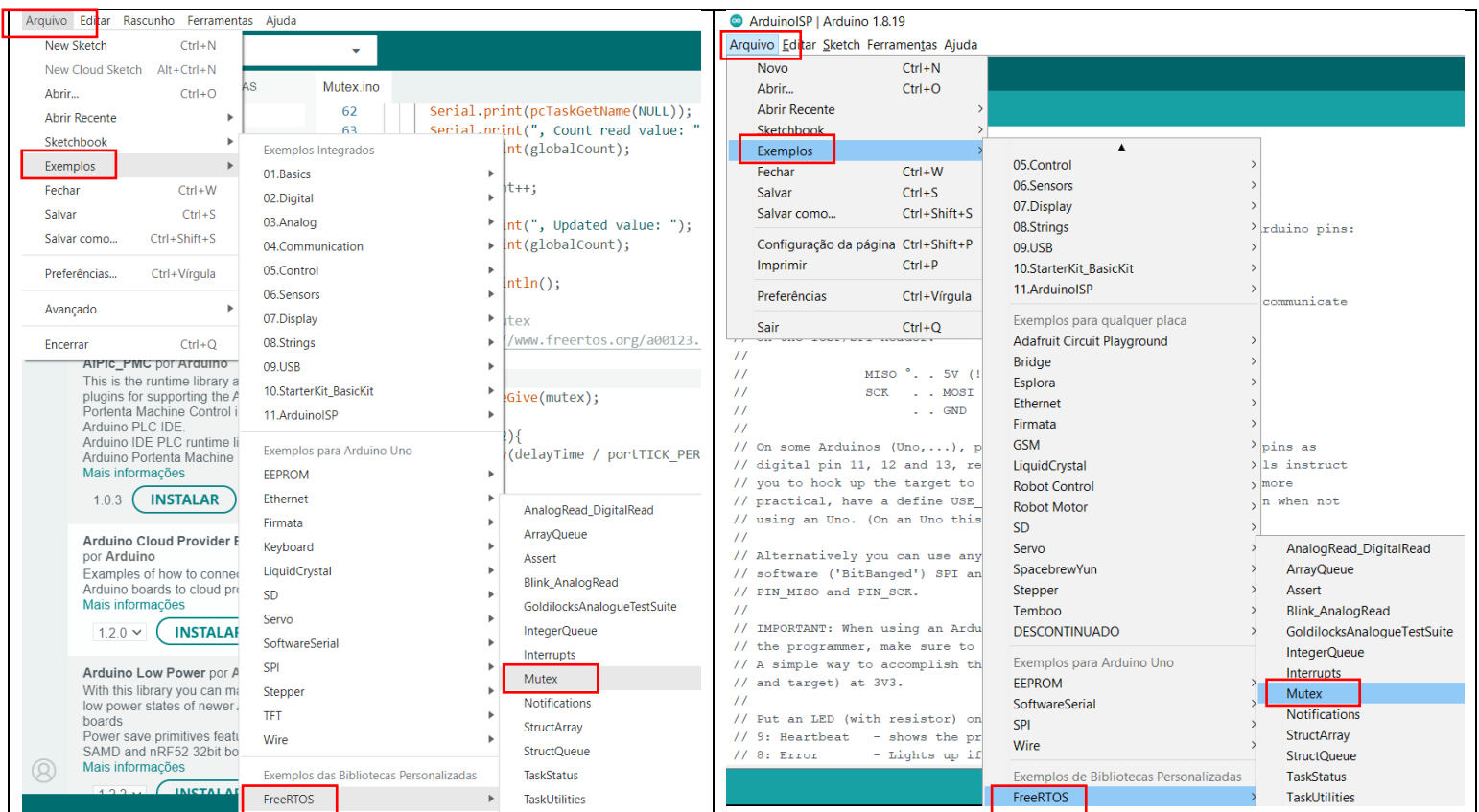
- a. Na versão 2.XX, clique em “Library Manager” ou “Gerenciador de Bibliotecas) e no campo de pesquisa digite FreeRTOS. Após isso instale a versão “FreeRTOS by Richard Barry” clicando em “Install” ou Instalar. (Obs: ao fazer o roteiro, já estava instalado na minha máquina).



- b. Na versão 1.XX, clique em “Ferramentas” e depois em “Gerenciar Bibliotecas”. No campo de pesquisa, digite FreeRTOS. Após isso instale a versão “FreeRTOS by Phillip Stevens” ou “FreeRTOS by Richard Barry” clicando em “Instalar”. (Obs: ao fazer o roteiro, já estava instalado na minha máquina). Outro local (Arduino IDE nos computadores da Univali) que opção de Gerenciador de Bibliotecas pode estar é indo no menu **Sketch** e depois em **Incluir Biblioteca**



3. Com a biblioteca instalada, vá em “Arquivo” e depois em “Exemplos”. Ao fim da lista, há a opção FreeRTOS. Nos exemplos disponíveis, escolha a opção Mutex.



4. Ao Clicar, irá abrir uma nova janela com o código do Arduino (Sketch com extensão .ino). Ali, haverá o código com três funções principais:
- `void setup()`: responsável por declarar parâmetros – constantes e variáveis.
 - `void loop()`: responsável por ter a parte comportamental do código em aplicações básicas de Arduino.
 - `void TaskMutex(void *pvParameters)`: função com o comportamento da thread (task ou processo no caso do FreeRTOS). O FreeRTOS é um sistema operacional chamada de RTOS (Real Time Operational System) e é voltado para sistemas embarcados que possuem bastante restrição de recursos. Por isso, ele não possui processos e threads, apenas task (que também são chamadas de threads nesse caso).

```
Mutex | Arduino 1.8.19
Arquivo Editar Sketch Ferramentas Ajuda

Mutex

/*
 * Example of a FreeRTOS mutex
 * https://www.freertos.org/Real-time-embedded-RTOS-mutexes.html
 */

// Include Arduino FreeRTOS library
#include <Arduino_FreeRTOS.h>

// Include mutex support
#include <semphr.h>

/*
 * Declaring a global variable of type SemaphoreHandle_t
 */
SemaphoreHandle_t mutex;

int globalCount = 0;

void setup() {

    Serial.begin(9600);
```

5. Dentro da função setup() há a criação das tasks/threads usando a função:

xTaskCreate(TaskFunction_t pvTaskCode, const char * const pcName, configSTACK_DEPTH_TYPE usStackDepth, void *pvParameters, UBaseType_t uxPriority, TaskHandle_t *pxCreatedTask)

Essa função possui os seguintes parâmetros:

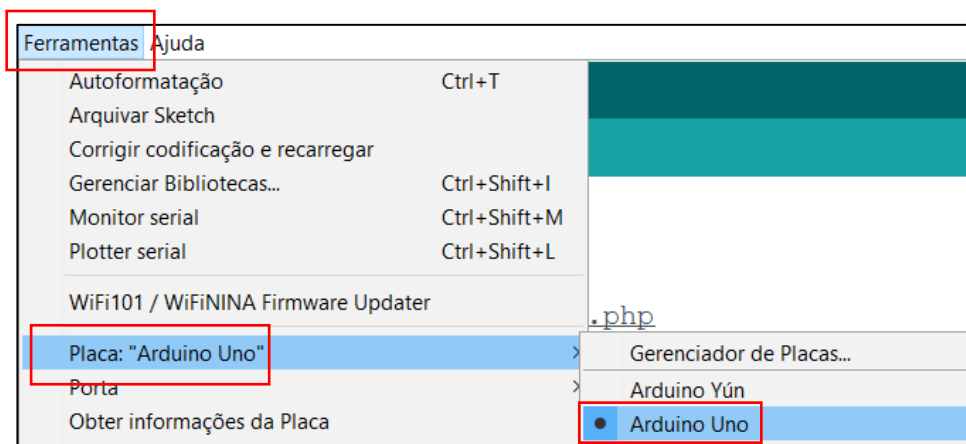
- TaskFunction_t pvTaskCode: Ponteiro para a função de entrada da tarefa (apenas o nome da função que implementa a tarefa).
- const char * const pcName: Um nome descritivo para a tarefa. Isso é usado principalmente para facilitar a depuração, mas também pode ser usado para obter um identificador de tarefa.
- configSTACK_DEPTH_TYPE usStackDepth: O número de palavras (não bytes!) a serem alocadas para uso como a pilha da tarefa. Por exemplo, se a pilha tiver 16 bits de largura e usStackDepth for 100, 200 bytes serão alocados para uso como a pilha da tarefa.
- void *pvParameters: Um valor que é passado como parâmetro para a tarefa criada.
- UBaseType_t uxPriority: A prioridade na qual a tarefa criada será executada.
- TaskHandle_t *pxCreatedTask: Usado para passar um identificador para a tarefa criada fora da função xTaskCreate() (uma espécie de ID de outra task). pxCreatedTask é opcional e pode ser definido como NULL.


```
*/
xTaskCreate(TaskMutex, // Task function
            "Task1", // Task name for humans
            128,
            1000, // Task parameter
            1, // Task priority
            NULL);

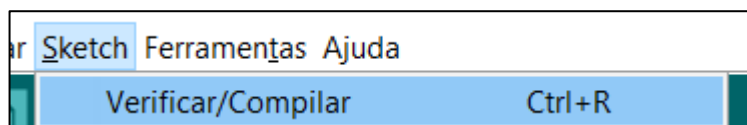
xTaskCreate(TaskMutex, "Task2", 128, 1000, 1, NULL);
```

6. Agora vamos verificar se o código está correto.

- Para isso, precisamos criar uma cópia desse código e salvar em um local de sua escolha (por exemplo na Área de Trabalho). Para isso vá em “Arquivo” e depois clique em “Salvar como..”. Esse salvamento é necessário para podermos alterar o código posteriormente. Salve como Mutex_Arduino.
Garanta que a compilação seja direcionada para a placa Arduino UNO. Para isso, na Arduino IDE vá em “Ferramentas” e depois em “Placa: “ selecione a Arduino UNO.



- d. Posteriormente ao salvamento, clique no botão verificar  ou vá em “Sketch” ou “Rascunho” e depois clique em Verificar/Compilar (o atalho é CTRL+R).



- e. Após verificar/compilar, ele irá executar e exibir uma mensagem como essa (Arduino IDE 2.XX):

```
Saída
Sketch uses 9852 bytes (30%) of program storage space. Maximum is 32256 bytes.
Global variables use 413 bytes (20%) of dynamic memory, leaving 1635 bytes for local variables. Maximum is 2048 bytes.
```

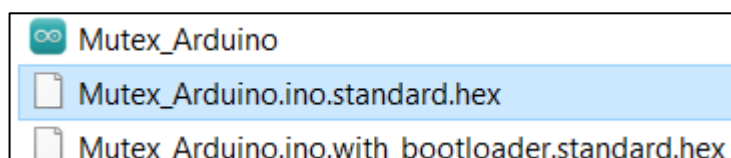
ou essa (Arduino IDE 1.8):

```
Compilação terminada.
C:\Users\leds\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-gcc -w -Os -g
"C:\Users\leds\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-objcopy" -O ih
"C:\Users\leds\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-objcopy" -O ih
Usando a biblioteca FreeRTOS na versão 10.5.0-0 na pasta: D:\OneDrive\OneDrive - UNIVALI\Documents\Arduino\libraries\FreeRTOS
"C:\Users\leds\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-size" -A "C:\
O sketch usa 9556 bytes (29%) de espaço de armazenamento para programas. O máximo são 32256 bytes.
Variáveis globais usam 413 bytes (20%) de memória dinâmica, deixando 1635 bytes para variáveis locais. O máximo são 2048 bytes.
```

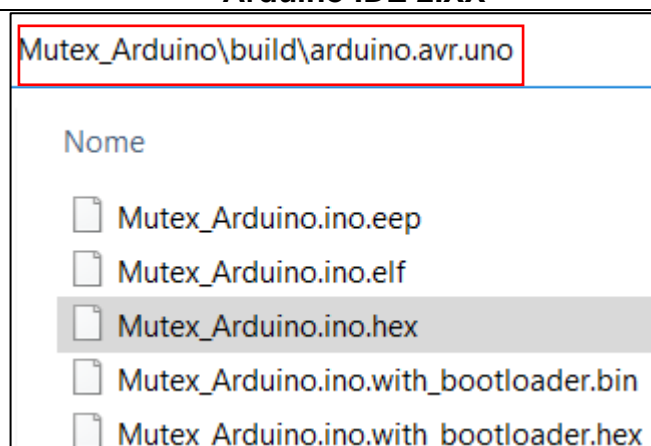
7. Agora que sabemos que o código compila, iremos simular a execução de uma placa Arduino e verificar o código com o SO rodando. Antes de vermos o simulador, vamos ter que gerar nossa “imagem” do SO para botar para executar no Arduino.

- Vá em “Sketch” ou em “Rascunho” e clique na opção “Exportar binário compilado”.
- Na pasta onde você salvou a sua cópia do seu projeto, irá ter o arquivo .HEX para podermos carregar no Arduino do simulador. Se a versão da Arduino IDE for a 2.XX, terá uma pasta como build/Arduino.avr.uno/ com o arquivo .HEX dentro.

Arduino IDE 1.8



Arduino IDE 2.XX

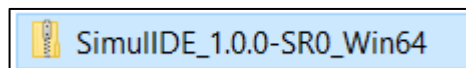


Início com SimulIDE e Arduino

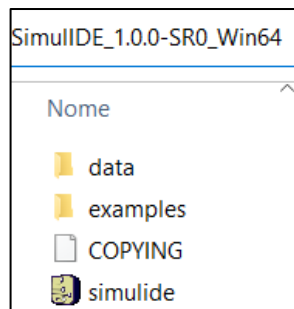
1. Baixe o simulador SimulIDE em um dos links fornecidos. Dê preferência pelo link do material da disciplina, já que não tem o processo de fornecer email e uma série de passos necessários para baixar.



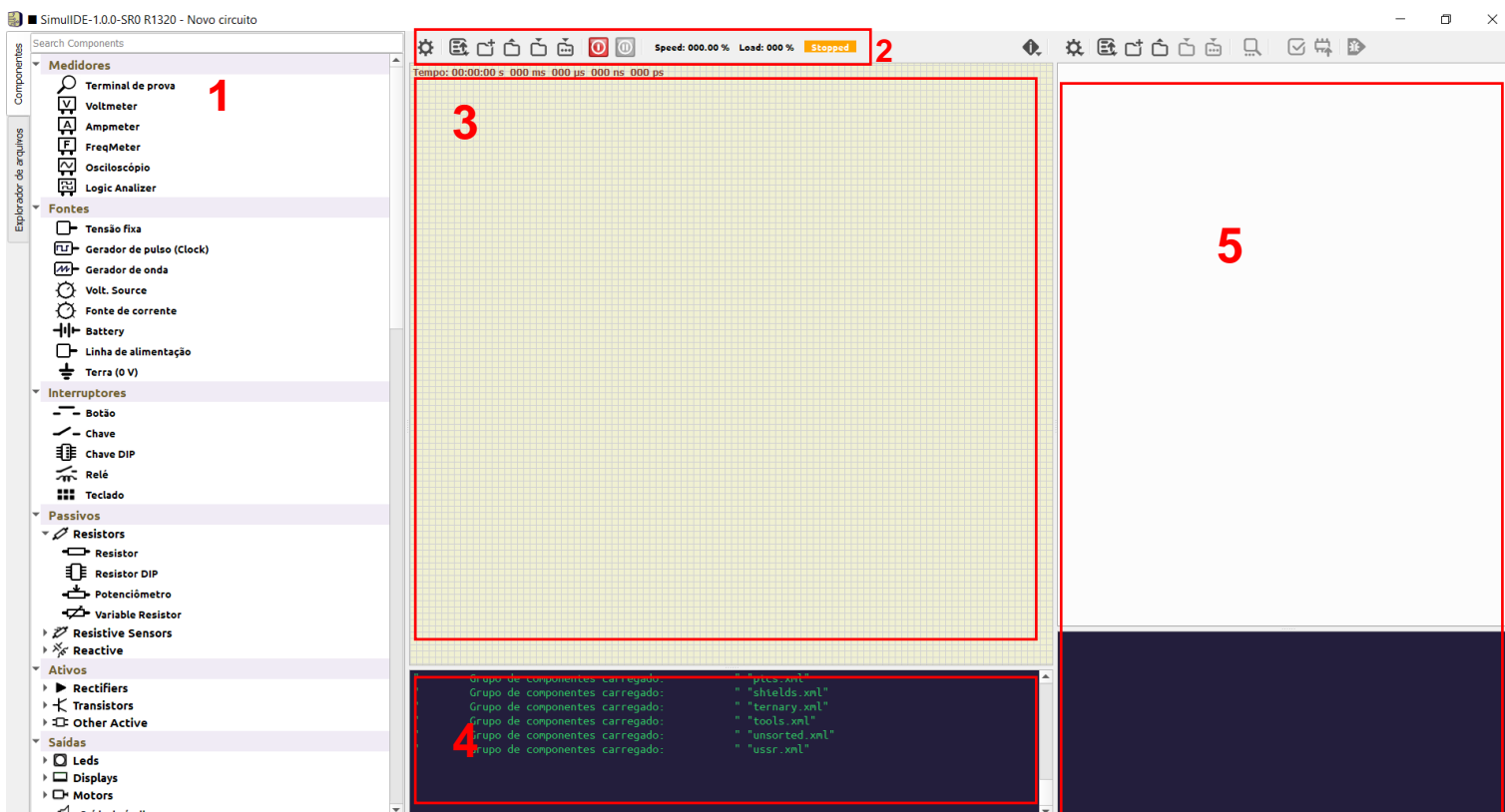
O SimulIDE não precisa de instalação, apenas descompacte ele em um local de interesse.



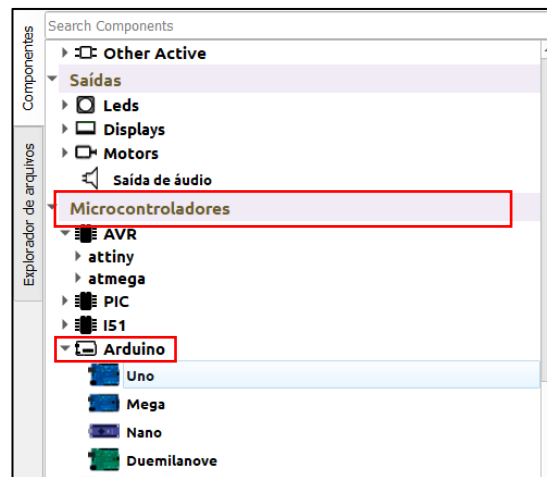
Você verá os seguintes arquivos:



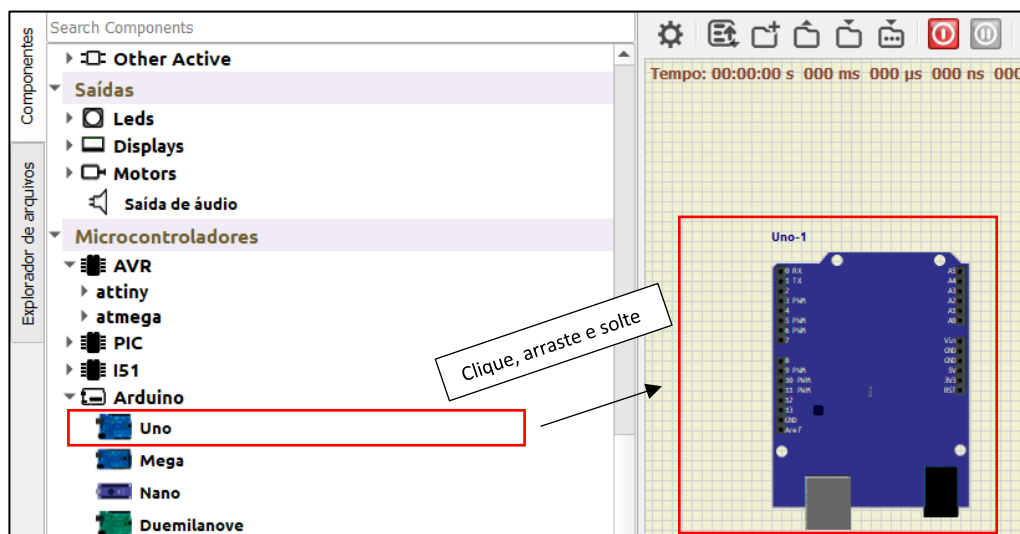
2. Abra o arquivo SimulIDE. Você verá uma tela como está:



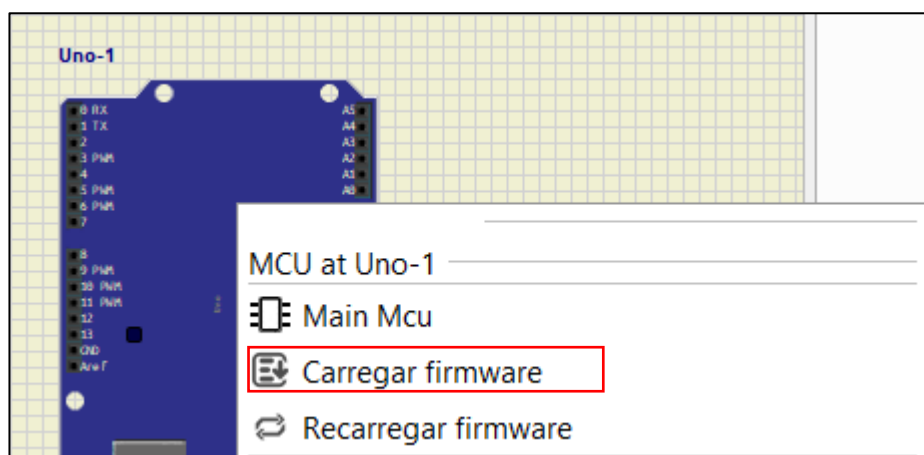
- a. Área 1: painel de navegação de componentes.
 - b. Área 2: local com botões de abrir, salvar, iniciar/parar simulação e pausar/retomar simulação.
 - c. Área 3: local para montar sua simulação de hardware.
 - d. Área 4: local com log de operações feitas no simulador.
 - e. Área 5: manipulação de arquivos e código (**não usaremos**).
3. Agora, no painel de navegação de componentes (1), procure pela Seção Microcontroladores e pela subseção Arduino.



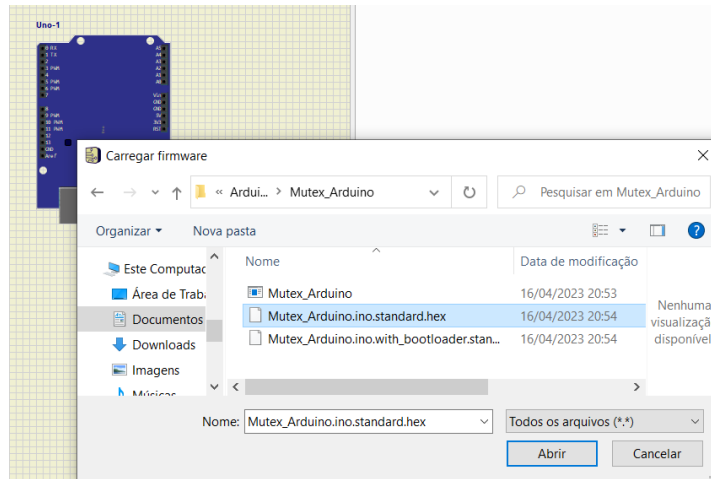
- a. Clique e arraste o Arduino Uno para a área de simulação de hardware (3).



4. Com o Arduino UNO na área de simulação, ele já está pronto para uso (não precisamos energizar ou conectá-lo a nada). Agora, clique com o botão direito do mouse sobre a placa Arduino UNO em (3) e clique na opção “Carregar firmware”.



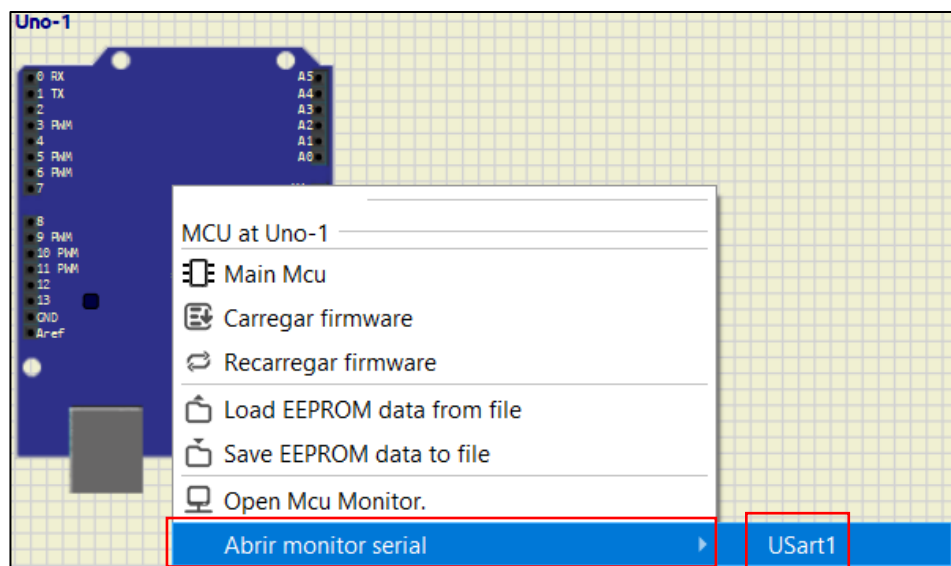
5. Na janela que abrir, procure a pasta onde está salvo o seu projeto Mutex_Arduino. Selecione o arquivo “Mutex_Arduino.ino.standard.hex” ou “Mutex_Arduino.ino.hex” (dependendo da sua versão da Arduino IDE).



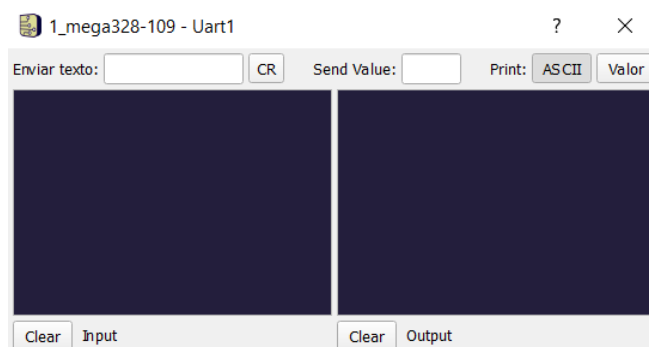
Você verá a seguinte mensagem em (4):

```
Loading hex file:  
"D:\...\br/>Mutex_Arduino/Mutex_Arduino.ino.standard.hex"  
Firmware succesfully loaded
```

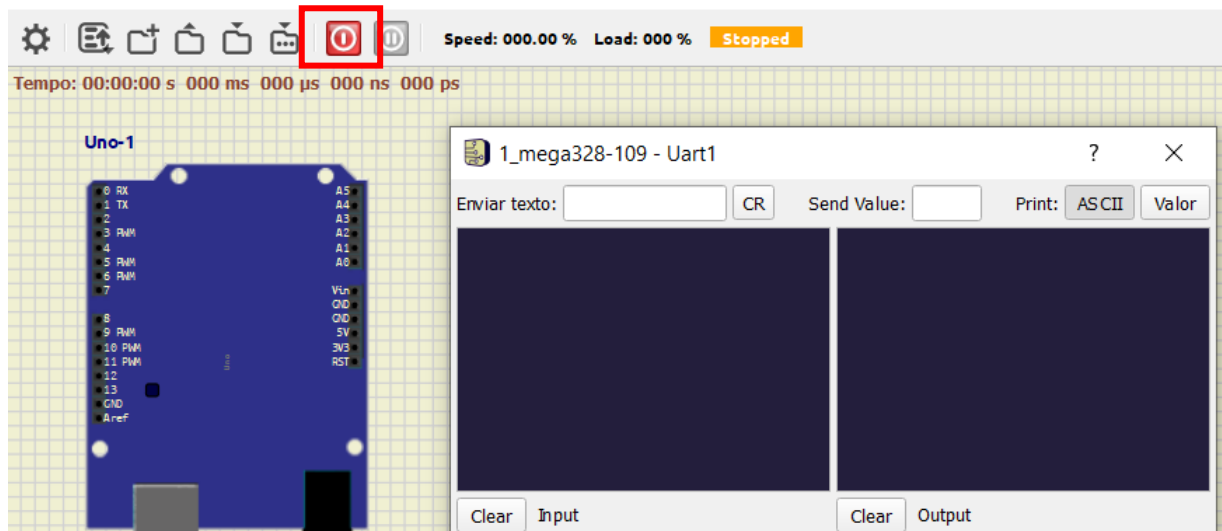
6. Com o firmware carregado, agora clique novamente com o botão direito do mouse sobre o Arduino UNO e vá na opção “Abrir monitor serial” e depois clique em “USart1”.



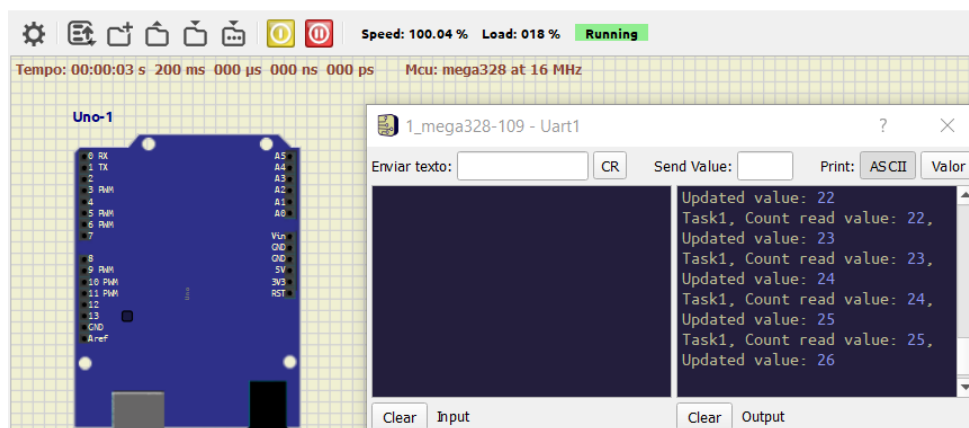
A tela de display de mensagens irá abrir:



7. Após carregar o firmware e abri a tela para exibir as mensagens, clique em iniciar ligar (ou iniciar a simulação):



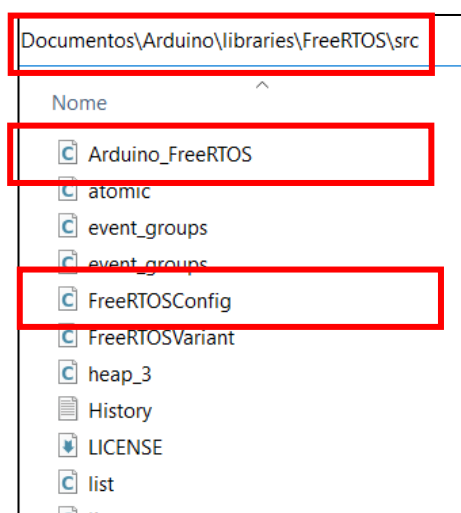
Após apertar o botão de ligar:



Agora você poder parar a simulação ou pausar ela. Na tela de exibir as mensagens irá ter o botão “Clear” para limpar o terminal caso você queira reiniciar a simulação.

Mexendo com o Escalonador do Arduino

1. Agora que vimos como adicionar o FreeRTOS ao Arduino e simular ele no SimulIDE, podemos verificar o comportamento do escalonador. Vá até a pasta onde está o código fonte do FreeRTOS para Arduino. Está pasta costuma ficar em **“Documentos\Arduino\libraries\FreeRTOS\src”**.



2. Nessa pasta, podemos alterar o comportamento do escalonador do FreeRTOS. A duas configurações principais que podemos alterar no arquivo “**FreeRTOSConfig.h**”:
- configUSE_PREEMPTION**: Defina como 1 para usar o escalonador RTOS preemptivo ou 0 para usar o escalonador RTOS cooperativo.
 - configUSE_TIME_SLICING**: Por padrão (se configUSE_TIME_SLICING não for definido ou se configUSE_TIME_SLICING for definido como 1), o FreeRTOS usa escalonamento preemptivo priorizado com divisão de tempo. Isso significa que o escalonador do RTOS sempre executará a tarefa de prioridade mais alta que está no estado Pronto e alternará entre tarefas de igual prioridade em cada interrupção de tick do RTOS. Se configUSE_TIME_SLICING for definido como 0, o escalonador RTOS ainda executará a tarefa de prioridade mais alta que está no estado Pronto, mas não alternará entre tarefas de igual prioridade apenas porque ocorreu uma interrupção de tick.

```
/* And on to the things the same no matter the AVR type... */
#define configUSE_PREEMPTION 1

#define configCPU_CLOCK_HZ ( ( uint32_t ) F_CPU )
#define configMAX_PRIORITIES 4
#define configIDLE_SHOULD_YIELD 1
#define configMINIMAL_STACK_SIZE ( 192 )
#define configMAX_TASK_NAME_LEN ( 8 )

#define configQUEUE_REGISTRY_SIZE 0
#define configCHECK_FOR_STACK_OVERFLOW 1

#define configUSE_TRACE_FACILITY 0
#define configUSE_16_BIT_TICKS 1

#define configUSE_MUTEXES 1
#define configUSE_RECURSIVE_MUTEXES 1
#define configUSE_COUNTING_SEMAPHORES 1
#define configUSE_TIME_SLICING 1
```

Exercícios:

- Investigue a função das outras funções do FreeRTOS usadas no código Mutex_Arduino.
- Como exercício para escalonador, faça:
 - Altere esses parâmetros e verifique se há diferença na simulação. Teste as 4 possibilidades de configuração:
 - configUSE_PREEMPTION = 1 e configUSE_TIME_SLICING = 1
 - configUSE_PREEMPTION = 0 e configUSE_TIME_SLICING = 1
 - configUSE_PREEMPTION = 1 e configUSE_TIME_SLICING = 0
 - configUSE_PREEMPTION = 0 e configUSE_TIME_SLICING = 0

Notou diferença em alguma delas?

- Agora altere a prioridade das tasks Task1 e Task2 para 2 e 3 e novamente testes as quatro possibilidades de configuração. Notou diferença em alguma delas?
- Agora faça o seguinte:
 - Duplique a função TaskMutex e chame a nova de TaskMutexModificada.
 - Dentro da TaskMutexModificada, adicione uma variável chamada sleep (pode colocar outro nome) e incremente a mesma dentro do laço de repetição **for(;;)**. Coloque a função vTaskDelay dentro de um if que verifica se sleep é menor que 2.
 - Agora teste novamente as 4 possibilidades de configuração de escalonamento. Consegue ver alguma diferença de comportamento?
- Faça uma pesquisa para dizer qual é a diferença entre escalonamento preemptivo e escalonamento cooperativo.

Exemplo da TaskMutexModificada modificada:

```
TickType_t delayTime = *((TickType_t*)pvParameters);
int sleep = 0;
for (;;)
{
    /**
     * Take mutex
     * https://www.freertos.org/a00122.html
     */
    if (xSemaphoreTake(mutex, 10) == pdTRUE)
    {
        Serial.print(pcTaskGetName(NULL)); // Get task name
        Serial.print(", Count read value: ");
        Serial.print(globalCount);

        globalCount++;

        Serial.print(", Updated value: ");
        Serial.print(globalCount);

        Serial.println();

        /**
         * Give mutex
         * https://www.freertos.org/a00123.html
         */
        sleep++;
        xSemaphoreGive(mutex);
    }
    if(sleep < 2){
        vTaskDelay(delayTime / portTICK_PERIOD_MS);
    }
}
```

Obs: Não esqueça de trocar a função usada pela Task1 ou Task2 (escolha uma delas) pela TaskMutexModificada.