SCALE 20X | MARCH 2023

Symmetric Multiprocessing (SMP) with FreeRTOS and Raspberry Pi Pico

Daniel Gross

Senior Developer Advocate
IoT Ecosystem Services
AWS

Agenda

- Overview of FreeRTOS
- Raspberry Pi Pico Dev Board
- AMP vs. SMP
- Multitasking and Scheduling
- Code Examples and Configuration
- SMP Specific APIs
- Resources

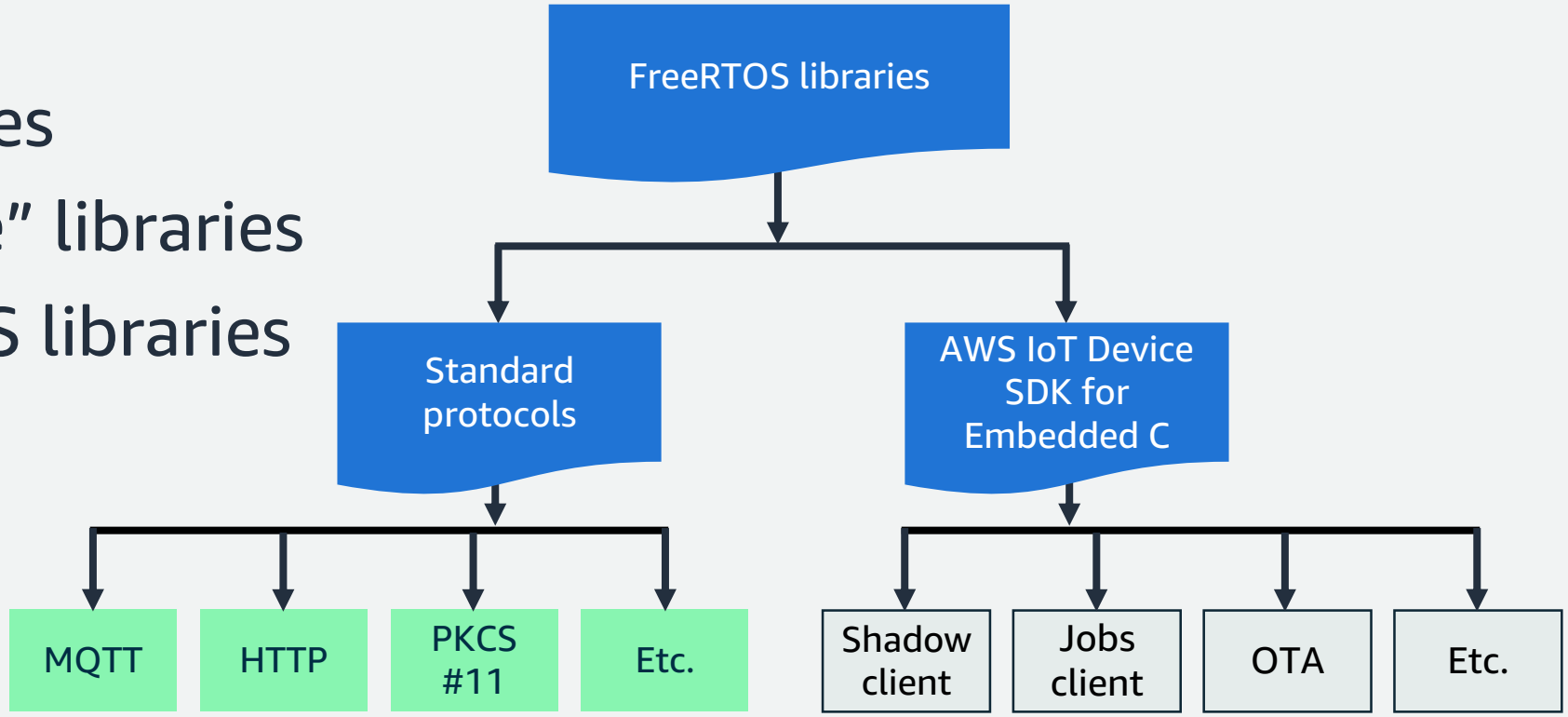
Overview of FreeRTOS

https://freertos.org
https://github.com/freertos

- 18+ years trusted, widely distributed (downloaded every 170 seconds)
- 40+ supported architectures, including Armv8-M (Cortex-M33) and RISC-V
- Broad partner ecosystem support
- Permissive MIT open source license
- Kernel plus modular libraries
- Improved Inter-Process Communication (IPC) capabilities with stream and message buffers

More than just a real-time kernel

- MIT licensed open-source software
- Real-time kernel
- FreeRTOS+ libraries
- Composable “core” libraries
- FreeRTOS for AWS libraries

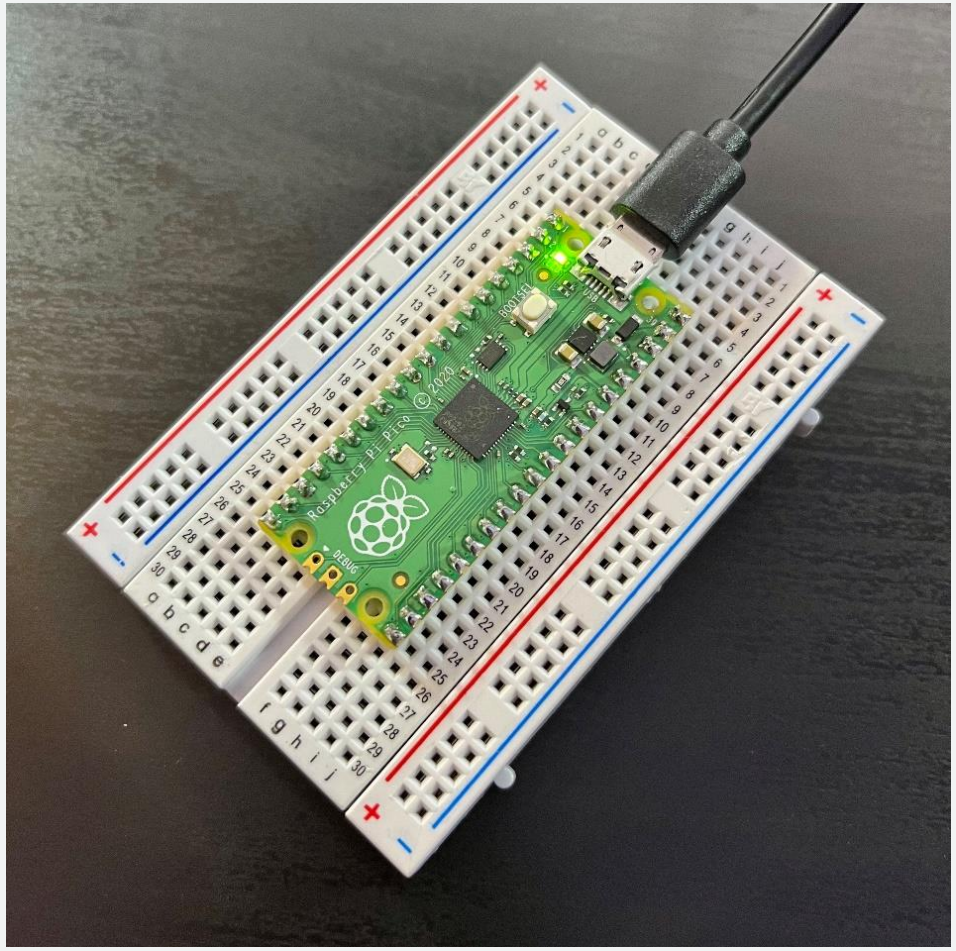


Intentionally unopinionated

- Take the code to the developer
- Basic C code only
- No library dependencies (other than the C library)
- No tools dependencies (but available within vendor IDEs and SDKs)
- Enables 8-, 16-, 32-, and 64-bit architectures

Raspberry Pi Pico Dev Board

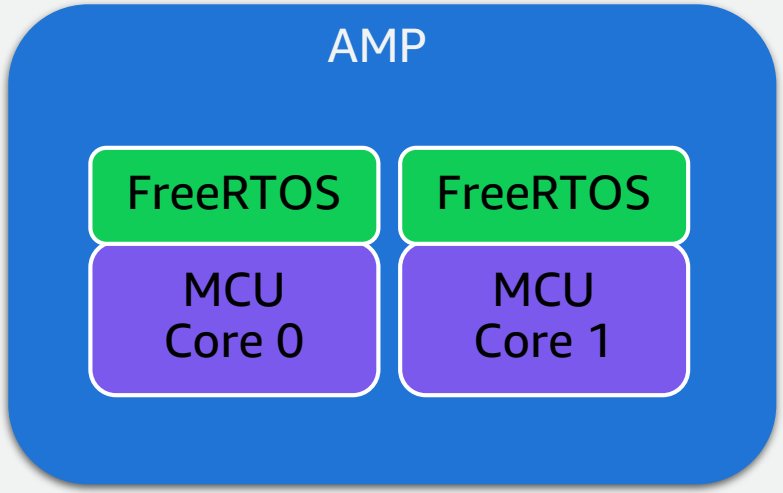
- RP2040 Microcontroller
- 133MHz **Dual-core** Arm Cortex-M0+
- 264KB on-chip SRAM
- 2MB on-board QSPI Flash
- 26 GPIO pins, 2x UART, 2x SPI, 2x I2C
- Open-source board design files
- FreeRTOS port and demos available
- C/C++ SDK and technical specification
  https://www.raspberrypi.com/documentation/microcontrollers/



AMP vs. SMP

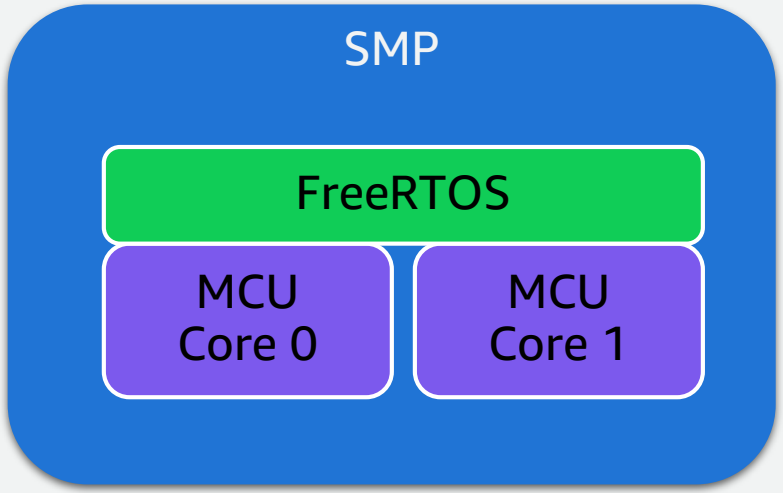
Asymmetric Multiprocessing (AMP)

Each processor core runs its own instance of FreeRTOS



Symmetric Multiprocessing (SMP)

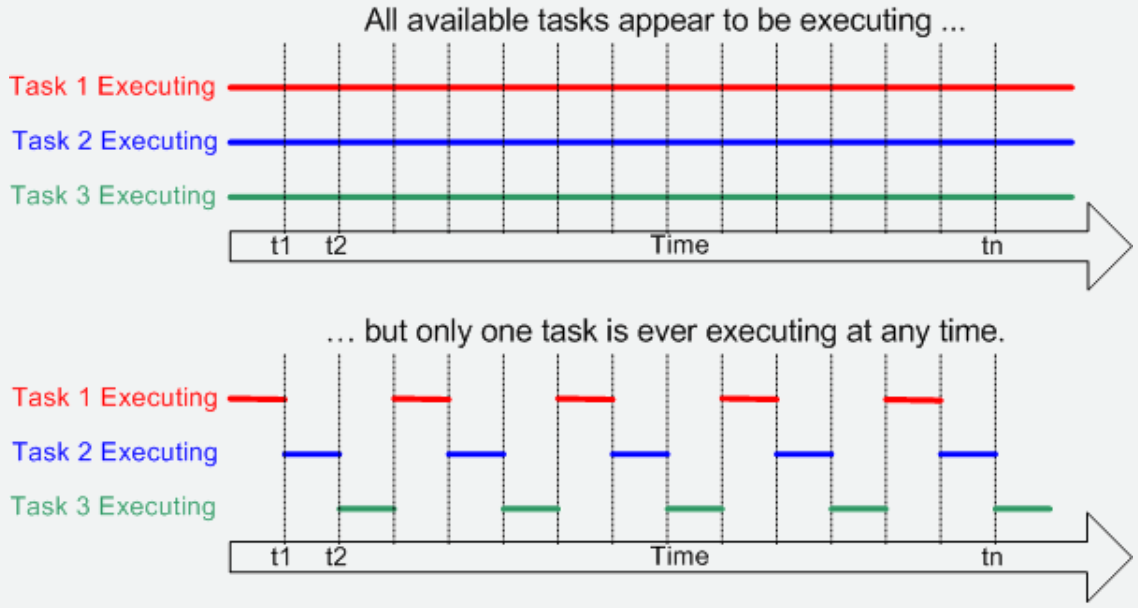
One instance of FreeRTOS that schedules tasks across multiple cores



Multitasking and Scheduling

A conventional processor can only execute a single task at a time - but by rapidly switching between tasks a multitasking operating system can make it **appear** as if each task is executing concurrently.

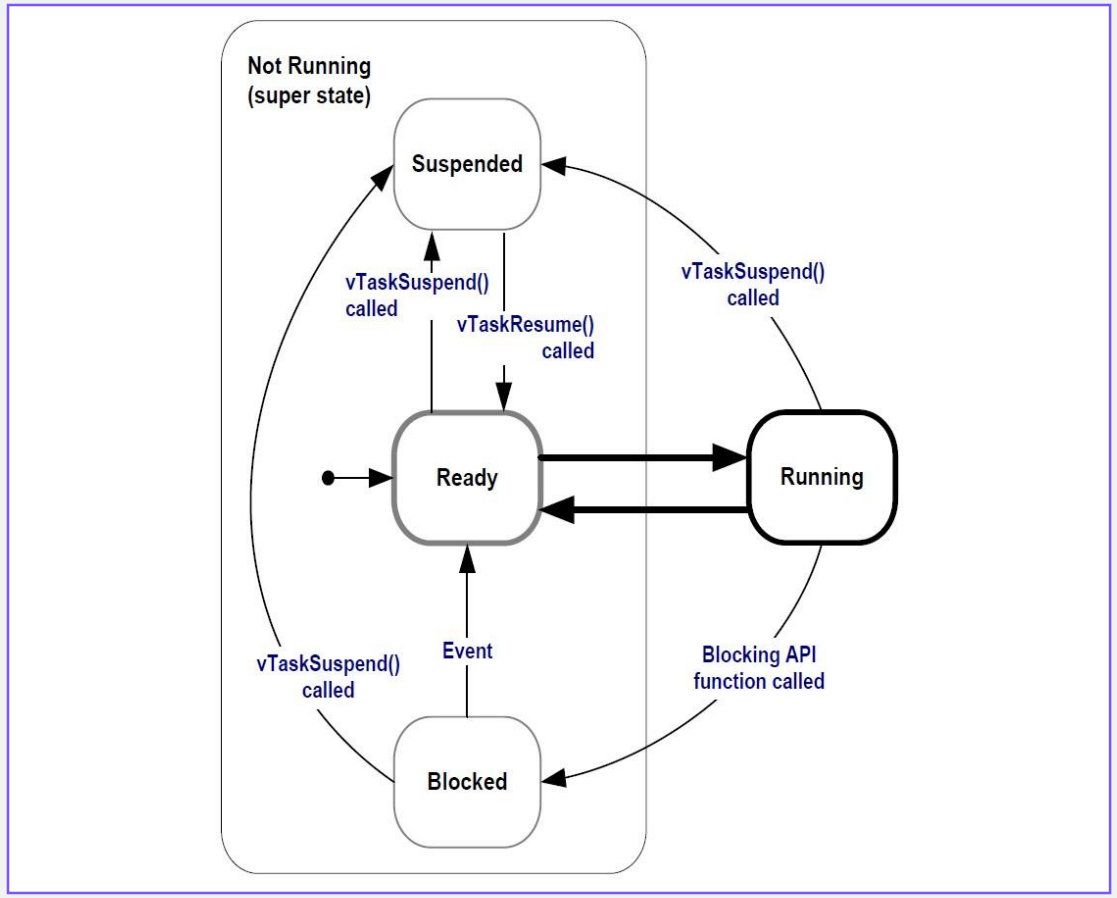
The **scheduler** is the part of the kernel responsible for deciding which task should be executing at any particular time. The kernel can suspend and later resume a task many times during the task lifetime.



FreeRTOS Task Scheduling

- By default, **FreeRTOS** uses a fixed-priority preemptive scheduling policy, with round-robin time-slicing of equal priority tasks.
- "**Fixed priority**" – The scheduler will not permanently change the priority of a task, although it may temporarily boost the priority of a task due to priority inheritance.
- "**Preemptive**" – The scheduler always runs the highest priority RTOS task that is able to run, regardless of when a task becomes able to run. A lower priority task can be "preempted" by a higher priority task.
- "**Round-robin**" – Tasks that share a priority take turns entering the **Running** state.
- "**Time sliced**“ – The scheduler will switch between tasks of equal priority on each tick interrupt - the time between tick interrupts being one time slice.

Full Task State Machine



- When a task is executing, it is in the Running state.
- When a task is not executing, it is in the Suspended, Blocked or Ready state.
- Tasks in the Blocked or Suspended state do not use any processing time and cannot be selected to enter the Running state.
- Unlike the single-core and AMP scenarios, **SMP** results in more than one task being in the Running state at any given time – there is one Running state task per core.

Task Priorities in FreeRTOS

- Each task is assigned a priority from 0 to (**configMAX_PRIORITIES** - 1), where **configMAX_PRIORITIES** is defined within **FreeRTOSConfig.h**
- Low priority numbers denote low priority tasks. The idle task has priority zero (**tskIDLE_PRIORITY**)
- API calls: xTaskCreate() & vTaskPrioritySet()
- The **FreeRTOS** scheduler ensures that tasks in the **Ready** or **Running** state will always be given processor (CPU) time in preference to tasks of a lower priority that are also in the ready state. In other words, the task placed into the **Running** state is always the highest priority task that is able to run
- Any number of tasks can share the same priority. If **configUSE_TIME_SLICING** is not defined, or if **configUSE_TIME_SLICING** is set to 1, then **Ready** state tasks of equal priority will share the available processing time using a time sliced round robin scheduling scheme

Example 1

Example 1 Configuration

FreeRTOSConfig.h

```
/* Application does not rely on task priorities and
uses synchronization primitives instead. */
#define configNUM_CORES                         2
#define configRUN_MULTIPLE_PRIORITIES           0
#define configUSE_CORE_AFFINITY                 1
#define configTICK_CORE                         0
#define configSUPPORT_PICO_SYNC_INTEROP         1
#define configSUPPORT_PICO_TIME_INTEROP         1
```

Example 1 Code

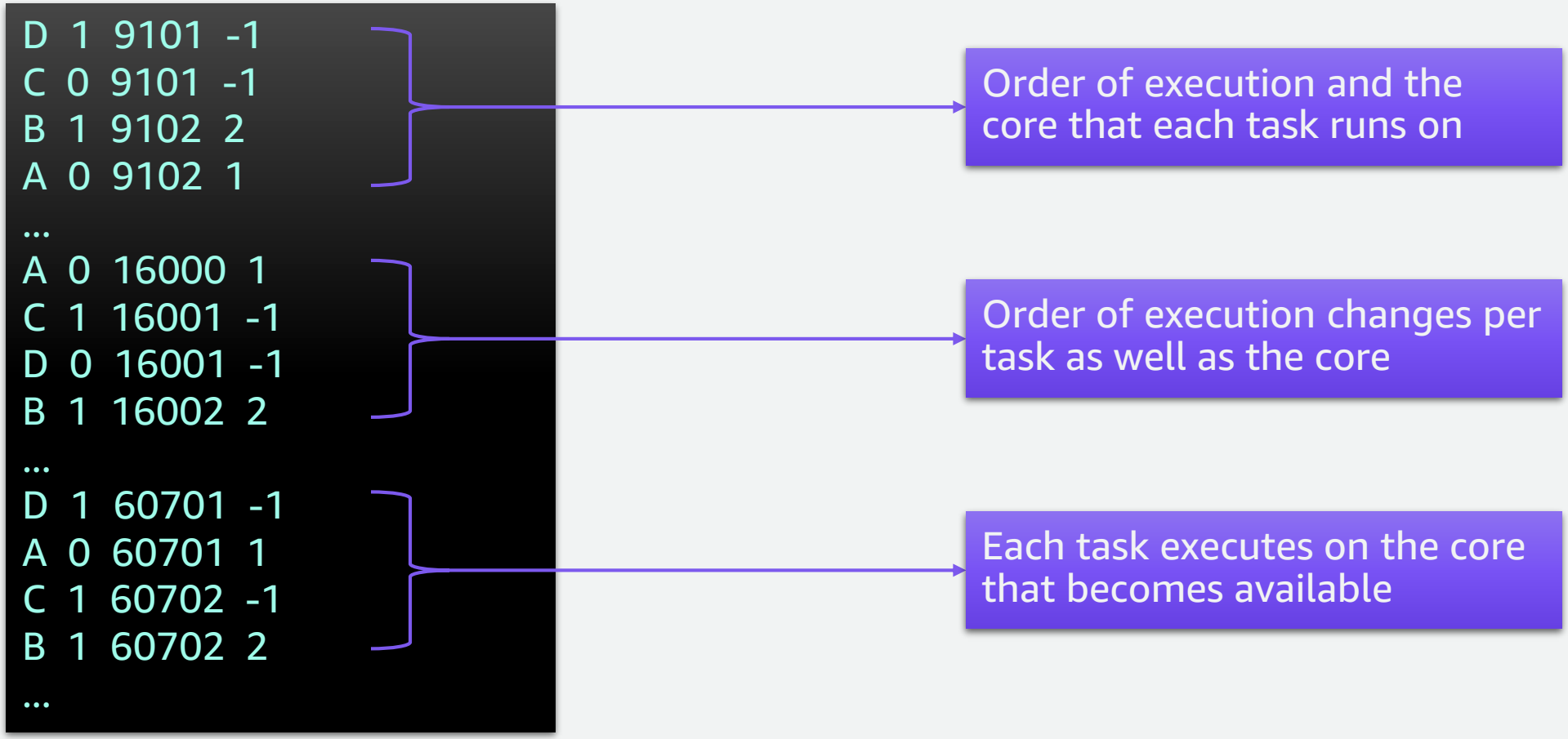
```c
#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/multicore.h"
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

const int task_delay = 200;
const int task_size = 128;

SemaphoreHandle_t mutex;

void vSafePrint(char *out) {
  xSemaphoreTake(mutex, portMAX_DELAY);
  puts(out);
  xSemaphoreGive(mutex);
}
```

```c
void vTaskSMP(void *pvParameters) {
  TaskHandle_t handle = xTaskGetCurrentTaskHandle();
  UBaseType_t mask = vTaskCoreAffinityGet(handle);
  char *name = pcTaskGetName(handle);
  char out[24];
  for (;;) {
    sprintf(out,"%s %d %d %d", name,
      get_core_num(), xTaskGetTickCount(), mask);
    vSafePrint(out);
    vTaskDelay(taskDelay);
  }
}
```

Example 1 Code (continued)

```
void main() {
  stdio_init_all();
  mutex = xSemaphoreCreateMutex();
  TaskHandle_t handleA;
  TaskHandle_t handleB;
  xTaskCreate(vTaskSMP, "A", taskSize, NULL, 1, &handleA);
  xTaskCreate(vTaskSMP, "B", taskSize, NULL, 1, &handleB);
  xTaskCreate(vTaskSMP, "C", taskSize, NULL, 1, NULL);
  xTaskCreate(vTaskSMP, "D", taskSize, NULL, 1, NULL);
  vTaskCoreAffinitySet(handleA, (1 << 0));
  vTaskCoreAffinitySet(handleB, (1 << 1));
  vTaskStartScheduler();
}
```

Example 1 Output



SMP Specific APIs – More Control

API	Description	Configuration
vTaskCoreAffinitySet	Sets the core affinity mask for a task, i.e. the cores on which a task can run. To ensure that a task can run on core 0 and core 1, set uxCoreAffinityMask to 0x03.	configUSE_CORE_AFFINITY 1
vTaskCoreAffinityGet	Gets the core affinity mask for a task, i.e. the cores on which a task can run. If a task can run on core 0 and core 1, the core affinity mask is 0x03.	configUSE_CORE_AFFINITY 1
vTaskPreemptionDisable	Disables preemption for a task. The task will not be preempted when it is executing code after this call.	configUSE_TASK_PREEMPTION_DISABLE 1
vTaskPreemptionEnable	Enables preemption for a task. The task can be preempted when it is executing code after this call.	configUSE_TASK_PREEMPTION_DISABLE 1

Example 2

Example 2 Configuration

FreeRTOSConfig.h

```
#define configNUM_CORES                         2
#define configRUN_MULTIPLE_PRIORITIES           1
#define configUSE_CORE_AFFINITY                 1
#define configTICK_CORE                         0
#define configSUPPORT_PICO_SYNC_INTEROP         1
#define configSUPPORT_PICO_TIME_INTEROP         1
```

Example 2 Code

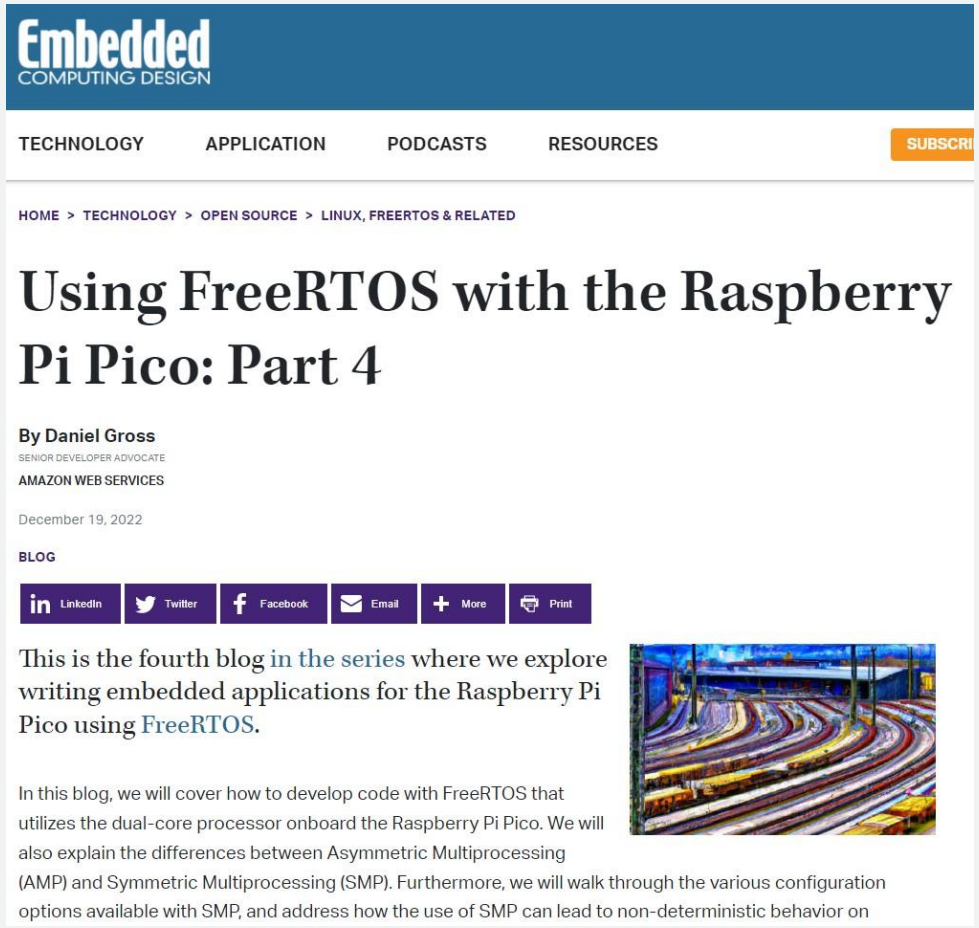
```
/* Task handle of the networking task - it is populated elsewhere. */
TaskHandle_t xNetworkingTaskHandle;
void vAFunction( void ) {
  TaskHandle_t xHandle;
  UBaseType_t uxNetworkingCoreAffinityMask;
  /* Create a task, storing the handle. */
  xTaskCreate( vTaskCode, "NAME", STACK_SIZE, NULL, tskIDLE_PRIORITY, &( xHandle ) );
  /* Get the core affinity mask for the networking task. */
  uxNetworkingCoreAffinityMask = vTaskCoreAffinityGet( xNetworkingTaskHandle );
  /* Here is a hypothetical scenario, just for the example. Assume that we have 2 cores - Core 0
   * and core 1. We want to pin the application task to the core that is not the networking task
   * core to ensure that the application task does not interfere with networking. */
  if ( ( uxNetworkingCoreAffinityMask & ( 1 << 0 ) ) != 0 ) {
    /* The networking task can run on core 0, pin our task to core 1. */
    vTaskCoreAffinitySet( xHandle, ( 1 << 1 ) );
  } else {
    /* Otherwise, pin our task to core 0. */
    vTaskCoreAffinitySet( xHandle, ( 1 << 0 ) );
  }
}
```

Additional SMP Support with FreeRTOS

XMOS XCORE.AI Explorer Board
https://freertos.org/smp-demo-for-xmos-xcore-ai-explorer-board.html

Espressif ESP-IDF FreeRTOS (SMP) for dual-core ESP32/ESP32-S3
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/freertos-smp.html

Resources 1 of 2


Embedded
COMPUTING DESIGN

TECHNOLOGY APPLICATION PODCASTS RESOURCES SUBSCRI

HOME > TECHNOLOGY > OPEN SOURCE > LINUX, FREERTOS & RELATED

Using FreeRTOS with the Raspberry Pi Pico: Part 4

By Daniel Gross
SENIOR DEVELOPER ADVOCATE
AMAZON WEB SERVICES

December 19, 2022

BLOG

LinkedIn Twitter Facebook Email More Print

This is the fourth blog in the series where we explore writing embedded applications for the Raspberry Pi Pico using FreeRTOS.

In this blog, we will cover how to develop code with FreeRTOS that utilizes the dual-core processor onboard the Raspberry Pi Pico. We will also explain the differences between Asymmetric Multiprocessing (AMP) and Symmetric Multiprocessing (SMP). Furthermore, we will walk through the various configuration options available with SMP, and address how the use of SMP can lead to non-deterministic behavior on

- Blog series published on Embedded Computing Design
- Part 4 covers SMP on the Raspberry Pi Pico specifically
- Parts 1-3 show how to setup your own environment and use other FreeRTOS features:
  - Queues
  - Message buffers
  - Semaphores
  - Event-driven design

Resources 2 of 2

- **Symmetric Multiprocessing (SMP) with FreeRTOS**
  https://freertos.org/symmetric-multiprocessing-introduction.html

- **FreeRTOS Scheduling**
  https://freertos.org/single-core-amp-smp-rtos-scheduling.html

- **SMP Demos for the Raspberry Pi Pico Board**
  https://freertos.org/smp-demos-for-the-raspberry-pi-pico-board.html

- **FreeRTOS SMP Demos on GitHub**
  https://github.com/FreeRTOS/FreeRTOS-SMP-Demos

- **Rapberry Pi Pico SDK for C/C++**
  https://github.com/raspberrypi/pico-sdk

Thank you!

Daniel Gross
 @dangross
 djgross@amazon.com