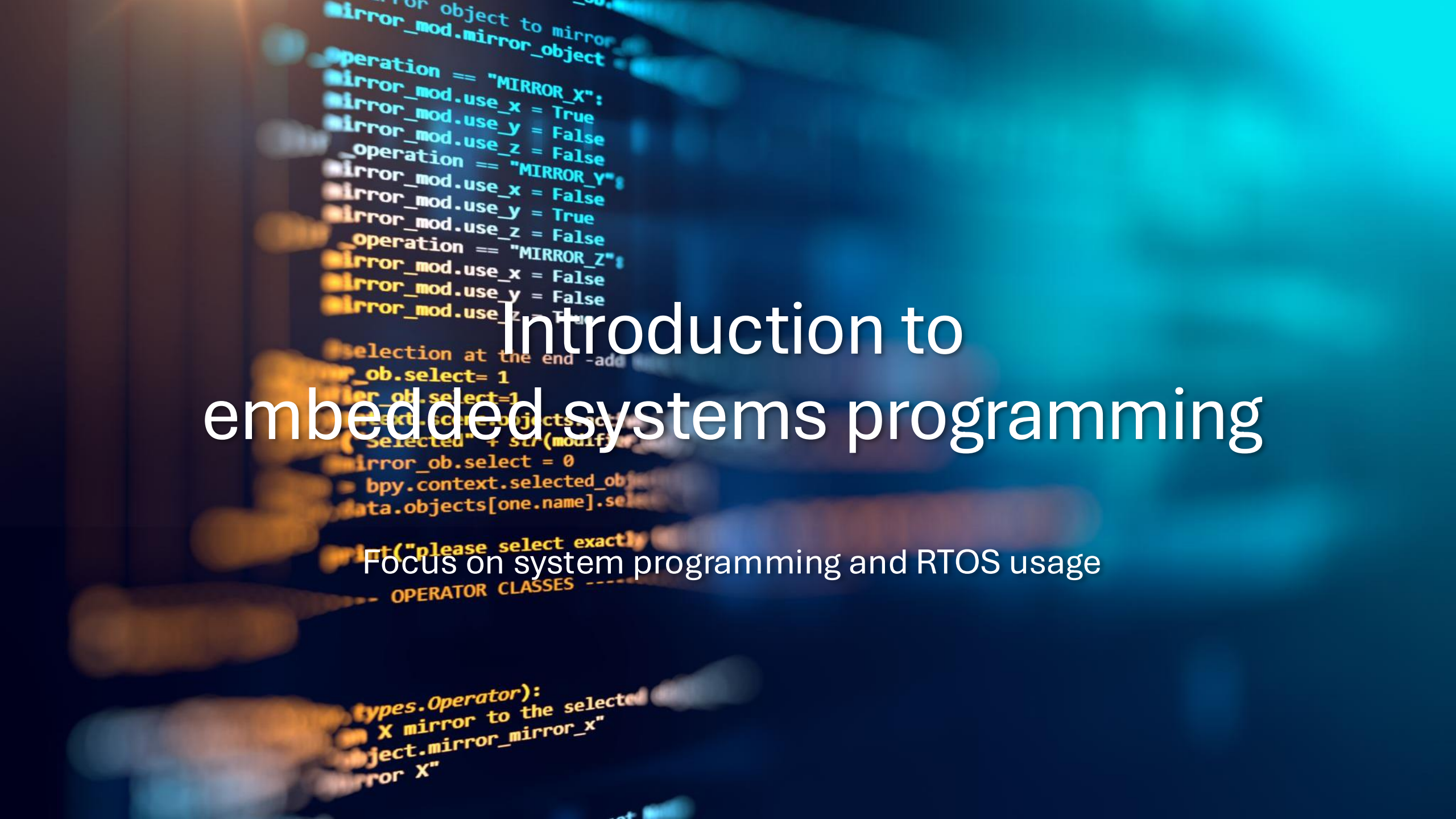
Applied Programming I

Programming computers in a nutshell.



Introduction to embedded systems programming

Focus on system programming and RTOS usage

Today's objectives

- Using an RTOS environment
- Working with files
- Command line interface
  - Starting a webserver

  on the microcontroller ESP32

RTOS

Real-time operating system

Operating system’s tasks

- An operating system (OS) controls the shared use of hardware between **multiple applications.**
- It provides resource management of CPU allocation, memory, storage.
- And offers application support by
  - Hardware abstraction,
  - Service provision.
- The kernel is the core component of an OS. It handles the creation, scheduling and termination of processes (applications).

General purpose operating systems

General purpose operating systems (e.g., Windows, Linux, macOS) provide:

- File system management
- Security and access control
- User interface, e.g., command line interface (CLI)
- Networking

Real-time operating system (RTOS)

- An RTOS provides scheduling of processes ensuring **deterministic timing**. It guarantees low latency (quick response).
- Operating systems for microcontroller can use only little resources as the microcontroller has limited resources.
- RTOS are reduced to the essentials and use only a small footprint.
- Essentials of an RTOS are **scheduler, queues and semaphores**.
- Scheduling of processes is **priority-based**.
- Configuration of an RTOS is done at compile time.

Advantages of using RTOS on microcontroller

An RTOS provides a framework for

- Scheduling
- Handling of timer and interrupts
- Usually different modules for specific functions.

Maintenance and portability of application is simple if the RTOS is maintained by its provider.

FreeRTOS

Example of an RTOS

FreeRTOS

- FreeRTOS is an example of an RTOS. It is very popular and supports many microcontroller architectures.
- It has a small kernel and is offered under the MIT open source license.
- There are two further variants of FreeRTOS, such as OpenRTOS and SafeRTOS. These are with commercial licenses and further features.

Core features of FreeRTOS

- FreeRTOS provides preemptive and co-operative scheduling of tasks (‘kind of programs that should share the microcontroller’) .
- Priorities of tasks can be flexible managed.
- Important data structures such as queues, semaphores and mutexes are provided.
- Heap and stack management is provided. Stack overflow checking is possible.

FreeRTOS: task

- A task is a simple C function. There can be many tasks running on the system. The tasks will share the CPU(s). The scheduler determines the order of running the tasks.
- Every task get its own stack.
- A task function has no return value and takes a void pointer as input:

```
void aTaskFunction(void *pvParameters);
```

FreeRTOS: task structure

```
void aTask(void *pvParameters) {
  /* task setup before the infinite loop */
  /* each instance of a task has it own local variables */
  int aVariable = 0;
  /* this is main / infinite loop of the task: */
  while(1) {
  /* task codehere */
  }
}
```

FreeRTOS: task generation

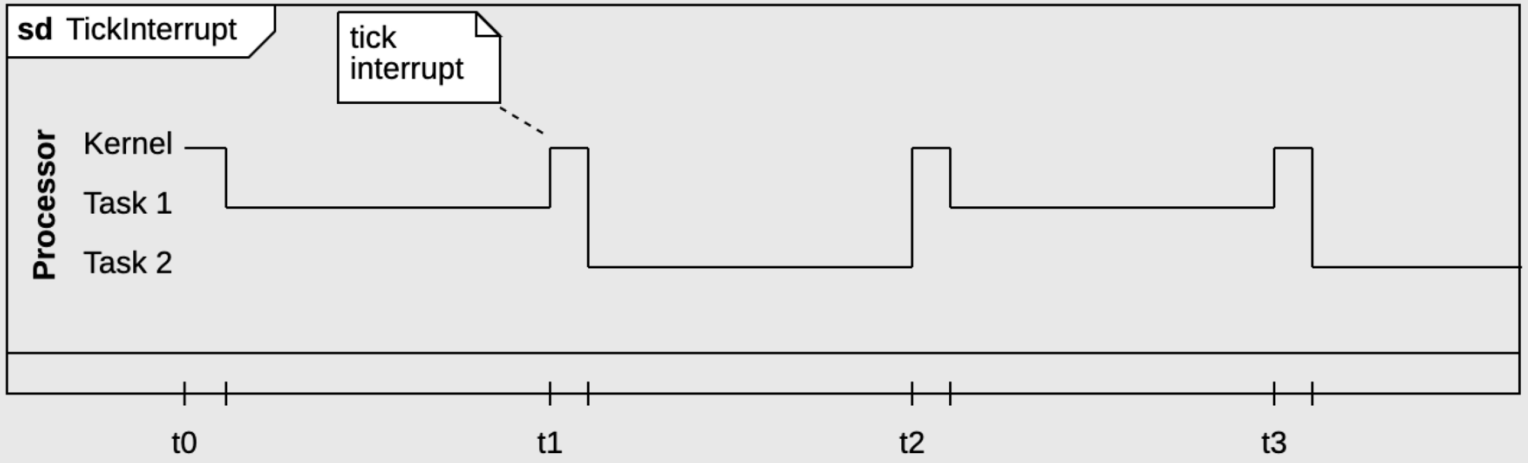
- A task has to be generated and register to be scheduled by FreeRTOS. This is done by the function xTaskCreate:

```
BaseType_t xTaskCreate(         /* return pdPASS if ok*/
  TaskFunction_t pxTaskCode,    /* C function of task */
  const char * const pcName,    /* name of task */
  const configSTACK_DEPTH_TYPE usStackDepth, /* size of stack */
  void * const pvParameters,    /* pointer to parameters for task */
  UBaseType_t uxPriority,       /* priority of task */
  TaskHandle_t * const pxCreatedTask ); /* handle of task */
```

FreeRTOS: task creation example

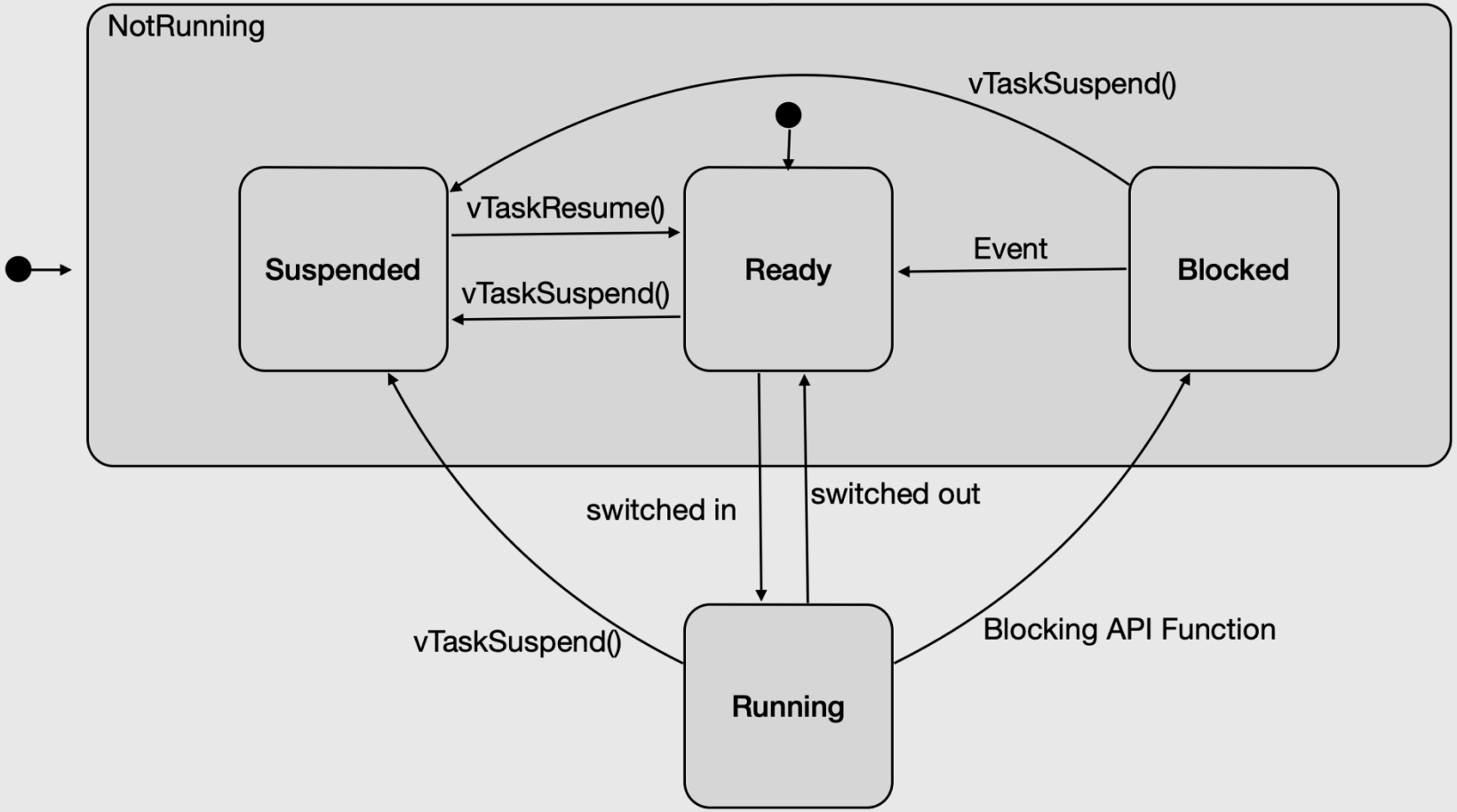
```
/* task function to be used */
void vTask1( void *pvParameters ) {
  int param = (int) pvParameters; /* cast and result: 1 */
  for( ;; ) {
    ...
  }
}
...
/* generate task for vTask1, name: "Task 1", stack size: 4096
parameter: 1, priority: 2 and no taskhandle (NULL) */
xTaskCreate(vTask1, "Task 1", 4096, (void*) 1, 2, NULL );
```

FreeRTOS: scheduling example



- Two tasks with similar priority are scheduled round-robin.
- After every time slice, the kernel takes some control by the tick interrupt.

FreeRTOS: States of a task

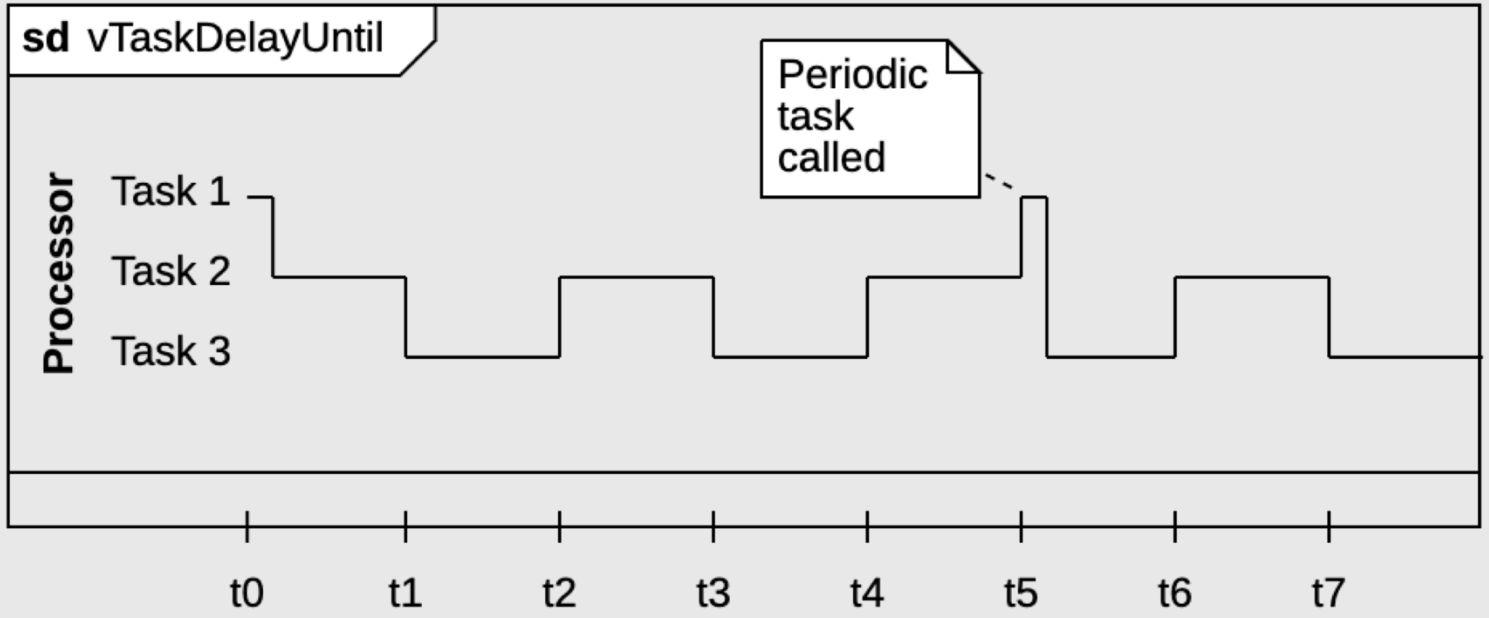


Cf. R. Barry, “Mastering the FreeRTOS Real Time Kernel”, V 1.1.0, 2023

FreeRTOS: States of a task

- Not-Running:
  - Blocked: wait for time event or synchronization event
  - Suspended: stopped of being scheduled by API call
  - Ready: can be scheduled
- Running: is executed

FreeRTOS: scheduling using vTaskDelayUntil



- Task 1 has higher priority than Task 2 and Task 3
- Task 1 blocks itself for 5 time slices by calling vTaskDelayUntil.

FreeRTOS: example scheduling

```
/* Task 1 has higher priority (2) than Task 2 and Task 3 (0)*/
xTaskCreate( vTask1, "Task 1", 4096, NULL, 2, NULL);
xTaskCreate( vTask2, "Task 2", 4096, NULL, 0, NULL);
xTaskCreate( vTask3, "Task 3", 4096, NULL, 0, NULL);
…

void vTask1( void *pvParameters ) {
 TickType_t xLastWakeTime; /* a point in time given in ticks */
 const TickType_t xDelay = pdMS_TO_TICKS(5000);
 xLastWakeTime = xTaskGetTickCount(); /* current time */
 for( ;; ) {
  /* periodic (it steps now back) for xDelay */
  vTaskDelayUntil( &xLastWakeTime, xDelay ); /* wait until (xLastW..T.. +xDelay) */
 } }
```

FreeRTOS: scheduling

- Fixed priority pre-emptive scheduling
  - Only the task itself can change priority
  - Pre-emption
  - Highest priority first
  - In case of same priority, round-robing
- Co-operative scheduling
  - Tasks need to release themselves

FreeRTOS: queues

- Queues are a means to communicate between tasks.
- Queues are used in FIFO mode (first in, first out)
- Accessing a queue has an optional waiting time
  - A task will be blocked until queue access is possible or time is over.
  - The task with highest priority accesses first in case of more tasks trying to access the queue.

FreeRTOS: queues usage / API

- Generation of a queue:

```
QueueHandle_t xQueueCreate(          /* return queue handle */
            const UBaseType_t uxQueueLength, /* number of items */
            const UBaseType_t uxItemSize );  /* size of an item in bytes */
```

- Send to / receive from a queue:

```
BaseType_t xQueueReceive( QueueHandle_t xQueue, /* queue to use */
            void * const pvBuffer,                  /* buffer for result */
            TickType_t xTicksToWait );          /* wait this time for an item */
BaseType_t xQueueSend( QueueHandle_t xQueue, /* queue to use */
            const void * const pvItemToQueue,    /* buffer containing item /*
            TickType_t xTicksToWait );         /* wait this time for space in queue */
```

FreeRTOS: queue example excerpt

```
/* Create a queue for uint32_t and store handle in blinkQueue */

blinkQueue = xQueueCreate( QUEUE_LENGTH, sizeof( uint32_t ) );

... /* in a sender task, */

void senderTask( void *pvParameters ) {

  u_int32_t data = 10;
...   /*  send the number stored at address data, the value 10 */

                xQueueSend(blinkQueue, &data, 0);
...}

...  /* in a receiver task, */

void receiverTask( void *pvParameters ) {

 u_int32_t recValue = 10;

...   /* if available, take a value from queue, wait max. 2s, store in recValue

      xQueueReceive(blinkQueue, &recValue, pdMS_TO_TICKS(2000));

...}
```

Your task: Work with queues in the example project

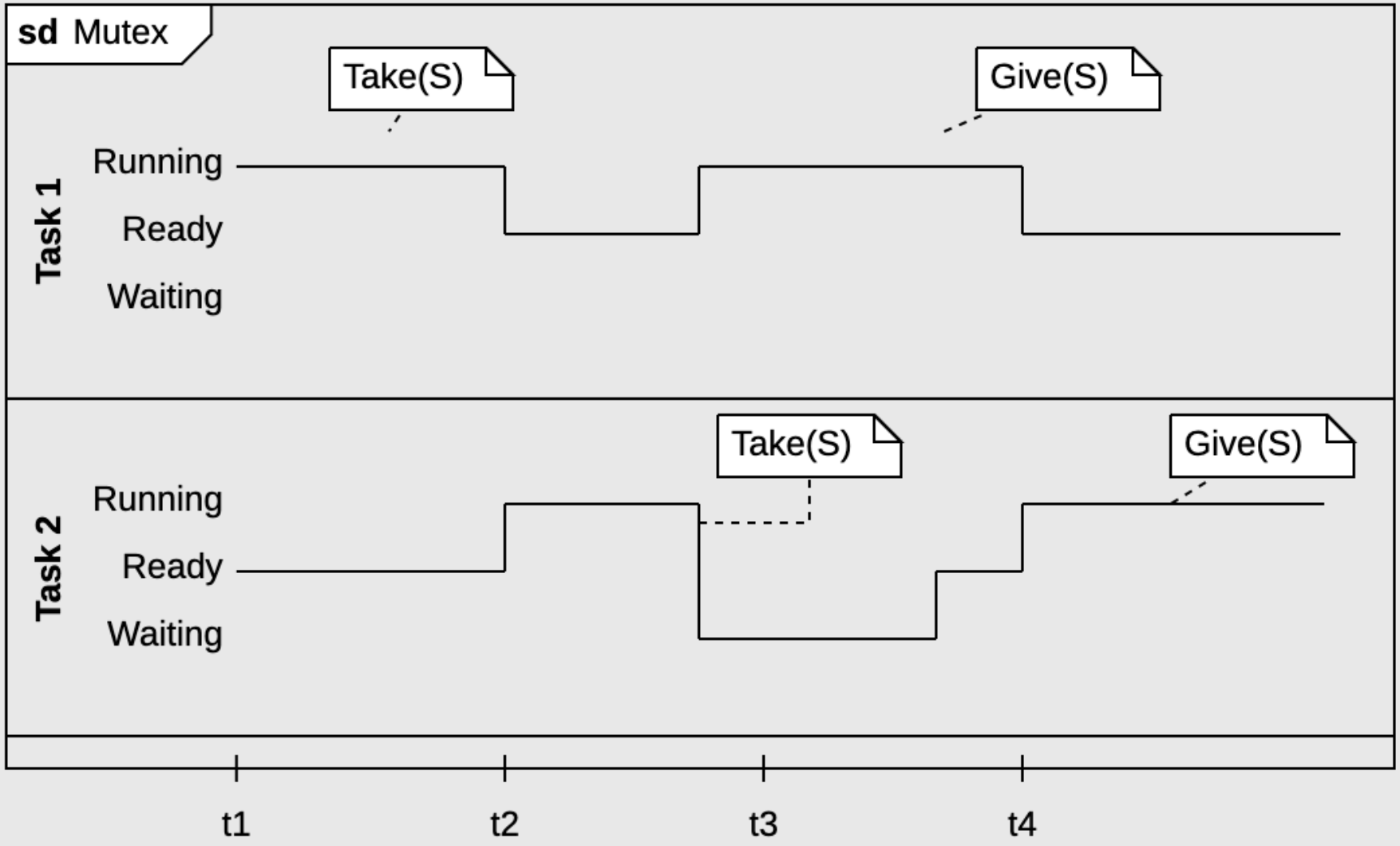
Group work, time: 20 Minutes.

1. Open folder ‘ap01/esp/freertos_queues_espidf’ and run it once.
2. Change the program so that the LED blinks once quickly for each received message. Use `vTaskDelay` in the receiver task.
3. Send two or three messages (the LED should two-/three-times blink).

Mutexes

- When at least two tasks access a shared resource, it can be a problem (called race condition). Hence, the critical sections (with the resource access) have to be exclusively used.
- A mutex (**mutual exclusion**) is a mean to allow only one task accessing the shared resource:
  - When a process uses the resource, it **takes** the mutex. When the process is done with the resource, it **gives** back the mutex.
  - Another task trying to access the resource cannot take the mutex and has to wait until it is given back. Only one task at a time can have the mutex.
- Side note: priority inversion is usually prevented by priority inheritance.

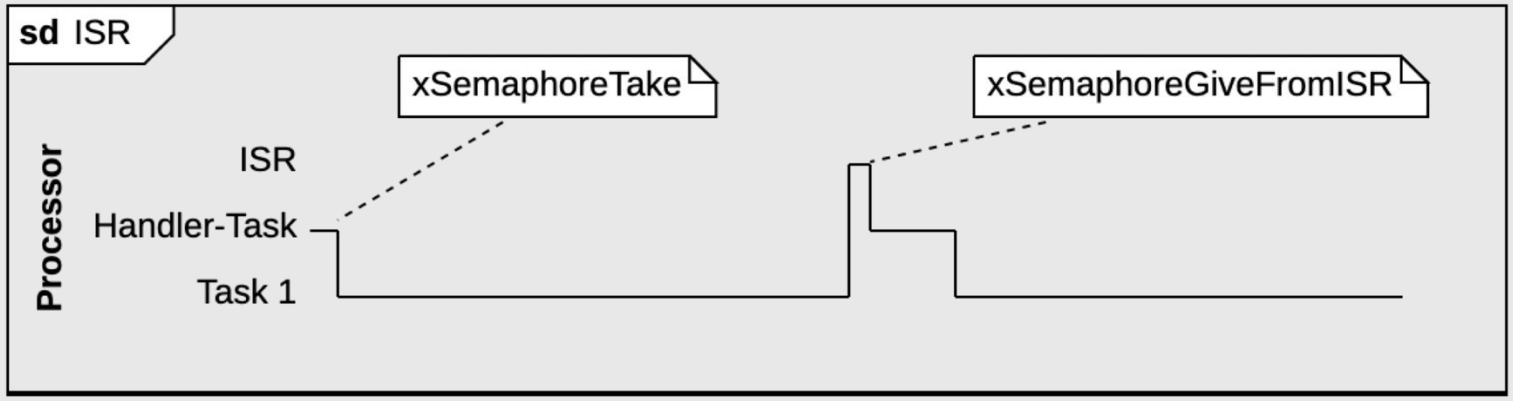
Mutex example (tasks of same priority)



Semaphores

- Semaphores are ’counting mutexes’ (or mutexes are ’binary semaphores’).
- Semaphores can be used to
  - allow access control if there a shared resource can be used by a limited number of tasks,
  - synchronize tasks.

FreeRTOS: semaphore for ISR handling



- Binary semaphores should be used to interrupt handling in FreeRTOS so that interrupt service routines can be short.
- A high priority task is activated by the semaphore to execute the code for the handler.

Your task: Understand the example project

Group work, time: 10 Minutes.

1. Open folder ‘ap01/esp/freertos_semaphore_espidf’ and run it once.
2. Find the API calls of the previous slide and discuss / explain them in your group.

Filesystems

Filesystem

- A filesystem is the component of an operating system that organizes the storage of data on storage devices.
- This includes how files and folders are organized in a (usually) hierarchical structure and how they are mapped on the device.
- Nowadays, a filesystem provides access control and data integrity as well.
- File systems are **mounted** by the system for usage and **unmounted** after usage. They can be mounted read-only.
- Examples are NTFS, FAT, ext4, APFS.

Filesystem on ESP32

- Filesystems on microcontrollers, such as the ESP32 groups, are usually mapped to be used in flash memory or on SD cards.
- The flash memory of a microcontroller is partitioned into different **partitions** (areas) so that different partition contain different data (e.g., program code binaries and filesystems).
- A **partition table** describes how the memory is configured.
- The application can **mount** a filesystem to use it.

Filesystem on ESP32

- Among other filesystems for embedded systems, ESP32 supports the FAT filesystem.
- Using a partition tool, filesystems can be created as part of the binary firmware to the microcontroller.
- The flash utility allows not only write a binary but also read it – so that data of the filesystem can be retrieved.

Filesystem on ESP32: partition table (1)

1. Step: create a partition table file with filesystem FAT for data :

```
# file partition.csv
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
storage,  data, fat,     ,       1M,
```

Filesystem on ESP32: partition table (1)

2. Step: configure project for partition table (in sdkconfig.default):

```
CONFIG_PARTITION_TABLE_CUSTOM=y
CONFIG_PARTITION_TABLE_CUSTOM_FILENAME="partitions.csv"
CONFIG_PARTITION_TABLE_FILENAME="partitions.csv"
CONFIG_ESPTOOLPY_FLASHSIZE_8MB=y
```

Filesystem on ESP32: partition table (2)

2. Step: configure project for partition table (in sdkconfig.default):

```
CONFIG_PARTITION_TABLE_CUSTOM=y
CONFIG_PARTITION_TABLE_CUSTOM_FILENAME="partitions.csv"
CONFIG_PARTITION_TABLE_FILENAME="partitions.csv"
CONFIG_ESPTOOLPY_FLASHSIZE_4MB=y
```

Filesystem on ESP32: files for partition (3)

3. Create a folder with file that should be in the FAT partition, e.g., folder ‘fatfs_image/test.txt’. And configure CMakeLists.txt:

```
...
# Create a FATFS image from the contents of the 'fatfs_image' directory
# that fits the partition named 'storage'. FLASH_IN_PROJECT indicates that
# the generated image should be flashed when the entire project is flashed to
# the target with 'idf.py -p PORT flash'.
set(image ../fatfs_image)
fatfs_create_spiflash_image(storage ${image} FLASH_IN_PROJECT)
```

Filesystem on ESP32: mount in application (4)

```
const char *base_path = "/fat"; /* base path to mount partition */

const esp_vfs_fat_mount_config_t mount_config = {
  .max_files = 4, /* no. that can be opened */
  .format_if_mount_failed = false, /* s. API documentation */
  .allocation_unit_size = CONFIG_WL_SECTOR_SIZE
};

esp_err_t err = esp_vfs_fat_spiflash_mount_rw_wl(
    base_path,     /* base, e.g., root file system, mount point */
    "storage",     /* partition to mount, see partition table */
    &mount_config, /* configuration, see above */
    &s_wl_handle); /* drive handle (for low-level use) */
```

Filesystem on ESP32: accessing files (5)

- Files on the filesystem can be accessed by the POSIX (Portable Operating System Interface) standard.
- Besides a lot of other things, POSIX allows a standard C API for file access (stdio.h):
  - Open and close a file by using its file name.
  - Reading from and writing to a file (when it is open).

Filesystem on ESP32: reading files (5)

```
const char *filename_read = "/fat/test.txt"; /* file name */
char line[128];                              /* buffer for text lines */
FILE *f;                                     /* file handle */
f = fopen(filename_read, "r");               /* open file for read */
if (f == NULL) { return; }                   /* error check*/

                                  /* read line by line until end and print */
while (fgets(line, sizeof(line), f) != NULL) {/* read character by character with fgets */
  printf("%s", line);
}
fclose(f);                                   /* close file */
```

Filesystem on ESP32: writing files (5)

```
const char *filename_write = "/fat/text.txt"; /* file name */
f = fopen(filename_write, "w");            /* handle is reused */

if (f == NULL) { return; }                 /* error check*/
                                /* write into file using fprintf */
fprintf(f, "Hello\n");
fprintf(f, "filesystem!\n");
fclose(f);

                                /* step (6): unmount FATFS */
esp_vfs_fat_spiflash_unmount_rw_wl(base_path, s_wl_handle);
```

Filesystem on ESP32: reading the partition

- The partition can be retrieved from the microcontroller and translated into files by:

```
parttool.py --port /dev/tty.usbmodem1101 read_partition --partition-type=data --partition-subtype=fat --output flash.img

$IDF_PATH/components/fatfs/fatfsparse.py flash.img
```

Your task: Make an application that saves sensor data.

Group work, time: 60 Minutes.

1. Open the folder ‘ap01/esp/fatfs_espidf’ and understand the code.
2. Open the folder ‘ap01/esp/fatfs_mpu6050_espidf’ and understand the code.
3. Write your code in the prepared setting between the /* TODO: begin */ and /* TODO: end */. The comments should help you how to do it.

Console

Console (or terminal)

- A console is a text-based interface to interact with the operating system.
- It implements a command line interface (CLI). Commands are typed in, executed and results displayed.
- Using a serial communication, a console can be easily implemented.

ESP32 and console

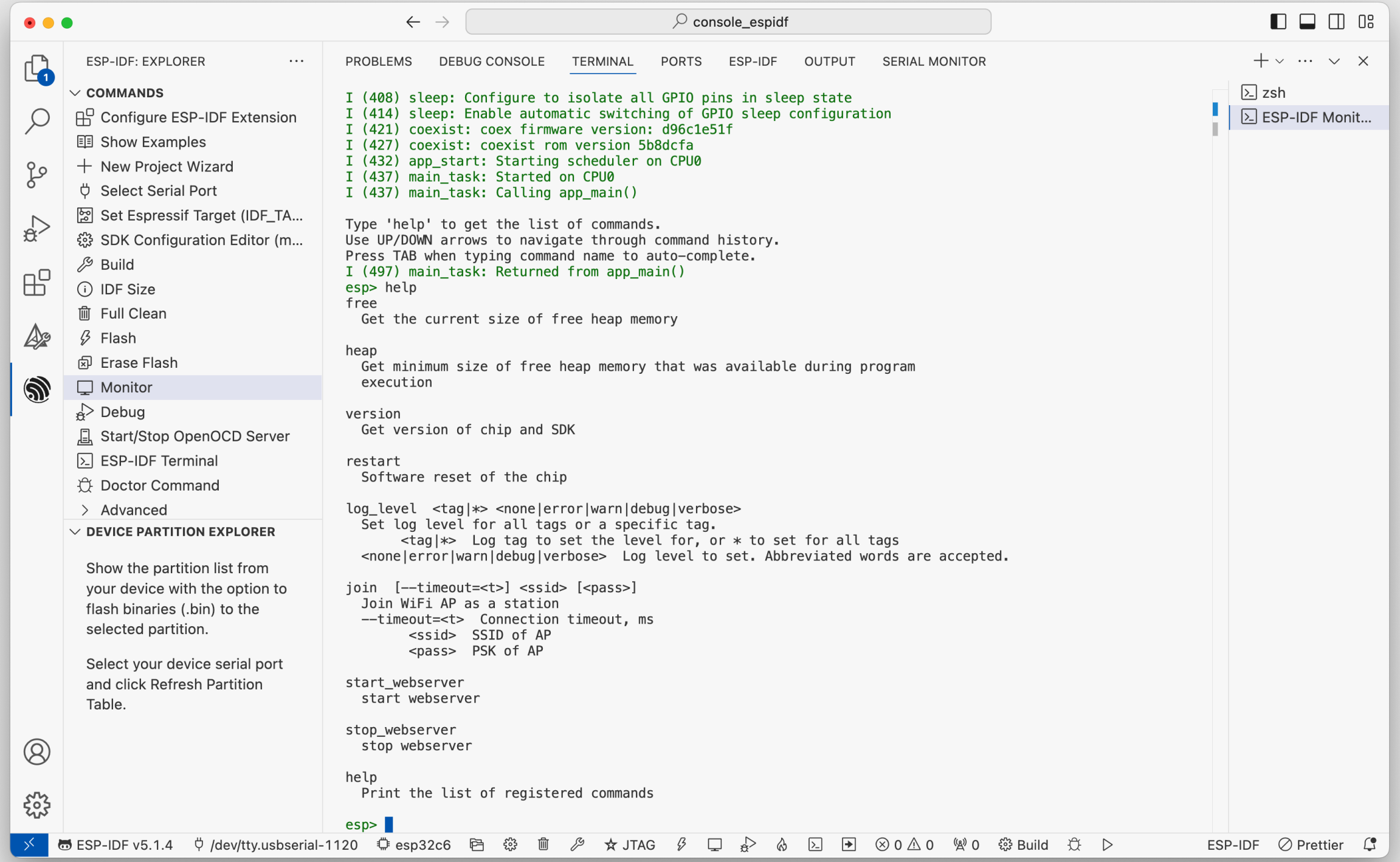
- ESP-IDF offers an API to implement a console.
- It provides basic commands that can be registered with a command ‘word’ to provide common used functionality.

ESP32: setting up a console

```
/* prepare console */
esp_console_repl_t *repl = NULL;
esp_console_repl_config_t repl_config = ESP_CONSOLE_REPL_CONFIG_DEFAULT();

/* register commands */
esp_console_register_help_command();
register_system_common();
register_wifi();

/* init console on uart */
esp_console_dev_uart_config_t hw_config = ESP_CONSOLE_DEV_UART_CONFIG_DEFAULT();
esp_console_new_repl_uart(&hw_config, &repl_config, &repl);

/* start console */
esp_console_start_repl(repl);
```

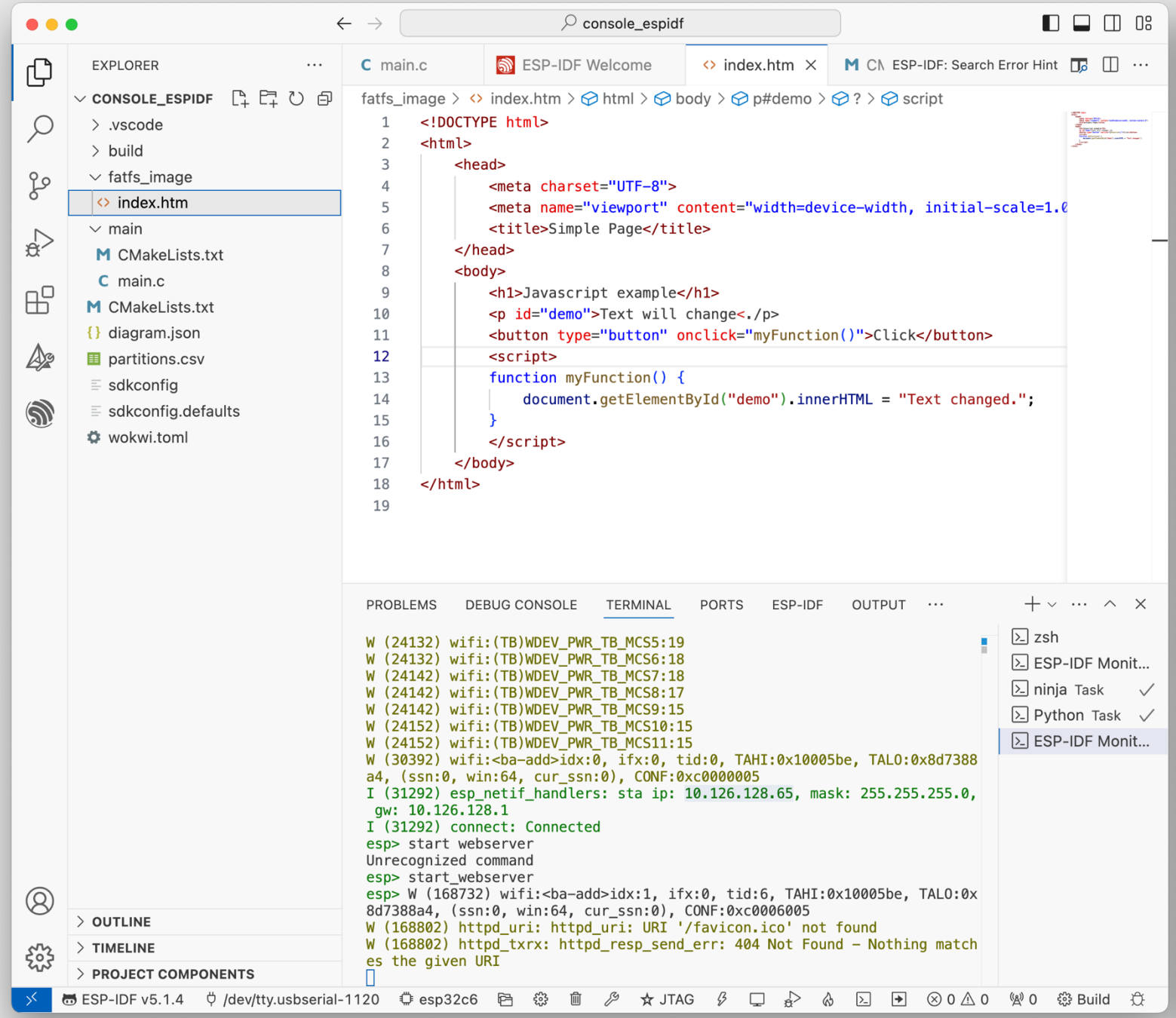
ESP32: register a console command

```
int start_webserver_func(int argc, char** argv);

void register_start_webserver() {
  /* structure defining the command */
  const esp_console_cmd_t start_webserver_cmd = {
    .command = "start_webserver", /* command name */
    .help = "start webserver", /* info text */
    .hint = NULL, /* no hint text */
    .func = &start_webserver_func, /* callback function */
  };
  /* register command */
  esp_console_cmd_register(&start_webserver_cmd);
}
```


console_espidf
ESP-IDF: EXPLORER
COMMANDS
Configure ESP-IDF Extension
Show Examples
New Project Wizard
Select Serial Port
Set Espressif Target (IDF_TA...
SDK Configuration Editor (m...
Build
IDF Size
Full Clean
Flash
Erase Flash
Monitor
Debug
Start/Stop OpenOCD Server
ESP-IDF Terminal
Doctor Command
Advanced
DEVICE PARTITION EXPLORER
Show the partition list from your device with the option to flash binaries (.bin) to the selected partition.
Select your device serial port and click Refresh Partition Table.
PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
ESP-IDF
OUTPUT
SERIAL MONITOR
I (408) sleep: Configure to isolate all GPIO pins in sleep state
I (414) sleep: Enable automatic switching of GPIO sleep configuration
I (421) coexist: coex firmware version: d96c1e51f
I (427) coexist: coexist rom version 5b8dcfa
I (432) app_start: Starting scheduler on CPU0
I (437) main_task: Started on CPU0
I (437) main_task: Calling app_main()
Type 'help' to get the list of commands.
Use UP/DOWN arrows to navigate through command history.
Press TAB when typing command name to auto-complete.
I (497) main_task: Returned from app_main()
esp> help
free
Get the current size of free heap memory
heap
Get minimum size of free heap memory that was available during program execution
version
Get version of chip and SDK
restart
Software reset of the chip
log_level <tag|*> <none|error|warn|debug|verbose>
Set log level for all tags or a specific tag.
<tag|*> Log tag to set the level for, or * to set for all tags
<none|error|warn|debug|verbose> Log level to set. Abbreviated words are accepted.
join [--timeout=<t>] <ssid> [<pass>]
Join WiFi AP as a station
--timeout=<t> Connection timeout, ms
<ssid> SSID of AP
<pass> PSK of AP
start_webserver
start webserver
stop_webserver
stop webserver
help
Print the list of registered commands
esp>
zsh
ESP-IDF Monit...
ESP-IDF v5.1.4
/dev/tty.usbserial-1120
esp32c6
JTAG
Build
ESP-IDF
Prettier

ESP32: essentials of running a webserver

```
esp_err_t index_get_handler(httpd_req_t *req) {
  ... httpd_resp_send(req, html_page, HTTPD_RESP_USE_STRLEN); ...
}
...
httpd_uri_t uri_index = {
  .uri = "/", /* uniform resource identifier */
  .method = HTTP_GET, /* http method: for request GET */
  .handler = index_get_handler, /* callback function */
  .user_ctx = NULL /* no user context here */ };
...
httpd_config_t config = HTTPD_DEFAULT_CONFIG();
if (httpd_start(&web_server_handle, &config) == ESP_OK) {
   httpd_register_uri_handler(web_server_handle, &uri_index);
}
```


console_espidf
EXPLORER
CONSOLE_ESPIDF
.vscode
build
fatfs_image
index.htm
main
CMakeLists.txt
main.c
CMakeLists.txt
diagram.json
partitions.csv
sdkconfig
sdkconfig.defaults
wokwi.toml
OUTLINE
TIMELINE
PROJECT COMPONENTS
main.c
ESP-IDF Welcome
index.htm
ESP-IDF: Search Error Hint
fatfs_image > index.htm > html > body > p#demo > ? > script
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0
6 <title>Simple Page</title>
7 </head>
8 <body>
9 <h1>Javascript example</h1>
10 <p id="demo">Text will change<./p>
11 <button type="button" onclick="myFunction()">Click</button>
12 <script>
13 function myFunction() {
14 document.getElementById("demo").innerHTML = "Text changed.";
15 }
16 </script>
17 </body>
18 </html>
19
PROBLEMS
DEBUG CONSOLE
TERMINAL
PORTS
ESP-IDF
OUTPUT
W (24132) wifi:(TB)WDEV_PWR_TB_MCS5:19
W (24132) wifi:(TB)WDEV_PWR_TB_MCS6:18
W (24142) wifi:(TB)WDEV_PWR_TB_MCS7:18
W (24142) wifi:(TB)WDEV_PWR_TB_MCS8:17
W (24142) wifi:(TB)WDEV_PWR_TB_MCS9:15
W (24152) wifi:(TB)WDEV_PWR_TB_MCS10:15
W (24152) wifi:(TB)WDEV_PWR_TB_MCS11:15
W (30392) wifi:<ba-add>idx:0, ifx:0, tid:0, TAHI:0x10005be, TALO:0x8d7388
a4, (ssn:0, win:64, cur_ssn:0), CONF:0xc0000005
I (31292) esp_netif_handlers: sta ip: 10.126.128.65, mask: 255.255.255.0,
gw: 10.126.128.1
I (31292) connect: Connected
esp> start webserver
Unrecognized command
esp> start_webserver
esp> W (168732) wifi:<ba-add>idx:1, ifx:0, tid:6, TAHI:0x10005be, TALO:0x
8d7388a4, (ssn:0, win:64, cur_ssn:0), CONF:0xc0006005
W (168802) httpd_uri: httpd_uri: URI '/favicon.ico' not found
W (168802) httpd_txrx: httpd_resp_send_err: 404 Not Found - Nothing match
es the given URI
zsh
ESP-IDF Monit...
ninja Task
Python Task
ESP-IDF Monit...
ESP-IDF v5.1.4
/dev/tty.usbserial-1120
esp32c6
JTAG
Build
10.126.128.65
Javascript example
Text will change<./p>
Click

Today’s summary

- Using FreeRTOS as example of an RTOS
- Working with files on a microcontroller (ESP32)
  - Partitions
  - Mounting
  - Standardized file I/O
- Console (command line interface) (ESP32)
  - Using standard functionality of system API
  - Running a web server

As always…
Questions?