Zephyr Meetup

User Mode in Zephyr:
Explained in Simple Words

Agenda

- **Overview**
- **Memory Domains**
- **Syscalls**

Introduction to User Mode

- **Keeping applications safe and reliable**
  - Enforcing memory access permissions
  - Restricting the execution of privileged instructions.
- **Definitions:**
  - User Mode:
    - Execution context where threads run with limited privileges (restrictions)
  - Kernel Mode:
    - Unrestricted access.
- **Zephyr brings convenience and simplicity to handling user threads.**
  - This is a big deal !

Key Features of User Mode

- **Limited Access:**
  - Access restricted to essential system resources to prevent unintended system alterations.
- **Isolation:**
  - Individual isolation of user mode threads to safeguard against faults and compromises in other threads.
- **Security:**
  - Requirement for explicit permissions for higher-privilege operations, enhancing overall system security.

User Mode in Zephyr

- **Depends on either MPU (Memory Protection Unit) or MMU (Memory Management Unit) based on system architecture.**
- **Two main features:**
  - Memory domains for managing different application permissions to memory.
  - Syscalls for performing operations, like kernel objects (e.g mutex) or device drivers
- **User mode restricts access to essential resources**
  - It requires explicit permissions to interact with hardware or memory outside its allocated range

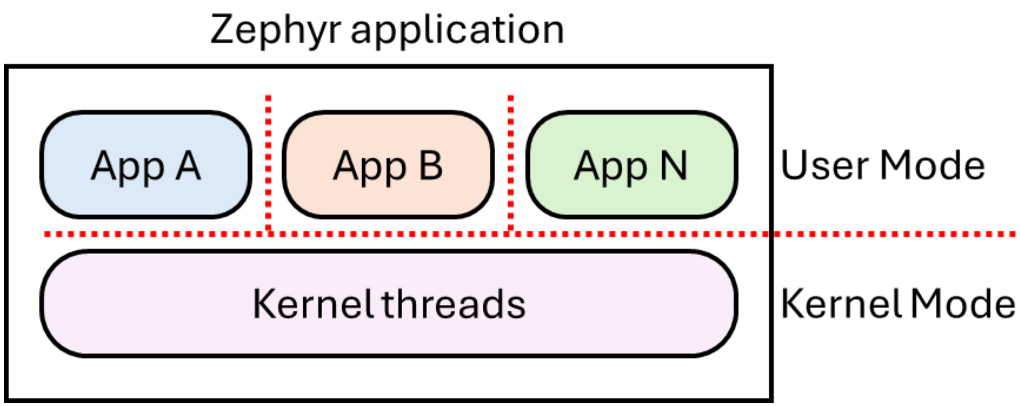
Agenda

- Overview
- Memory Domains
- Syscalls

User mode application structure

- **The term "app" refers to your project that contains all the code you're working on, part of the build system.**

- **User mode allows the creation of multiple "logical apps".**
  - Collections of user space threads grouped under the same memory domain.

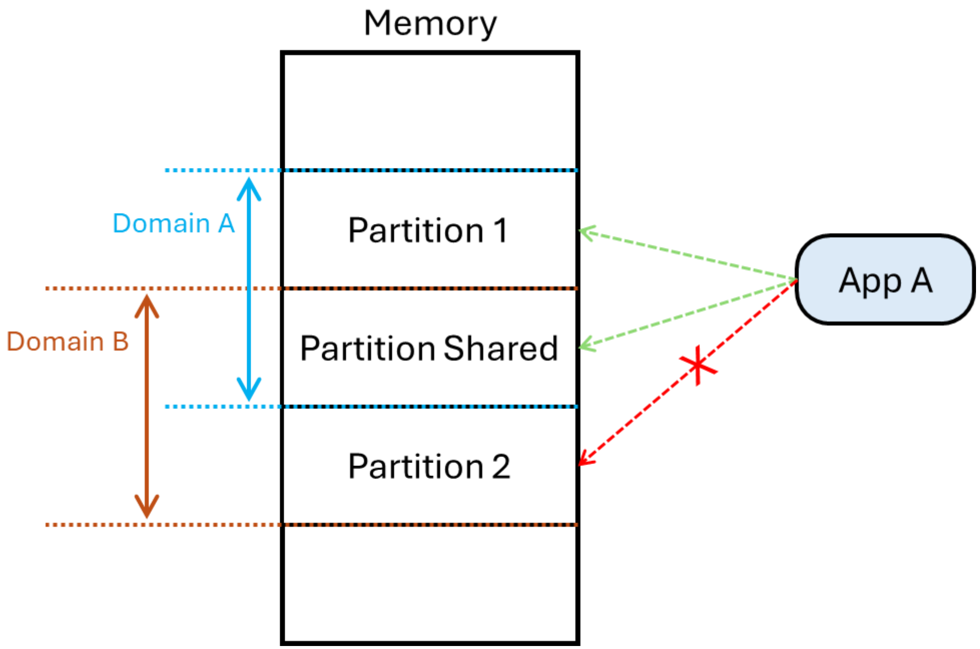


- **Threads in each logical app are isolated from those in another logical app**
  - Preventing them from accessing variables defined in different logical apps
  - Kernel threads have the ability to access all memory addresses

Memory domains and partitions

- **Memory domains in Zephyr are designed to control memory access from user threads.**

- **Each domain consists of one or more partitions.**
  - A partition is a contiguous memory region where global variables are defined.
  - The same partition can be specified in multiple memory domains (shared).



Memory Domains

- **Memory domains are not intended to control access to memory from supervisor (kernel) mode.**
- **APIs are accessible only in supervisor mode, not in user mode.**
- **Threads and Memory Domains**
  - All threads, including supervisor threads, are members of a memory domain.
    - The default domain, k_mem_domain_default

Memory Partitions in Memory Domains

- **Partitions are intended to control access to system RAM.**

- **Each partition consists of a memory address, a size, and permission**
  - They must represent regions programmable by MPU/MMU.
  - Partitions within the same memory domain must not overlap.
  - The same partition may be specified in multiple memory domains.

- **Two methods for defining memory partitions:**
  - Manual or automatic; it is usually done automatically

Automatic Memory Partitions

- **Automatic memory partitions are created by the Zephyr build system.**
  - Globals requiring specific memory partitions are tagged accordingly.
- **Characteristics of Automatic Memory Partitions:**
  - They are defined using K_APPMEM_PARTITION_DEFINE().
  - Global variables are directed to the partition using K_APP_DMEM() for initialized data and K_APP_BMEM() for BSS
- **During boot, the system zeroes any BSS variables within the memory block.**

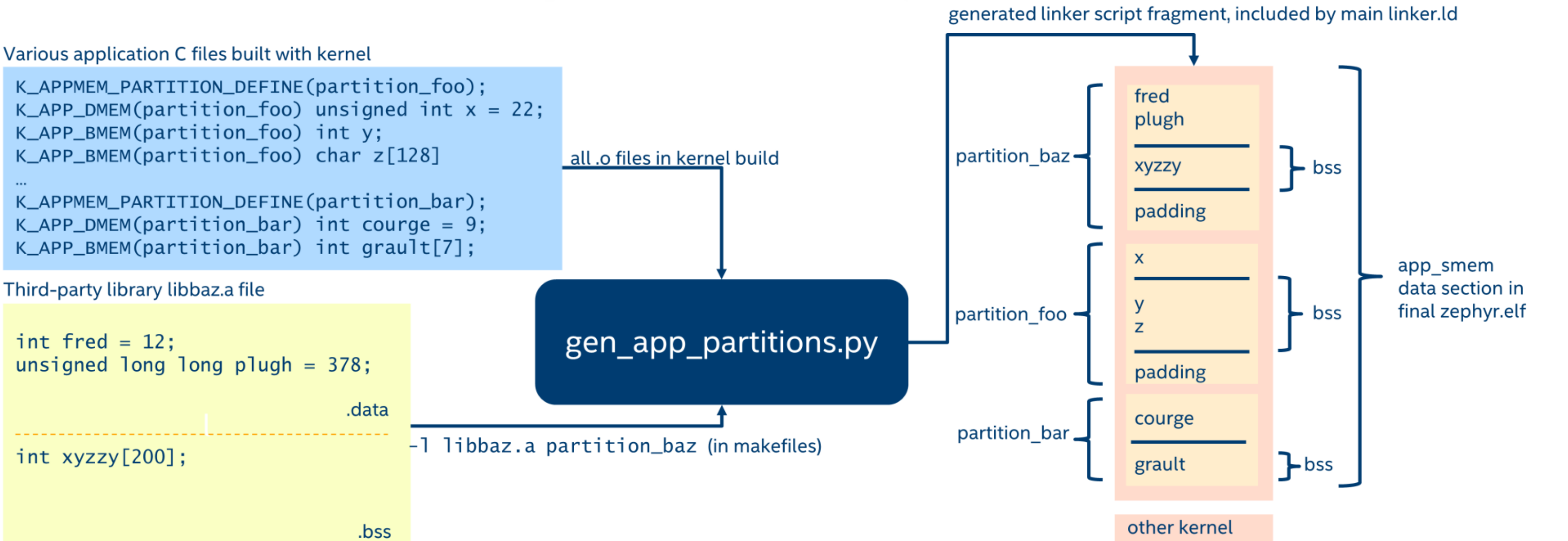
Example (1/2)

```c
/* Memory partitions definitions */
K_APPMEM_PARTITION_DEFINE(partition1);

/* Variables in specific memory partitions */
K_APP_DMEM(partition1) int var_1 = 11;

/* Thread functions for application A */
void app_a_threads(void *arg1, void *arg2, void *arg3)
{
 printk("App A, Thread %d: can access var_1 = %d and var_shared = %d\n", \
               (int) arg1, var_1, var_shared); // OK

 printk("App A, Thread %d: cannot access var_2\n", (int) arg1); // fatal
}
```

Example (2/2)

```c
/* Memory domains declarations */
struct k_mem_domain domain_a;

/* Memory partition configuration arrays */
struct k_mem_partition *app_a_partitions[] = { &partition1, other...};

int main(void)
{
 /* Initialize and assign partitions to domains */
 k_mem_domain_init(&domain_a, ARRAY_SIZE(app_a_partitions), app_a_partitions);

 /* Add app1 threads to domain a */
 k_mem_domain_add_thread(&domain_a, tid_app_a1);
}
```

Automatic Memory Domain build flow



Partition	Raw Size	Adjusted Size for Cortex-M7
partition_foo	128 + 4 + 4 = 136	256 (next power of 2)
partition_bar	4 + (4 * 7) = 32	32
partition_baz	(200 * 4) + 8 + 4 = 812	1024 (next power of 2)

Agenda

Overview

Memory Domains

- Syscalls

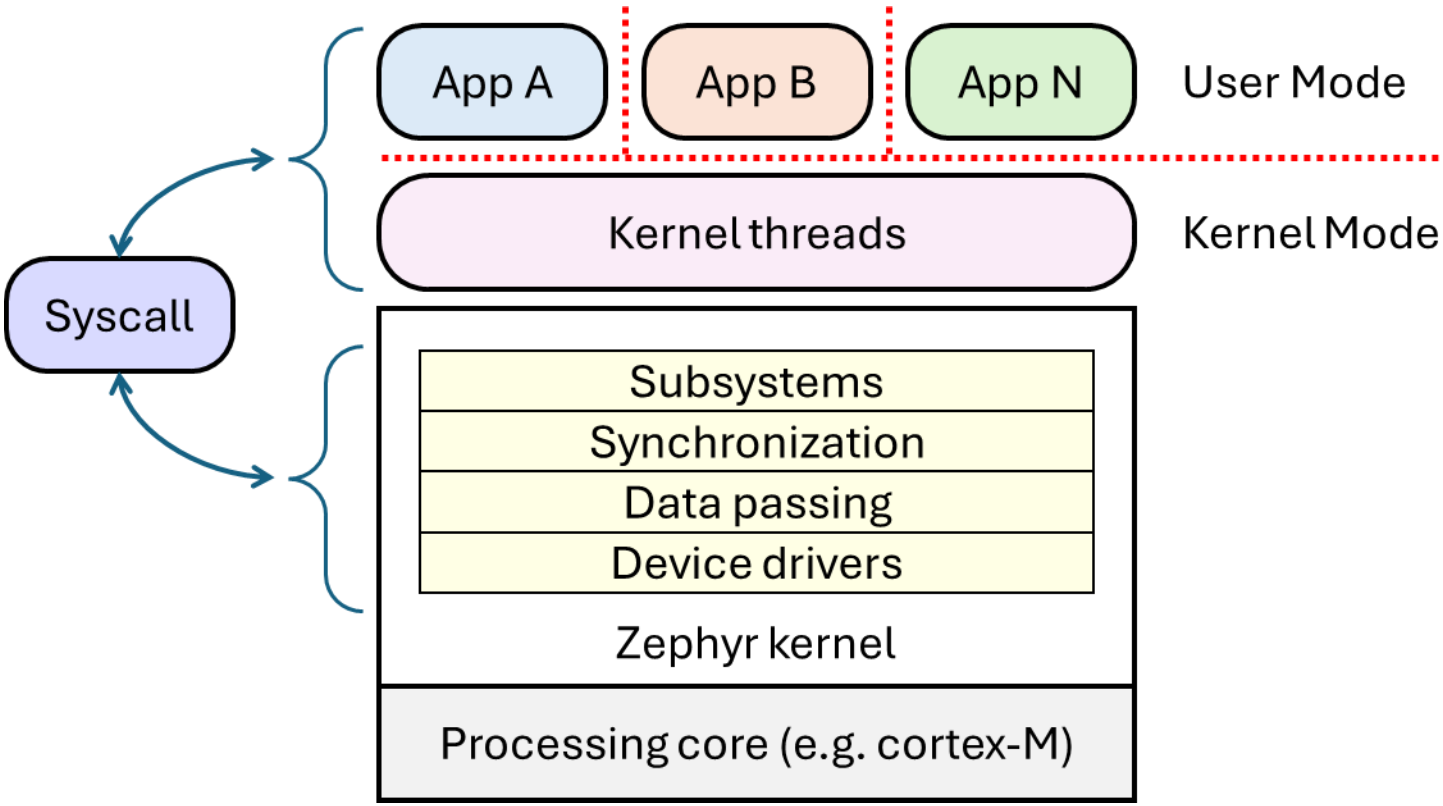
Kernel objects in a nutshell

- **Kernel objects are zephyr's core components**
  - like mutexes, semaphores, and device drivers, among others.
- **User threads must have <u>explicit</u> permissions to access these objects**
  - This is a crucial aspect of Zephyr's security model
  - Permissions are granted on a per-object basis
    - Each thread can interact with objects while being restricted from others
- **Supervisor threads have unrestricted access to any kernel object**

The concept

- **System calls are special functions to interact with the core features.**
- **Each time an application makes a system call, Zephyr checks all the information provided to ensure that it is correct and safe.**
- **Zephyr checks whether a system call originates from a user thread or a supervisor thread.**
  - User thread: Zephyr verifies whether it has the explicit permission.
    - If the permission is granted, the operation proceeds;
    - if not, the system call returns an error.
- **Note:**
  - Granting permissions to kernel objects operates independently from logical applications or memory domains.

The concept


App A
App B
App N
User Mode
Kernel threads
Kernel Mode
Syscall
Subsystems
Synchronization
Data passing
Device drivers
Zephyr kernel
Processing core (e.g. cortex-M)

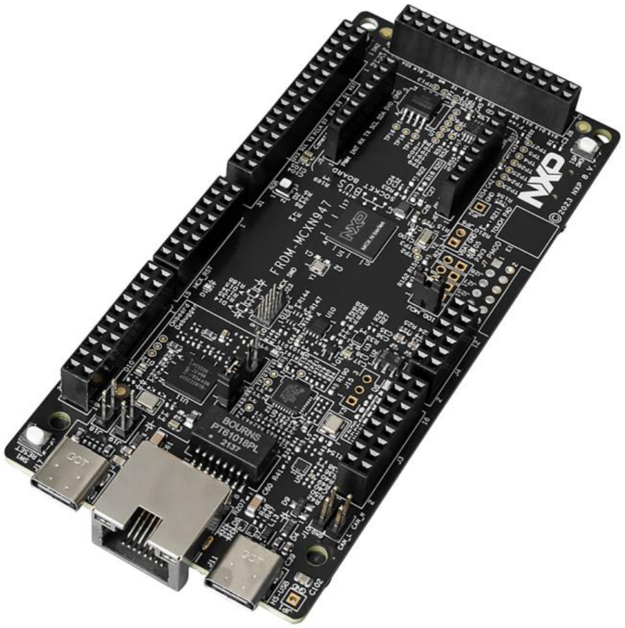
Example

```c
/* Define the semaphore (kernel object) */
K_SEM_DEFINE(my_sem, 0, 1);

/* User thread1 entry function */
void user_thread1(void *p1, void *p2, void *p3) {
  if (k_sem_take(&my_sem, K_FOREVER) == 0) {
    printk("User thread1: Successfully accessed the semaphore.\n");
  }
}

/* Fatal error handler */
void k_sys_fatal_error_handler(unsigned int reason, const z_arch_esf_t *esf) {
  if(reason == K_ERR_KERNEL_OOPS) {
    printk("Kernel OOPS in : %s\n", k_thread_name_get(k_current_get()));
  }
}

/* main is a kernel thread */
int main(void) {
  k_object_access_grant(&my_sem, user_thread1_id);
}
```

Test it on FRDM-MCXN947

- **Install MCUXpresso for Visual Studio Code**

- **Visit our github and clone projects**
  - https://github.com/Ac6Embedded/Zephyr-Examples



Summary

- Overview
- Memory Domains
- Syscalls