5TC option AUD

Embedded Programming Basics

Romain Michon, Tanguy Risset

Labo CITI, INSA de Lyon, Dpt Télécom

GRAME-CNCM INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

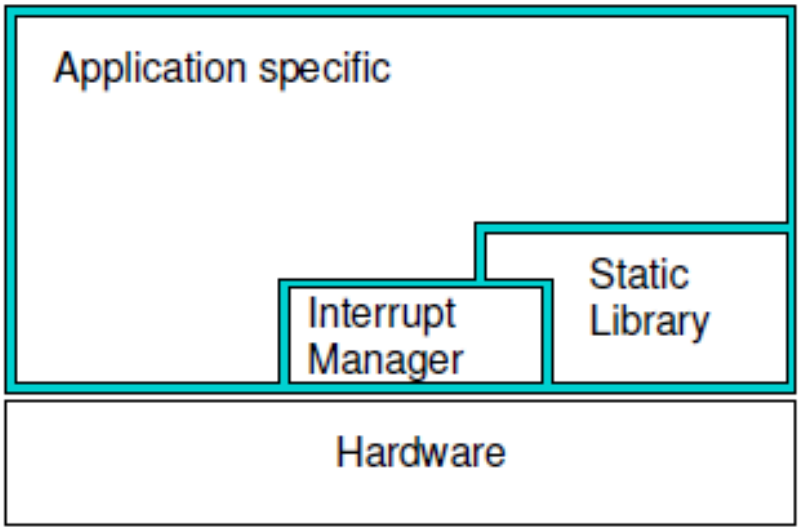
GRAME CENTRE NATIONAL DE CRÉATION MUSICALE, LYON

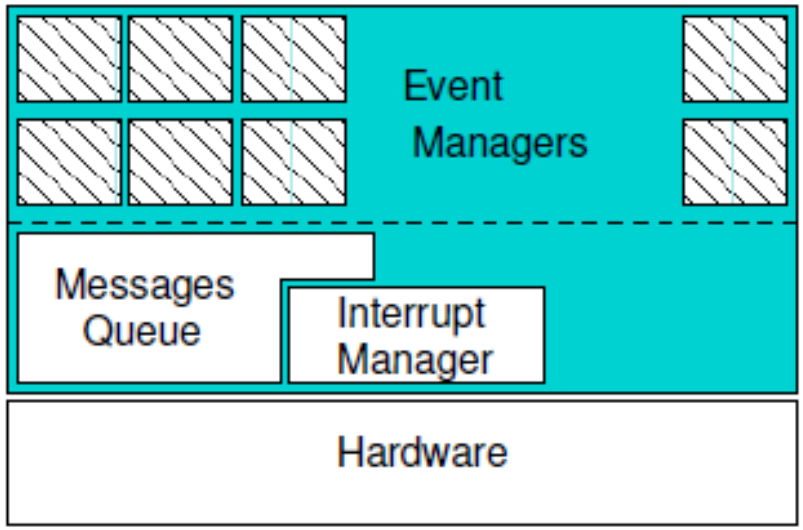
7 septembre 2020

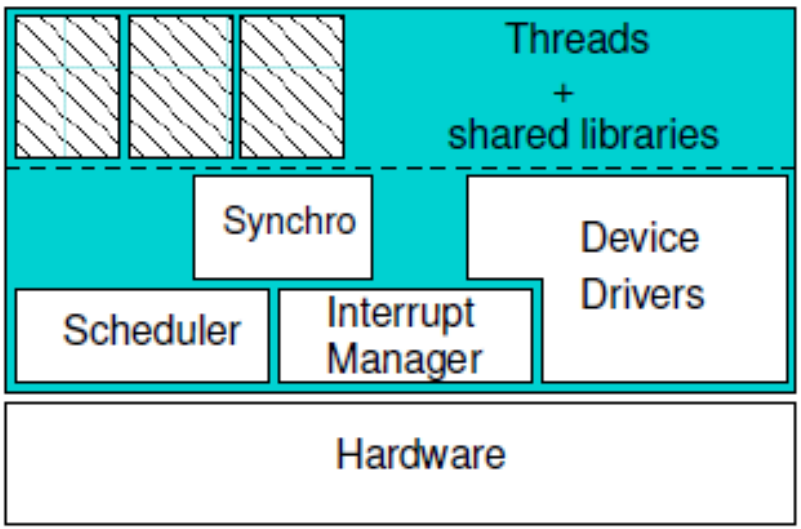
Systèmes d'exploitation légers

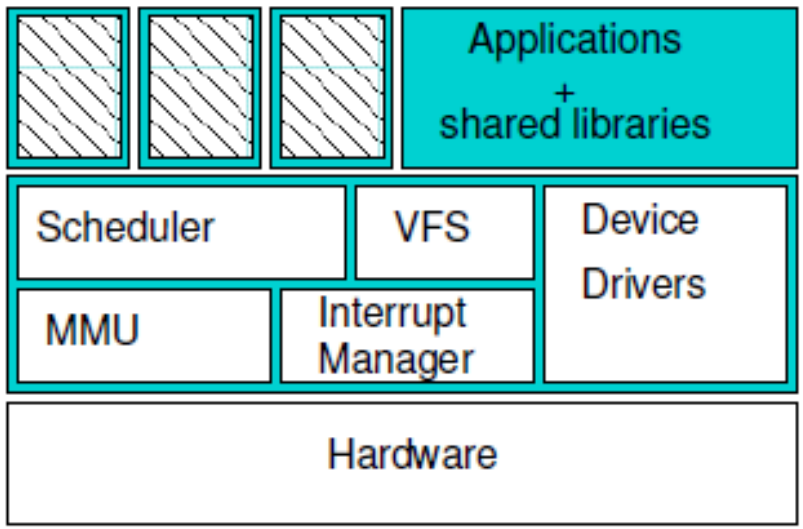
- Les systèmes d'exploitation peuvent aller d'une bibliothèque spécifique pour une application à un système générique type Unix.
- Les applications sans système d'exploitation représentent une part importante des systèmes déployés aujourd'hui.
- Il existe tout de même deux grandes catégories de système
  - modèle "Event driven"
  - modèle "Thread"

Catégories des systèmes









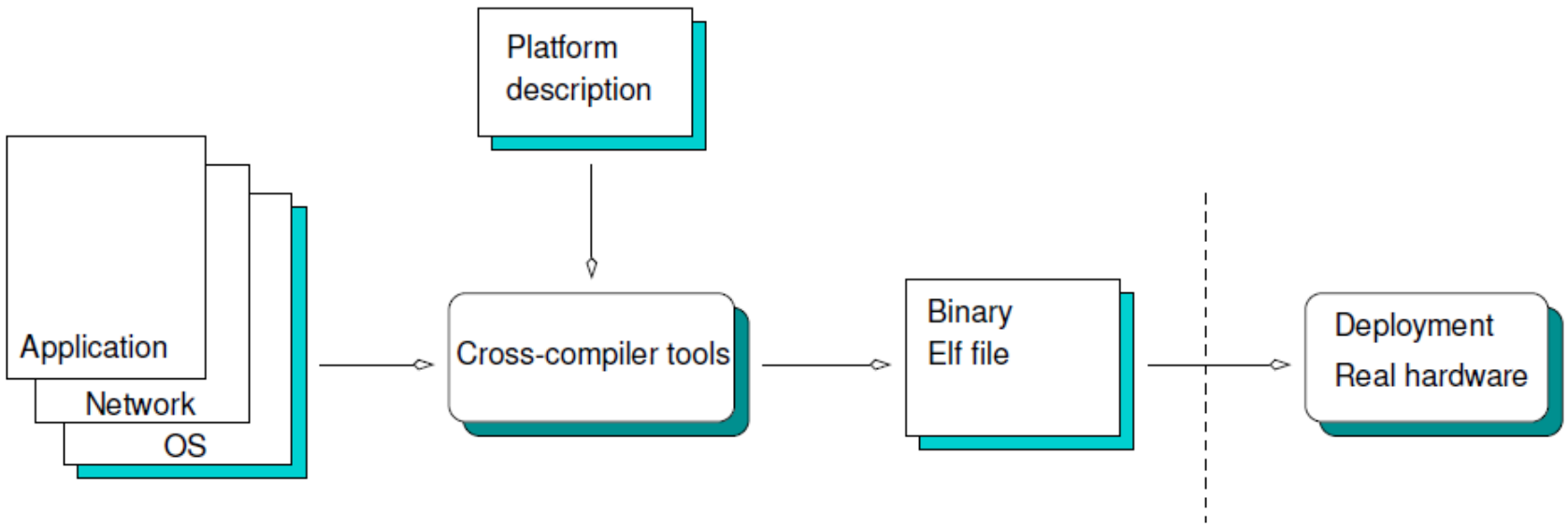
Modèles de programmation et d'exécution

- Evénements :
  - Les événements matériels démarrent des fonctions qui s'exécutent sans interruption (*run to completion*).
  - Les changements de contexte, la gestion de pile, l'ordonnancement et la gestion de priorité sont simplifiés.
  - Exemples : TinyOS 1 & 2

- File de programme / *Thread* :
  - Proche du modèle de programmation classique.
  - Mémoire partagée, piles séparées.
  - Changement de contexte.
  - Exemples : FreeRTOS, Contiki

Environnement logiciel

Les applications sont souvent simples. Les deux modèles sont fait pour être liés statiquement au programme et embarqués dans le système.



Pourquoi utiliser un OS ?

Quels services demander à un OS ?

- Gestion de Tâches/Files = ordonnanceur
- Pilotes de périphériques = interface matériel
  - Gestionnaire d'interruption
  - Gestion du temps et des timers

- Gestion des modes de veille
- Pile réseau intégrée

- Environnement de programmation et outils

Préemption : exemple FreeRTOS 4.x

- Opérations de base :
  - Gestion de tâches
  - **Ordonnancement préemptif**
  - Timers & Synchronisation (mutex)

- Utilisation de priorités
- Ordonnancement préemptif
- Primitives de synchronisation
- **Piles séparées par thread**
- Tâche “idle” de plus faible priorité

- Pas de pilote de périphérique

FreeRTOS 4.x main loop (1/2)

```
portTASK_FUNCTION(task_periodic_send, pvParameters) {
    const portTickType xDelay = 1000 / portTICK_RATE_MS;
    for(;;) {
        send_temperature();
        vTaskDelay(xDelay);
    }
}

int main( void ) {
  prvSetupHardware();
  vParTestInitialise();   // start Idle Task
  xTaskCreate(task_periodic_send, "RADIO", STACK_SIZE,
        & ParameterToPass, TASK_PRIORITY, &task_handle );
  vTaskStartScheduler(); // never returns
  return 0;
}
```

FreeRTOS 4.x main loop (2/2)

```
interrupt (TIMERA0_VECTOR) prvTickISR( void )
{
  portSAVE_CONTEXT();
  vTaskIncrementTick();
  vTaskSwitchContext();
  portRESTORE_CONTEXT();
}
```

Cette interruption est appelée périodiquement par un *timer*.

FreeRTOS integration in IDF