

SON@ENS-Lyon 2025 (CR14)

Embedded Operating Systems

Romain Michon, Tanguy Risset

Labo CITI, INSA de Lyon, Inria, GRAME-CNCM



GRAME
CENTRE NATIONAL
DE CRÉATION
MUSICALE, LYON

29 août 2025

Table of Contents

OS embarqués légers

- Catégories d'OS

- OS à Événements

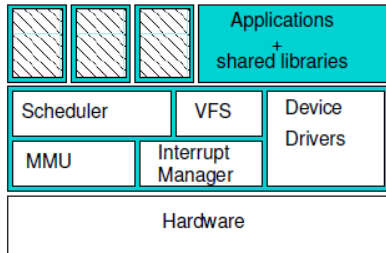
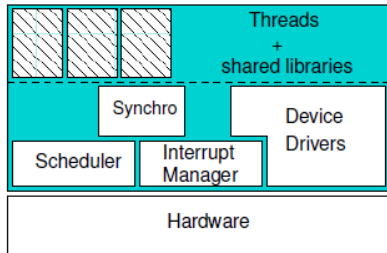
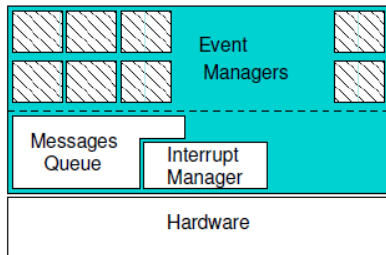
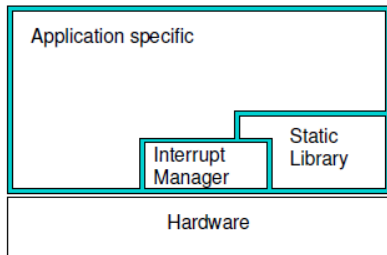
- OS à Coroutines

- Protothread Adam Dunkel

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" (OS événementiels)
 - modèle "Thread"

Catégories des systèmes

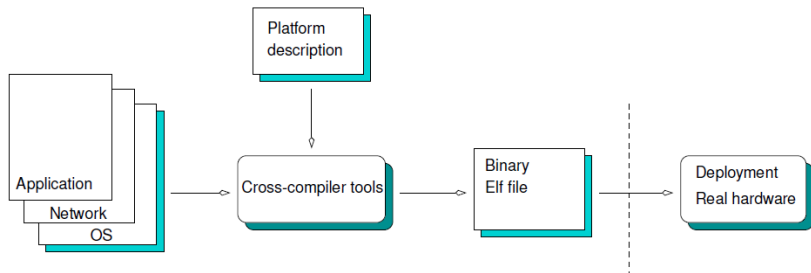


Modèles de programmation et d'exécution

- Événements :
 - 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 (ou pas).
 - 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

Aperçu des systèmes

3 exemples de systèmes utilisés dans les petits objets.

1. TinyOS : modèle à événements
2. Contiki : modèle à protothread (Anciennement Co-routine)
3. FreeRTOS : modèle à thread

Événements : exemple TinyOS

- Gestion des événements
- Fréquences fixes, mode basse conso simple
- Propose des abstractions pour
 - les communications
 - les timers
 - le stockage
- Modèle d'exécution : run to completion
- **Utilisation d'une seule pile d'exécution**
- TinyOS 2.x légère amélioration du système de mise en veille

TinyOS 1.x main loop (1/2)

```
int main(void)
{
    MainM$hardwareInit();
    TOSH_sched_init();
    MainM$StdControl$init();
    MainM$StdControl$start();
    __nesc_enable_interrupt();
    for (; ; ) {
        TOSH_run_task();
    }
}
```

TinyOS 1.x main loop (2/2)

```
bool TOSH_run_task(void)
{
    void (*func)(void );
    __nesc_atomic_t fInterruptFlags = __nesc_atomic_start();
    uint8_t old_full = TOSH_sched_full;
    func = TOSH_queue[old_full].tp;
    if (func == NULL) {
        __nesc_atomic_sleep();
        return 0;
    }
    TOSH_queue[old_full].tp = NULL;
    TOSH_sched_full = (old_full + 1) & TOSH_TASK_BITMASK;
    __nesc_atomic_end(fInterruptFlags);
    func();
    return 1;
}
```

Coroutines / Protothread : exemple Contiki

Coroutines

- Multi-tâche coopératif.
- L'application reste maître de l'ordonnancement.

Protothread

- Modèle très proche des coroutines
- Modèle mixte, orienté événements
 - “run to completion”
 - **Changement de fil sur opération bloquante**
 - **Pile d'exécution unique**
 - Les “thread” n'ont pas d'état (variables locales)

Coroutines / Protothread : exemple Contiki

Coroutines

- Multi-tâche coopératif.
- L'application reste maître de l'ordonnancement.

Protothread

- Modèle très proche des coroutines
- Modèle mixte, orienté événements
 - “run to completion”
 - **Changement de fil sur opération bloquante**
 - **Pile d'exécution unique**
 - Les “thread” n'ont pas d'état (variables locales)

Principe des Coroutines utilisées Contiki

- Chaque *thread* va *rendre la main* à interval régulier lors de son execution, pour permettre l'execution des autres threads (on appelle ça *yield*)
- En général quand le thread se met en attente sur une condition
- Comme il n'y a qu'une pile, lors de sa reprise le thread aura probablement perdu l'état de ses variables, donc ce sont des threads sans état.
- Problème : on sait comment sortir d'une fonction à n'importe quelle ligne (`return`), mais comment *reprendre* une fonction à n'importe quelle ligne ?

Machine de Duff (1/2)

En 1984 Tom Duff travaillant pour LucasFilm cherche à accélérer le code suivant pour copier une image :

```
send(int *to, int *from, int count)
{
    do
        *to = *from++;
    while(--count>0);
}
```

Machine de Duff (1bis/2)

Technique classique : dérouler la boucle

```
send(int *to, int *from, int count)
{
    register n=(count+7)/8;
    do{
        *to++ = *from++;
        *to++ = *from++;
        *to++ = *from++;
        *to++ = *from++;
        *to++ = *from++;
        *to++ = *from++;
        *to++ = *from++;
        *to++ = *from++;
    }while(--n>0);
}
```

Pb : la taille du tableau doit être un multiple de 8...

Machine de Duff (2/2) : la solution

```
send(int *to, int *from, int count)
{
    register n=(count+7)/8;
    switch(count%8){
        case 0: do{ *to++ = *from++;
        case 7:  *to++ = *from++;
        case 6:  *to++ = *from++;
        case 5:  *to++ = *from++;
        case 4:  *to++ = *from++;
        case 3:  *to++ = *from++;
        case 2:  *to++ = *from++;
        case 1:  *to++ = *from++;
                }while(--n>0);
    }
}
```

<http://www.lysator.liu.se/c/duffs-device.html>

Contiki 2.x pthread example

```
#define PT_WAIT_UNTIL(pt, condition) \
do { \
    LC_SET((pt)->lc); \
    if(!(condition)) { \
        return PT_WAITING; \
    } \
} while(0)

static PT_THREAD(thread_periodic_send(struct pt *pt)) {
    PT_BEGIN(pt);
    while(1) {
        TIMER_RADIO_SEND = 0;
        PT_WAIT_UNTIL(pt, node_id != NODE_ID_UNDEFINED &&
            timer_reached( TIMER_RADIO_SEND, 1000));
        send_temperature();
    }
    PT_END(pt);
}
```

Protothread Adam Dunkel : lc-switch.h

```
#define LC_INIT(s) s = 0;

#define LC_RESUME(s) switch(s) { case 0:

#define LC_SET(s) s = __LINE__; case __LINE__:

#define LC_END(s) }
```

Protothread Adam Dunkel : pt.h

```
#include "lc.h"
```

```
typedef unsigned short lc_t;
```

```
struct pt {
```

```
    lc_t lc;
```

```
};
```

```
#define PT_INIT(pt)    LC_INIT((pt)->lc)
```

```
#define PT_THREAD(name_args) char name_args
```

```
#define PT_BEGIN(pt) { char PT_YIELD_FLAG = 1;
```

```
LC_RESUME((pt)->lc)
```

```
#define PT_WAIT_UNTIL(pt, condition)
```

```
do {
```

```
    LC_SET((pt)->lc);
```

```
    if(!(condition)) {
```

```
        return PT_WAITING;
```

```
    }
```

```
} while(0)
```

```
#define PT_END(pt) LC_END((pt)->lc); PT_YIELD_FLAG = 0; \
```

```
PT_INIT(pt); return PT_ENDED; }
```

Hands-on : Protothread in SON@ENS-Lyon

- An example `protothread_blink` is available on SON website.
 - Crazy Sine with two timers (one for blinking, one for frequency selection)
 - Two protothreads associated
- TODO :
 1. Understand the existing code
 2. Add a counter to *count elapsed time* (i.e. seconds) and print out elapsed time on Serial monitor
 3. Add a control to *mute sound* from your keyboard (in a new protothread).
- (Check for more instructions on the website)

End of embedded OS course

End of Embedded OS slides