Trampoline

Protection mémoire et protection temporelle

Protection mémoire

Protection mémoire

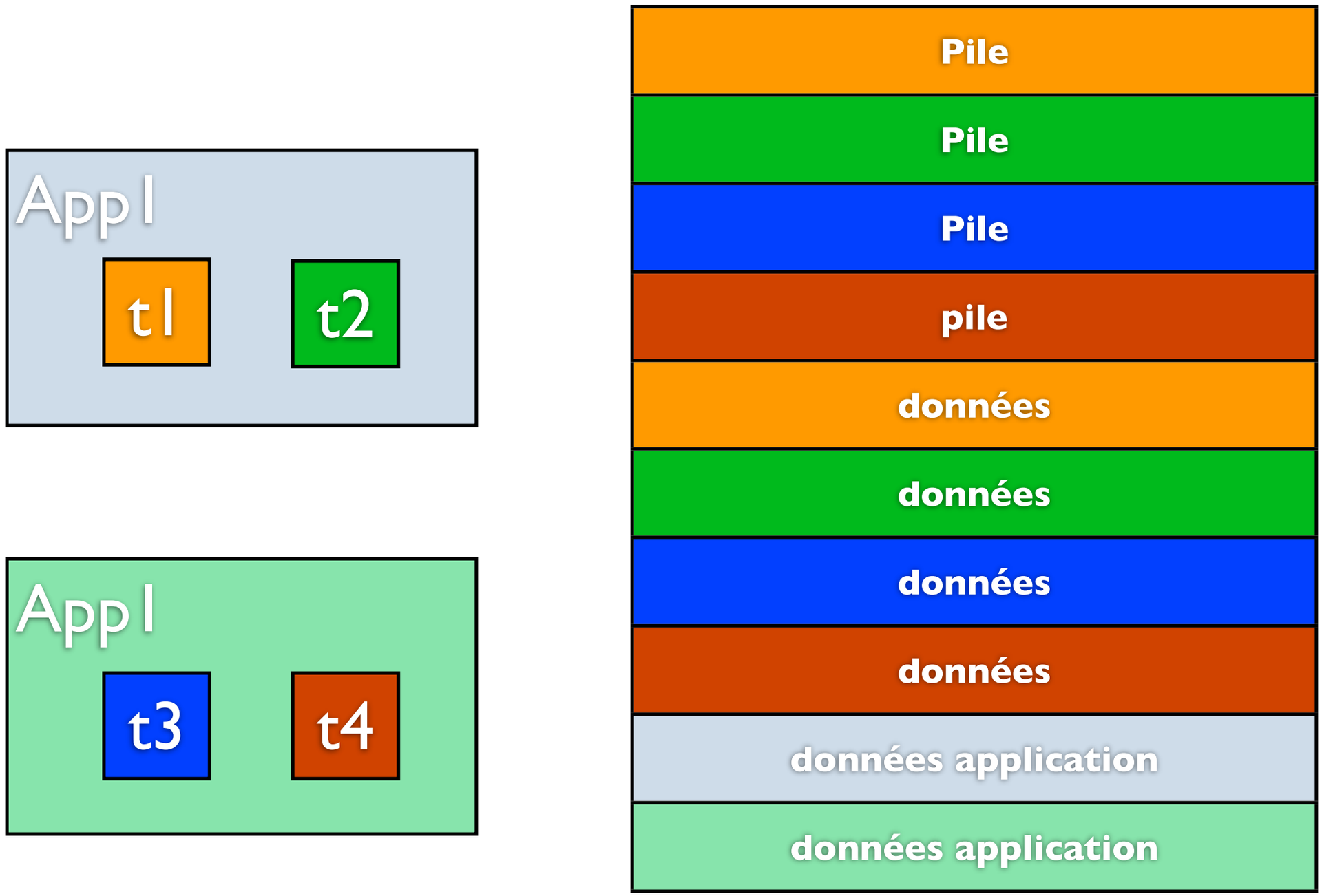
- MPU : ensemble de paires de registres délimitant des régions mémoire
- À chaque région on associe des droits.
- exemple sur ARM Cortex :
  - 8 régions
  - Chaque région est définie par une adresse de base et une taille (les deux sont en puissance de 2, min 32 octets, max 4Go !)
  - Une région démarre à une adresse divisible par sa taille
  - Droits superviseur et utilisateur : execute / read / write / aucun accès
  - Exception si accès incompatible avec les droits

PM : régions définies

- 3 régions actuellement dans Trampoline :
  - pile du processus (tâche ou ISR2)
  - données du processus
  - données de l'OS Application
- Le création de ces régions est faite dans le link script.
  - Exemple avec le PowerPC :

```
foreach proc in PROCESSES do
%  .% !proc::KIND %_% !proc::NAME %_SEC_VAR : {
    . = ALIGN(32);
    __SEG_START_% !proc::KIND %_% !proc::NAME %_VAR_RGN = .;
/* Initialized variables section of % !proc::KIND % % !proc::NAME % */
%
  for DATA_SIZE in "32BIT","16BIT","8BIT","BOOLEAN","UNSPECIFIED" do
    for SECTION_KIND in "_NOINIT_","_POWER_ON_INIT_","_FAST_","_" do
%    *(.% !proc::KIND %_% !proc::NAME %_SEC_VAR% !SECTION_KIND !DATA_SIZE %)
%
    end for
  end for
%    __SEG_END_% !proc::KIND %_% !proc::NAME %_VAR_RGN = ALIGN(32) - 1;
  } > ram
%
```

PM : Répartition des régions



PM : Description des régions

- Descripteur de région

```
typedef struct TPL_MEM_REGION {
  P2VAR(void, TYPEDEF, OS_APPL_DATA) start;
  P2VAR(void, TYPEDEF, OS_APPL_DATA) end;
} tpl_mem_region;
```

```
struct TPL_MEM_PROT_DESC {
    VAR(tpl_mem_region, TYPEDEF) proc_var;
    VAR(tpl_mem_region, TYPEDEF) proc_stack;
#if WITH_AUTOSAR == YES
    VAR(tpl_mem_region, TYPEDEF) osap_var;
#endif
};
```

- Les droits sont implicites

PM : génération des descripteurs

- Templates de génération des descripteurs

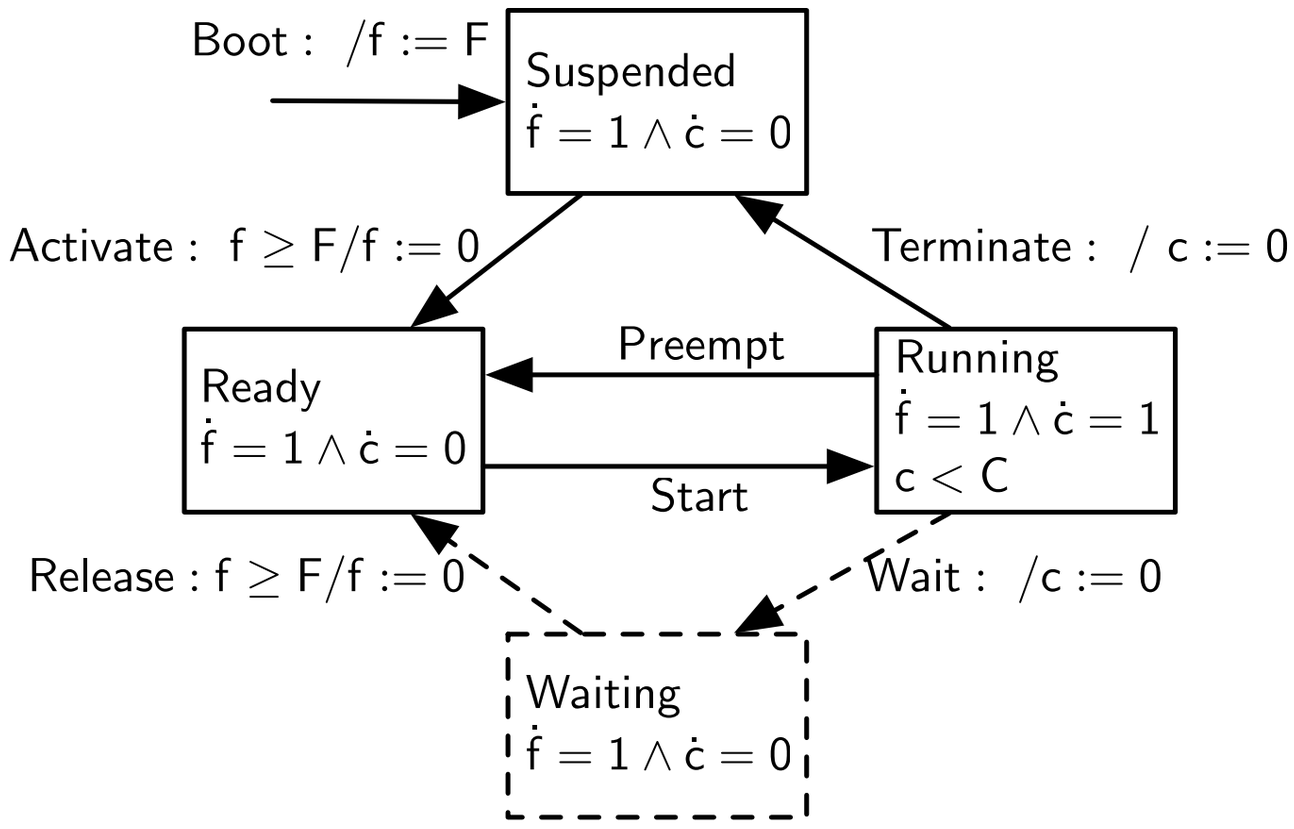
```
#define OS_START_SEC_CONST_UNSPECIFIED
#include "tpl_memmap.h"
CONST(tpl_mem_prot_desc, OS_CONST) % !proc::NAME %_mp_desc = {
  { /* proc_var memory region */
    &__SEG_START_% !proc::KIND %_% !proc::NAME %_VAR_RGN,
    &__SEG_END_% !proc::KIND %_% !proc::NAME %_VAR_RGN
  },
  { /* proc_stack memory region */
    &__SEG_START_% !proc::KIND %_% !proc::NAME %_STACK_RGN,
    &__SEG_END_% !proc::KIND %_% !proc::NAME %_STACK_RGN
  }%
if AUTOSAR then
%,
  { /* osap_var memory region */
    &__SEG_START_OS_APP_% !proc::APPLICATION %_VAR_RGN,
    &__SEG_END_OS_APP_% !proc::APPLICATION %_VAR_RGN
  }
%
end if
%};
#define OS_STOP_SEC_CONST_UNSPECIFIED
#include "tpl_memmap.h"
```

PM : fonctions noyau

- Initialiser le MPU : `tpl_init_mp`
- Activer la protection mémoire : `tpl_user_mp`
- Désactiver la protection mémoire : `tpl_kernel_mp`
- Programmer le MPU en fonction du processus qui est ordonnancé : `tpl_set_process_mp`

Protection Temporelle : les processus (1)

- Chaque processus est caractérisée par :
  - un budget d'exécution (c)
  - un *timeframe*, intervalle de temps entre deux activations (f)



Protection Temporelle : les processus (2)

- Ces attributs sont renseignables dans le fichier OIL

```
TASK t3 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 3;
 ACTIVATION = 1;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET = 10000;
   TIMEFRAME = 1;
   MAXOSINTERRUPTLOCKTIME = 1;
   MAXALLINTERRUPTLOCKTIME = 1;
};
```

Protection Temporelle : les interruptions

- Deux attributs permettent de borner la durée de désactivation des interruptions.

```
TASK t3 {
 AUTOSTART = TRUE { APPMODE = std; };
 PRIORITY = 3;
 ACTIVATION = 1;
 SCHEDULE = FULL;
 TIMING_PROTECTION = TRUE {
   EXECUTIONBUDGET = 10000;
   TIMEFRAME = 1;
   MAXOSINTERRUPTLOCKTIME = 1;
   MAXALLINTERRUPTLOCKTIME = 1;
};
```

Protection temporelle : les ressources

- Le but est de borner la durée de détention d'une ressource
  - non implémenté pour l'instant

Les fonctions noyaux

- Les fonctions utilisables pour ajouter ses propres services sont :
  - `tpl_activate_task` : active une tâche (la tâche passe de l'état `suspended` à `ready`)
  - `tpl_terminate` : termine la tâche appelante (la tâche passe de l'état `running` à `suspended`)
  - `tpl_block` : bloque la tâche appelante (la tâche passe de l'état `running` à `waiting`)
  - `tpl_release` : débloque une tâche (la tâche passe de l'état `waiting` à `ready`)
  - `tpl_preempt` : préempte la tâche en cours d'exécution (la tâche passe de l'état `running` à `ready`)
  - `tpl_start` : démarre la tâche de plus forte priorité (la tâche passe de l'état `ready` à `running`)
  - `tpl_schedule_from_running` : effectue un réordonnancement.

Implantation du portage (1)

- fichiers C et assembleur localisés dans `machines/<jeu d'instructions>/<micro>/<carte>`
- templates de production de code localisés dans `goilv2/templates/code/<jeu d'instructions>/<micro>/<carte>`
- OIL de configuration localisés dans `goilv2/templates/config/<jeu d'instructions>/<micro>/<carte>`
- templates de compilation localisés dans `goilv2/templates/compiler/<compilateur>/<jeu d'instructions>/ <micro>/<carte>`
- templates d'édition de liens localisés dans `goilv2/templates/linker/<linker>/<jeu d'instructions>/<micro>/<carte>`

Implantation du portage (2)

- Fichiers C et assembleur :
  - Handler d'appel système
  - Handler d'interruption
  - Initialisation de contexte `tpl_init_context`
  - Fonctions d'activation et de désactivation des IT `tpl_enable_interrupts` et `tpl_disable_interrupts`
  - Fonction de la tâche idle `idle_function`
  - Définitions de la pile système
  - Définition du contexte

Implantation du portage (3)

- Templates de production de code
  - Génération des services : `tpl_invoque_s.goilTemplate`, `service_call.goilTemplate`, `instruction_specific.goilTemplate` et `invoque_specific.goilTemplate`
  - Génération des instances de la structure de stockage de contexte et de la pile : `process_specific.goilTemplate`
  - Génération de la table des routines d'interruption : `interrupt_table.goilTemplate`

Implantation du portage (4)

- OIL de configuration.
  - Les fichiers config.oil le long du chemin sont agrégés.

```
CPU thumb2_ressource {
  COUNTER SystemCounter {
    SOURCE = SysTick;
  };
};
```

Implantation du portage (5)

- Templates de compilation
  - Génération du fichier `Compiler_Cfg.h` à partir de `Compiler_Cfg_h.goilTemplate`. Attributs pour la segmentation mémoire.
  - Génération du fichier `Compiler.h` à partir de `Compiler_h.goilTemplate`. Directives de placement des objets en mémoire.
  - Génération du fichier `MemMap.h` à partir de `MemMap_h.goilTemplate`. Définition des sections mémoire

Implantation du portage (6)

- Templates d'édition de liens
  - Production d'un link script à partir du template script.goilTemplate.

```
/*
 * code and consts of the processes of the applications
 */
. = ALIGN(4);
  apptext : {
    __PROGCONST_SECTION_START = .;
    __SEG_START_APP_CODE_CONST_RGN = .;
  . = ALIGN(4);
    *(.osApiConst)  /* API constants    */
    *(.rodata)      /* litteral strings (constants, strings, etc.) */
    *(.rodata*)     /* litteral strings (constants, strings, etc.) */
    . = ALIGN(4);
    *(.osApiCode)   /* API functions    */
    /* Sections for code of tasks and ISR */%
foreach proc in PROCESSES do
%
    *(.% !proc::KIND %_% !proc::NAME %Code)
%
end foreach
%
  } >FLASH
```