ChibiOS

SE302

Samuel TARDIEU
samuel.tardieu@telecom-paris.fr
Septembre 2019

Plan

Services du système d'exploitation

Anatomie d'une application

Caractéristiques principales (1/2)

- logiciel libre ;
- supporte un grand nombre de micro-contrôleurs ;
- système temps-réel multi-tâches préemptif ;
- se combine avec l'application finale lors de la compilation et de l'édition de liens ;
- existe en deux variantes : **RT** (complet, mais très efficace) et **NIL** (minuscule, avec moins de services de haut niveau) ;
- existe sous cinq licences : GPL (pour évaluation et développement interne), GPL + exception non contaminante, version commerciale gratuite (en échange de publicité pour ChibiOS) et versions commerciales payantes petits/moyens volumes ou grands volumes.

Caractéristiques principales (2/2)

- initialisation statique possible de toutes les structures de données ;
- aucune limite arbitraire codée en dur ;
- peu d'opportunités d'erreurs dans l'API ;
- possède un HAL (*hardware abstraction layer*) permettant de s'abstraire des opérations de bas niveau et un EX (*external devices*) gérant les composants externes (IMU, etc.) ;
- se charge des opérations d'initialisation du système ;
- s'intègre avec d'autres logiciels libres (systèmes de fichiers, piles réseau, etc.) ;
- est activement maintenu (ChibiOS 19.1.3 est sorti en juillet 2019).

Threads

ChibiOS permet d'utiliser un nombre quelconque de threads qui s'exécuteront de manière concurrente.

- Chaque thread a une priorité, qui peut être dynamique si cette option est activée.
- Seuls les threads de plus haute priorité parmi ceux qui sont prêts à s'exécuter ont accès au CPU, en mode *round-robin* configurable (quantum de temps) ou désactivable..
- Les threads peuvent être suspendus par eux-même ou par d'autres (**à éviter !**) et redémarrés.
- Les threads peuvent se synchroniser, se donner rendez-vous, etc.

En SE302 et en projet, on souhaite ne jamais stopper un thread depuis un autre.

Gestion du temps

ChibiOS offre un système complet de gestion du temps :

- Le programme a accès aux timers matériel : timers précis, configurables, compteurs, décompteurs, enregistrant la date d'événements externes.
- Le programme a accès à un nombre quelconque de timers virtuels, implémentés en utilisant un timer matériel, avec possibilité de *callback* (RT).
- Les primitives bloquantes de synchronisation vues par la suite disposent toutes de possibilité d'expiration (*timeout*).
- Le système peut fonctionner sans timer périodique (mode *tickless*).

Outils de synchronisation (1/2)

ChibiOS offre un large choix d'outils de synchronisation :

- Les **sémaphores** (compteurs, et binaires dans RT) permettent de produire des ressources ou d'attendre leur production.
- Les **verrous** (*mutex*, RT) permettent de protéger une section critique. On ne peut relâcher que le dernier mutex acquis.
- Les **variables conditionnelles** (RT) permettent de synchroniser des tâches sur une condition à vérifier, à utiliser avec les mutex.

Outils de synchronisation (1/2)

ChibiOS offre un large choix d'outils de synchronisation :

- Les **sémaphores** (compteurs, et binaires dans RT) permettent de produire des ressources ou d'attendre leur production.

  $\Rightarrow$ déblocage d'un thread par une ISR

- Les **verrous** (*mutex*, RT) permettent de protéger une section critique. On ne peut relâcher que le dernier mutex acquis.

- Les **variables conditionnelles** (RT) permettent de synchroniser des tâches sur une condition à vérifier, à utiliser avec les mutex.

Outils de synchronisation (1/2)

ChibiOS offre un large choix d'outils de synchronisation :

- Les **sémaphores** (compteurs, et binaires dans RT) permettent de produire des ressources ou d'attendre leur production.

  $\Rightarrow$ déblocage d'un thread par une ISR
- Les **verrous** (*mutex*, RT) permettent de protéger une section critique. On ne peut relâcher que le dernier mutex acquis.

  $\Rightarrow$ accès à un écran LCD partagé
- Les **variables conditionnelles** (RT) permettent de synchroniser des tâches sur une condition à vérifier, à utiliser avec les mutex.

Outils de synchronisation (1/2)

ChibiOS offre un large choix d'outils de synchronisation :

- Les **sémaphores** (compteurs, et binaires dans RT) permettent de produire des ressources ou d'attendre leur production.
  $\Rightarrow$ déblocage d'un thread par une ISR
- Les **verrous** (*mutex*, RT) permettent de protéger une section critique. On ne peut relâcher que le dernier mutex acquis.
  $\Rightarrow$ accès à un écran LCD partagé
- Les **variables conditionnelles** (RT) permettent de synchroniser des tâches sur une condition à vérifier, à utiliser avec les mutex.
  $\Rightarrow$ attente d'une étape dans un programme

Outils de synchronisation (2/2)

- Les **événements** permettent d'associer librement des producteurs d'événements à des consommateurs.
- Les **boîtes à lettres** (RT) permettent de poster des informations (idéalement immutables) destinées à d'autres threads de manière asynchrone.
- Les **messages** (RT) permettent d'échanger des informations (qui peuvent être mutables) de manière synchrone entre threads.

Outils de synchronisation (2/2)

- Les **événements** permettent d'associer librement des producteurs d'événements à des consommateurs.
  $\Rightarrow$ appui sur un bouton poussoir
- Les **boîtes à lettres** (RT) permettent de poster des informations (idéalement immutables) destinées à d'autres threads de manière asynchrone.

- Les **messages** (RT) permettent d'échanger des informations (qui peuvent être mutables) de manière synchrone entre threads.

Outils de synchronisation (2/2)

- Les **événements** permettent d'associer librement des producteurs d'événements à des consommateurs.
  $\Rightarrow$ appui sur un bouton poussoir
- Les **boîtes à lettres** (RT) permettent de poster des informations (idéalement immutables) destinées à d'autres threads de manière asynchrone.
  $\Rightarrow$ caractères lus depuis RS232 envoyés par l'ISR
- Les **messages** (RT) permettent d'échanger des informations (qui peuvent être mutables) de manière synchrone entre threads.

Outils de synchronisation (2/2)

- Les **événements** permettent d'associer librement des producteurs d'événements à des consommateurs.
  $\Rightarrow$ appui sur un bouton poussoir
- Les **boîtes à lettres** (RT) permettent de poster des informations (idéalement immutables) destinées à d'autres threads de manière asynchrone.
  $\Rightarrow$ caractères lus depuis RS232 envoyés par l'ISR
- Les **messages** (RT) permettent d'échanger des informations (qui peuvent être mutables) de manière synchrone entre threads.
  $\Rightarrow$ passage sans copie d'un buffer réseau

Gestion de la mémoire

Par défaut, dans ChibiOS, tout est géré en statique. Il existe toutefois, **lorsque cela est indispensable** :

- des primitives pour gérer un ou plusieurs tas séparés, utilisant l'algorithme *first fit* ;
- des primitives pour gérer des allocateurs de taille fixe (*pools*), à préférer aux tas ;
- des primitives pour créer dynamiquement des threads.

Il est fortement conseillé de se restreindre à des allocations purement statiques : lorsque l'édition de liens se termine avec succès, on **sait** qu'aucun problème d'allocation mémoire ne pourra plus survenir.

Threads, mémoire et pile

Lors de la création d'un thread, il faut fournir une zone utilisable comme pile pour stocker les données temporaires et la pile d'appel :

- Si cette zone est trop petite, il y a un risque de débordement de pile, entraînant en général une corruption silencieuse de la mémoire.
- Si elle est trop grande, il y a gaspillage de ressources.

Cette taille est compliquée à évaluer, et il faut utiliser tous les outils mis à disposition pour vérifier qu'elle est suffisante. Il faut également limiter l'allocation d'objets volumineux sur la pile (utilisation de `static` et de synchronisation).

Flux et canaux de communication

ChibiOS fournit, en C, une interface en mode « orienté objet » permettant d'utiliser (en RT) :

- des flux abstraits ;
- des canaux d'entrées/sorties abstraits construits sur les flux ;
- des files d'entrées/sorties construites sur les canaux.

Voir la documentation pour plus d'informations.

Pilotes de périphériques

Les pilotes de périphériques (*device drivers*) de ChibiOS sont divisés en deux catégories :

1. Les pilotes de périphériques physiques, conçus en deux parties :

   le haut niveau qui assure une API commune ;
   le bas niveau qui est spécifique à l'architecture.

2. Les pilotes de périphériques complexes, qui utilisent d'autres pilotes pour accéder au matériel.

Pilotes de périphériques

Les pilotes de périphériques (*device drivers*) de ChibiOS sont divisés en deux catégories :

1. Les pilotes de périphériques physiques, conçus en deux parties :

   le haut niveau qui assure une API commune ;
   le bas niveau qui est spécifique à l'architecture.

   $\Rightarrow$ un pilote SPI sur STM32

2. Les pilotes de périphériques complexes, qui utilisent d'autres pilotes pour accéder au matériel.

Pilotes de périphériques

Les pilotes de périphériques (*device drivers*) de ChibiOS sont divisés en deux catégories :

1. Les pilotes de périphériques physiques, conçus en deux parties :

   le haut niveau qui assure une API commune ;
   le bas niveau qui est spécifique à l'architecture.

   $\Rightarrow$ un pilote SPI sur STM32
   $\Rightarrow$ un pilote SPI en *bit-banging*

2. Les pilotes de périphériques complexes, qui utilisent d'autres pilotes pour accéder au matériel.

Pilotes de périphériques

Les pilotes de périphériques (*device drivers*) de ChibiOS sont divisés en deux catégories :

1. Les pilotes de périphériques physiques, conçus en deux parties :

   le haut niveau qui assure une API commune ;
   le bas niveau qui est spécifique à l'architecture.

   $\Rightarrow$ un pilote SPI sur STM32
   $\Rightarrow$ un pilote SPI en *bit-banging*

2. Les pilotes de périphériques complexes, qui utilisent d'autres pilotes pour accéder au matériel.

   $\Rightarrow$ un pilote de SD-card qui utilise un pilote SPI

Classes d'interruptions

ChibiOS supporte trois types d'interruptions :

Interruptions classiques : elles peuvent être temporairement masquées par le noyau pour des périodes très courtes.

Interruptions rapides : celles-ci ne peuvent pas être retardées par le noyau qu'elles peuvent préempter. Par contre, elles ne peuvent pas utiliser ses services, car le noyau ChibiOS pouvant être interrompu à n'importe quel moment, les structures internes peuvent ne pas être dans un état stable. Elles peuvent utiliser des zones de mémoire partagées avec l'application.

Interruptions non masquables : elles sont très rapides, mais n'ont aucune possibilité d'accès aux services du système.

États du système

Un système ChibiOS, après l'initialisation, est dans un des état suivants :

Thread : toutes les API normales sont utilisables, les threads s'exécutent.

S-Locked : les interruptions classiques sont masquées (mais pas les rapides), seules les primitives de classe S et de classe I peuvent être invoquées.

I-Locked : toutes les interruptions masquables sont masquées, seules les primitives de classe I peuvent être invoquées.

Pour la liste complète, voir
`http://chibios.sourceforge.net/html/concepts.html`

Primitives du noyau

Les primitives sont construites selon la notation `ch`*<group><action><suffix>*, comme par exemple `chSemSignalI()`. Les suffixes sont :

sans suffixe : utilisables depuis l'état **thread** (normal).

S : utilisables depuis l'état **S-locked**.

I : utilisables depuis les états **I-locked** et **S-locked**. Depuis l'état **S-locked**, il faut forcer un *rescheduling* avant de sortir de la zone critique.

X : utilisables depuis les trois états ci-dessus.

FromIsr : (en nombre réduit) utilisables depuis certaines routines de gestion d'interruptions classiques.

Certaines fonctions sans suffixe exposent leurs contraintes dans la documentation.

Niveaux de priorités

Il est indispensable, pour chaque tâche, de réfléchir au niveau de priorité souhaité. Les niveaux prédéfinis sont, du moins prioritaire au plus prioritaire :

`IDLEPRIO` : réservé à l'idle thread ;

`LOWPRIO` : plus bas niveau utilisable pour le code utilisateur ;

`NORMALPRIO` : au centre de la hiérarchie, destiné à être utilisé en relatif (`NORMALPRIO - 1`, `NORMALPRIO + 4`) ;

`HIGHPRIO` : plus haut niveau utilisable pour le code utilisateur ;

`ABSPRIO` : maximum absolu de l'espace des priorités réservées au système.

Initialisation statique

ChibiOS fournit des macros suffixées par `_DECL` permettant de déclaré des objets dans un état initialisé.

```
// Case 1: semaphore "sem" needs to be initialized at run time.
static semaphore_t sem;
int main() {
  chSysInit();                  // Initialize ChibiOS run time
  ...
  chSemObjectInit(&sem, 5);     // Initialize semaphore sem
  ...
}

// Case 2: semaphore "sem" is declared and initialized
// at compilation time in the "data" section through the
// SEMAPHORE_DECL macro.
static SEMAPHORE_DECL(sem, 5);
```

Utilitaires divers

ChibiOS vient avec des utilitaires qui peuvent être utilisées dans une application :

- Des **evénéments périodiques** peuvent être générés et utilisés par plusieurs threads.
- Un **interpréteur de commande** (*command shell*) extensible peut être connecté à un flux d'entrée-sortie.
- Un **formatteur** similaire à `sprintf` permet de formatter des données diverses sur un flux de sortie.

Plan

Services du système d'exploitation

Anatomie d'une application

Organisation des répertoires

Les répertoires importants sont :

- `boards/` contient un répertoire par carte (*board*) décrivant les caractéristiques propres à la carte
- `os/` contient le noyau de ChibiOS et les pilotes de périphériques
- `<votre projet>/` contiendra votre projet (directement ou dans un sous-répertoire), faisant indirectement référence aux autres répertoires

Tous les répertoires que vous créérez peuvent l'être à l'extérieur des sources de ChibiOS, facilitant ainsi la migration vers une nouvelle version de l'OS.

Description d'une carte

Les fichiers décrivant une carte sont :

`board.mk`	fragment de `Makefile` décrivant comment accéder aux fichiers source et aux fichiers d'entête
`board.h`	valeur initiale de configuration des périphériques (GPIO) et prototypes des méthodes spécifiques à cette carte
`board.c`	fonctions d'initialisations spécifiques de la carte
autres	fichiers implémentant les accès aux périphériques de la carte

Configuration d'une application

La configuration d'une application ChibiOS se fait à l'aide de quatre fichiers (on part en général d'un modèle) :

`chconf.h` sélectionne et paramètre les services du noyau ChibiOS utilisés par l'application ;

`halconf.h` sélectionne et paramètre les périphériques utilisés par l'application ;

`mcuconf.h` contrôle la configuration du micro-contrôleur (horloges, etc.).

`Makefile` indique les répertoires source et les fichiers utilisés, ainsi que les fragments à inclure (par exemple le `board.mk`) décrit ci-avant.

Outils de debug

Un certain nombre de paramètres permettent de configurer les vérifications effectuées par l'exécutif de ChibiOS. Citons par exemple :

`CH_DBG_SYSTEM_STATE_CHECK` : vérifie la classe (normal, S, I, FromIsr) des fonctions et du contexte d'appel

`CH_DBG_ENABLE_CHECKS` : vérifie la validité des paramètres donnés à l'API de ChibiOS

`CH_DBG_ENABLE_ASSERTS` : insère des points de vérification de cohérence du système

`CH_DBG_ENABLE_STACK_CHECK` : vérifie l'absence de débordement de pile

Ces paramètres sont généralement utilisés lors du développement (**obligatoires en SE302 et en projet**) et désactivés lors de la mise en production.