Master Réseaux Mobiles et Services

Système d’exploitation des capteurs TinyOS

B.KADRI, STIC Lab, Dép GEE, Université de Tlemcen

TinyOS

- TinyOS est un système d'exploitation open-source conçu pour des réseaux de capteurs sans-fil. Il respecte une architecture basée sur une association de composants, réduisant la taille du code nécessaire à sa mise en place. Cela s'inscrit dans le respect des contraintes de mémoires qu'observent les réseaux de capteurs.
- La bibliothèque de composant de TinyOS est particulièrement complète puisqu'on y retrouve des protocoles réseaux, des pilotes de capteurs et des outils d'acquisition de données.
- L'ensemble de ces composants peut être et adapté à une application précise.
- En s'appuyant sur un fonctionnement événementiel, TinyOS propose à l'utilisateur une gestion très précise qui permet de mieux s'adapter à la nature aléatoire de la communication sans fil entre interfaces physiques.

Caractéristiques

- Disponibilité : TinyOS est un système principalement développé et soutenu par l'université américaine de Berkeley, qui le propose en téléchargement sous une licence libre et en assure le suivi. Ainsi, l'ensemble des sources sont disponibles pour de nombreuses cibles matérielles.
- Event-driven : Le fonctionnement d'un système basé sur TinyOS s'appuie sur la gestion des évènements. Ainsi, l'activation de tâches, leur interruption ou encore la mise en veille du capteur s'effectue à l'apparition d'évènements, ceux-ci ayant la plus forte priorité. Ce fonctionnement évènementiel (event-driven) s'oppose au fonctionnement dit temporel (time-driven) où les actions du système sont gérées par une horloge donnée.
- Langage : TinyOS utilise comme langage de programmation NesC .

Caractéristiques

- Préemptif : Le caractère préemptif d'un système d'exploitation précise que celui-ci permet l'interruption d'une tâche en cours. TinyOS ne gère pas ce mécanisme de préemption entre les tâches mais donne la priorité aux interruptions matérielles. Ainsi, les tâches entre-elles ne s'interrompent pas mais une interruption peut stopper l'exécution d'une tâche.
- Temps réel : Lorsqu'un système est dit temps réel celui-ci gère des niveaux de priorité dans ses tâches permettant de respecter des échéances données par son environnement. Dans le cas d'un système strict, aucune échéance ne tolère de dépassement contrairement à un système temps réel souple. TinyOS se situe au-delà de ce second type car il n'est pas prévu pour avoir un fonctionnement temps réel.
- Consommation : TinyOS a été conçu pour réduire au maximum la consommation en énergie du capteur. Ainsi, lorsqu'aucune tâche n'est active, il se met automatiquement en veille.

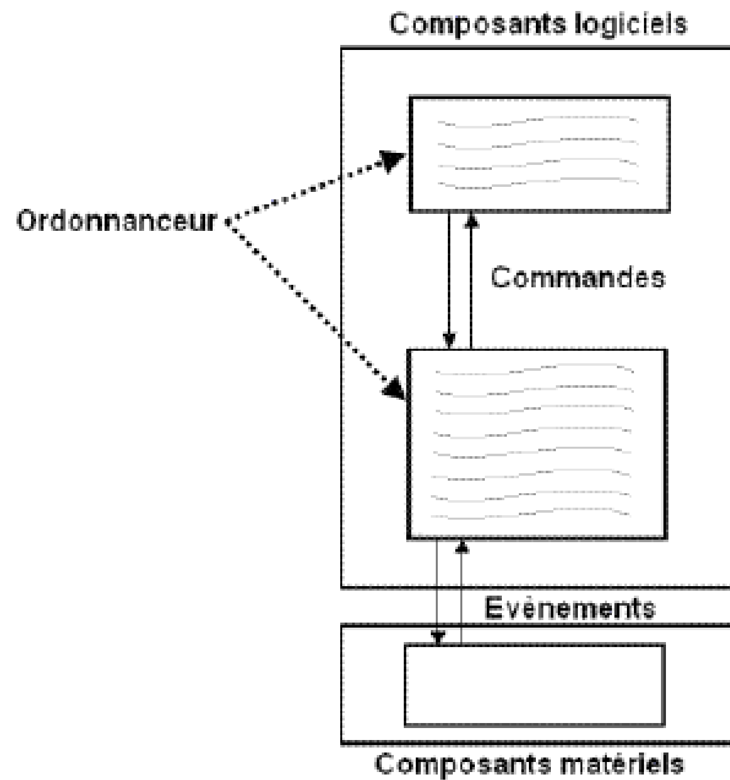
Allocation de la mémoire

- Il est important de préciser de quelle façon un système d'exploitation aborde la gestion de la mémoire.
- C'est encore plus significatif lorsque ce système travaille dans un espace restreint.
- TinyOS a une empreinte mémoire très faible puisqu'il ne prend que 300 à 400 octets dans le cadre d'une distribution minimale. En plus de cela, il est nécessaire d'avoir 4 Ko de mémoire libre
  - La pile : sert de mémoire temporaire au fonctionnement du système notamment pour l'empilement et le dépilement des variables locales.
  - Les variables globales : réservent un espace mémoire pour le stockage de valeurs pouvant être accessible depuis des applications différentes.
  - La mémoire libre : pour le reste du stockage temporaire.
- La gestion de la mémoire possède de plus quelques propriétés. Ainsi, il n'y a pas d'allocation dynamique de mémoire et pas de pointeurs de fonctions. Bien sur cela simplifie grandement l'implémentation.
- Par ailleurs, il n'existe pas de mécanisme de protection de la mémoire sous TinyOS ce qui rend le système particulièrement vulnérable aux crash et corruptions de la mémoire.

Structure logicielle

- Le système d'exploitation TinyOS s'appuie sur le langage NesC. Celui-ci propose une architecture basée sur des composants, permettant de réduire considérablement la taille mémoire du système et de ses applications.
- Chaque composant correspond à un élément matériel (LEDs, timer, ADC . . .) et peut être réutilisé dans différentes applications.
- Ces applications sont des ensembles de composants associés dans un but précis.
- Les composants peuvent être des concepts abstraits ou bien des interfaces logicielles liées aux entrées-sorties matérielles de la cible étudiée (carte ou dispositif électronique).
- L'implémentation de composants s'effectue en déclarant des tâches, des commandes ou des évènements.

Structure logicielle



Les tâches, commandes, évènements

- Les tâches sont utilisées pour effectuer la plupart des blocs d'instruction d'une application. A l'appel d'une tache, celle-ci va prendre place dans une file d'attente de type FIFO (First In First Out) pour être exécutée.
- Comme nous l'avons vu, il n'y a pas de mécanisme de préemption entre les tâches et une tache activée s'exécute en entier. Ce mode de fonctionnement permet d'éviter des opérations pouvant bloquer le système (inter-blocage, famine, . . .). Par ailleurs, lorsque la file d'attente des taches est vide, le système d'exploitation met en veille le dispositif jusqu'au lancement de la prochaine interruption (on retrouve le fonctionnement (event-driven).
- Les évènements sont prioritaires par rapport aux tâches et peuvent interrompre la tache en cours d'exécution. Ils permettent de faire le lien entre les interruptions matérielles (pression d'un bouton, changement d'état d'une entrée, . . .) et les couches logicielles que constituent les taches.

L'ordonnanceur TinyOS

- Le choix d'un ordonnanceur déterminera le fonctionnement global du système et le dotera de propriétés précises telles que la capacité à fonctionner en temps réel.
- L'ordonnanceur TinyOS a deux niveaux de priorité (bas pour les tâches, haut pour les évènements), une file d'attente FIFO (disposant d'une capacité de 7)
- Par ailleurs, entre les tâches, un niveau de priorité est déni permettant de classer les tâches, tout en respectant la priorité des interruptions (ou évènements). Lors de l'arrivée d'une nouvelle tache, celle-ci sera placée dans la file d'attente en fonction de sa priorité (plus elle est grande, plus le placement est proche de la sortie).
- Dans le cas ou la file d'attente est pleine, la tâche dont la priorité est la plus faible est enlevée de la FIFO

Le langage NesC

- Dans la pratique, NesC permet de déclarer 2 types de composants : les modules et les configurations :
  - Les modules constituent les briques élémentaires de code et implémentent une ou plusieurs interfaces.
  - Les configurations pour regrouper les fonctionnalités des modules.
  - Un fichier top-level configuration permet de faire le lien entre tous les composants.

Construction d’une application en NesC

- Toutes les applications ont besoin d’un fichier de configuration de haut niveau.
- Par exemple **NomApplication.nc** est le fichier de configuration de l’application et le fichier source que le compilateur nesC utilise pour générer l’exécutable.
- **NomApplicationM.nc** quant à lui correspond à l’implémentation à proprement dit de l’application.
- Les composants nesC peuvent se relier les uns aux autres, et ici c’est le fichier **NomApplication.nc** qui permet de faire cette connexion entre le module **NomApplicationM.nc** et les autres composants auxquels l’application fait appel.
- La convention de TinyOS veut alors que **NomApplication.nc** représente la configuration et **NomApplicationM.nc** représente le module correspondant.
- Il faut donc respecter cette convention pour utiliser le Makefile livré avec la distribution de TinyOS.

Construction d'une application en NesC

- **Le fichier de configuration « NomApplication.nc » :**
- **NomApplication.nc :**
- **configuration NomApplication {**
- **}**
- **implementation {**
- **components ……………………………;**
- **……………………………………………………;**
- **…………………………………………………… ;**
- **}**

Construction d’une application en NesC

- La première chose à noter est le mot clé « configuration » qui indique qu’il s’agit d’un fichier de configuration dont le nom est « NomApplication ».
- A l’intérieur des accolades, il est possible de spécifier des interfaces requises ou offertes par la configuration.
- La véritable configuration est implémentée au sein des deux accolades qui suivent le mot clé « implementation ».
- La ligne « components » spécifie les différents composants auxquels la configuration fait référence.
- Le reste de l’implémentation consiste à connecter les interfaces utilisées par certains composants aux interfaces fournies par les autres.

Construction d'une application en NesC

- Le composant Main est le premier exécuté dans une application TinyOS. Pour être plus précis, la commande **Main.StdControl.init()** est la première fonction à être exécuté suivi de **Main.StdControl.start()**.
- Donc toute application TinyOS doit avoir un composant **"Main"** dans sa configuration. StdControl est une interface de base utilisée afin d'initialiser et de démarrer des composants TinyOS.
- **interface StdControl {**
- **command result_t init();**
- **command result_t start();**
- **command result_t stop();**
- **}**
- Nous pouvons voir que « StdControl » définit trois commandes, « init() », « start() » et « stop() ».
- « init() » La première est appelée quand un composant est initialisé pour la première fois,
- « start() » quand il est démarré, c'est à dire exécuté pour la première fois.
- « stop() » quand le composant est arrêté, par exemple pour éteindre le dispositif physique qu'il contrôle.

Construction d'une application en NesC

- **Le module « NomApplicationM.nc »:**
- **NomApplicationM.nc**
- **includes ……;**
- **module NomApplicationM {**
- **provides {**
- **interface StdControl;**
- **}**
- **uses {**
- **interface Timer ;**
- **………**
- **}**
- **}**
- **implementation {**
- **……………………… ;**
- **…………**
- **}**

Construction d’une application en NesC

- La première ligne c’est l’inclusion d’un fichier d’en tête.
- La deuxième signale qu’il s’agit d’un module appelé « NomApplicationM » et déclare les interfaces fournies et utilisées.
- Nous voyons qu’il fournit l’interface « StdControl » par exemple. elle est nécessaire pour initialiser et démarrer du composant. Il utilise ensuite plusieurs interfaces: par exemple : le Timer.
- Ces déclarations donnent accès à « NomApplicationM » aux commandes de ces interfaces, et l’oblige à implémenter les événements (events) déclarés dans ces interfaces.

Exemple: Blink.nc

- Blink.nc
- /**
- * Blink is a basic application that toggles the leds on the mote on every clock interrupt.
- * The clock interrupt is scheduled to occur every second. The initialization of the clock
- * can be seen in the Blink initialization function, StdControl.start().<p>
- *
- * @author tinyos-help@millennium.berkeley.edu
- **/
- configuration Blink {
- }
- implementation {
- components Main, BlinkM, SingleTimer, LedsC;
- Main.StdControl -> SingleTimer.StdControl;
- Main.StdControl -> BlinkM.StdControl;
- BlinkM.Timer -> SingleTimer.Timer;
- BlinkM.Leds -> LedsC;
- }

Exemple: BlinkM.nc

- BlinkM.nc
- /**
- * Implementation for Blink application. Toggle the red LED when a Timer fires.
- **/
- module BlinkM {
- provides {
- interface StdControl;
- }
- uses {
- interface Timer;
- interface Leds;
- }
- }

Exemple: BlinkM.nc

- implementation {
- /**
- * Initialize the component.
- *
- * @return Always returns <code>SUCCESS</code>
- **/
- command result_t StdControl.init() {
- call Leds.init();
- return SUCCESS;
- }
- /**
- * Start things up. This just sets the rate for the clock component.
- *
- * @return Always returns <code>SUCCESS</code>
- **/
- command result_t StdControl.start() {
- // Start a repeating timer that fires every 1000ms
- return call Timer.start(TIMER_REPEAT, 1000);
- }

Exemple: BlinkM.nc

- /**
- * Halt execution of the application. This just disables the clock component.
- *
- * @return Always returns <code>SUCCESS</code>
- **/
- command result_t StdControl.stop() {
- return call Timer.stop();
- }
- /**
- * Toggle the red LED in response to the <code>Timer.fired</code> event.
- *
- * @return Always returns <code>SUCCESS</code>
- **/
- event result_t Timer.fired()
- {
- call Leds.redToggle();
- return SUCCESS;
- }
- }

Le Makefile

- COMPONENT=Blink
- include ../Makerules
- $ make mica2 /* installer l'application sur un capteur mica2 */
- ou
- $ make pc /* compiler l'application sur pc pour la tester avec le simulateur tossim cela genererera un dossier build/pc/ dans lequel se trouve votre main.exe */
- Pour lancer un vizualisateur graphique :
- $ tinyviz -run build/pc/main.exe 10

Travail demandé

- Réaliser un programme de la diffusion directe dans lequel chaque capteur périodiquement diffuse un message a ses voisins directs « dans sa porté », qui font la même chose pour leurs voisins directes.