Tiny Internet Interface

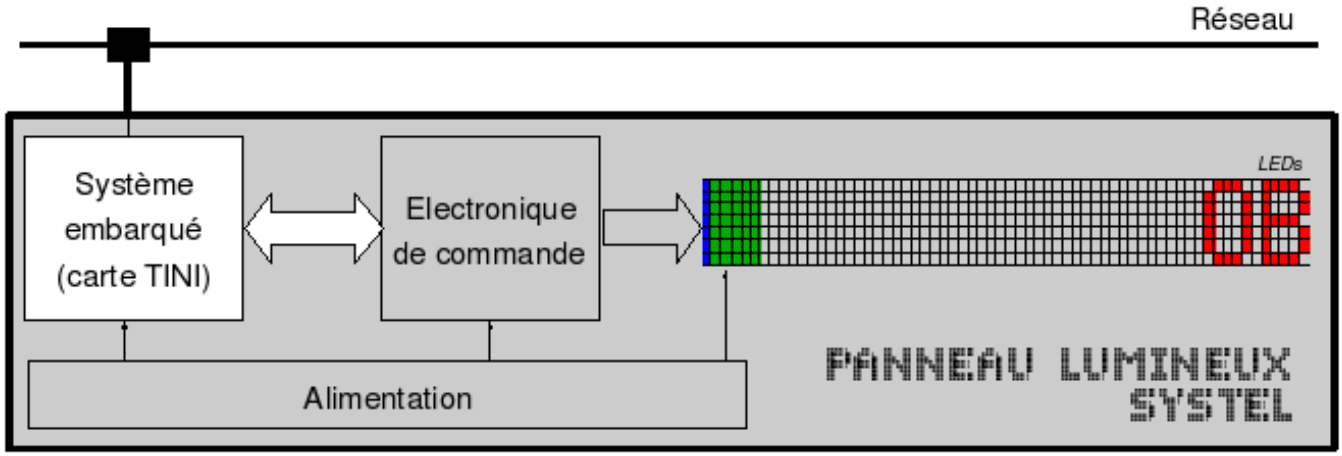
Table des matières

La carte d'expérimentation Taylec TutorIO

MISE EN SITUATION

Une carte TINI (*TIny Network Interface*) est un système embarqué basé sur un circuit hybride fabriqué par Dallas Semiconductor et destiné à faciliter la conception d'équipements pouvant dialoguer sur un réseau de type TCP/IP (Internet).

Le composant est fourni avec un système d'exploitation appelé TINI_OS, ainsi qu'un interpréteur de commande appelé SLUSH et une machine virtuelle JAVA. Cette dernière permet à cette carte de pouvoir être programmé en langage Java.



Exemple d'un panneau à messages variables intelligent utilisant un système embarqué TINI

SYSTÈME EMBARQUÉ

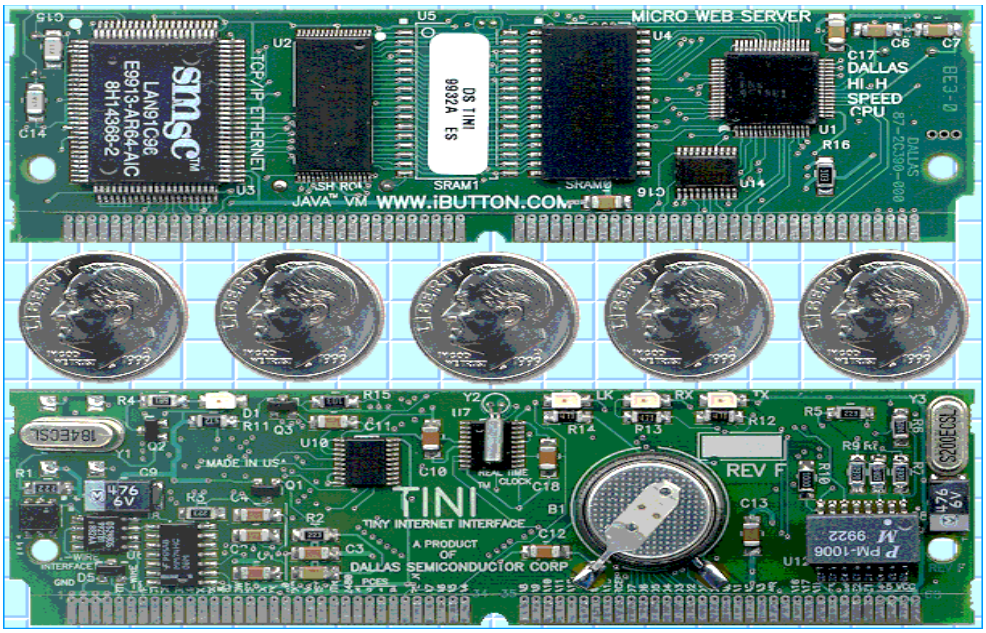
Un système embarqué (ou système enfoui) peut être défini comme un système électronique et informatique autonome, qui est dédié à une tâche bien précise.

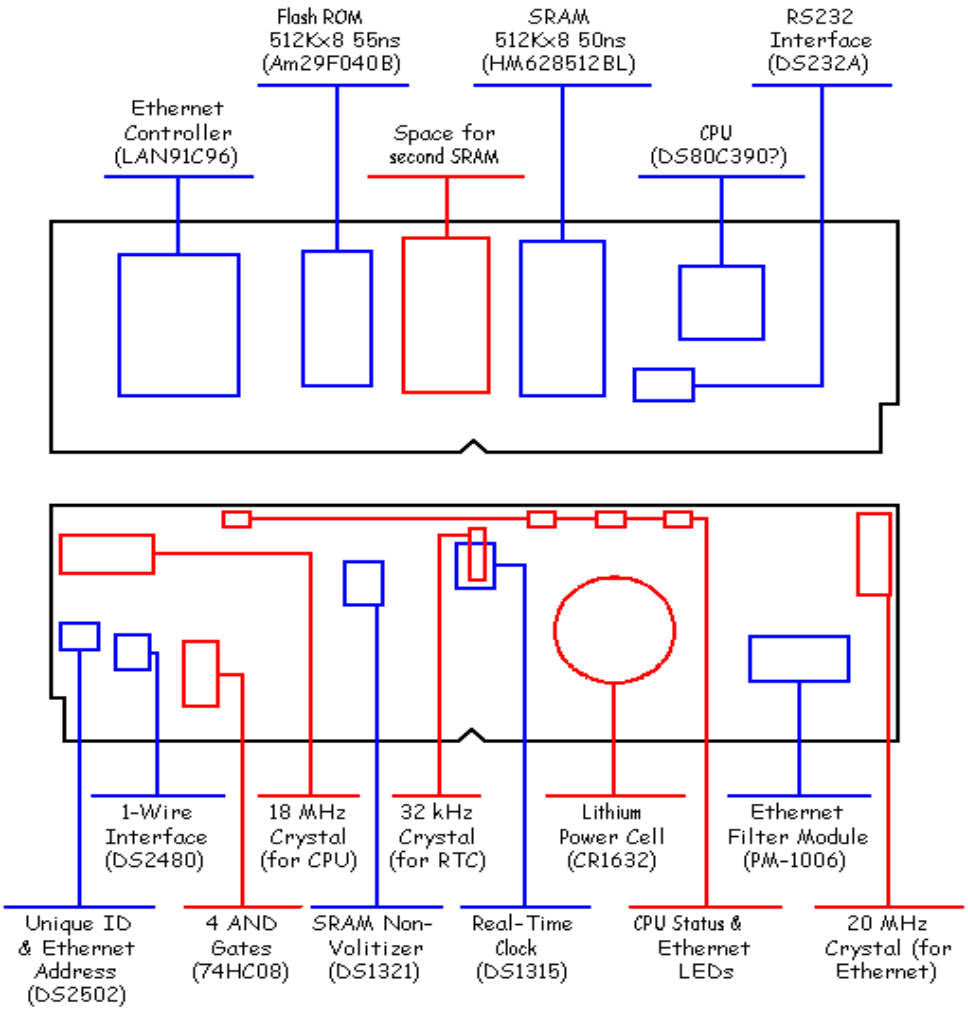
Ses ressources disponibles sont généralement limitées (coût faible, taille réduite, consommation restreinte, espace mémoire limité de l'ordre de quelques Mo maximum, puissance de calcul juste nécessaire pour répondre aux besoins, ...).

Exemple de système mebarqué basé sur un composant de Dallas Semi-Conductor :

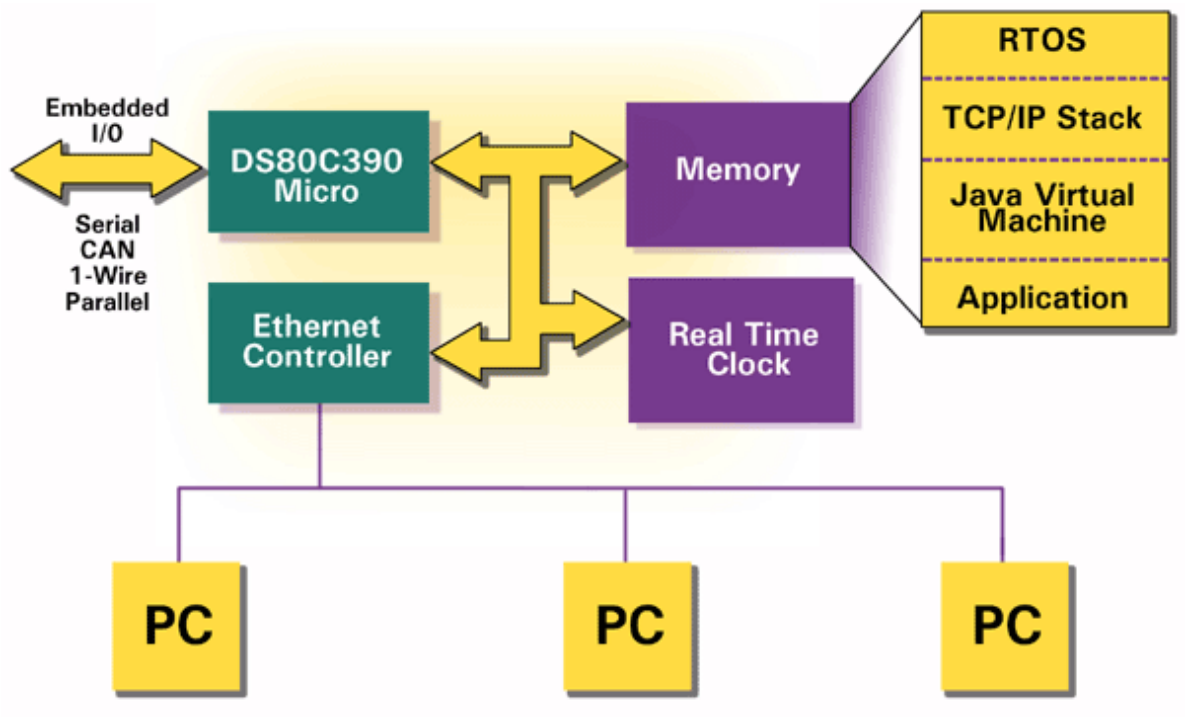
- µC 40Mhz (DS80C390) ou µC 75Mhz (DS80C400)
- Jeu d'instructions : 8051 (+ special functions)
- Coprocesseur mathématique 16/32-bits
- 3 x 16-bits timer/counters
- 4 x Ports 8 bits E/S
- 2 x Controlleur bus CAN
- Espace d'adressage : Max 4MB Data + Program

LES COMPOSANTS DE LA TINI

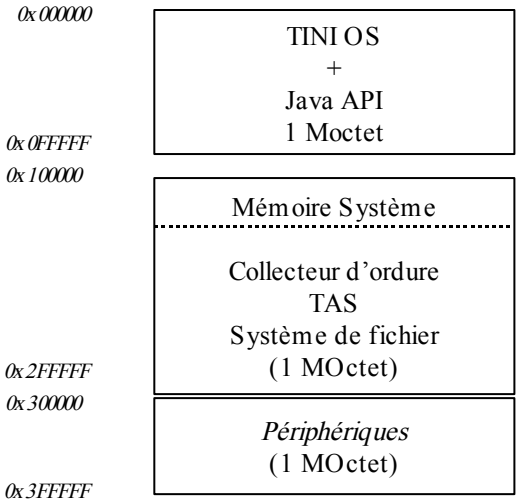




L'ARCHITECTURE



La cartographie mémoire (*map*) est la suivante :

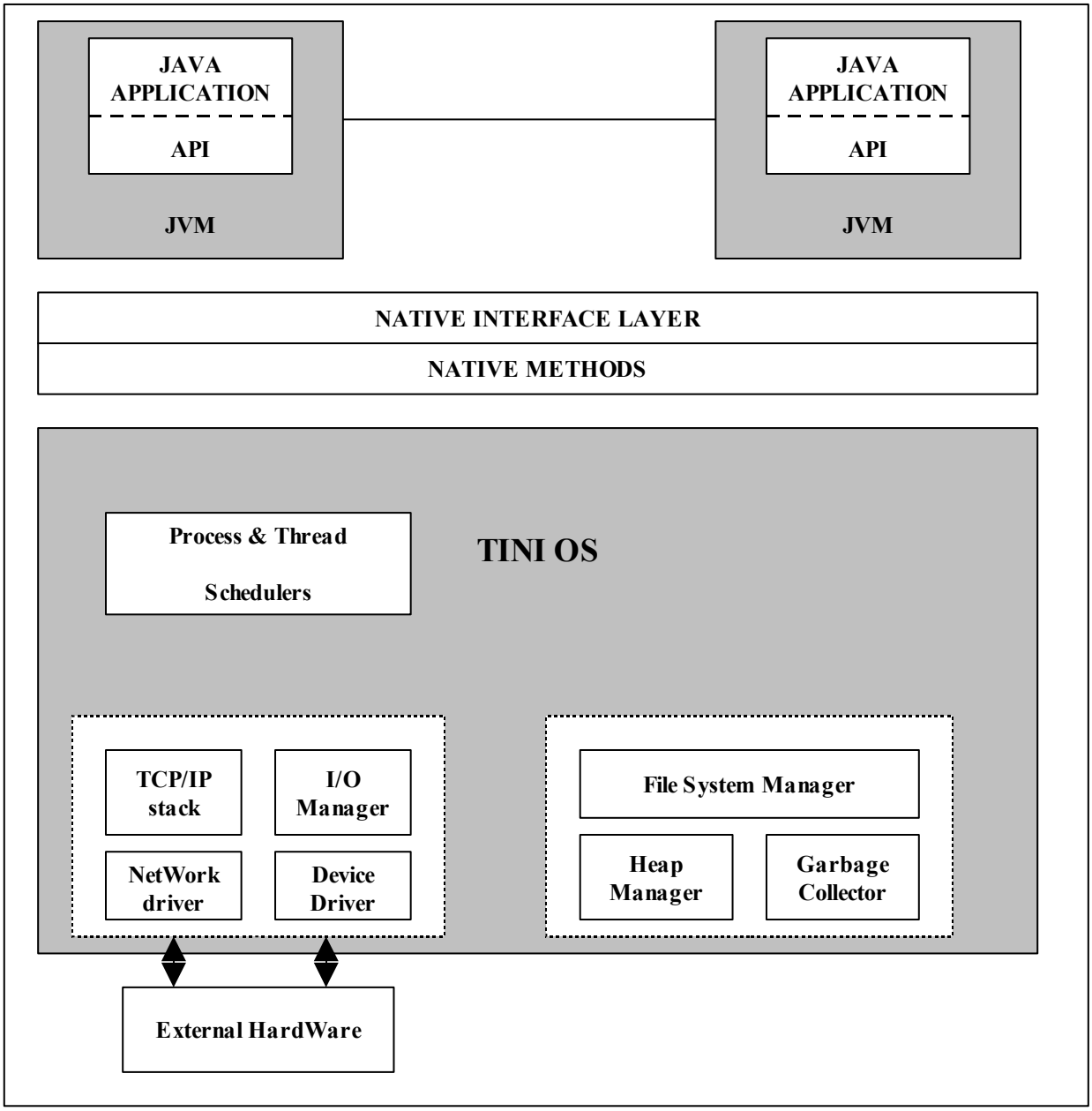


LE SYSTÈME D'EXPLOITATION TINIOS

C'est la couche la plus basse de l'environnement de distribution. Il est responsable de la gestion des ressources « système » : l'accès mémoire, gestion des tâches (sheduler), interaction avec l'environnement extérieur.

On peut résumer cet OS par trois composants :

- Le gestionnaire de tâches (sheduler)
- Gestion de la mémoire
- Gestion des entrées/sorties



Le gestionnaire de tâches (sheduler)

Le système d'exploitation contient un gestionnaire de processus et un gestionnaire de tâches. Ces gestionnaires sont des applications chargées par le microcontrôleur lorsque son timer (ISR : Interrupt Service Routine) génère une interruption à priorité haute toutes les millisecondes. A chaque évènement de l'ISR (accessible par la méthode uptimeMillis() dans la classe com.dalsemi.system.TINIOS), on a :

- Mise à jour du système toutes les millisecondes
- Chargement du gestionnaire de Thread toutes les 2 millisecondes
- Exécution des drivers toutes les 4 millisecondes
- Chargement du gestionnaire de Processus toutes les 8 millisecondes

Chaque processus a donc un temps d'exécution de 8 ms. Après expiration de ce temps, le processus est envoyé à la queue de la liste des processus et attend son tour. Chaque processus a son propre gestionnaire de tâches (Sheduler).

La gestion des Thread est de type coopératif : tous les threads ont un temps d'occupation CPU de 2 ms. En fait, en Java, les Threads sont préemptifs puisque ce sont eux qui s'approprient le CPU toutes les deux milisecondes.

La gestion des thread est moins « gourmande » que la gestion des processus ; c'est pourquoi la majorité des applications gèrent plusieurs Threads plutôt que plusieurs processus. L'implémentation de la synchronisation est alors plus simple puisqu'il n'y a pas à gérer les IPC (InterProcessusCommunication).

Plusieurs processus peuvent utiliser le système de fichier et les sockets puisque les méthodes de gestion sont « synchronisées » (gestion de la réentrance).

Puisqu'il est possible de gérer plusieurs processus, pendant l'exécution d'une application, il est possible d'exécuter des commandes « shell ».

Il est à noter que le « Garbage Collector » (ramasse miettes) est un processus distinct.

La gestion de la mémoire

Le gestionnaire de mémoire comprend les trois tâches :

- Allocation d'une mémoire « Heap » (Tas) pour chaque processus
- Allocation de zone de « Garbage » provoqué par les programmes Java
- Gestion des fichiers.

La mémoire « Heap » (voir plan mémoire) est utilisée pour stocker les objets créés. Cette mémoire est libérée par le « Garbage Collector » qui constitue un processus distinct. Le « Garbage Collector » est créé au « boot » du système.

Dans des conditions normales d'utilisation de la mémoire, le processus « Garbage Collector » reste dans un état inactif ; il n'utilise donc pas de ressource CPU. Il est réveillé dans l'une des trois conditions suivantes :

- Explicitement par la méthode gc() de la classe java.lang.System
- La fabrication d'un objet engendre un dépassement mémoire.
- La terminaison d'un processus.

A l'exécution du processus « Garbage Collector », toute la mémoire « Heap » n'est pas « nettoyée » ; il nettoie uniquement le « garbage » du processus en cours d'exécution. Ainsi, chaque processus a son « garbage » au sein de la mémoire « Heap ».

Tous les fichiers peuvent être créés, effacés, lus et écrits par les applications Java, en utilisant les classes du paquetage « java.io » comme File et FileOutputstream. La mémoire utilisée lors de la création de fichiers est identique à celle utilisée lors de la création d'objets, à savoir la « Heap ». Quand le gestionnaire de fichiers alloue la mémoire, un « tag » est positionné pour indiquer qu'il s'agit d'un fichier.

Le fait que le système de fichiers utilise le même espace que les objets (« Heap ») entraîne plusieurs conséquence : d'abord, la lecture et l'écriture est plus rapide que sur un disque dur puisque la « Heap » se trouve dans une SRAM. Ensuite, les cartes TINI non pourvus de « SRAM » (mais de RAM) perdent les fichiers à l'arrêt de l'alimentation. Enfin, plus il y a de fichiers, moins il reste de place pour la création d'objets dans les applications.

Le système de fichier contient aussi les fichiers Java exécutables « .tini » qui peuvent être exécutés par la JVM. Les « gros » fichiers sont fragmentés en des blocks de 512 octets ; Ils occupent donc une plage non-contigüe de la mémoire. Pour être exécuté, un fichier doit pourtant être en un seul block. Si bien que, la première fois qu'un fichier exécutable est exécuté, le système de fichier le défragmente pour générer un fichier binaire contigüe.

Les entrées/sorties du système

Le système d'entrées/sorties est divisé en deux composants :

- Le réseau TCP/IP
- Les bus de terrain CAN et 1-Wire.

Ces deux composants sont gérés par des processus distincts avec un « sheduler » indépendant. L'ISR « du sheduler » a une période de 4 millisecondes.

La pile TCP/IP constitue le plus gros du code natif. Il fournit une plus grande partie des applications réseaux et est suffisamment riche en classes dans le paquetage « jana.net ». La pile peut utiliser deux interfaces de communication : Ethernet (la plus rapide) pour une communication dans un réseau local, et PPP (Point to Point Protocol) pour commander un MODEM.

L'INTERPRÉTEUR DE COMMANDES SLUSH

Slush est un interpréteur de commandes proche de ceux existants sous Unix. Ses principales caractéristiques sont :

- Multi-threads, multi-utilisateurs (authentification par login et mot de passe).
- Accessible par port série ou par telnet (serveur telnetd)
- Offre des commandes de gestion de fichiers (visualisation, suppression, modification des permissions, etc.), mais aussi de configuration de l'interface réseau.
- Possède un serveur ftp, telnet et série
- Est exécuté à chaque « boot » de la carte TINI.

Il est à noter que slush est une application Java qui est interprétée par la machine virtuelle Java de la carte TINI.

Les fichiers de configuration

Lors de la première séquence de « boot », slush crée trois fichiers de configuration. Ces derniers sont placés sous le répertoire /**etc** :

```
TINI /etc> ls -l
total 5
drwxr-x  1 root  admin    3 Jan 27 15:14 .
drwxr-x  1 root  admin    1 Jan 27 15:13 ..
-rwxr--  1 root  admin   28 Jan 27 15:14 .tininet
-rwx---  1 root  admin  225 Jan 27 15:14 .startup
-rwxr--  1 root  admin  101 Jan 27 15:14 passwd
```

Description :

- **passwd** : qui définit les comptes d'identification (login et mots de passe crypté avec l'algorithme Secure Hash Algorithm « SHA1 ») présents sur le système

```
root:f8491b67e91f837c13c3444965281bcee5fca964:128
guest:1bb6e3a2abc20f654fe62fd139790c06394885d3:0
```

- **.tininet** : qui définit le nom d'hôte et de domaine du système

```
HostName:TINI
DomainName:
```

- **.startup** : est interprété par le « Slush » à chaque démarrqge (*boot*). Ce fichier permet de configurer les variables d'environnement des services (FTP, Telnet et série) et de lancer automatiquement une application

```
#Autogen'd slush startup file
setenv FTPServer enable
setenv TelnetServer enable
setenv SerialServer enable
initializeNetwork
```

Explications :

Les lignes précédés d'un « # » sont des commentaires.
Les trois lignes commençant par « setenv » valident (enable) respectivement les serveurs FTP, Telnet et Série.
Si une application a besoin d'utiliser la liaison série SERIAL0 que Slush utilise par défaut comme serveur série, l'administrateur devra mettre cette ligne en commentaire.

Des applications peuvent être chargées à l'interprétation de ce fichier, par une commande appropriée comme, par exemple :

```
java MonApplication.tini > debug.log
```

Cette commande demande à Slush d'exécuter le programme MonApplication.tini et de rediriger les flux java.lang.System.out et java.lang.System.err vers un fichier de journalisation (log) nommé debug.log.

Toutes les applications lancées à partir de ce fichier le seront en tâche de fond.

Les commandes

La liste complète des commandes supportées par slush peut être obtenue par la commande help :

```
TINI /> help
Available Commands:
append  arp     cat     cd
chmod   chown   clear   copy
cp      date    del     df
dir     downserver      echo        ftp
gc      genlog  help    history
hostname        ipconfig            java        kill
ls      md      mkdir       move
mv      netstat nslookup    passwd
ping    ps      pwd     rd
reboot  rm      rmdir   sendmail
setenv  source  startserver stats
stopserver      su      touch       useradd
userdel wall    wd      who
whoami
```

Pour une description plus complète de la commande, il suffit d'employer help suivie du nom de la commande. Par exemple :

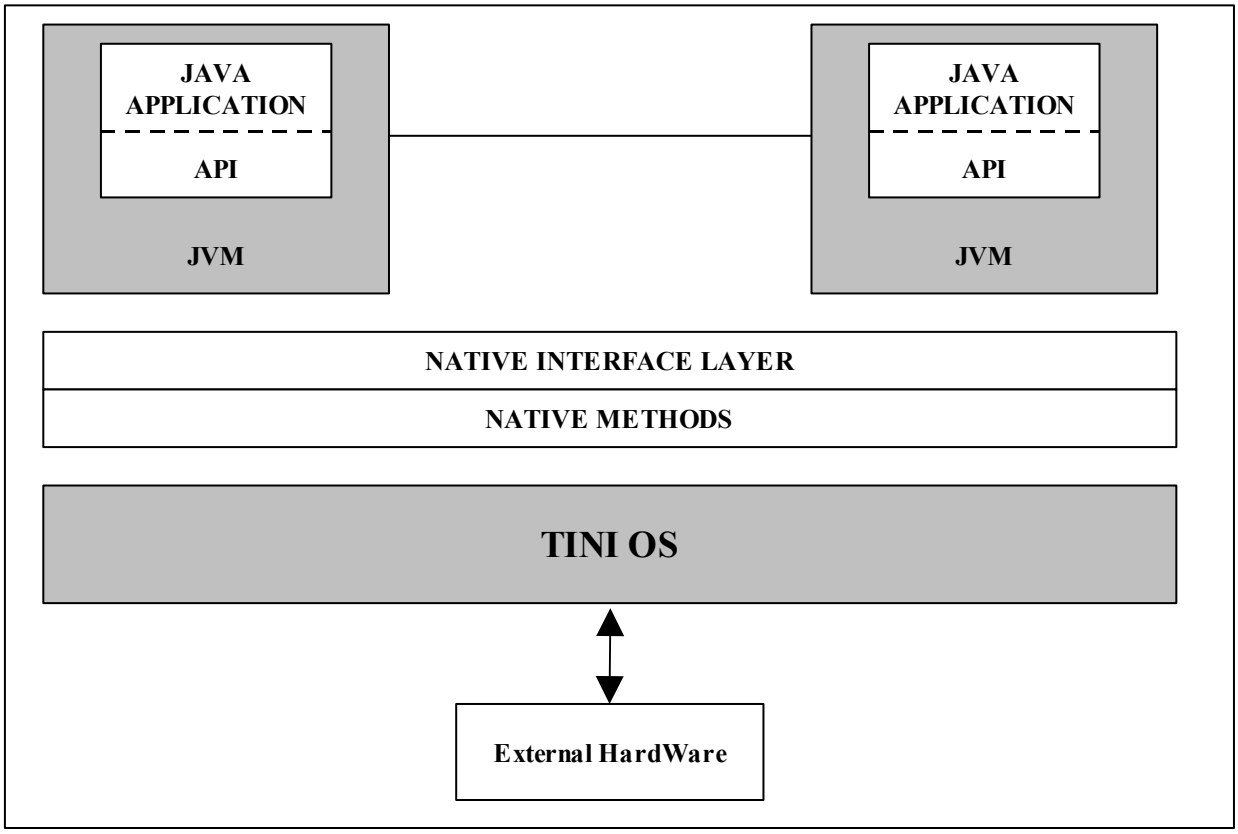
```
TINI /> help java
java FILE [&]
Executes the given Java application.
'&' indicates a background process.
```

LA TECHNOLOGIE JAVA TINI

Cet environnement peut être divisé en deux catégories :

- Code Natif : exécuté directement par le microcontrôleur
- API « ByteCodes » interprété par la machine virtuelle JAVA (JVM)

Le code des applications est écrit en JAVA et utilise l'API pour exploiter le code natif et les ressources intégrées (bus CAN, bus Serie, Bus I2C, etc .). Il est aussi possible d'écrire du code natif qui peut être utilisé au sein d'une application.



L'API TINI

Cet API couvre l'implémentation des paquetages (issus du JDK 1.1.8) :

- java.lang
- java.io
- java.net
- java.util
- java.lang.reflect
- javax.comm

ATTENTION : Les paquetages qui ne sont pas listés ci-dessus ne sont pas implémentés dans l'environnement d'exécution de TINI.

Remarques : des évolutions (classes supplémentaires) sont possibles dans le futur. Se renseigner sur le site : dalsemi.com

The TINI® runtime environment is copyrighted :

http://www.maxim-ic.com/products/tini/software/soft_ order.cfm

TINI Firmware Download : http://files.dalsemi.com/tini/index.html

Le paquetage com.dalsemi

Le paquetage com.dalsemi est propre à TINI et comporte les paquetages :

- **com.dalsemi.system** : ce paquetage fournit les classes permettant, notamment, l'accès aux entrées/sorties (bus du microcontrôleur) et l'utilisation du bus One-Wire. Il contient aussi les classes de gestion de l'horloge temps réel, du chien de garde et des interruptions extérieures.
- **com.dalsemi.tininet** : ce paquetage fournit la classe « TININet » contenant des méthodes statiques pour communiquer par réseau TCP/IP ou encore le configurer (adresse IP, masque de sous-réseau, atc.). Les sous-paquetage de com.dalsemi.tininet fournissent les méthodes pour implémenter les protocoles de haut niveau, comme DHCP, ICMP et DNS.
- **com.dalsemi.shell** : les classes de ce paquetage et ses sous-paquetages implémentent les commandes shell de l'interpréteur de commande « Slush ». On peut aussi y trouver les classes permettant d'utiliser des serveurs Telnet et FTP.
- **com.dalsemi.comm** : ce paquetage contient les classe de bas niveau permettant d'accéder au contrôleur du bus CAN. Il contient également les classes permettant la communication avec les bus séries. L'accès aux ports séries utilise l'implémentation de « JAVA Sun Communication API», issue du paquetage javax.comm.
- **com.dalsemi.onewire** : c'est le paquetage de base dans la hiérarchie pour l'utilisation du bus One-Wire. Comme les paquetages cités ci-dessus, l'API du bus One-Wire est aussi supporté par d'autres plateformes que TINI. Le paquetage com.dalsemi.onewire.container contient les classes (à utiliser avec des conteneurs) implémentant la communication avec le boîtier « One-Wire ». Pour éviter d'utiliser inutilement de l'espace mémoire, les conteneurs ne sont pas inclus dans ce package et doivent donc l'être dans l'application.

La machine virtuelle (JVM)

La place mémoire occupée par la JVM dans la carte TINI est inférieur à 40 Koctets. Malgré sa petite taille, toutes les fonctionnalités d'une JVM sont gérées, excepté les objets « finaux » et le chargement dynamique de classes (voir page 13 du document « The TINI's Specification Guide ».

Les méthodes natives

La couche native représente une collection de méthodes natives pour gérer l'environnement de la carte TINI : gestion de trames réseau TCP/IP (sockets), du chien de garde, de l'horloge temps réel.

Entre les méthodes natives et le code interprété Java se trouve une couche très fine : « Native Method Interface » ; cette couche fournit un mécanisme (appelé TNI) qui permet le changement de contexte entre le code exécuté sur la machine virtuelle Java et les méthodes natives. Ce mécanisme de passage est identique à la « JNI » (Java Native Interface) sur une JVM classique. Toutefois, « TNI » est plus léger et donc moins souple.

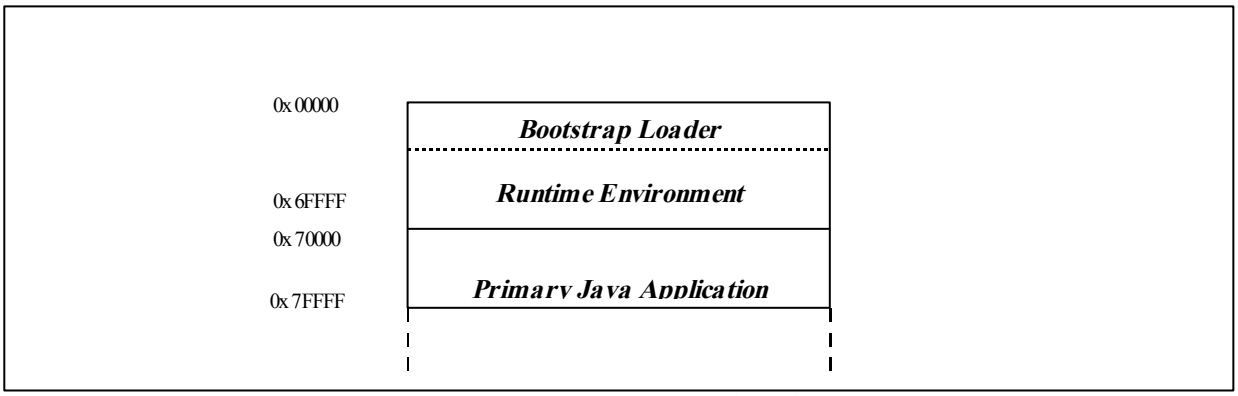
Les applications nécessitant des méthodes natives supplémentaires peuvent les charger en utilisant la méthode (dans la classe java.lang.Runtime) où « libname » est le nom de la librairie native :

```
Public static void loadLibrary(String libname)
```

La description du processus d'écriture de méthodes natives se trouvent dans deux fichiers « Native_Methods.txt » et Native_API.txt » fournies dans le SDK de TINI.

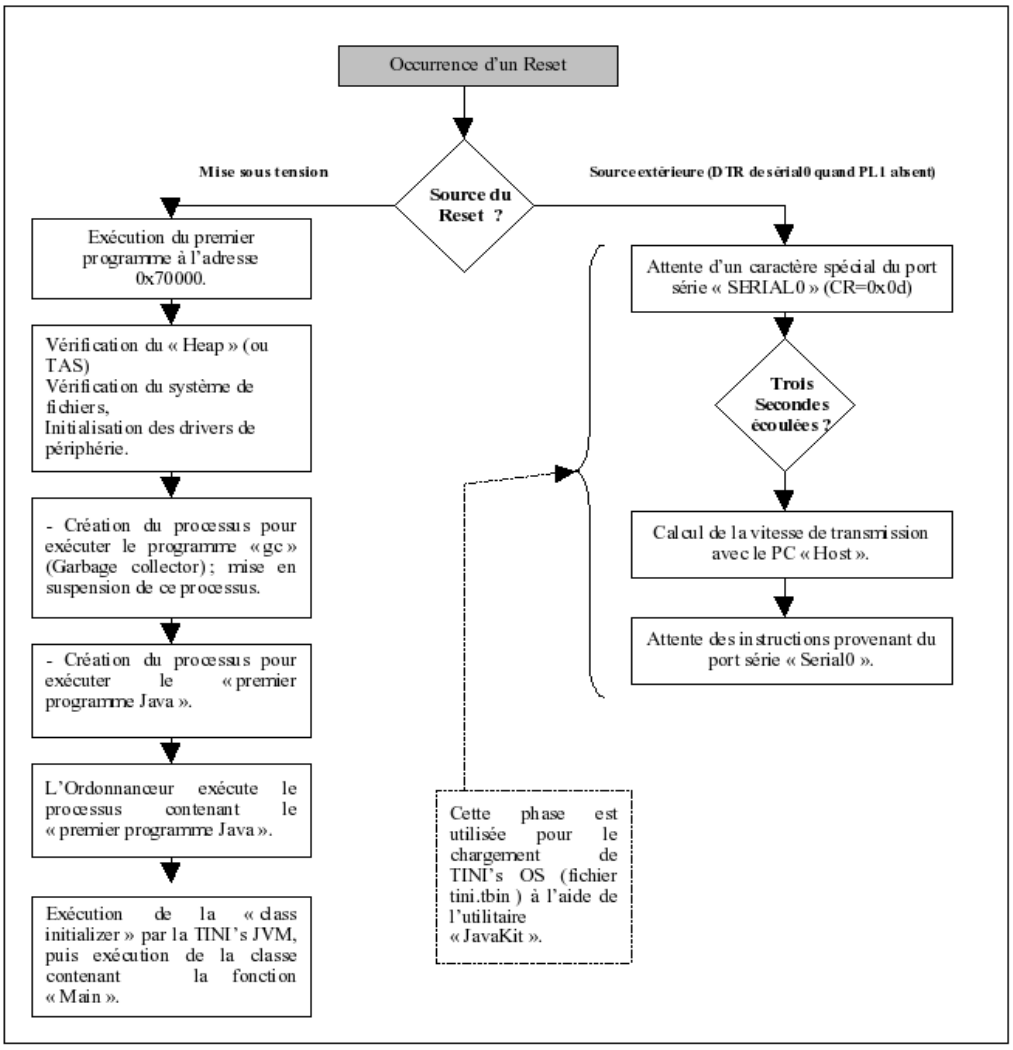
LE CHARGEMENT DU SYSTÈME

Le « bootstrap loader » est le premier programme (environ 4 KO) exécuté par le microcontrôleur, à la mise sous tension, et qui contrôle le chargement de la JRE et du premier programme dans la mémoire Flash (qui se trouve toujours à l'adresse 0x70000).



Cartographie mémoire (map)

Dans les conditions normales de fonctionnement, le contrôle est ensuite donné à la JRE. Après avoir exécuté certaines routines d'initialisation, celle-ci « charge » la première application Java. Les étapes de « Boot » peut être représenté par l'organigramme suivant :



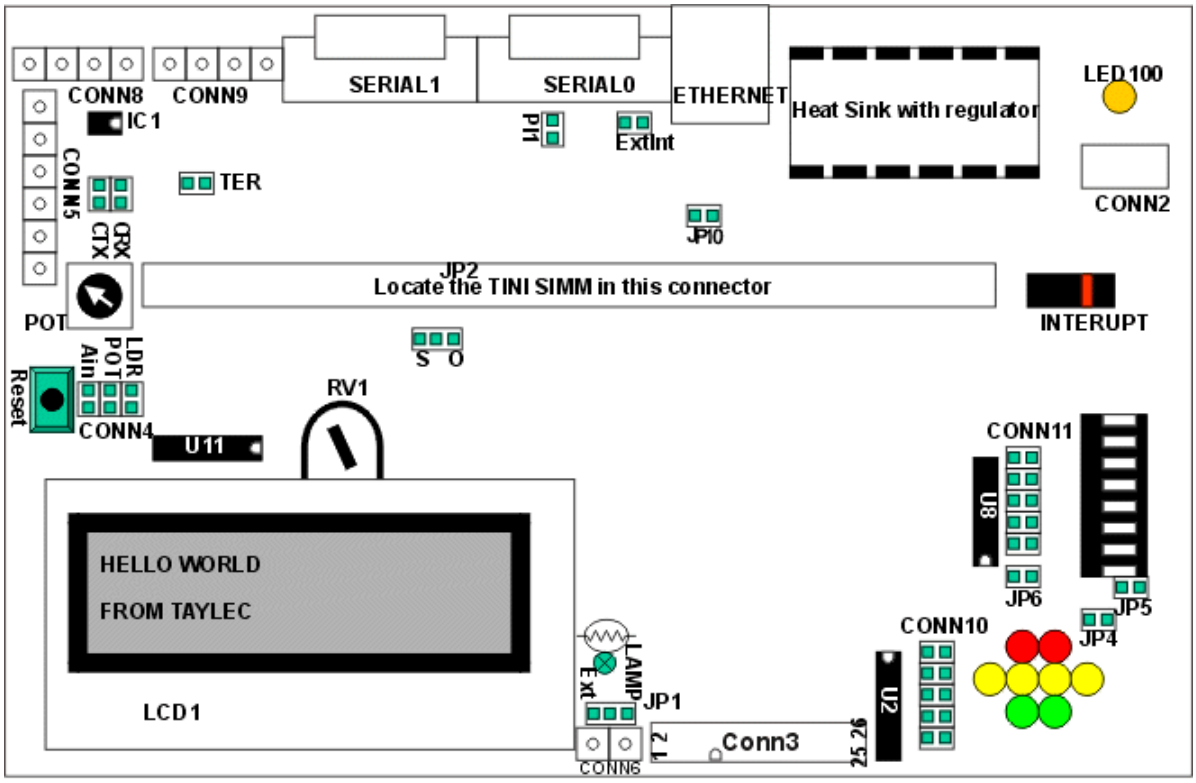
Remarque : le chargement de l'environnement TINI est décrit dans le manuel d'installation

LA CARTE TAYLEC TUTORIO

Cette carte d'expérimentation offre un environnement d'accueil à la carte TINI (JP2) permettant le développement d'applications pour ce système embarqué. Elle fournit notamment un connecteur SIMM pour insérer la carte TINI et des connecteurs d'extension : DB9 femelle (DCE) et mâle (DTE) pour la liaison série, RJ45 pour la liaison Ethernet ainsi que des connecteurs pour les bus CAN, One-Wire et I2C.

Comme souvent, cette carte d'expérimentation offre quelques fonctionnalités supplémentaires :

- *un afficheur LCD 2 lignes*
- *une interface analogique avec un DAC 8 bits et un ADC 8 bits, un potentiomètre, une lampe et une LDR* (Light-Dependent Resistor)
- *une interface numérique 24 bits E/S (*82C55A PIA*), 8 DELs et 8 micro-interrupteurs*
- *etc ...*



Description : (voir la documentation Taylec_ MB-TutorIO_ Board.pdf)

CONN5	*Tension d'alimentation non régulée (avec GND). De 7,5 v à 15 volts. Entrée non protégée contre les inversions de tension.* *GND : Masse* *OwIO : Bus 1 Fil (One Wire)* *OwR : Retour du bus 1 fil* *Vcc : + 5V DC* *Ain : Analog Input : entrée du convertisseur ADC. De 0 à 5 volts (voir connecteur CONN4)*
CONN6	*Sortie du convertisseur DAC. La tension de sortie varie de 0 à 5 volts. Celle-ci se retrouve en sortie si JP1 est connecté entre le centre et EXT.*
CONN8	*Connecteur pour le bus I2C (SDA, SCL, Vcc, GND).* *Pour utiliser ce bus, voir les états des cavaliers CRX, CTX et TER.*
CONN9	*Connecteur pour le bus CAN (CANH, CANL, Vcc, GND)* *Pour utiliser ce bus, voir les états des cavaliers CRX, CTX et TER.*
CRX *CTX*	*Ces cavaliers sont utilisés pour utiliser soit le bus CAN, soit le bus I2C, sachant qu'ils ne peuvent être utilisés en même temps. Pour utiliser le bus CAN, il suffit de mettre en place ces cavaliers. Pour le bus I2C, ils ne doivent pas être en place.*
JP1	*Ce cavalier est utiliser pour soit envoyer la sortie de CNA vers un circuit extérieur, soit vers une lampe de test.*
JP2	*Ce cavalier sélectionne soit le bus « One Wire », soit la liaison série SERIAL1.*
JP4	*Ce cavalier est utilisé pour sélectionner les LEDs ou non.*
JP5	*Ce cavalier est utilisé pour sélectionner les SWITCH ou non.*
JP6	*Ce cavalier est utilisé pour sélectionner les résistances de Pull-up ou non.*
PL1	*Ce cavalier est utilisé pour déconnecter la ligne DTR de la carte TINI. Si le DTR est connecté, la carte TINI provoque un RESET. Ainsi, pour communiquer avec la carte avec un autre équipement, il est recommandé d'enlever ce cavalier.*
RV1	*Potentiomètre de contraste de l'afficheur LCD.*
SERIAL0	*Port Série utilisé pour télécharger l'OS dans la carte à l'aide de l'application « JavaKit ». Il est à noter que l'entrée DTR sur ce port série a une utilisation bien spéciale ; s'il est utilisé et forcé à « 0 », ceci provoque un RESET de la carte. L'application « JavaKit » utilise le DTR pour provoquer un RESET de la carte avant de télécharger l'OS et après ce téléchargement. Si PL1 est absent, il n'est pas possible de télécharger d'OS.* *TX : pin 2, RX : pin 3, DTR : pin 4, GND : pin 5*
SERIAL1	*Second Port série ; Contient les signaux de contrôle (DTR, CTS, RTS, DCD).* *DCD : pin 1, TX : pin 2, RX : pin 3, DTR : pin 4, GND : pin 5 RTS : pin 7 CTS : pin 8*
JP10	*Connecté à IOS1 sur la carte TINI*
TER	*Ce cavalier est utilisé pour la terminaison du bus CAN. Si la technologie du bus est « filaire », ce cavalier, doit être en place. Sinon, il doit être enlevé pour prévenir d'une mauvaise terminaison.*