heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ReDS
Reconfigurable & embedded
Digital Systems

libmarklin

Kevin Georgy et Daniel Molla

Département TIC
Institut REDS
Version du 03 mars 2012

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale

Révisions

Date	Version	Ingénieur	Révision
17/01/09	v1.0	KGY	Version initiale
26/03/09	v1.1	KGY	Adaptation pour version paquet Debian
03/03/12	v1.2	DMO	Adapation pour la distribution 11.10

Table 1 – Révisions

Mise en forme

- Les noms propres sont écrits en Petites Capitales
- Les fichiers, dossiers ou commandes sont en `chasse fixe`
- Les termes étrangers, les nouveaux termes ou les termes techniques sont en *emphasis*
- Le *listing* de code prend la forme suivante :
  - Pour des commandes ou un extrait de code source :

```
./configure
make -j8
make install
```

  - Pour du code sources :

```
1 #include <stdio.h>
2
3 int main(int argc, char ** argv)
4 {
5     print("Un exemple...\n");
6     return 0;
7 }
```

Table des matières

1 Introduction

libmarklin est une bibliothèque qui fournit les accès aux maquettes Märklin utilisées dans le cadre des laboratoires de programmation concurrente de la Heig-vd. Elle permet de contrôler les locomotives et les aiguillages, en utilisant le port série, et de recevoir les activations des contacts par le port parallèle.

Installation 2

2.1 Matériel

La figure 2.1 illustre les connexions à réaliser avec la maquette.

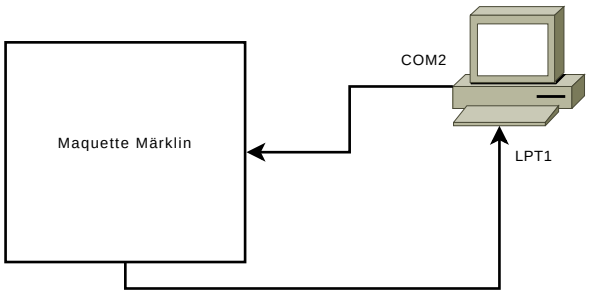

FIGURE 2.1 – Branchement

2.2 Logiciel

La bibliothèque est uniquement disponible pour la distribution GNU/LINUX UBUNTU 8.10. Cette distrbution a été la première utilisée pour le laboratoire de programmation concurente. Elle fonctionne également sur d'autres distributions, dont nottamenent la distribution GNU/LINUX UBUNTU 11.10 actuellement utilisée. Un paquet d'installation est fournit (`libmarklin_x.x.x-0ubuntux_i386.deb`). Il suffit d'ouvrir ce dernier avec GDEBI pour procéder à son installation/mise-à-jour.

Pour que les utilisateurs puissent avoir accès au port LPT, utilisé pour la reception des contacts, il faut qu'ils soient ajoutés au groupe "lp". Pour ce faire, utiliser la commande suivante en spécifiant <user_name> par le nom d'utilisateur devant utiliser la librairie :

```
1 sudo adduser <user_name> lp
```

Il est également nécessaire de donner aux utilisateurs l'accès au port série (COM) utilisée pour l'envoie de commandes aux aiguillages et locomotives. Pour ce faire, il faut récupérer le nom du groupe représantant le port série utilisé. La commande suivante permet ensuite d'ajouter les droits nécessaires en spécifiant <user_name> par le nom d'utilisateur et <nom_group_port> par nom du groupe représantant le port série utilisé (généralement `dialout`) :

```
1 sudo adduser <user_name> <nom_group_port>
```

La librairie est désormais prête à l'emploi.

Configuration 3

Il est possible de modifier certains paramètres utilisés par la librairie après son installation. Le fichier de configuration (*/etc/libmarklin/markinl.conf*) a la forme suivante :

```
#Comport device, used to send railroad commands
comport = /dev/ttyS1

# Lptport device, used to get contacts activation
lptport = /dev/parport0

# Minimum time between two command sends
cmd_tempo = 50000

# Time between speed increase or decrease when setting speed step by step
vit_prog_delay = 0

# Minimum time of switch power supply
min_aig_alim = 500000
```

Listing 3.1 – Fichier de configuration.

Le premier paramètre (`comport`) défini le fichier *device* du port COM à utiliser (*/dev/ttyS0 = COM1*, */dev/ttyS1 = COM2*, ...). Le second (`lptport`) défini le fichier *device* du port LPT à utiliser (*/dev/parport0 = LPT1*, */dev/parport0 = LPT2*, ...). Les trois derniers paramètres permettent de règler différents temps d'attente :

- `cmd_tempo` est le temps minimum à attendre entre l'envoi de deux commandes.
- `vit_prog_delay` définit le temps entre chaque incrémentation ou décrémentation de vitesse lors d'appels à `mettre_vitesse_progressive()`.
- `min_aig_alim` est le temps d'alimentation minimum des aiguillages.

4 Utilisation avec QtCreator

Cette documentation se base sur l'environnement Qt SDK 4.7.3.

4.1 Récupération du projet

Il faut récupérer le projet exemple Qt disponible ou le projet actuel de l'utilisateur. Une fois récupéré, il est possible de lancer QtCreator à partir du fichier `QtrainSim.pro` du projet ou alors d'ouvrir le projet à partir de QtCreator.

4.2 Passage en mode maquette

Une fois le projet récupéré, il faut éditer le fichier `QtrainSim.pro` du projet et décommenter la ligne suivante :

```
1 CONFIG += MAQUETTE
```

Le simulateur est alors prêt à être utilisé en mode maquette. Avant tout lancement de l'application, il est préférable de contrôler que les numéros de maquette, de locomotives ainsi que l'emplacement initiales de celles correspondent avec la maquette et aux locomotives utilisés.

4.3 Interaction utilisateur

Une fois l'application lancée en mode maquette, l'utilisateur a la possibilité de cliquer sur les aiguillages de l'interface graphiques afin de changer en temps réel leur direction. Il va sans dire qu'une telle possibilité doit être utlilisée avec prudence afin de ne pas endomager le materiel mis à diosposition (locomotives, rails et aiguillages).

L'utilisateur dispose également d'un bouton d'arrêt d'urgence. La figure4.1présente l'emplacement de ce bouton.

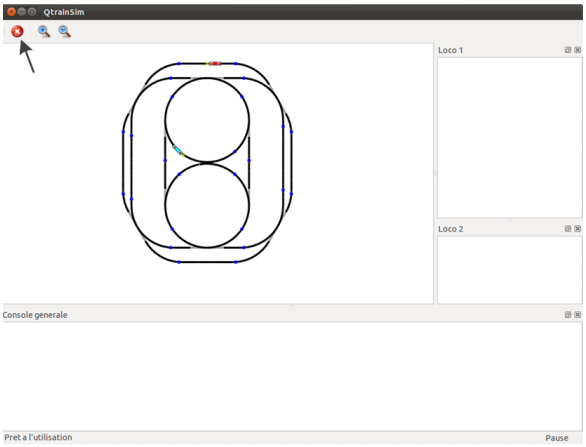

FIGURE 4.1 – Bouton d'arrêt d'urgence

L'implentation de l'action à réaliser lors de la pression du bouton revient à l'utilisateur au sein de la méthode *emergency_stop()* en fonction du fonctionnent de son application.

A relever que les locomotives de l'application graphique ne se déplace pas durant l'exécution en mode maquette et que la gestion des *logs* pour les locomotives est à gérer par l'utilisateur.

4.4 Écriture d'un programme

Lors de l'utilisation du simulateur en mode maquette, les fonctions fournies par la librairie définie dans `ctrain_handler.h` (*listing* 4.1) sont utilisées. A noter que les fonctions *selection_maquette* et *assigner_loco* ne sont pas présentes dans la librairie de la maquette et sont récupérée à partir de la librairies du simulateur.

```
1  /**  \file ctrain_handler.h
2   *
3   * Fournit les fonctions de controle du simulateur/maquette
4   * de trains. Mappage en C des fonctions fournies par
5   * "train_handler.ads".
6   *
7   * \author Kevin Georgy
8   * \date 9.2.2009
9   */
10
11 #ifndef H_CTRAIN_HANDLER
12
13 #define H_CTRAIN_HANDLER
14
15 /** Vitesse Ã  l'arret */
16 #define VITESSE_NULLE (0)
17
18 /** Vitesse minimum */
19 #define VITESSE_MINIMUM (3)
20
21 /** Vitesse maximum */
22 #define VITESSE_MAXIMUM (14)
23
24 /** Numero max. d'aiguillage */
25 #define MIN_AIGUILLAGES (1)
26
27 /** Numero max. d'aiguillage */
28 #define MAX_AIGUILLAGES (80)
29
30 /** Numero max. de contact */
31 #define MAX_CONTACTS (64)
32
33 /** Numero min. de loco */
34 #define MIN_LOCOS (1)
35
36 /** Numero max. de loco */
37 #define MAX_LOCOS (80)
38
39 /** Direction des aiguillages DEVIE*/
40 #define DEVIE (0)
41
```

```
42 /** Direction des aiguillages TOUT_DROIT*/
43 #define TOUT_DROIT (1)
44
45 /** Etat des phares eteints*/
46 #define ETEINT (0)
47
48 /** Etat des phares allumes*/
49 #define ALLUME (1)
50
51 /**
52  * Initialise la communication avec la maquette/simulateur.
53  * A appeler en debut de programme client.
54  */
55 void init_maquette(void);
56
57
58 /**
59  * Met fin a la simulation. A appeler en fin de programme client
60  */
61 void mettre_maquette_hors_service(void);
62
63 /**
64  * Realimente la maquette. Inutil apres init_maquette
65  */
66 void mettre_maquette_en_service(void);
67
68 /**
69  * Change la direction d'un aiguillage.
70  * \param[in] no_aiguillage No de l'aiguillage a diriger.
71  * \param[in] direction Nouvelle direction. (DEVIE ou TOUT_DROIT)
72  * \param[in] temps_alim Temps l'alimentation minimal du bobinage de l'aiguillage.
73  */
74 void diriger_aiguillage(int no_aiguillage, int direction, int temps_alim);
75
76 /**
77  * Attend l'activation du contact donne.
78  * \param[in] no_contact No du contact dont on attend l'activation.
79  */
80 void attendre_contact(int no_contact);
81
82 /**
83  * Attend l'activation d'un contact.
84  */
85 int attendre_contact_x(void);
86
87 /**
88  * Arrete une locomotive (met sa vitesse a VITESSE_NULLE).
89  * \param[in] no_loco No de la loco a stopper.
90  */
91 void arreter_loco(int no_loco);
92
93 /**
94  * Change la vitesse d'une loco par palier.
95  * \param[in] no_loco No de la loco a stopper.
96  * \param[in] vitesse_future Vitesse apres changement.
97  *
98  * \attention Dans le simulateur cette procedure agit comme la
99  * fonction "Mettre_Vitesse_Loco". c'est-a-dire que
100 * l'acceleration est immediate( de la vitesse actuelle
101 * a la vitesse specifiee )
102 */
103 void mettre_vitesse_progressive(int no_loco, int vitesse_future);
104
105 /**
106  * Permettre d'allumer ou d'eteindre les phares de la locomotive.
107  * \param[in] no_loco No de la loco a controler.
108  * \param[in] etat Nouvel etat des phares. (ETEINT ou ALLUME)
109  *
110  * \attention Dans le simulateur cette fonction n'a aucun effet.
111  * Les locomotive representee par des rectangles
112  * possedent une partie jaune indiquant le sens de
113  * deplacement. L'utilisation des phares n'est donc
114  * plus utile.
115  */
116 void mettre_fonction_loco(int no_loco, char etat);
117
118 /**
119  * Inverse le sens d'une locomotive, en conservant sa vitesse.
120  * \param[in] no_loco No de la loco a inverser.
121  */
122 void inverser_sens_loco(int no_loco);
123
124 /**
125  * Change la vitesse d'une loco.
126  * \param[in] no_loco No de la loco a controler.
127  * \param[in] vitesse Nouvelle vitesse.
128  */
129 void mettre_vitesse_loco(int no_loco, int vitesse);
```

```
130
131 /**
132  * Indique au simulateur de demander une loco Ã  l'utilisateur. L'utilisateur
133  * entre le numero et la vitesse de la loco. Celle—ci est ensuite placee entre
134  * les contacts "contact_a" et "contact_b".
135  * \param[in] contact_a Contact vers lequel la loco va se diriger.
136  * \param[in] contact_b Contact Ã  l'arriere de la loco.
137  * \param[out] no_loco Numero de loco choisi par l'utilisateur.
138  * \param[out] vitesse Vitesse choisie par l'utilisateur.
139  */
140 void demander_loco(int contact_a, int contact_b, int *no_loco, int *vitesse);
141
142 #endif
```

Listing 4.1 – ctrain_handler.h

Le *listing* 4.2 montre un fichier source minimal pour l'utilisation de la librairie. On remarque l'inclusion <*ctrain_handler.h*> qui fournit l'accès au fonctions. L'appel à *init_maquette()* et *mettre_maquette_hors_service()* sont à appeler obligatoirement en début et fin de programme.

```
1  #include <pthread.h>
2  #include "ctrain_handler.h"
3  #include <errno.h>
4
5
6  // structure qui definit une locomotive
7  typedef struct
8  {
9    int no;
10   int vitesse;
11 } Locomotive;
12
13
14 // Declaration des deux locomotives
15 Locomotive loco1;
16
17
18 void emergency_stop()
19 {
20   printf("\nSTOP!");
21
22   // on arrete les locomotives.
23   arreter_loco(loco1.no);
24 }
25
26
27 // Contacts a parcourir
28 #define NB_CTS 7
29 int parcours[] = {6, 11, 10, 13, 14, 19, 3};
30
31
32 void cmain()
33 {
34   int ct;
35
36   // Initialisation de la maquette (a effectuer une seule fois en debut de programme)
37   init_maquette();
38
39   loco1.no =1;
40   loco1.vitesse = 12;
41
42   selection_maquette("MAQUET_B");
43
44   // Demande au simulateur de placer une loco entre les contacts 6 et 11
45   // Recupere le numero et la vitesse saisis par l'utilisateur.
46   assigner_loco(  parcours[1],
47                   parcours[0],
48                   loco1.no,
49                   loco1.vitesse);
50
51   // Dirige les aiguillages sur le parcours
52   diriger_aiguillage(7,TOUT_DROIT,0);
53   diriger_aiguillage(8,DEVIE,0);
54   diriger_aiguillage(5,TOUT_DROIT,0);
55   diriger_aiguillage(9,DEVIE,0);
56   diriger_aiguillage(10,TOUT_DROIT,0);
57   diriger_aiguillage(14,TOUT_DROIT,0);
58   diriger_aiguillage(13,DEVIE,0);
59   diriger_aiguillage(1,TOUT_DROIT,0);
60
61
62   // Demarre la loco
63   mettre_vitesse_progressive(loco1.no, loco1.vitesse);
64
65   // Attend que la loco passe sur les differents contacts de son parcours.
66   for (ct = 1; ct < NB_CTS; ct++) {
```

```
67      attendre_contact(parcours[ct]);
68      printf("Loco %d de vitesse %d a atteint le contact %d.\n", loco1.no, loco1.vitesse, ct);
69    }
70
71    // Stoppe la loco
72    arreter_loco(loco1.no);
73
74    // Fin de la simulation (a effectuer une seule fois en fin de programme, sans effet
75    // sur le simulateur, mais necessaire sur les maquettes reelles).
76    mettre_maquette_hors_service();
77  }
```

Listing 4.2 – Fichier source minimal