Informatique temps-réel – Projet d'implémentation

Gauvain Devillez

Ce projet consiste en l'implémentation du logiciel de contrôle d'une ligne d'assemblage de voitures. Il s'agira bien sûr d'une simulation. Il sera à réaliser par **groupes de trois étudiants**.

Vous trouverez sur Moodle les fichiers `assembly.c` et `assembly.h` qui implémentent le code de base à utiliser pour votre projet. Le premier contient l'implémentation et vous n'avez pas besoin de le lire. Le second contient les fonctions à utiliser ainsi que les constantes utiles.

Ce document présente les fonctionnalités à implémenter et décrit le fonctionnement du code de base disponible sur Moodle.

1 La ligne d'assemblage

La ligne d'assemblage est composée d'un tapis roulant et de bras robot. Une voiture sera déplacée sur le tapis roulant de position en position. Des deux côtés du tapis roulant se trouvent des bras robots. Chacun pourra être configuré pour installer une partie de la voiture ou rester non-configuré et ne rien faire. À la fin du tapis roulant, si tout s'est bien passé, toutes les pièces de la voiture devraient avoir été installées (voir Section 1.2 pour des explications sur les pièces).

Par exemple, dans la Figure 1, on a six bras robots. Les deux bras en position 1 sont configurés pour installer le chassis et les roues. Le bras de gauche en position 2 est configuré pour installer les fenêtres[1]. Les autres bras ne sont pas configurés.

1.1 L'assemblage

L'assemblage d'une voiture est découpée en plusieurs étapes: l'initialisation, l'assemblage proprement dit et la vérification. Ces trois étapes se répètent en boucle.

Lors de l'initialisation, une voiture "vide" est placée en position zéro. Ensuite, pendant l'assemblage, le tapis roulant va avancer la voiture d'une position à la fois. Chaque fois que la voiture se trouve à une position donnée, les bras robots à la même position pourront effectuer l'installation de leur pièce. Si la voiture ne se trouve pas à la même position que le bras, l'installation ne se fera pas.

Une fois la dernière position atteinte (juste après le dernier bras qui a été configuré), une dernière étape sera la vérification. On va vérifier si toutes les pièces nécessaires ont été installées et afficher un message avec le résultat. Après cette étape, on recommence la procédure à l'initialisation.

Entre chaque déplacement du tapis roulant, celui-ci attendra un temps donné (`BELT_PERIOD`). Cela inclu le moment après l'initialisation (position 0) et après la vérification. Ici, nous supposons qu'une seule voiture est assemblée à la fois.

Dans l'exemple, la dernière position est la position 3 car plus aucun bras robot n'est configuré à partir de cette position. Imaginez la dernière position comme la position à partir de laquelle plus aucune pièce ne sera installée.

[1] Notez que les dépendances entre les pièces ne sont pas respectées.

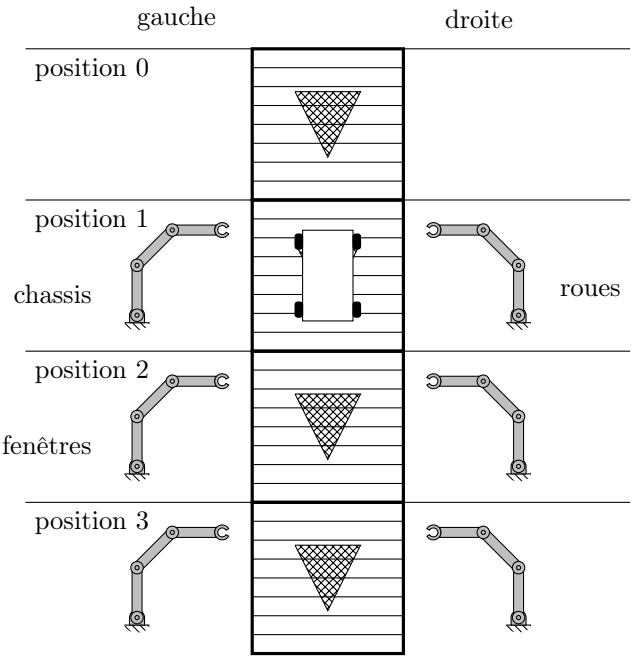

Figure 1: Exemple de ligne d'assemblage

Un déroulement type pour cet exemple serait le suivant:

- position 0: initialisation de la voiture
- position 1: installation du chassis et des roues
- position 2: installation des fenêtes
- position 3: vérification
- retour à la position 0

Entre chaque point, un délai de `BELT_PERIOD` millisecondes sera ajouté.

1.2 Les pièces de la voiture

Pour que la voiture soit correctement assemblée, chaque pièce devra être installée par un bras robot. Mais toutes les pièces ne peuvent pas être installées en même temps.

La Table 1 reprends la liste des pièces ainsi que les constantes qui y sont associées dans le code. La dernière colonne contient les dépendances de chaque pièce. Par exemple, la carrosserie est représentée par la constante `PART_BODY` et ne peut être installée qu'après que le moteur ait été installé.

2 Le projet

Afin de pouvoir implémenter le système de contrôle, vous devrez implémenter un programme C avec sa fonction `main` qui utilise le code donné sur Moodle. Votre code principal se trouvera dans un fichier `main.c` mais pourra inclure d'autres fichiers. Ce code devra être accompagné d'un petit rapport au format PDF.

Le rapport doit expliquer votre raisonnement et vos choix d'implémentation. Il permet d'expliquer les détails utiles que le code ne peut pas transmettre. Par exemple, comment vous avez décidé de gérer les blocages et

Pièce	Constante	Dépendances
Chassis	`PART_FRAME`	$\emptyset$
Moteur	`PART_ENGINE`	`PART_FRAME`
Roues	`PART_WHEELS`	`PART_FRAME`
Carrosserie	`PART_BODY`	`PART_ENGINE`
Portières	`PART_DOORS`	`PART_BODY`
Fenêtres	`PART_WINDOWS`	`PART_DOORS`
Phares	`PART_LIGHTS`	`PART_BODY`

Table 1: Pièces de la voiture

pourquoi vous avez choisi cette solution. Le rapport est un supplément au code et ne doit pas nécessairement être long. Vous devrez indiquer dans le rapport votre numéro de groupe et les noms des membres du groupe.

Pour ce projet, il est autorisé et même encouragé de réutiliser du code des séances d'exercice précédentes ainsi que les notions qui y ont été présentées.

Tous les fichiers nécessaires devront être remis dans une archive au format .zip sur Moodle au plus tard le mercredi 9 avril 2025.

Les sections ci-dessous présentent les différents éléments à implémenter.

2.1 Configuration des bras robots

Par défaut, les bras robots n'installent rien. Votre code devra commencer par configurer les bras robots via la fonction `setup_arm` qui prend en paramètre la ligne d'assemblage, une pièce de voiture, un côté (`LEFT` ou `RIGHT`) et une position (entre 1 et `MAX_POSITION` inclus).

La configuration des bras robots devra être compatible avec les dépendences des différentes pièces. Par exemple, un bras en position 4 ne pourra pas installer sa pièce avant un bras en position 2.

Tous les bras robots ne doivent pas nécessairement être configurés. Il y a plus de positions (`MAX_POSITION` * 2) que de pièces.

2.2 Installation des pièces

L'installation des pièces n'est pas gérée par le code de base. Vous devrez donc créer des tâches qui s'occuperont de cela.

Pour ce faire, il vous est demandé de créer un thread par pièce à installer. Ce thread va, de façon périodique, appeler la fonction `trigger_arm` en lui donnant sa position et son côté. Pour que l'installation se fasse, il faudra que toutes les dépendances pour la pièce soient satisfaites mais aussi que la voiture soit à la bonne position. Pour que le bras robot en position 3 puisse installer sa pièce sur la voiture, celle-ci doit se trouver en position 3.

Vous devrez vous-même décider des délais de tâches et des positions à associer à chaque pièce. La voiture commence toujours en position zéro et avancera d'une position toutes les `BELT_PERIOD` millisecondes. N'hésitez pas à faire un schéma si besoin.

2.3 Blocages de la ligne

De façon aléatoire, un bras robot peut rester bloqué après l'installation. Dans ce cas, le tapis roulant ne pourra plus avancer et les autres bras robots pourraient également être bloqués.

En pratique, les threads seront bloqués par un sémaphore. Il ne sera donc pas possible de le détecter depuis les threads eux-mêmes[2].

Dans le cas d'un blocage, il faudra alors appeler la fonction `shutdown_assembly` qui permettra de débloquer les threads. Quand cette fonction se termine, le tapis roulant est remis à la position zéro et la ligne d'assemblage est arretée. Les tâches des différents bras robots ne seront pas stoppée par cette fonction et vous devrez le gérer vous-même.

Une fois la ligne d'assemblage à l'arrêt, la fonction `run_assembly` pourra être appelée pour la relancer.

2.4 Arrêt propre

Lorsque le signal `SIGINT` est reçu, (`CTRL+C`), le programme doit s'arrêter et libérer ses ressources. Il faudra donc appeler la fonction `shutdown_assembly` et chaque thread devra afficher un message avant de se terminer, une fois ses ressources libérées.

2.5 Affichage des statistiques

La fonction `print_assembly_stats` permet d'afficher les statistiques de la ligne d'assemblage. Celle-ci devra être appelée lorsque le signal `SIGUSR1` est reçu.

Attention, les autres threads ne devraient pas être retardés par la gestion du signal. Nous avons vu comment éviter cela lors des exercices.

2.6 Évaluation

Vous serez évalués sur le fonctionnement du code, sa qualité et sa résistance aux erreurs ainsi que la qualité des solutions apportées.

Dans une situation "normale" (c'est à dire sans blocage ou interruption), chaque tâche devra se déclencher de façon synchrone avec le tapis roulant et les pièces devront être installées dans un ordre compatible avec les dépendances.

Si un blocage survient, la ligne d'assemblage devra correctement être stoppée et relancée et l'installation des pièces devra reprendre et continuer à fonctionner comme avant (une fois le blocage résolu).

Enfin, la gestion des signaux devra être réalisée de manière correcte et avoir le comportement demandé. Vous devez éviter les problèmes de concurrence mais la réentrance des gestionnaires de signaux n'est pas nécessaire.

Les fichiers `assembly.c` et `assembly.h` ne peuvent pas être modifés à l'exception des constantes définies dans `assembly.h`. Les changements dans ces constantes devront être documentés et justifiés dans le rapport. Notez que votre code devra pouvoir fonctionner même si ces constantes sont modifiées de façon raisonable (par exemple, augmenter ou réduire légèrement le délai du tapis roulant). Assurez-vous donc d'éviter des valeurs "en dur" dans votre code.

Il devra être possible de compiler et d'exécuter votre code sans le modifier.

3 Le code de base

Cette section détaille le code fourni sur Moodle. Pour l'utiliser, vous devrez inclure le fichier `assembly.h` dans votre code avec l'instruction `#include` et ajouter `assembly.c` aux fichiers à compiler.

[2] C'est un problème qui a été rencontré dans les exercices.

3.1 Ce qu'il fait

Le code fourni sur Moodle implémente la gestion du tapis roulant, de la vérification de l'assemblage et garde des statistiques sur les assemblages réussis et échoués. Certains messages en console seront également affichés pour voir ce qui se passe.

Il contient également des sémaphores pour éviter les problèmes de concurrences entre les threads. Les fonctions `trigger_arm`, `print_assembly_stats` et `shutdown_assembly` ne devraient pas causer de problème en cas d'accès concurrent.

Les fonctions permettant la configuration des brats robots et leur déclenchement sont présentes et fonctionnent mais ne sont pas appelées.

3.2 Ce qu'il ne fait pas

Le code de base ne déclenche pas les bras robots pour effectuer l'assemblage et ne les configure pas non plus automatiquement. Ce sera à vous de les configurer et de les déclencher quand la position de la voiture sur le tapis roulant le permet.

La gestion des signaux devra être réalisée par votre code et vous devrez également vous assurer de stopper la ligne d'assemblage en cas de blocage.

3.3 Fichier d'entête

Le fichier d'entête fourni sur Moodle contient les constantes suivantes:

- `NUM_PARTS`: Nombre de pièces de voiture. Utile pour créer un tableau. Attention, une pièce spéciale `PART_EMPTY` est présente. Celle-ci représente un bras robot qui n'a pas été configuré.
- `MAX_POSITION`: Nombre maximum (inclusif) de positions sur le tapis roulant de chaque côté. Il y a uniquement des bras robots entre les positions 1 et `MAX_POSITION`.
- `BELT_PERIOD`: Période du tapis roulant. Il avance d'une position toutes les `BELT_PERIOD` millisecondes.
- `MIN_DELAY`: Délai minimum pour une installation. Un appel à `perform_assembly` prendra au moins `MIN_DELAY` millisecondes.
- `MAX_DELAY`: Délai maximum pour une installation. Un appel à `perform_assembly` prendra au plus `MAX_DELAY` millisecondes.
- `ONE_IN_BLOCK_CHANCE`: Probabilité de blocage d'une installation. Un appel à `perform_assembly` aura 1/`ONE_IN_BLOCK_CHANCE` de chance de se bloquer.

Pendant le développement, vous pouvez modifier ces constantes (sauf les deux premières) pour tester différentes configurations. Par exemple, augmenter `ONE_IN_BLOCK_CHANCE` permet de diminuer la probabilité d'un blocage tant qu'ils ne sont pas gérés.

Pour s'assurer de la robustesse de votre code dans des configurations différentes, utilisez ces constantes dans votre code plutôt que des valeurs explicites. Par exemple, configurez les délais de vos bras robots en fonction de `BELT_PERIOD`.

Plusieurs types de données sont définies. Tout d'abord, le type `error_t` correspond à une énumération qui sera retournée par les fonctions du code de base. Chaque valeur possible correspond à un nombre entier. L'avantage de cette approche est que l'on peut facilement s'assurer que seules des valeurs valides soient retournées. Le type `part_t` correspond à une énumération qui encode chaque pièce de la voiture ainsi que la pièce spéciale `PART_EMPTY`. Enfin, le type `side_t` correspond à une énumération qui encode le côté gauche ou droit de la ligne d'assemblage.

Le type principal est `assembly_line_t` qui correspond à un pointeur vers une structure contenant les informations sur la ligne d'assemblage. Comme le contenu de la structure n'est pas important dans ce contexte, on utilise un pointeur afin de "masquer" le contenu de la structure.

Pour intéragir avec la ligne d'assemblage, vous disposerez des fonctions suivantes:

`init_assembly_line` Initialise la ligne d'assemblage. La fonction prend en paramètre le pointeur vers la ligne d'assemblage à initialiser.

`free_assembly_line` Libère la mémoire allouée par la ligne d'assemblage. La fonction prend en paramètre le pointeur vers la ligne d'assemblage à libérer.

`setup_arm` Configure un bras robot pour qu'il installe une pièce de voiture. La fonction prend en paramètre la ligne d'assemblage, la pièce de la voiture, le côté (gauche ou droit) et la position (entre 1 et `MAX_POSITION`).

`run_assembly` Lance la ligne d'assemblage. La fonction prend en paramètre la ligne d'assemblage.

`trigger_arm` Tente d'installer une pièce de la voiture. La fonction prend en paramètre la ligne d'assemblage, le côté (gauche ou droit) et la position (entre 1 et `MAX_POSITION`).

`shutdown_assembly` Arrête la ligne d'assemblage. La fonction prend en paramètre la ligne d'assemblage.

`print_assembly_stats` Affiche les statistiques de la ligne d'assemblage. La fonction prend en paramètre la ligne d'assemblage.

3.4 Compatibilité

Les librairies utilisées sont les mêmes que celles vues en séances d'exercice. A priori, vous ne devriez pas rencontrer de problème à part les deux exceptions présentées ci-dessous.

Si vous trouvez un problème ou un bug dans le code de base, merci de me prévenir rapidement.

3.4.1 Fonction `clock_nanosleep` introuvable (macOS)

Lors des exercices, il s'est révélé que la fonction `clock_nanosleep` n'était pas disponible sur macOS. Pour remédier à cela, le code possède deux implémentations. L'une utilisant `clock_nanosleep` et l'autre utilisant seulement `nanosleep`.

Cette dernière peut être activée en ajoutant décommentant la ligne `#define MACOS_SLEEP` dans le fichier `assembly.h`. La seconde implémentation sera alors utilisée. La fonction `nanosleep` est un peu moins précise que `clock_nanosleep` mais devrait suffire pour les besoins de ce projet.

3.4.2 Affichage en couleur dans la console

Pour rendre l'affichage plus facile à lire, certains messages sont affichés en couleur dans la console. Les fonctions utilisées sont `printf_green` et `printf_red`. Elles s'utilisent exactement comme `printf`[3].

Cette fonctionnalité n'est pas toujours supportée sur tous les terminaux et vous pourriez donc avoir des symboles bizarres dans le terminal. Si c'est le cas, commentez la ligne `#define COLOR_PRINT` dans le fichier `assembly.h`.

[3]Ces fonctions se contentent d'afficher des séquences d'échappement ANSI avant et après un appel à `printf` (voir `https://en.wikipedia.org/wiki/ANSI\_escape\_code\#In\_C`).