Informatique temps-réel – Projet

Gauvain Devillez

1 Introduction

Pour ce projet, il vous est demandé, **par groupe de 2 étudiants**, de développer un système automatique de protection contre les menaces pour un éleveur de poules. Ces poules sont élevées dans une prairie et sont libre d'aller et venir. Cependant, un renard et un aigle tentent fréquemment de voler des poules. Il faut donc un système capable de détecter leur présence et de les chasser.

Ce projet demandera deux étapes distinctes:

1. Une analyse sur papier pour obtenir un **ordonnancement off-line** des tâches.
2. Une implémentation respectant cet ordonnancement.

Les détails pour l'analyse d'ordonnancement se trouvent dans la section 2. Une fois votre ordonnancement prêt, les détails sur le code qui vous est fourni et les contraintes pour l'implémentation se trouvent dans la section 3. Les modalités de remise et les critères d'évaluation sont détaillés dans la section 4.

2 Ordonnancement

Pour ce projet, les tâches utiliseront un ordonnancement off-line. Vous devrez donc d'abord réaliser l'ordonnancement à la main et insérer des délais (`clock_nanosleep`) pour respecter l'ordonnancement. Attention à bien tenir compte des différentes situations.

Le système sera implémenté à l'aide de deux tâches principales, chacune représentée par un thread qui s'exécute en boucle infinie:

- **La tâche renard**: dédiée à la détection et la chasse du renard.
- **La tâche aigle**: dédiée à la détection et la chasse de l'aigle.

Attention, le renard et l'aigle sont indépendants de ces tâches et leur comportement est déjà implémenté.

Ces deux menaces (renard et aigle) se comportent de la façon suivante: ils choisissent une direction depuis laquelle ils vont tenter de voler une poule et, après un temps donné, volent la poule. Il s'agit donc d'un comportement périodique avec une seule attaque par période pour chaque menace.

Pour les stopper, les deux tâches devront utiliser un senseur pour savoir dans quelle direction se trouve chaque menace et une alarme pour chasser la menace. Une direction spéciale est `away`, qui signifie que la menace ne chasse pas pour cette période.

Les durées Afin de vous aider dans l'analyse d'ordonnancement, nous définissons les durées suivantes pour les menaces:

- **Période du Renard**: `FOX_TIME`. Le renard attaque toutes les `FOX_TIME` millisecondes. Il choisit une direction (north, south, east ou away) et tente de voler une poule si la direction n'est pas `away`. L'alarme doit être déclenchée avant ce délai.

- **Période de l'Aigle**: `EAGLE_TIME`. L'aigle est deux fois plus rapide : `EAGLE_TIME` = `FOX_TIME`/2. Il choisit une direction (above ou away) et tente de voler une poule si la direction n'est pas `away`.

Les deux outils (le senseur et l'alarme) que vos tâches utiliseront sont également soumis à quelques contraintes. Leur utilisation est mutuellement exclusive, c'est-à-dire, on ne peut pas utiliser les deux en même temps, et ils sont mono-direction (une seule direction à la fois). Les deux opérations qui sont la détection (fonction `sense`) et l'alarme (`sound_alarm`) prennent un temps non-négligeable que nous définissons comme `STEP_TIME` = `FOX_TIME`/8.

Contraintes de l'ordonnancement Afin de pouvoir empêcher le vol de poules, votre ordonnancement off-line devra respecter les contraintes suivantes:

- Mono-processeur: on ne peut pas utiliser le senseur ou l'alarme en même temps ni dans plus d'une direction à la fois.
- Respect des périodes: Le travail des tâches doit se terminer avant sa prochaine période.
- Préemptif: Pour respecter les périodes, il est possible qu'une de vos tâches doive se mettre en pause pour laisser le temps à l'autre tâche de s'exécuter dans les temps.

Conseils Avec ces considérations, vous devrez réaliser un ordonnancement. Considérez des unités de temps égales à `STEP_TIME` et tentez de répondre aux questions ci-dessous. Notez bien que, dans le pire des cas, on peut devoir tester plusieurs directions avec le senseur afin de trouver le renard.

1. Quelles sont les périodes des deux tâches?
2. Quelles sont les quatre étapes de la tâche dédiée au renard dans le pire cas?
3. Quelles sont les deux étapes de la tâche dédiée à l'aigle dans le pire cas?
4. Quelle est le temps d'exécution de chaque tâche dans le pire des cas (pour une seule activation)?

Une fois que vous aurez répondu à ces questions, vous devriez avoir asez d'informations pour construire un ordonnacement de vos deux tâches qui respecte leurs périodes.

3 Implémentation

Votre implémentation devra reproduire l'ordonnancement de la section 2. Comme il s'agit d'un ordonnancement off-line, vous pourez obtenir un comportement similaire en insérant des délais (voir `delay_until` du TP 2). Le code fourni sur Moodle (section3.3) est prévu pour effectuer un appel à `exit` si le nombre de poules atteint zéro, causant la fin du programme.

Assurez-vous que vous respectez cet ordonnancement même si une tâche n'est pas dans son pire cas (par exemple, si on détecte le renard dés la première direction testée ou si on ne détecte pas l'aigle). Si votre ordonnancement est préemptif, découpez la tâche et ajoutez un délai entre les deux parties pour permettre à l'autre tâche de s'exécuter.

Vos deux tâches devront utiliser les fonctions `sense` et `sound_alarm` qui vont respectivement détecter une menace dans une direction et la chasser. En pratique, ces deux fonctions prendront un temps `STEP_TIME` − `JITTER` où `JITTER` est un petit délai (quelques millisecondes) qui permet de garder un peu de marge. Par exemple, pour éviter des délais introduits par un élément externe (un autre programme sur votre machine). Dans votre implémentation, si vous utilisez des délais absolus (`TIMER_ABSTIME`), cela ne change rien à votre code.

Les fonctions `sense` et `sound_alarm` utilisent déjà un mutex commun pour éviter la concurrence et ainsi forcer le charactère mono-processeur. Vous ne devez donc pas vous soucier de cela.

3.1 Remplacer les poules perdues

Une fois votre implémentation fonctionnelle, vous verrez que certaines étapes des tâches (senseur et alarme) n'ont pas toujours besoin d'être effectuées, ce qui laisse parfois un moment où le système ne fait rien.

Pour profiter de ces instants, ajoutez une troisième tâche qui se comportera comme un gestionnaire d'interruption différé. C'est à dire qu'elle restera en attente sur un sémaphore la plupart du temps et sera libérée par une de vos deux autres tâches lorsque c'est possible. Par exemple, s'il n'est pas nécessaire de chasser une des deux menaces, on a du temps pour cette troisième tâche.

Cette tâche devra, lorsqu'elle est libérée, vérifier combien de poules restent et, s'il en reste moins que le nombre initial, en ajouter une. Cette tâche utilisera la fonction `get_chickens` pour détecter combien de poules restent et `add_chicken` pour ajouter une.

Vous pouvez supposer que ces deux fonctions ont une durée très courte, bien inférieure à `STEP_TIME` et sont thread-safe grâce à un mutex (indépendant de celui de `sense` et `sound_alarm`).

Si votre implémentation est correcte, cette tâche ne devrait jamais avoir besoin de remplacer les poules perdues.

3.2 Gestion du signal SIGINT

Lorsque votre programme reçoit le signal `SIGINT` (Ctrl+c), il devra stopper les différentes tâches et libérer les ressources avant de se terminer.

3.3 Code fourni sur Moodle

Sur Moodle, vous trouverez deux fichiers: un fichier d'entête `chickens.h` et un fichier de code `chickens.c`. Le fichier `chickens.h` contient toutes les fonctions et types utiles à votre implémentation. Celui-ci est documenté et accompagné d'un fichier pdf produit par *doxygen*, un outil de documentation similaire à javadoc. Ce pdf vous permettra de chercher dans la documentation plus facilement.

Votre code devra inclure le fichier d'entête afin de disposer de toutes les fonctions et constantes nécessaires. Vous ne pouvez **pas modifier le fichier** `chickens.c`, ni copier ou inclure (avec une directive `include`) son code. Dans `chickens.h`, vous pouvez uniquement modifier les constantes (`define`) et, si besoin, ajouter de nouvelles erreurs dans l'enum `error`.

Dans le fichier `chickens.h`, vous verrez que toutes les structures ne sont pas définies. Par exemple, le type `sensors_t` est défini mais pas la structure sous-jacente `struct sensors`. Cette structure est définie dans `chickens.c`. C'est une technique d'encapsulation (type opaque) qui vous permet d'utiliser des pointeurs vers ces structures `sensors_t` sans pouvoir en modifier les attributs.

Attention, les fonctions `sense` et `sound_alarm` utilisent un mutex pour empêcher les utilisations concurrentes et un délai logiciel pour simuler un travail. Elles ont donc un temps d'exécution non-négligeable d'au plus `STEP_TIME`.

Vous trouverez également un fichier `main-template.c` qui donne un exemple d'initialisation du code.

4 Modalités et évaluation

Vous devrez remettre votre projet sur Moodle pour le **27 novembre au plus tard**. Celui-ci sera remis sous la forme d'une archive zip contenant:

- Votre code (fichiers .c et .h),
- un fichier `README.txt` (ou un pdf) contenant

- les noms des deux membres du groupe,
- les réponses aux questions de la section 2,
- votre ordonnancement,
- une courte explication sur l'implémentation et le déclenchement de la tâche de la section 3.1,
- et les autres décisions qui vous semblent utiles ou importantes.

Tous les fichiers nécessaires à la compilation et l'exécution du code doivent être fournis.

Votre projet sera principalement évalué suivant les critères suivants:

- respect des consignes,
- correctitude (est-ce que le code fonctionne et respecte l'ordonnancement?),
- qualité du code (lisible, documenté, souci d'efficacité, gestion des erreurs),
- et l'ordonnancement effectué (correct, respecté par l'implémentation).

5 Glossaire

menace: L'aigle ou le renard. Un animal qui tente de voler une poule.

senseur: Le système de détection de menaces. Ne peut tester qu'une direction à la fois.

direction: Un des "côtés" de la prairie ou au-dessus.

- Directions possibles pour le renard: `north`, `south`, `east`, et `away`.
- Directions possibles pour l'aigle: `above`, et `away`.
- Note: Un large mur se trouve à l'ouest, empêchant les menaces de ce côté.

alarme: Le système permettant de chasser une menace en l'effrayant. Ne peut se déclencher que dans une direction à la fois. Mutellement exclusif avec le senseur.

prairie: L'endroit où se trouvent les poules.