Modèle d'un système de producteurs - consommateurs

TP de programmation concurrente

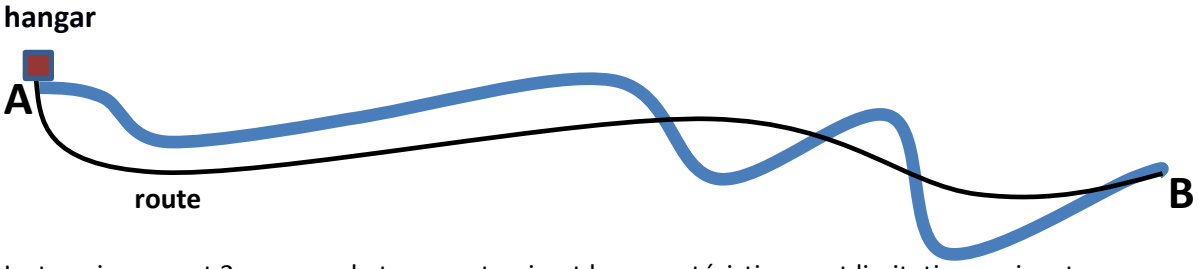
V. Pilloux

Objectif pédagogique

Implémenter un système de producteurs – consommateurs concurrents et asynchrones en respectant les contraintes de développement.

Enoncé

On aimerait modéliser un système qui simule des convois militaires qui transportent hommes et matériel pour se rendre du point géographique A au point B.



Le terrain permet 3 moyens de transport qui ont les caractéristiques et limitations suivantes:

Voie	Moyen de transport	Charge transportée x [kg]	Lieu d'arrivée (B)
Air	Avion	$x \leq 10000$	Aéroport
Route	Camion	$10000 < x < 30000$	Parking
Fleuve	Bateau	$x \geq 30000$	Port

Table 1

Chaque convoi doit impérativement transporter un conducteur/pilote, du matériel et des troupes avec des poids respectifs dans les plages suivantes :

Type	Poids nominal [kg]	Ecart de poids maximum* [kg]	Temps de préparation* [µs]
Conducteur ou pilote	80	±20	10 ± 2
Matériel	25000	±24500	10 ± 2
Troupes	8000	±7000	10 ± 2

Table 2

* l'incertitude sur ces valeurs indique qu'il faut en choisir une au hasard dans la plage indiquée, arrondie à l'unité.

Au point A, un hangar est utilisé pour préparer les convois. Celui-ci permet d'accueillir au maximum 6 conducteurs, 6 lots de matériel et 6 troupes. Le temps de préparation indiqué dans la Table 2 est l'intervalle de temps entre la disponibilité de deux conducteurs, lots de matériel ou de troupes successifs. Il peut être simulé avec la fonction `usleep(x)`. En revanche la durée du temps de transport est ignorée. Ainsi, dès qu'un convoi est prêt dans le hangar, il peut partir et se retrouver « instantanément » au point B sur l'un des lieus d'arrivée.

Notre système doit être modélisé par des threads concurrents qui simulent 3 producteurs (préparation d'un conducteur, de matériel et des troupes) et 3 consommateurs qui sont les lieus d'arrivée au point B (aéroport, parking, et port). 200 convois doivent être simulés au total.

- Avant de commencer à coder, produisez **deux organigrammes** : l'un décrivant **en partie** ce que doit faire l'un des producteurs et l'autre celui de l'un des consommateurs (voir « contenu et format du rapport à rendre »).

- Comme les poids sont déterminés au hasard, le nombre respectif de livraisons à l'aéroport, au parking et au port seront différents à chaque exécution du programme, mais leur somme à la fin de l'exécution devra toujours être égale à 200.

- Lorsque votre programme fonctionne, ôtez les attentes concernant les temps de préparation et relancez-le à plusieurs reprises : il doit encore fonctionner. Vous pouvez le faire facilement avec la macro suivante : `#define usleep(x) {}`

- Affichez au minimum les poids des « producteurs », ceux des « consommateurs » ainsi que le numéro de chaque convoi qui a été effectué. Ôtez les affichages de débogage sur le code rendu.

Conseils de développement

1. réfléchissez bien à la logique de votre organigramme avant de coder, puis vérifiez que votre code lui corresponde. Un exemple d'organigramme est fourni sur Cyberlearn.

2. n'hésitez pas à écrire des routines qui aident au débogage, mais que l'on peut retirer du code ensuite (comme l'affichage du tampon qui gère les convois par exemple).

3. si vous êtes bloqué sur un problème, utilisez un débogueur (par exemple ddd ou autre) pour pouvoir mettre des points d'arrêts dans votre code.

4. Proposition pour les prototypes de fonctions qui placent ou retirent un convoi :

```
void put(unsigned type, unsigned weight)

unsigned get(int weight_category)
```

où type est choisi parmi:

```
enum {
      DRIVER=0,
      CARGO,
      TROUP
```

```
};
```
Et `weight_category` est :
```
enum {
      LIGHT,
      MEDIUM,
      HEAVY
};
```

Contraintes de développement

- **Les queues Posix ne doivent pas être utilisées (c'est-à-dire *mqd_t*)**

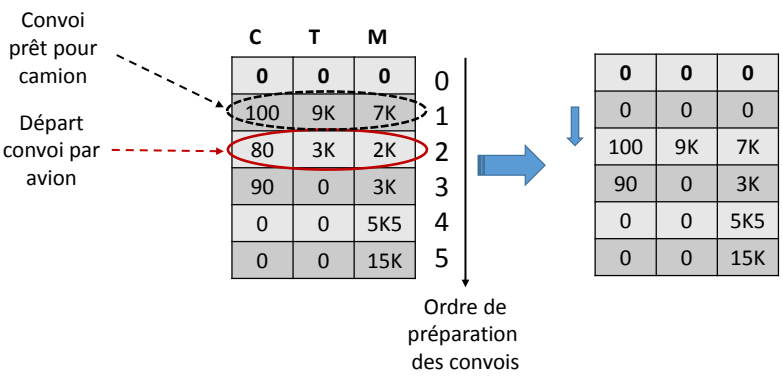
- Le modèle doit utiliser **un seul tampon commun** et de petite taille simulant le hangar pour stocker les charges à transporter. Par exemple ainsi :
  ```
  #define NB_SLOTS_IN_DEPOT 6
  // order of weight storage: driver, cargo, troups :
  unsigned convoy_weights[NB_SLOTS_IN_DEPOT][3];
  ```
  Lorsque le tampon est plein, il doit bloquer les producteurs. Cela peut arriver par exemple lorsque 6 conducteurs sont prêts, alors qu'aucune troupe n'est disponible.
- Les producteurs sont asynchrones, et les convois peuvent partir en parallèle, **mais seulement s'ils ont des destinations différentes**. Sinon ils doivent partir **dans l'ordre chronologique dans lequel ils ont été constitués**. Par exemple si x représente un indice de temps, C un conducteur, M un lot de matériel et T une troupe et que chronologiquement les ces « producteurs » sont prêts dans cet ordre :

  C1, T1, C2, T2, T3, M1, M2 il y a 2 cas de figure :

  a) Le poids de (C1,T1,M1) et (C2,T2,M2) correspondent à un transport de même catégorie : dans ce cas, (C1,T1,M1) doit partir avant (C2,T2,M2)

  b) Les poids des 2 convois appartiennent à des catégories différentes : on laisse alors l'OS choisir lequel des 2 convois part en premier. Cela pourrait être (C2,T2,M2).

  Autre exemple d'utilisation du tampon de gestion des convois: le thread « *plane* » parcourt la ligne 1 et 2, trouve sa catégorie de poids sur la ligne 2, l'ôte et décale la ligne 1.



Un camion pourrait aussi partir si la tâche gérant le parking le demandait. Notez que l'illustration ci-dessus est juste un exemple : votre implémentation du tampon peut être différente, mais elle doit respecter l'ordre de départ des convois pour une même destination.

- Lorsque tous les convois ont eu lieu, chaque thread doit se terminer correctement, sans avoir à utiliser pthread_cancel(), pthread_exit() ou exit().
- Le fonctionnement de votre code doit refléter la description de vos organigrammes (voir rapport à rendre)
- **Toutes les attentes doivent être passives**
- Votre programme doit aussi pouvoir fonctionner en ôtant les attentes indépendantes de la synchronisation (du type `uspleep()`).
- Propreté du code : voir les « consignes pour l'écriture de code C » données sur Cyberlearn

Travail à rendre

Une seule archive .zip contenant le rapport, vos fichiers sources et un makefile permettant de compiler votre programme sans warning et sans erreur.

Bon travail !

Format et contenu du rapport à rendre

Nommez le fichier contenant le rapport ainsi:

G<votre numéro de groupe>_report.pdf

Le rapport ne doit pas dépasser **4 pages A4, illustrations comprises et doit être au format PDF**. Il doit contenir les points suivants :

- Titre : mention du cours, du TP, numéro du groupe, noms des auteurs et date.
- Indiquez de façon concise l'état général de votre développement au moment où le code a été rendu. Si le code n'est pas terminé, expliquez brièvement pourquoi et quelle aurait été votre démarche pour pouvoir le finir.
- Présentez un organigramme pour la fonction `put()` et un autre pour la fonction `get()` avec une petite explication si nécessaire.
- Si vous avez observé des anomalies (sur le code rendu uniquement), **donnez une piste pour les expliquer et/ou un moyen qui permettrait de les résoudre.**
- **Il est inutile de répéter la spécification du TP dans le rapport,** mais vous pouvez y faire référence.