SDE-TP 3 et 4
Gestion d'un carrefour

Durée encadrée prévue : 8 heures

1. But du TP

Ce TP a pour but de vous initier à la programmation multi-processus et aux IPC (Inter-Process Communication) en C sous Linux.

2. Présentation du sujet

2.1. Vue utilisateur (cahier des charges) :

Soit un carrefour composé par le croisement perpendiculaire de 2 routes (une allant du Nord au Sud, et l'autre d'Est en Ouest). Ce croisement est géré par 4 feux bicolores (un à chaque angle) – les feux d'une même route sont, à chaque instant, de la même couleur et sont de la couleur opposée sur la seconde route. Il s'agit donc d'une synchronisation classique : quand c'est rouge d'un côté, c'est vert de l'autre (l'orange ne sera pas considéré).
Lorsqu'une voiture arrive, elle respecte le code le la route, à savoir qu'elle passe au vert et qu'il y a priorité à droite dans le croisement, quand la voiture veut tourner.
On considèrera qu'il n'y a jamais de voiture qui reste coincée au milieu du carrefour (cas des embouteillages).
Si un véhicule prioritaire (pompiers, ambulances, …) arrive au carrefour, ce véhicule doit pouvoir passer immédiatement. Pour cela, dès que le véhicule approche du carrefour, il est détecté et les feux changent de couleur pour lui permettre de passer dans la direction voulue (dans ce cas là, seul un des 4 feux est au vert, pour lui permettre de passer rapidemment).

2.2. Modélisation :

On a 4 processus.

- *Générateur_traffic* simule la génération du trafic normal. Pour chaque voiture générée, il choisit au hasard sur quelle portion de route elle est et où elle va. Ces informations sont transmises au *Coordinateur*.
- *Générateur_traffic_prioritaire* simule la génération du trafic prioritaire. Pour chaque véhicule prioritaire généré, il choisit au hasard sur quelle portion de route il est et où il va. Ces informations sont transmises au *Coordinateur.*
- *Coordinateur* permet aux véhicules (prioritaires ou non) de passer en fonction du code de la route, et de l'état des feux. Il peut faire changer la couleur des feux si un véhicule prioritaire approche.
- *Feux*, grâce à un timer, fait changer les feux de couleur et signale les changements à *Coordinateur*. Si un véhicule prioritaire approche, il est prévenu par Coordinateur, et met les feux à la couleur voulue.

Les processus *Générateur_traffic* et *Coordinateur* communiquent via une boîte aux lettres.
Les processus *Feux* et *Générateur_traffic_prioritaire* communiquent avec *Coordinateur* via des signaux et une mémoire partagée qui contient l'état des feux (et pour des questions pratiques, le pid de *Coordinateur*).

Attention, pour que ce soit compréhensible pour l'utilisateur, chaque processus doit faire des affichages à l'écran… Donc pour chaque action importante effectuée, un affichage doit avoir lieu, en mettant bien le nom du processus en début d'affichage.

3. VOTRE TRAVAIL

3.1. Conception

Faire la conception de cet exercice. Représentez en Lâcatre les interactions entre processus, écrivez les macro-algorithmes des fonctions utiles à chaque processus, définissez les clés des éléments d'IPC…
Ne mettez des relations père-fils entre processus que si elles sont indispensables.
Prévoyez les tests à effectuer à toutes les étapes.

3.2. Préparation de l'implémentation

Prévoyez un plan d'implémentation **très** précis. Nous vous conseillons **fortement,** dans un premier temps, de réaliser chaque IPC séparément, c'est à dire de prévoir d'écrire, pour chaque IPC, deux processus effectuant l'IPC entre eux, avec des informations « en dur », puis seulement après d'ajouter le traitement réel de l'information, puis alors de lier à un autre processus via une IPC déjà implantée et validée séparément…
Réfléchissez à comment vous allez lancer cette application (comment l'automatiser ?), à comment la quitter proprement, que faire si votre application « plante » ?..

3.3. implémentation

Suivez votre plan d'implémentation, notez vos tests…
Il est conseillé de vérifier l'état des processus et des IPC chaque fois que vous commencez à travailler... (**ipcs** pour savoir et **ipcrm** pour détruire).
Laisser votre machine « propre » à chaque fois que vous arrêtez ce TP (processus et IPC). Pour cela, utiliser kill -SIGINT `/sbin/pidof nomProcessus` (attention, c'est un accent grave!!!).
Bien penser, dans chaque processus, à rajouter une fonction fin() qui, entre autres, détruit les IPC et permet de sortir correctement.

3.4. Indice

Comment prévoyez-vous de faire pour que toutes les IPC soient détruites proprement dans tous les cas de figures?

3.5. Date échéances et livrables

9/11/15 : Distribution des sujets.
30/11/15 à 8h (du matin) au plus tard : Préparation à rendre au secrétariat des études (cette préparation comprend les parties « Conception » et « Préparation à l'implémentation »). Nous vous rendons cette préparation corrigée lors de la première séance de TP.
25/1/16 à 8h au plus tard : Compte rendu du TP complet à rendre au secrétariat des études. Rédigez votre compte-rendu final selon les conseils donnés sur la fiche « Un compte-rendu de TP » donnée dans les documents de référence du PIT. Longueur max. hors listing : 10 pages. Rendu des codes sous Moodle, sous forme d'une archive.
Semaine du 25 janvier 2016 : démonstration (penser à s'inscrire sur le planning !).