Simulation d'un système de transport

1 Objectif général

L'objectif de ce travail pratique est de vous familiariser avec les appels système Unix/Linux concernant la communication et la synchronisation des processus et des threads. Il s'agit de simuler un circuit d'un système de transport en commun, composé d'un autobus, d'un métro et de 3 taxis.

2 Description : Circuit d'autobus et de métro

Le système à simuler est composé d'un trajet d'autobus et d'un trajet de métro (voir figure 1). Ces trajets se composent de 5 stations d'autobus (stations 0 à 4) et de 3 stations de métro (stations 5 à 7) dans lesquelles les utilisateurs peuvent descendre, monter ou tout simplement attendre l'autobus ou le métro. Comme le montre la figure 1, l'autobus fait continuellement le tour des stations 0 à 4 (0, 1, 3, 4, 0, 1, ...) tandis que le métro parcourt les stations 5 à 7 puis revient dans la direction opposée (5, 6, 7, 6, 5, ...). Les stations 0 et 5 permettent aux passagers d'effectuer des transferts entre les trajets de metro et d'autobus. Notez qu'un passager qui utilise l'autobus pour se rendre à la station 5, arrive à destination lorsque l'autobus le débarque à la station 0. Il en est de même pour le passager qui utilise le métro pour se rendre à la station 0. Les véhicules (autobus et métro) ont chacun une capacité maximale (8 passagers pour le train et 5 passagers pour l'autobus). Aucun passager ne peut monter dans un véhicule lorsque ce dernier aura atteint sa capacité maximale. Dans un tel cas, le passager devra attendre le prochain passage du véhicule (dans la bonne direction) ou appeler un taxi.

3 Ajout de taxis

Supposez maintenant que 3 taxis sont à la disposition des passagers pour les conduire vers les destinations de leurs choix. On suppose également que :

1. Chaque taxi a une capacité maximale de 1 passager.
2. Une course en taxi étant plus dispendieuse qu'une course en autobus ou en metro, un passager attendra un certain temps, dans une station, avant de décider d'appeler un taxi. S'il appelle un taxi, le passager quitte définitivement la station pour se mettre en attente d'un taxi.

Implantation

Ce système de transport est simulé par un processus principal et un processus taxis (processus fils du processus principal). Le processus principal communique avec le processus taxis via un pipe nommé. La figure 2 présente le schéma conceptuel du système.

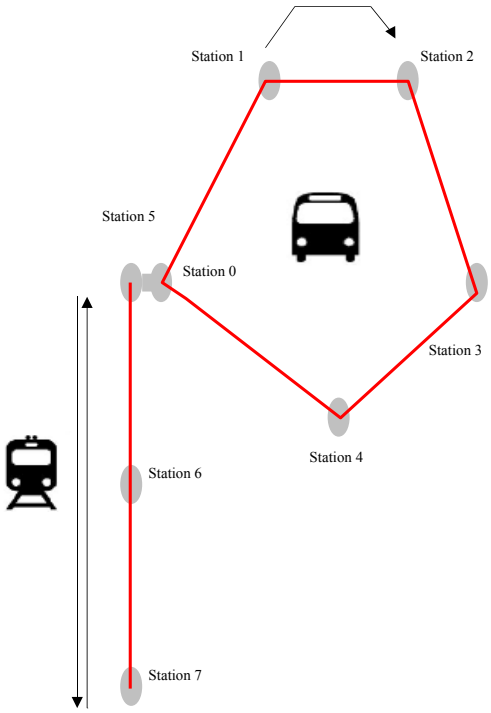

FIGURE 1 – *Exemple d'un circuit ż 5 stations d'autobus et 3 stations de métro. L'automus fait contunuellement le tour du circuit dans la même direction. Tandis que le métro fait des allers-retours sur la même ligne de métro*

3.1 Processus principal

Le processus principal se compose de trois threads : métro, autobus et vérificateur. L'autobus circule dans le sens horaire (indiqué par une flèche dans la figure 1). À chaque station d'autobus est associée une file d'attente de passagers.

Le métro assure le transport des passagers entre les stations 5, 6 et 7. Les usagers du métro peuvent attendre dans une même station mais pour une direction différente. Pour ce faire, chaque station de métro possède deux files d'attente (une file pour chaque direction). Toutes les files d'attente des stations sont des queues de type FIFO.

Les passagers sont représentés par une structure contenant les informations suivantes :

— Numéro d'identification unique
— Station de départ.
— Station d'arrivée
— Temps d'attente écoulé
— Un transfert entre le circuit de métro et d'autobus est requis (valeur booléenne)
— Temps d'attente maximal

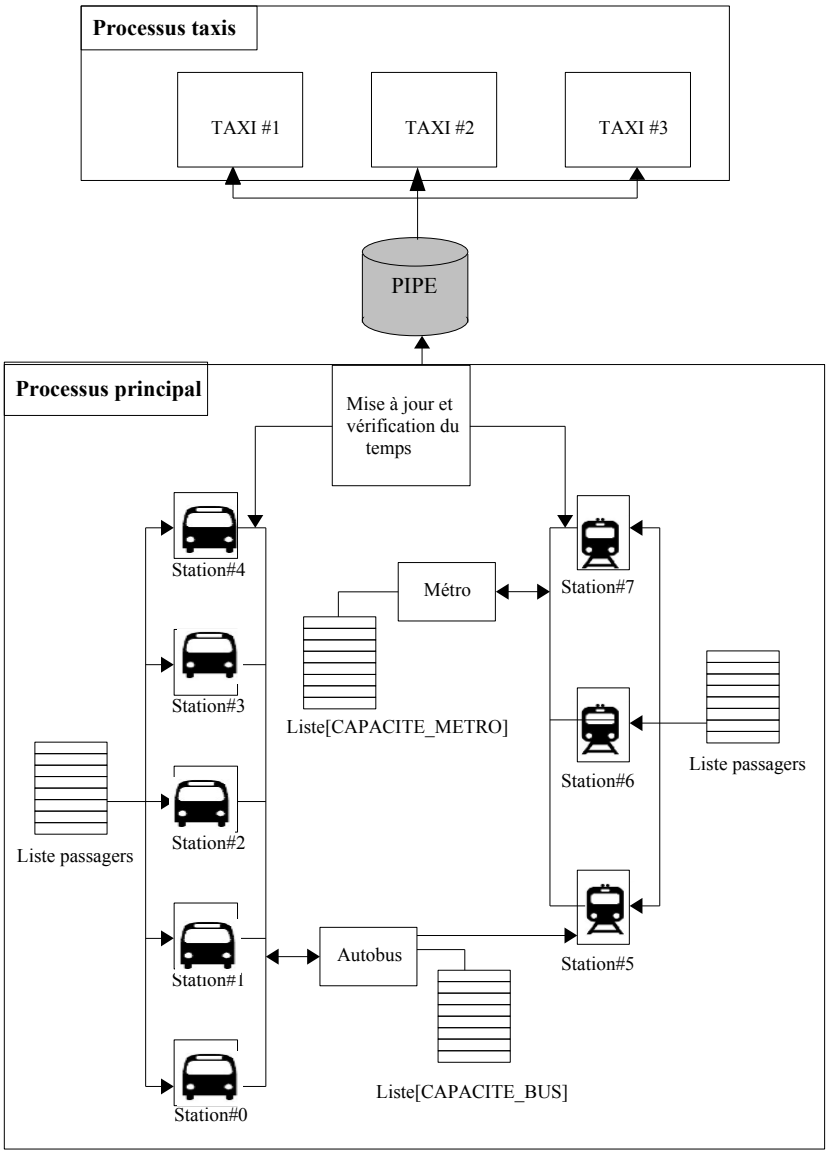

FIGURE 2 – *Schéma conceptuel du système de transport en commun.*

Les passagers sont lus à partir d'un fichier texte. Le processus principal reçoit le nom du fichier à la ligne de commande, crée les passagers et les insère dans la bonne file d'attente. Dans le fichier texte, la première ligne fixe le nombre de passagers tandis qu'une ligne précédée par un '#' identifie un passager. Sur une même ligne, l'ordre des informations lues correspond à l'ordre des variables dans la structure de données passager. Par exemple, la ligne "# 21 3 6 0 1 8" identifie le passager 21 qui se trouve à la station 3 (depuis 0 unités de temps) et veut

se rendre à la station 6. Son temps d'attente maximal est 8 unités de temps. Après lecture et répartition des passagers dans les files d'attente des stations, le processus principal crée, dans l'ordre, le tube de communication nommé, le processus taxis et les threads.

3.2 Les threads autobus et métro

Les threads autobus et métro sont répétitifs (cycliques). À chaque cycle, chaque thread (autobus et métro) réalise, dans l'ordre, les opérations suivantes :

1. Incrémenter un compteur de station en fonction de la direction du véhicule (metro, autobus).
2. Vérifier dans sa liste si un passager est arrivé à sa destination. Le cas échéant, il assure le débarquement du (des) passager(s) en question en ajustant sa liste de passagers et affiche les actions exécutées. Au besoin, transférer les passagers vers la queue des stations 0 ou 5. Le format de l'affichage est :

   ```
   [bus ou metro] : [embarque ou debarque] le passager {id_passager}
   [bus ou metro] : transfert passager {id} vers station {id_station}
   ```

3. Faire l'embarquement des nouveaux passagers en examinant la queue qui correspond au compteur de station. Attention, à la capacité maximale du véhicule.
4. Donner le contrôle au vérificateur une fois leur exécution terminée via un rendez-vous bilatéral (voir plus loin la description d'un rendez-vous bilatéral). Chaque cycle du thread autobus s'exécute en concurrence avec un cycle du thread métro. Ces deux cycles sont toujours suivis d'un cycle du thread vérificateur.

3.3 Le thread vérificateur

Le vérificateur est un thread répétitif. à chaque cycle, il réalise, dans l'ordre, les actions suivantes :

1. Parcourir toutes les files d'attente des passagers et incrémenter le temps d'attente de chaque passager.
2. Comparer le temps d'attente de chaque passager avec son temps d'attente maximal.
3. Transférer le passager de la queue autobus/métro (si son temps d'attente maximal est atteint) vers le tube nommé mémorisant les passagers en attente de taxis. Cela suppose que le passager demande un taxi lorsque son temps d'attente maximal est atteint. Le thread affiche alors le message suivant :

   ```
   verificateur : transfert du passager {id_ passager}vers le taxi {id_taxi}
   ```

4. Donner le contrôle à l'autobus et au métro via un rendez-vous bilatéral. Les deux threads autobus et métro sont inactifs pendant que le vérificateur exécute son cycle.

3.4 Le processus taxis

Le processus taxis est composé de trois threads (un pour chaque taxi) qui se chargent de récupérer les demandes du pipe (lecture bloquante). Lorsqu'un thread taxi récupère une demande de la part d'un passager, il exécute `usleep(10)` puis affiche :

```
taxi#{id_taxi} : passager {id_ passager} est rendu a la station {id_station}
```

Note : l'action `usleep(10)` simule l'action de reconduire un passager.

4 Rendez-vous bilatéral

Deux threads peuvent se synchroniser en utilisant deux sémaphores comme le montre la figure 3. Ceci est appelé rendez-vous bilatéral. Un rendez-vous bilatéral est similaire à un rendez-vous unilatéral sauf que les deux threads exécutent un à la suite de l'autre.

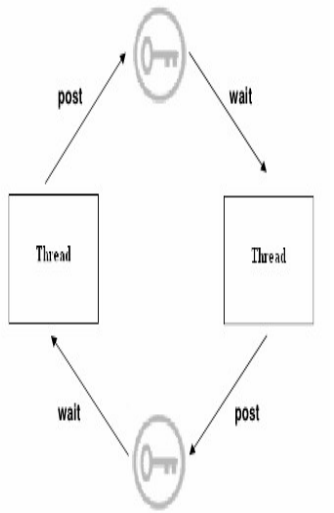

FIGURE 3 – *Rendez-vous bilatéral.*

5 Terminaison

Le programme doit se terminer lorsque tous les passagers seront arrivés à bon port. À la terminaison, un affichage de profit est requis. Le coût d'une course en autobus ou en métro est de1$. Les transferts ne sont pas gratuits. En cas de transfert, le passager doit encore débourser le montant requis. Le coût pour une course en taxi est de 3 $, peu importe le trajet parcouru.