TP 3 (à rendre)

1 Introduction

On souhaite modéliser le fonctionnement d'un musée à l'aide de processus, matérialisés par des commandes indépendantes travaillant sur un ensemble d'objets IPC System V (mémoire partagée et sémaphores).

Dans cette modélisation, le musée est constitué des objets IPC System V. Son directeur crée le musée avec une capacité (nombre de visiteurs) donnée, déclare le musée ouvert ou fermé, et peut le supprimer. En outre, un contrôleur surveille les entrées et les sorties, en vérifiant que la capacité du musée ne soit pas dépassée. Enfin, les visiteurs peuvent se présenter et entrer dans le musée s'ils sont autorisés par le contrôleur ; ils passent leur chemin si la file d'attente est trop longue.

2 Travail à réaliser

On demande de réaliser 4 programmes (mono-processus, mono-threadés) :

1. le programme `directeur` possède les fonctionnalités suivantes :
   — appelé avec la syntaxe « `directeur creer` *`capacité file`* », il crée le musée (c'est-à-dire les objets IPC System V) et les initialise avec la capacité (nombre maximum de visiteurs admis dans le musée) et la taille de la file d'attente (nombre maximum de visiteurs attendant à l'extérieur du musée, au delà duquel les nouveaux visiteurs déclarent forfait), le musée est fermé à l'issue de la création,
   — appelé avec l'argument « `ouvrir` » ou « `fermer` », il déclare le musée ouvert ou fermé,
   — appelé avec l'argument « `supprimer` », le musée est supprimé. On ne tient pas compte des autres programmes pouvant éventuellement être en attente d'un sémaphore du musée ;
2. le programme `controleur` est appelé sans argument : le processus doit être « en poste » avant l'ouverture du musée. Dès que le musée est déclaré ouvert, le contrôleur vérifie les entrées et les sorties des visiteurs pour ne pas excéder la capacité du musée ; lorsque le musée est déclaré fermé, le contrôleur stoppe l'entrée de nouveaux visiteurs et n'autorise plus que la sortie des visiteurs actuellement dans le musée ; lorsque le dernier visiteur a quitté le musée, le processus contrôleur se termine ;
3. le programme `visiteur` simule l'arrivée d'un nouveau visiteur. Il est appelé autant de fois qu'il y a de visiteurs. Un visiteur se termine si la file d'attente excède la limite fixée lors de la création. Si ce n'est pas le cas, le visiteur demande l'autorisation d'entrer au musée qui lui est accordée par le contrôleur ; il entre alors dans le musée, y reste un temps (en milli-secondes) spécifié en argument, puis sort du musée. Si le musée est fermé ou si l'autorisation n'est pas donnée par le contrôleur, le visiteur reste en attente jusqu'à la prochaine ouverture du musée. Les visiteurs peuvent éventuellement se doubler dans la file d'attente, on n'impose pas de respecter strictement l'ordre d'arrivée ;
4. le programme `dump` affiche l'état du musée, c'est à dire l'état détaillé de tous les objets IPC System V. C'est un outil de mise au point et de visualisation indispensable.

Bien évidemment, vous éviterez toute attente active, même ralentie. Vous utiliserez un segment de mémoire partagée pour stocker les informations devant être partagées entre plusieurs processus. Vous utiliserez les sémaphores IPC System V comme unique mécanisme de synchronisation. On suppose qu'il n'y a qu'un seul directeur et un seul contrôleur (on ne demande pas de gérer la concurrence entre deux processus contrôleurs, par exemple).

Vous rédigerez un rapport comprenant notamment la description des synchronisations entre les différents acteurs ainsi que la justification des informations placées en mémoire partagée.

3 Implémentation

Le code de retour de vos programmes devra indiquer le succès ou l'échec de l'action associée. En cas d'erreur, vous adopterez la stratégie simple de terminer l'exécution du programme concerné avec un message explicite et un code de retour adéquat.

On vous demande de mettre en place un système d'affichage de l'état de vos programmes sous forme de messages de débogage contrôlés par la variable d'environnement `DEBUG_MUSEE`. Si celle-ci existe, elle doit contenir un entier. Si elle n'existe pas, ou si la valeur est 0, vos programmes doivent être muets (sauf pour les erreurs, bien évidemment). Si la valeur égale 1, les programmes doivent afficher un court message lors des étapes importantes. Avec des valeurs supérieures, vos programmes doivent afficher des informations de débogage plus complètes, notamment le détail des synchronisations. N'oubliez pas d'utiliser `fflush` pour avoir ces messages dès leur affichage lorsque la sortie est redirigée dans un fichier.

Un ensemble de jeux de tests est à votre disposition. Vous ne devez en aucun cas le modifier, mais vous pouvez le compléter avec de nouveaux scripts.

4 Modalités de remise

Vous déposerez votre TP, avec votre rapport, sous forme d'une archive (au format « *login*.tar.gz » où *login* est votre nom de login sur Turing) sur Moodle, dans l'espace de devoirs prévu à cet effet. Vous supprimerez les fichiers binaires et autres fichiers de log. Les rendus présentant de trop fortes simulitudes seront sanctionnés.