TP 6 (à rendre)

Note : ce TP est un travail individuel, à rendre avant le 15 décembre 2021 à 20h (voir modalités plus bas). Aucun rendu ne sera accepté en retard.

1 Introduction

L'Organisation Mondiale de la Santé signale l'arrivée prochaine du Covid-23 avec un risque de pandémie interplanétaire. La société Covid Corp Inc., chargée de la vaccination par le gouvernement nouvellement nommé de la planète Terre, doit simuler la mise en place de vaccinodromes. Malheureusement, les sources des programmes de simulation des vaccinodromes de la précédente pandémie, le Covid-19, ont été perdus derrière une pile de masques. C'est pour cela que le gouvernement a décidé de conclure un accord avec le département d'informatique de l'UFR de mathématique et d'informatique de l'université de Strasbourg, et c'est vous qui êtes chargé-e de rédiger les programmes de simulation.

2 Fonctionnement du vaccinodrome

Le vaccinodrome est constitué d'une salle d'attente comprenant n sièges où s'installent les patients au fur et à mesure de leur arrivée en attendant qu'un des médecins les prenne en charge pour la vaccination. Celle-ci prend un temps t. Le médecin doit noter le nom de chaque patient pour la mise à jour de son pass sanitaire. Une fois la vacccination effectuée, le patient peut partir. Si un patient tente d'entrer dans le vaccinodrome alors que les n sièges sont occupés, il doit attendre à l'extérieur.

Le vaccinodrome est dimensionné pour m médecins qui peuvent arriver plus ou moins rapidement en fonction des embouteillages sur la route ou dans le tram : le nombre de médecins est donc dynamique. Pour simplifier, on supposera qu'un médecin ayant commencé son service dans le vaccinodrome doit y rester jusqu'à la fermeture complète : le nombre de médecins peut ainsi croître de 0 jusqu'à m, mais ne peut pas décroître.

Enfin, lorsque le vaccinodrome doit fermer, tous les patients encore en attente dans le vaccinodrome ou en cours de vaccination doivent terminer le parcours vaccinal. Les médecins ne peuvent partir que lorsque le dernier patient a quitté le vaccinodrome. Les patients attendant à l'extérieur, quant à eux, repartent en maugréant sans avoir été vaccinés.

3 Travail à réaliser

On demande de réaliser les programmes suivants en langage C :

1. Le programme `ouvrir` ouvre le vaccinodrome. Il admet les arguments n, m et t (exprimé en millisecondes) et doit créer tous les éléments nécessaires (mémoire partagée, sémaphores, etc.) sauf s'ils existent déjà auquel cas une erreur doit être générée. Au démarrage, il n'y a encore aucun médecin dans le vaccinodrome.
2. Le programme `fermer` donne l'ordre de fermer le vaccinodrome : plus aucun patient n'est admis, mais tous ceux qui sont déjà engagés dans le processus vaccinal doivent le terminer dans les règles. Lorsque ce programme termine son exécution, tous les patients et les médecins doivent avoir quitté le vaccinodrome et les objets (sémaphores, segments de mémoire partagée, etc.) doivent être supprimés.
3. Le programme `nettoyer` supprime toutes les structures de données rémanentes (sémaphores, segments de mémoire partagée, etc.) au cas où une précédente exécution aurait laissé de telles structures en place.
4. Le programme `medecin` simule un médecin. Chaque médecin se voit attribuer un box numéroté dans l'ordre d'arrivée, à partir de 0 jusqu'à $m - 1$, que le patient doit connaître. Le programme affiche le nom du patient en cours de vaccination. Lorsque la fermeture est ordonnée, les médecins doivent attendre que tous les patients aient été vaccinés pour se terminer.

5. Le programme `patient` simule un patient. Il admet un argument, le nom du patient, et affiche sur sa sortie standard les différentes étapes du parcours vaccinal : numéro du siège ($0 \ldots n-1$) dans la salle d'attente et identité (numéro) du médecin qui l'a pris en charge. Si tous les sièges de la salle d'attente sont occupés, le programme doit attendre à l'extérieur. Si le vaccinodrome est fermé avant que le patient ne puisse y rentrer, le programme se termine avec un code de retour non nul. Le patient ne sait pas combien de temps dure la vaccination, c'est le médecin qui doit lui indiquer quand il peut partir.

Deux stratégies sont possibles pour la rencontre entre le patient et le médecin : soit le médecin choisit un des patients en attente, soit le patient choisit un box libre. Bien évidemment, vous éviterez toute attente active, même ralentie. Pour partager des informations entre plusieurs processus, vous n'utiliserez que la mémoire partagée POSIX, à l'exclusion de tout autre mécanisme tel que fichier, tube, etc. De même, pour la synchronisation, vous n'utiliserez que les sémaphores POSIX, à l'exclusion de tout autre mécanisme tel que barrière, signal, verrou, etc.

Le code de retour de vos programmes devra indiquer si une erreur s'est produite ou non. En cas d'erreur, vous adopterez la stratégie simple de terminer l'exécution du programme concerné avec un message explicite.

Pour simplifier l'implémentation, on supposera que les conditions suivantes sont vérifiées. Sauf mention contraire, on ne vous demande pas de les vérifier vous-mêmes :

— il n'y a qu'un seul vaccinodrome ;
— les noms des patients sont limités à 10 octets (limite à vérifier) ;
— il n'y a pas besoin de vérifier qu'un patient a déjà été vacciné.

Vous trouverez sur Moodle deux fichiers `asem.c` et `asem.h` constituant la « bibliothèque ASE » et dont le but est de vous aider dans la mise au point de vos programmes et de faciliter l'évaluation de votre travail. Ces deux fichiers ne doivent en aucun cas être modifiés (vous ne les déposerez d'ailleurs pas sur Moodle) et leur utilisation est impérative. Ils contiennent deux familles de fonctions :

— un système de debug (fonctions `ainit` et `adebug`) permettant d'afficher ou non des messages suivant un niveau indiqué par la variable d'environnement `DEBUG_ASE`. Si celle-ci existe, elle doit contenir un entier. Si elle n'existe pas ou si sa valeur est 0, les programmes ne doivent rien afficher sauf en cas d'erreur. Si la valeur égale 1, les programmes doivent afficher les synchronisations. Pour les valeurs supérieures à 1, les programmes affichent également toute information complémentaire que vous jugerez utile.
— des fonctions comparables aux fonctions `sem_*` sur des sémaphores non nommés : elles utilisent un type `asem_t` qui contient, outre le type `sem_t` habituel, un nom explicite que vous définirez pour rendre le suivi des synchronisations plus facile. L'utilisation directe des fonctions `sem_*` est interdite.

Vous rédigerez un rapport comprenant notamment la description des synchronisations (événement caractéristique de la synchronisation, acteurs concernés, mécanisme utilisé) ainsi que la justification des informations placées en mémoire partagée. Vous utiliserez le modèle fourni sur Moodle pour ce faire (vous pouvez utiliser LaTeX ou tout autre outil de votre choix, à condition de fournir un rapport au format PDF).

4 Jeux de tests

Un ensemble de jeux de tests est à votre disposition. Ceux-ci constituent des spécifications complémentaires, notamment en matière de messages attendus à l'affichage. Vous ne devez en aucun cas modifier ces tests, mais vous pouvez éventuellement les compléter avec de nouveaux scripts. La cible `couverture-et-tests` du `Makefile` peut vous donner des idées pour ajouter des tests.

Les scripts fournis incorporent des durées et sont donc sensibles à la vitesse de l'ordinateur qui les exécute. Tels quels, ils fonctionnent sur `turing.unistra.fr` qui sert de référence. Vous pouvez utiliser la variable `MARGE` pour agir sur certaines tolérances, (par exemple : `MARGE=60 make test`), mais la réussite n'est en aucun cas garantie.

5 Modalités de remise

Pour faciliter l'évaluation de votre travail, il est impératif d'utiliser le répertoire qui vous est fourni sur Moodle. Vous devrez rédiger les fichiers `.c` correspondant aux différents programmes demandés, ainsi que les fichiers `shm.c` et `shm.h` contenant les définitions communes utilisées dans au moins deux programmes. Votre rapport doit être rendu au format PDF. Pour construire l'archive à déposer sur Moodle, vous utiliserez la cible `devoir.tgz` du `Makefile`. Les rendus présentant de trop fortes similitudes seront sanctionnés.