Rapport de TP

« Vaccinodrome »

Date de rendu : 15/12/2021

Matthieu FREITAG

UFR Math-Info - Université de Strasbourg

3ième année de Licence Informatique

Architecture des systèmes d'exploitation

https://github.com/Zapharaos/Vaccinodrome

Table des matières

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.

On se propose donc, pour palier à cette perte, d'étudier et de réaliser l'implémentation en language C d'une simulation des vaccinodromes. Pour ce faire, on se servira de la mémoire partagée, d'une implémentation préfaite des semaphores et d'un ensemble de jeux de tests préfourni.

On implémentera `ouvrir.c` pour initaliser la mémoire partagée et ouvrir un vaccinodrome, `fermer.c` pour signaler la fermeture du vaccinodrome avant d'attendre le départ des protagonistes et de nettoyer la mémoire partagée. On se servia également de `nettoyer.c` pour la nettoyer la mémoire partagée et de `patient.c` (respectivement `médecin.c`) pour simuler un patient (respectivement médecin).

2 Structure de données

2.1 Structures de données partagées

La structure principale où se trouve ma mémoire partagée est `vaccinodrome_t` :

Champ	Type	Init	Description
n	int	–	le nombre de sieges
m	int	–	le nombre de box
t	useconds_t	–	la duree d'une vaccination (en millisecondes)
status	vac_status_t	`OUVERT`	le statut du vaccinodrome (cf. documentation de `vac_status_t`)
med_count	int	0	le nombre de médecin dans le vaccinodrome
pat_count	int	0	le nombre de patient dans le vaccinodrome
salle_count	int	0	le nombre de patient dans la salle d'attente
vide	asem_t	0	sémaphore pour indiquer que le vaccinodrome est vide
pat_vide	asem_t	0	sémaphore pour indiquer qu'il ne reste aucun patient aux médecins
fermer	asem_t	1	sémaphore pour indiquer qu'il ne reste aucun patients à fermer
salle_m	asem_t	0	sémaphore pour simuler la salle d'attente (côté médecin)
salle_p	asem_t	n	sémaphore pour simuler la salle d'attente (côté patient)
edit_salle	asem_t	1	sémaphore pour les sections critiques communes
medecin	asem_t	1	sémaphore pour les sections critiques des médecins
patient	patient_t []	–	tableau de taille n*m pour simuler les patients en salle d'attente et dans les box des médecins

La structure `vaccinodrome_t` fait appel à l'enumération `vac_status_t` :

Champ	Valeur	Description
`FERME`	0	le vaccinodrome est fermé
`OUVERT`	1	le vaccinodrome est ouvert

La structure `vaccinodrome_t` fait également appel à un tableau de structures `patient_t` :

Champ	Type	Init	Description
status	pat_status_t	`LIBRE`	le statut du patient (cf. documentation de `pat_status_t`)
nom	char []	–	le nom du patient
id_medecin	int	–	l'id du medecin qui traite le patient
s_patient	asem_t	0	sémaphore pour indiquer que le patient doit attendre
s_medecin	asem_t	0	sémaphore pour indiquer que le médecin doit attendre

La structure `patient_t` fait appel à l'enumération `pat_status_t` :

Champ	Valeur	Description
`LIBRE`	0	la place est libre
`OCCUPE`	1	la place est occupée
`TRAITEMENT`	2	la place est un box et le patient s'y fait vacciner

2.2 Structures de données non partagées

L'implémentation pour laquelle je me suis décidé n'implique pas d'utiliser des données non partagées. Chaque variable ou structure de données utilisée est accessible via la mémoire partagée, mis à part deux variables dont je me sers pour stocker les identifiants des médecins et des patients. J'estime que les détailler ici ne serait pas pertinent étant donné que leur utilisation est triviale.

3 Synchronisations

3.1 Arrivée d'un patient

Le patient est le seul à intérargir ici.

Champ	Type	Description
salle_p	asem_t	sémaphore représentant la salle d'attente (voir `vaccinodrome_t`)
status	vac_status_t	le status actuel de vaccinodrome (voir `vaccinodrome_t`)
salle_count	int	nombre de patients dans la salle d'attente (voir `vaccinodrome_t`)
pat_count	int	nombre de patients dans le vaccinodrome (voir `vaccinodrome_t`)
n	int	nombre de sièges (voir `vaccinodrome_t`)
m	int	nombre de médecins (voir `vaccinodrome_t`)
patient	patient_t []	tableau de taille n*m pour simuler la salle d'attente (voir `vaccinodrome_t`)
id_patient	int	indentifiant du patient (variable locale pour chaque patient)

```
P (salle_p)
SI status == FERME ALORS fin du patient FINSI
incrementer_compteurs() // salle_count et pat_count
installation(patient, n+m) // trouver un siege libre et s'installe
// intéractions avec les médecins
V (salle_p)
```

Le patient arrive et attend qu'une place se libère dans la salle d'attente. Si le vaccinodrome est fermé, le patient repart. Sinon, il incrémente les compteurs qui correspondent au nombre de patients. Lors de son installation, le patient parcourt le tableau `patient` de taille `n+m` jusqu'à trouver un siège libre, s'y installe et y renseigne ses informations. Il y aura forcément au moins une place de libre car le sémaphore `salle_p` a été passé. Après son installation, le patient prévient les médecins de son installation et attend qu'on médecin vienne le chercher.

`incrementer_compteurs()` est dans une section critique car les compteurs sont communs à tous. De la même manière, `installation()` est également dans une section critique car la liste des places est commune à tous. Les intéractions enre le patient et les médecins ne seront pas détaillées ici, mais elles le seront plus tard.

3.2 Arrivée d'un médecin

Le médecin est le seul à intérargir ici.

Champ	Type	Description
status	vac_status_t	le status actuel de vaccinodrome (voir `vaccinodrome_t`)
med_count	int	nombre de médecins dans le vaccinodrome (voir `vaccinodrome_t`)
m	int	nombre de médecins (voir `vaccinodrome_t`)
id_m	int	indentifiant du médecin (variable locale pour chaque médecin)
salle_m	asem_t	sémaphore représentant la salle d'attente (voir `vaccinodrome_t`)
salle_count	int	nombre de patient dans la salle d'attente (voir `vaccinodrome_t`)

```
SI status == FERME ALORS fin du médecin FINSI
SI m == med_count ALORS fin du médecin FINSI // trop de médecins
incrementer() // med_count
TANTQUE VRAI
    SI status == FERME ET salle_count = 0 ALORS fin du médecin FINSI
    P (salle_m) // patient ou fermer
    SI status == FERME ET salle_count = 0 ALORS fin du médecin FINSI
    // intéractions avec les patients
FIN TANTQUE
```

Le médecin arrive et vérifie le statut du vaccinodrome, s'il est fermé il repart, s'il est ouvert il vérifie combien de médecins sont déjà présent et agit en fonction. S'il y a autant de médecins que de box, il repart, sinon il récupère son identifiant de médecin et incrémente le compteur puis il entre dans la boucle.

A l'intérieur de cette boucle, il vérifie si le vaccinodrome ne s'est pas fermé entre temps et que le nombre de patients dans la salle d'attente n'est pas nul, si oui il repart. Sinon, il attend avec le sémaphore salle_m. Il se peut qu'aucun patient n'entre et donc que le médecin attende de manière infinie. C'est pourquoi `fermer.c`

peut débloquer les médecins. Auquel cas, le médecin le détecte avec la condition suivante : si le vaccinodrome est fermé et que le nombre de patients dans la salle d'attente est nul alors il peut partir. Sinon, c'est qu'un patient est entré dans la salle d'attente et le médecin va le traiter.

`incrementer()` récupère l'identifiant du médecin et incrémente le compteur `med_count`. Ce qui se trouve avant la boucle `TANTQUE` est dans une section critique car les variables utilisées sont communes entre les médecins. Les intéractions enre le médecin et les patients ne seront pas détaillées ici, mais elles le seront plus tard.

3.3 Interactions entre patients et médecins

J'ai décidé d'implémenter la deuxième option : le médecin trouve un patient dans la salle d'attente.

Médecin lors de l'intéraction avec un patient :

Champ	Type	Description
status	vac_status_t	le status actuel de vaccinodrome (voir `vaccinodrome_t`)
salle_count	int	nombre de patient dans la salle d'attente (voir `vaccinodrome_t`)
salle_m	asem_t	sémaphore représentant la salle d'attente (voir `vaccinodrome_t`)
n	int	nombre de sieèges (voir `vaccinodrome_t`)
m	int	nombre de médecins (voir `vaccinodrome_t`)
patient	patient_t []	tableau de taille n*m pour simuler la salle d'attente (voir `vaccinodrome_t`)
patient.status	pat_status_t	statut du patient `patient_t`)
patient.id_medecin	int	medecin du patient `patient_t`)
patient.s_patient	asem_t	semaphore attente du patient `patient_t`)
patient.s_medecin	asem_t	semaphore attente du médecin `patient_t`)
id_p	int	variable locale pour identifier le patient sélectionné

```
TANTQUE VRAI
    SI status == FERME ET salle_count = 0 ALORS fin du médecin FINSI
    P (salle_m) // patient ou fermer
    SI status == FERME ET salle_count = 0 ALORS fin du médecin FINSI
    id_p = trouver_patient()
    V (patient[id_p].s_patient) // convoque patient
    P (patient[id_p].s_medecin) // attend arrive du patient
    vaccination()
    V (patient[id_p].s_patient) // libere le patient
FIN TANTQUE
```

Patient lors de l'intéraction avec le médecin :

Champ	Type	Description
salle_p	asem_t	sémaphore représentant la salle d'attente (voir `vaccinodrome_t`)
patient	patient_t []	tableau de taille n*m pour simuler la salle d'attente (voir `vaccinodrome_t`)
patient.nom	char[]	le nom du patient `patient_t`)
patient.id_medecin	int	medecin du patient `patient_t`)
patient.s_patient	asem_t	semaphore attente du patient `patient_t`)
patient.s_medecin	asem_t	semaphore attente du médecin `patient_t`)
id_patient	int	variable locale pour identifier le siège sélectionné

```
P (patient[id_patient].s_patient) // attend arrive du medecin
V (salle_p) // libere place salle d'attente
afficher_medecin()
V (patient[id_patient].s_medecin) // previent medecin de son arrive dans le box
P (patient[id_patient].s_patient) // attend la fin de la vaccination
```

Pour cela, le médecin attend qu'un patient entre dans la salle d'attente puis il parcourt le tableau `patient` jusqu'à trouver un siège `OCCUPE`. Il y en aura forcément un étant donné que le sémaphore a été débloqué. Puis le médecin change le statut du patient à `TRAITEMENT` et y insère son identifiant de médecin. Ensuite, le médecin affiche le nom du patient, il prévient ce même patient qui se trouve dans la salle d'attente et l'attend dans son box (il n'y a pas de structure box à proprement parler mais c'est un moyen pour mieux décrire l'implémentation).

Comme indiqué précedemment, après être entré dans le vaccinodrome, le patient s'installe dans la salle d'attente et attend qu'un médecin vienne le chercher. Lorsque le patient est convoqué par le médecin, il affiche l'identifiant du médecin, le rejoint dans son box et attend la fin du processus de vaccination avant de quitter la vaccinodrome.

Après que le patient ait rejoint le médecin dans son box, le médecin va démarrer le processus de vaccination (usleep dans notre cas) avant de libérer le patient et attendre l'arrivée d'un nouveau patient. Il se peut qu'aucun autre patient arrive, auquel cas le médecin sera débloqué par la fermeture.

3.4 Fermeture du vaccinodrome

La fermeture est le seul programme à intérargir ici.

Champ	Type	Description
status	vac_status_t	le status actuel de vaccinodrome (voir `vaccinodrome_t`)
salle_m	asem_t	sémaphore représentant la salle d'attente (voir `vaccinodrome_t`)
med_count	int	nombre de médecins dans le vaccinodrome (voir `vaccinodrome_t`)
m	int	nombre de médecins (voir `vaccinodrome_t`)
vide	asem_t	sémaphore indique si le vaccinodrome est vide (voir `vaccinodrome_t`)

```
status = FERME
Si pat_count > 0 ALORS P (fermer) FINSI
SI med_count > 0 ALORS
    LOOP : 0 à m
        V (salle_m)
    FIN LOOP
    SI med_count != 0 ALORS P(vide) FINSI
FINSI
clean_file()
```

Lors du lancement de `fermer.c`, il faut d'abord annoncer la fermeture du vaccinodrome. Ensuite, il faut attendre que tous les patients aient quité le vaccinodrome puis attendre que le vaccinodrome soit vide.

C'est pourquoi on vérifiera s'il reste un nombre positif de patients. S'il en reste, on attendra qu'il n'en reste aucun. Sinon, on vérifiera s'il reste un nombre positif de médecins dans le vaccinodrome. S'il ne reste aucun médecin, on peut quitter normalement. Sinon, cela signifie peut-être que des médecins sont bloqués dans une attente infinie et `fermer.c` va les débloquer via le semaphore dans `LOOP`. Après cette boucle, si le nombre de médecins est différent de 0 on attend qu'ils se terminent (ce qui arrivera forcément puisqu'on vient de les débloquer) puis on peut quitter normalement.

3.5 Patients après fermeture

Le patient est le seul à intérargir ici.

Champ	Type	Description
pat_count	int	nombre de patients dans le vaccinodrome (voir `vaccinodrome_t`)
status	vac_status_t	le status actuel de vaccinodrome (voir `vaccinodrome_t`)
pat_vide	asem_t	sémaphore pour indiquer qu'il ne reste aucun patient (voir `vaccinodrome_t`)

```
pat_count = pat_count - 1
SI status == FERME ET pat_count = 0 ALORS
    V (pat_vide)
    V (fermer
FINSI
```

A partir du moment où un patient est installé dans la salle d'attente, il doit obligatoirement quitter la vaccinodrome en étant vacciner. Ainsi, seul la fin diffère du déroulement classique puisque après avoir décrementé le nombre de patient total dans le vaccinodrome, s'il s'avère que le patient était le dernier à quitter le vaccinodrome et que le vaccinodrome est fermé, il doit prévenir les médecins qu'il ne reste plus aucun patient et qu'ils peuvent, à leur tour, quitter le vaccinodrome. Il préviendra également `fermer.c` pour lui permettre de, peut-être, débloquer des médecins.

3.6 Médecins après fermeture

Le médecin est le seul à intérargir ici.

Champ	Type	Description
salle_m	asem_t	sémaphore représentant la salle d'attente (voir `vaccinodrome_t`)
status	vac_status_t	le status actuel de vaccinodrome (voir `vaccinodrome_t`)
salle_count	int	nombre de patient dans la salle d'attente (voir `vaccinodrome_t`)
pat_count	int	nombre de patients dans le vaccinodrome (voir `vaccinodrome_t`)
pat_vide	asem_t	sémaphore pour indiquer qu'il ne reste aucun patient (voir `vaccinodrome_t`)
med_count	int	nombre de médecins dans le vaccinodrome (voir `vaccinodrome_t`)
vide	asem_t	sémaphore pour indiquer que le vaccinodrome est vide (voir `vaccinodrome_t`)

```
TANTQUE VRAI
    SI status == FERME ET salle_count = 0 ALORS fin du médecin FINSI
    P (salle_m) // patient ou fermer
    SI status == FERME ET salle_count = 0 ALORS fin du médecin  FINSI
    traiter_patient()
FIN TANTQUE

SI pat_count == 0 ALORS V (pat_vide) FINSI
med_count = med_count - 1
P (pat_vide)
SI med_count == 0 ALORS V (vide) FINSI
```

Après la fermeture, les médecins continuent de traiter les patients qui étaient déjà dans l'enceinte du vaccinodrome avant la fermeture. Ainsi, tant qu'il y a des patients dans la salle d'attente, le médecin passe le sémaphore salle_m et traite un patient.

Or, s'il n'y a plus de patients, le semaphore est bloquant et les médecins ne peuvent pas finir. C'est ici que `fermer.c` intervient, je ne vais pas revenir sur son fonctionnement ici mais `fermer.c` permet de débloquer les médecins. En toute logique, quand `fermer.c` débloque les médecins, il modifie également le statut du vaccinodrome, il ne reste plus de patient dans la salle d'attente et donc le médecin sort de la boucle `TANTQUE`.

Après cela, s'il reste encore des patients (en cours de vaccination, et non dans la salle d'attente), les médecins doivent attendre. Une fois que le dernier patient a quitté le vaccinodrome, les médecins peuvent également quitter le vaccinodrome et décrémentent le compteur. Ainsi, le dernier médecin à quitter le vaccinodrome prévient `fermer.c` qu'il peut nettoyer le vaccinodrome.

4 Remarques sur l'implémentation

Tous les accès ou modifications effectuées concernant la mémoire partagée sont encadrés dans des sections critiques, sauf oubli de ma part ou exception, qui sera à ce moment là précisée dans les commentaires.

En ce qui concerne le choix d'un siège pour le patient et le choix d'un patient pour le médecin, j'exécute une boucle `FOR` qui itère dans mon tableau jusqu'à trouver le statut recherché. J'admet que ça n'est pas réaliste puisque ce sont toujours les premières cases du tableau qui sont utilisées. Pour palier à ce problème, on pourrait implémenter une file mais je n'ai pas souhaité l'implémenter par soucis de simplicité.

Concernant mon tableau `patient` et sa taille `n+m`. Je me suis décidé pour cette implémentation car d'après moi, j'aurai de toute manière été obligé d'utiliser deux tableaux, un de taille `n` pour représenter les sièges du vaccinodrome et un autre de taille `m` pour représenter les boxs des médecins. Or, cette imlémentation ne m'a pas enchanté et j'ai jugé qu'utiliser un tableau de taille `n+m` serait plus optimal.

En effet, il y a au maximum `n+m` patients à l'intérieur du vaccinodrome. Ainsi, j'utilise l'énumération (voir 2.1) pour spécifier quel statut je souhaite que chaque case de mon tableau ait. En combinant cela à mes compteurs, je suis assuré de ne jamais modifier une case de manière involontaire. De plus, utiliser un tableau me garantit un accès rapide aux cases souhaitées et lorsqu'un patient quitte le vaccinodrome, il me suffit de modifier le statut de la case jusqu'à ce qu'un nouveau patient vienne remplacer les données. Le médecin ira de lui même s'inscrire comme étant le médecin d'un patient et, de ce fait, il aura accès aux sémaphores ce qui assure une communication simple entre les deux protagonistes.

La fermeture telle que je la vois fonctionne de la manière suivante : avant de nettoyer la mémoire partagée, il faut attendre que le vaccinodrome soit vide, c'est à dire qu'il ne doit rester aucun médecin, ni patient. De la même manière, les médecins ne peuvent quitter le vaccinodrome uniquement lorsqu'il n'y a plus de patients dans le vaccinodrome.

C'est pour cela que, dans ma fermeture, je vérifie d'abord qu'il ne reste aucun patient (s'il en reste, j'attends) avant de vérifier qu'il ne reste aucun médecin (s'il en reste, je dois les débloquer, puis j'attends).

Les médecins, eux, bouclent tant que le vaccinodrome est ouvert ou qu'il reste des patients à traiter, mais une fois que le dernier patient a été traité, ils attendent indéfiniemment jusqu'à ce que la fermeture les débloque. Ensuite, ils vérifient qu'il ne reste aucun patient (s'il en reste, ils attendent) puis le dernier médecin autorise la fermeture à procéder au nettoyage du vaccinodrome.

Ainsi, lorsque le dernier patient quitte le vaccinodrome, il débloque la fermeture et les médecins qui attendaient que tous les patients aient quitté le vaccinodrome.

En revanche, je ne suis pas certain que l'attente des patients par les médecins soit utilisée, mais je laisse tout de même la gestion de ce cas dans mon code.

5 Remarques sur les jeux de tests

J'ai ajouté la commande `make long` dans le makefile qui permet de lancer un nombre prédéfini de fois `make test`. Le test est vraiment long à exécuter puisque j'ai attendu 45 minutes pour 100 tests.

5.1 Machine personnelle

Lors de l'execution sur ma machine personnelle (WSL2), j'ai rencontré 22 erreurs sur 250 tests. C'était exclusivement aux tests 190 et 200 (onze fois chacun) :

```
test-190.sh
==> Échec du test 'test-190' sur 'Reste 0 patients non terminés sur 1 attendus à 9030 ms'.
==> Voir détails dans le fichier test-190.log
==> Exit

test-200.sh
==> Échec du test 'test-200' sur 'Reste 0 patients non terminés sur 1 attendus à 7830 ms'.
==> Voir détails dans le fichier test-200.log
==> Exit
```

5.2 Turing

En revanche, lors de l'exécution sur turing (qui est à privilégier j'imagine), je n'ai échoué qu'une seule fois sur 250 tests et c'était au test 110 avec l'erreur suivante :

```
test-110.sh
==> Échec du test 'test-110' sur 'fermer : /dev/shm pas dans l'état initial'.
==> Voir détails dans le fichier test-110.log
==> Exit
```

Concernant les erreurs aux tests 190 et 200, je ne les ai jamais rencontré lors de l'exécution sur turing et je ne comprends pas trop d'où pourrait provenir mon erreur.

Je n'ai aucune idée d'où a pu provenir cette erreur au test 110 et je ne sais pas non plus comment y remedier. C'est la première fois qu'elle m'apparaît, j'espère que mon code n'est pas fautif et que c'est arrivé suite à l'exécution d'un programme par une autre personne sur le serveur ssh.

6 Conclusion

Je trouve que mon implémentation comporte beaucoup de sections critiques, peut-être trop. De ce fait, j'imagine que si la capacité d'accueil et la fréquentation du vaccinodrome venait à augmenter, on pourrait rencontrer des soucis de performances. Malheureusement, je ne sais pas comment y remedier, ni même si c'est possible.

J'ai notamment rencontré des difficultés lors de l'initialisation et de l'allocation de la mémoire partagée. C'était des erreurs plutôt absurde mais cela m'a demandé du temps afin de m'en rendre compte. J'ai également eu du mal à finaliser mon implémentation puisque j'avais tout simplement oublié d'implémenter la plupart des sections critiques ce qui me faisait régulièrement échouer certains tests.

J'imagine qu'il n'y a pas tellement de possibilités d'implémentation différente car la ligne directrice est assez claire, mais mon implémentation me semble tout de même être simple et logique à la fois et j'en suis plutôt fier. Notamment mes choix concernant la structure et le tableau de la mémoire partagée. En effet, le fait de m'être servis d'une unique structure avec un unique tableau pour représenter à la fois les sièges de la salle d'attente et les boxs des médecins me semble être optimal et je pense que cela facilite la communication entre un patient et son médecin. Après avoir conversé avec certains camarades qui eux ont utilisé plusieurs tableaux voir même plusieurs structures via des `mmap` différents, j'estime que mon implémentation n'est pas mauvaise.

Dans l'objectif d'une amélioration future, il faudrait commencer par apporter toutes les modifications que vous, en tant que correcteur, jugerez nécessaire. Ensuite, si c'est effectivement réalisable, il faudrait corriger la limite mentionnée au début de la conclusion. Il pourrait surtout être intéressant d'implémenter une version plus complète comme mentionnée lors de l'introduction : plusieurs vaccinodromes, des médecins et des patients identifiés via une base de données, l'implémentation d'un pass sanitaire, ... Pour accompagner une implémentation aussi importante et rendre ce projet plus réaliste, on pourrait également réaliser une version graphique.