#(Nom)
Note : 2.75/20

+108/1/49+

Programmation Concurrente

Contrôle Terminal

Durée totale : 1h30

Toute communication (orale, téléphonique, par messagerie, *etc.*) avec les autres étudiants est interdite. Aucun document autorisé.

Vous rendrez le sujet complet agrafé. Vous reporterez votre **NUMÉRO D'ÉTUDIANT** sur la première page (ci-dessous).

— Pour les parties rédigées, vous répondrez obligatoirement dans les parties prévues pour, et seulement en cas d'extrême nécessité sur les blancs en bas de pages (dernière page par exemple).

Consignes :

- Utilisez un **stylo à bille noir ou bleu foncé.**
- **Noircir ou bleuir** la/les cases, sans dépasser sur les autres cases !
- Pour corriger (dernier recours) : effacez proprement la case.
- Ne pas oublier de noter votre **numéro d'étudiant.**

Numéro étudiant à coder (8 chiffres, il est sur votre carte étudiant !)

- Notez-le ici :

Numéro d'étudiant :

[redacted]

.....

- Encodez-le ci-contre (chiffre des unités tout à droite, en remplaçant p de votre login par 1) : par exemple, pour un numéro *p*1234567, ie 11234567 vous grisez le 1 de la première colonne, le 1 de la deuxième, le 2 de la troisième...).

[redacted]

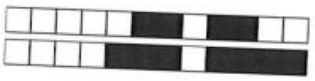
Rappels sur C++11 et les threads

Pour vous aider, voici un rappel de la syntaxe C++11 pour les threads :

```
// Création et attente de terminaison d'un thread :
int main () {
        // ...
        std::thread t(f, 42, std::ref(x));
        // ...
        t.join();
}

// Déclaration d'un mutex
std::mutex m;

// Verrouillage/déverrouillage d'un mutex :
m.lock();
// ...
m.unlock();

// Instantiation d'un verrou :
{
        std::unique_lock<std::mutex> l(m);
        // ...
}

// Opérations sur une variable de condition :
std::condition_variable c;
c.wait(l); // l de type verrou (std::unique_lock par exemple)
c.notify_one();
c.notify_all();

// Opérations sur une variable de condition :
std::condition_variable_any c;
c.wait(m); // m de type mutex
c.notify_one();
c.notify_all();
```

Exemple d'utilisation de std::queue

```
std::queue<int> f;
f.push(42); f.push(12);
cout << f.front(); // 42
cout << f.front(); // toujours 42
f.pop(); // Retire la première valeur
cout << f.front(); // 12
```



+108/3/47+

1 Administration système

Question 1 **(0.5 point)** Que représente la variable `PATH` ?

☐ Faux ☐ Partiel ☐ OK *Réservé* 0/0.5

On stock dans PATH les variables environnementales

Question 2 **(0.5 point)** Comment modifier cette variable `PATH` tout en conservant le contenu original ?

☐ Faux ☐ Partiel ☐ OK *Réservé* 0/0.5

On peut modifier la variable PATH à l'intérieur des fichier bash.* qui sont exécutés à chaque lancement de shell

Question 3 **(0.5 point)** Afin de pérenniser la mise à jour du `PATH`, on ajoute la ligne précédente au `.bashrc`. D'ailleurs, quelle est la différence entre sourcer (`source .bashrc`) et exécuter (`./mon-script.sh`) un script ?

☐ Faux ☐ Partiel ☐ OK *Réservé* 0.25/0.5

Sourcer change l'environnement dans le shell actuel alors que exécuter ouvre un nouveau shell avec l'environnement modifié

Question 4 **(0.5 point)** Que se passe t-il si on enlève les droits en lecture sur `/etc/passwd` et que l'on essaie de se connecter ? Pourquoi ?

☐ Faux ☐ Partiel ☐ OK *Réservé* 0.25/0.5

Si quelqu'un utilisateur Other essaiera de se connecter, il ne pourra pas car on ne pourra pas à accéder à la liste des utilisateurs donc on ne trouvera pas ces informations de l'utilisateur en question

Question 5 **(1 point)** Dans quel fichier sont stockés les *hash* des mots de passe ? Quel est l'intérêt d'avoir un fichier différent de `/etc/passwd` ? D'ailleurs comment est calculé le *hash* et pourquoi ?

☐0 ☐1 ☐2 ☐3 ☐4 *Réservé* 0.25/1

Les hash des mots de pass sont stocké dans /etc/shadow. Car peut arriver que 2 utilisateurs aient le même mot de passe, donc on crée un fichier avec ces hash pour augmentes la securités des comptes

On considère le fichier `-r--r--r-- 1 chaprot chaprot 160 Apr 12 14:26 fichier`

Question 6 **(0.5 point)** Avec quelle commande `chaprot` peut-il changer les droits du fichier pour s'ajouter le droit en écriture ? Donnez deux façons de le faire.

☐Faux ☐Partiel ☐OK *Réservé* 0/0.5

chmod g+w

chmod 464

Question 7 **(0.5 point)** Quels sont les autres utilisateurs qui peuvent changer les droits sur ce fichier ?

☐Faux ☐Partiel ☐OK *Réservé* 0.25/0.5

les utilisateurs du groupe chaprot et root

Question 8 **(1 point)** On suppose qu'un utilisateur `you` a accès à une machine appelée `REMOTE` (en utilisant le même protocole que celui qui vous a servi à accéder aux VMs en TP). Comment `you` se connecte t-il à la machine pour y exécuter des commandes ? Donnez une façon d'envoyer un fichier `myfile.txt` (initialement sur la machine locale) à la racine de son compte sur la machine `REMOTE`.

☐Faux ☐Partiel ☐OK *Réservé* 0/1

you @ etudiant

ssh "adresse ip de la VM" @ etudiant : myfile.txt

+108/5/45+

Question 9 **(0.5 point)** you souhaite maintenant installer le package `apache2`. Quelle commande permet de connaître le descriptif du package avant installation ? Quelle commande permet de savoir quels fichiers ont été installés par le package `apache2` ?

☐Faux ☐Partiel ☐OK *Réservé* 0/0.5

- .....
- dpkg -l apache2 .....

Question 10 **(0.5 point)** Pour finir, you souhaite mettre à jour son système. Comment faut-il procéder ?

☐Faux ☐Partiel ☐OK *Réservé* 0.5/0.5

apt update .....
apt upgrade .....

2 Ordonnancement

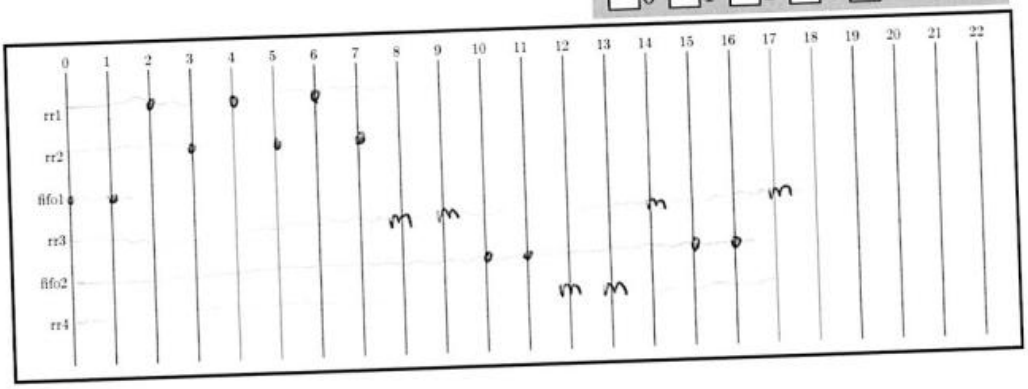
Nous utilisons un ordonnancement préemptif avec priorité (plus la valeur de priorité est importante, plus la tâche est prioritaire) qui se tient à chaque unité de temps sur un système monoprocesseur. Le quantum de temps est de **1 unité de temps**. Les tâches partagent un mutex M. Quand une tâche SCHED_FIFO ou SCHED_RR arrive alors qu'il y a déjà d'autres tâches de même priorité en attente, la nouvelle tâche est insérée en fin de liste.

Tâche	Date d'arrivée	Politique	Priorité	Durée	Remarque
fifo1	0	SCHED_FIFO	1	3	
rr1	2	SCHED_RR	3	2	
rr2	2	SCHED_RR	3	2	
rr3	8	SCHED_RR	5	4	prend le mutex M durant toute son exécution
fifo2	10	SCHED_FIFO	5	4	
rr4	12	SCHED_RR	10	2	prend le mutex M durant toute son exécution

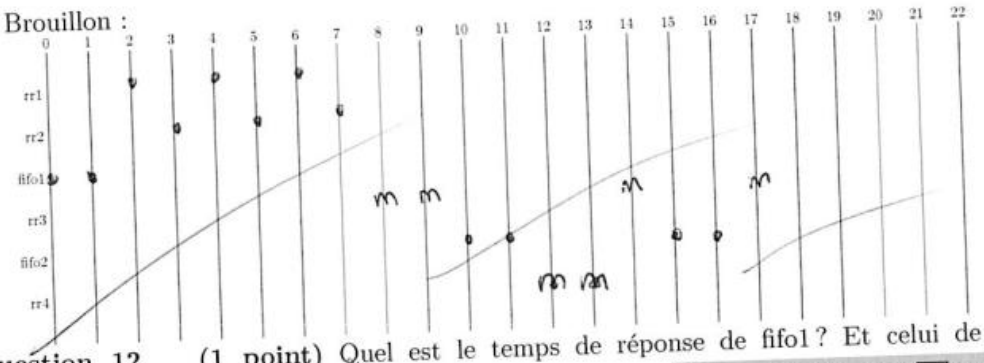
Faire l'ordonnancement de ces tâches. Vous pouvez utiliser le brouillon si besoin. Barrez la réponse incorrecte si vous répondez plusieurs fois. Si vous avez vu une autre présentation en TD (sur une seule ligne à la place d'une ligne par tâche), vous pouvez représenter votre ordonnancement de cette manière.

Question 11 **(2 points)**

☐0 ☐1 ☐2 ☐3 ☐4 *Réservé* 0/2



Brouillon :



Question 12 **(1 point)** Quel est le temps de réponse de fifo1 ? Et celui de rr4 ?

☐Faux ☐Partiel ☐OK *Réservé* 0.5/1

fifo1 = 2 rr4 = 2

Question 13 **(1 point)** Qu'est-ce qu'une inversion de priorité ? Y en a-t-il une ici ? Si oui à quel moment ?

☐Faux ☐Partiel ☐OK *Réservé* 0/1

On a une inversion de priorité si une tâche moins prioritaire détient un mutex partagé avec une tâche plus prioritaire mais qui ne pourra pas s'exécuter tant que le mutex n'est pas relâché. Ici il n'y a pas une inversion de tâche à cause des mutex, mais à l'instant 14 rr3 s'exécute alors que fifo2 de priorité plus grande devrait finir son exécution mais il est mis en attente

3 Threads

La plupart des parties de cette section peuvent être traitées indépendamment. Lisez bien les questions jusqu'à la fin !

`nb_athletes` athlètes (modélisés chacun par un thread `athlete`) vont participer à une course. Cette course se déroule en plusieurs étapes comme suit :

1. Chaque athlète se prépare pendant un certain temps avant de se rendre sur la ligne de départ.
2. Une fois que tous les athlètes sont arrivés sur la ligne de départ, le départ peut être donné.
3. La première étape de la course consiste à récupérer un numéro de dossard en passant un tourniquet (modélisé par un thread `tourniquet`) parmi `nb_tourniquets`.
4. Pour la seconde étape de la course, chaque athlète arrive dans un couloir d'une piste d'athlétisme avec un témoin sur sa droite (partagé avec le concurrent à sa droite[108]) et un autre témoin sur sa gauche (partagé avec le concurrent à sa gauche). Il doit prendre les deux témoins pour courir un tour de piste puis les redéposer.
5. Pour finir, un jury (modélisé par un thread `jury`) note les résultats des coureurs une fois la ligne d'arrivée passée. Chaque coureur peut consulter son classement (en même temps que d'autres coureurs et sauf si le jury est en train de le modifier) sans avoir le droit de modifier les résultats. Comme les athlètes sont pressés de connaitre leur classement, il est possible qu'un coureur tente de consulter son résultat avant que le jury ne l'ai enregistré : ce n'est pas grave, il pourra le relire plus tard. Afin d'effectuer les opérations couteuses hors des sections critiques, on vous conseille de découper les opérations de cette manière : 1) `resultats.debut_consultation()` ; 2) Consultation effective et 3) `resultats.fin_consultation()` (de manière similaire pour le `jury` afin de noter les résultats).

Voici un pseudo code correspondant à ce descriptif :

```
// id est le numéro du thread athlete
void athlete(int id, ...) {
  // 1) Se préparer

  // 2) Attendre le début de la course
  cout << "J'attends␣que␣tout␣le␣monde␣soit␣prêt" << endl;
  ...
  cout << "C'est␣parti␣!" << endl;

  // 3) Passer un des tourniquets
  ...
  cout << "Tourniquet␣passé␣pour␣" << id << endl;

  // 4) Prendre le témoin à droite et le témoin à gauche
  ...
  cout << "Athlete␣" << id << "␣a␣eu␣les␣témoins" << endl;
  // Courir un tour puis rendre les deux témoins
  ...
  cout << "Athlete␣" << id << "␣a␣rendu␣les␣témoins" << endl;

  // 5) Consulter son résultat
  ...
}

// id est le numéro du thread tourniquet
void tourniquet(int id, ...) {
  while (pasfini){
```

108. "En athlétisme, le témoin est un objet que se transmettent les coéquipiers dans des courses de relais, notamment dans les disciplines olympiques du 4 × 100 mètres et du 4 × 400 mètres". Ici pas de relais mais on utilise quand même **deux** témoins pour courir.

```
        // Préparer un nouveau dossard
  }
}

void jury(...) {
  for (int i=0; i<nb_athletes; i++){
        // Attendre arrivée d'un athlète puis noter son résultat
        ...
}
```

Question 14 **(2 points)** Identifiez à quels problèmes classiques de concurrence correspondent les étapes 2), 3), 4) et 5)

0/2

■0 □1 □2 □3 □4 *Réservé*

Dans ces etapes 3) et 4) et 5) on se retrouve dans des cas d'accès concurrents et de problèmes d'accès concurrent des données partagés. A l'etape 2) on pourrait avoir des problèmes d'attente active

Pour chacun de ces problèmes, il vous est demandé de proposer une solution (structure de données, code...) sous la forme d'un moniteur de Hoare. Il faut bien entendu une solution sans risque de deadlock ni de famine et qui soit la plus efficace possible. **Les quatre questions qui suivent sont indépendantes, donc ne restez pas bloqué sur une question.**

Question 15 **(1.5 points)** Proposez une solution sous la forme d'un moniteur de Hoare pour gérer l'étape 2), la ligne de départ. Explicitez comment modifier un thread `athlete` pour utiliser le moniteur.

☒0 ☐1 ☐2 ☐3 ☐4 *Réservé* 0/1.5

```
vector<thread> athlete;
class Depart {                          void athlete(int id, bool pret)
private:
mutex m;                                {  //
int NB_PARTICIPANTS;
bool tous_prets = false;
condition_variable_any c;
public:
Depart(int nb_participants){
    NB_PARTICIPANTS = nb_participants
for(int i=0; i<NB_PARTICIPANTS; i++)
{athlete[i] = thread(athlete, i, false)
}
for(int i=0; i<NB_PARTICIPANTS; i++)
{athlete[i].join();}

bool sont_prets(bool fin_preparation){
m.lock();
tous_prets = fin_preparation;
c.notify_all();
m.unlock();
return tous_prets;}
bool get_prets(){
for(int i=0; i<NB_PARTICIPANTS; i++){
while(athlete[i].pret == false){
    c.wait(m)}
}
return true;
}
```

Question 16 **(1.5 points)** Proposez une solution sous la forme d'un moniteur de Hoare pour gérer l'étape 3), les tourniquets. Explicitez comment modifier un thread `athlete` et un thread `tourniquet` pour utiliser le moniteur.

☒0 ☐1 ☐2 ☐3 ☐4 *Réservé*

0/1.5

On...ma dirige...la...fonction.....athlete...de...façon..qu'elle...prenne
en....paramètre....un....boolean.....qui....renvoie....s'il...a...pu
.ou...pas....un....tourniquet.....

class..Passage.Tourniquet.{.....
private:
.mutex.m;.....
.condition.-.variable.-any.c;.....
..int....NB.-.TOURNIQUETS.;.....
.bool.....pas.-fini.=.true;.....
.Passage.Tourniquet.().{.....

Question 17 **(2 points)** Proposez une solution *efficace* sous la forme d'un moniteur de Hoare pour gérer l'étape 4), l'accès aux témoins. Explicitez comment modifier un thread `athlete` pour utiliser le moniteur.

☒0 ☐1 ☐2 ☐3 ☐4 *Réservé*

0/2

```
void athlete (int id, bool et_prat, bool temoin_d, bool temoin_g)
{ // ... }

class AccesAuxTemoins {
private:
mutex m;

condition_variable cv_d, cv_g;
bool t_d, t_g;
bool peut_partir = false;

public:

bool temoin_droite() {
while (!t_d) {
cv_d.wait(m); }
return true; }

bool temoin_gauche() {
while (!t_g)
{ cv_g.wait(m); }
return true; }

bool get_t_d() {
m.lock();
t_d = true;
m.unlock();
cv_d.notify_all();
return t_d;

bool get_t_g() {
m.lock();
t_g = true;
m.unlock();
cv_g.notify_all();
return t_g;
}

bool etape4() {
if (get_t_d() && get_t_g())
return true;
return false;
```

Question 18 **(3 points)** Proposez une solution *efficace* sous la forme d'un moniteur de Hoare pour gérer l'étape 5), la gestion des résultats. Explicitez comment modifier un thread `athlete` et le thread `jury` pour utiliser le moniteur.

0.75/3

☐0 ☒1 ☐2 ☐3 ☐4 *Réservé*

```
thread Jury;
class Consultation {                    void Jury(bool a_lire)
private:                                { //... }
mutex m;
condition_variable cv;
bool consultable = false;
public:
Consultation() {
Jury = thread(jury, false);
consultable = false; }
bool get_consultable(bool est_consultable)
m.lock();
consultable = est_consultable;
m.unlock();
cv.notify_all();
return true; }
bool attente() {
  m.lock()
while(!consultable)
    cv.wait(m);
  m.unlock()
return true;
}
```

Question 19 **(1.5 points)** Écrire le code du `main` qui instancie *tous* les threads et les moniteurs dont vous avez besoin. Assurez-vous que ce code termine proprement.

☑ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 *Réservé*

0/1.5