	Ministère de l'enseignement supérieur et de la recherche scientifique Ecole Nationale Polytechnique (ENP) 2ème Année Data Science et Intelligence Artificielle	Date : 27/10/2025

Module : Systèmes Distribués

TP 1 : Horloges Logiques et Coupures des Systèmes

Objectifs :

- ✓ Comprendre et implémenter les horloges logiques (Lamport, vectorielles et matricielles).
- ✓ Appliquer ces horloges dans des simulations de communication distribuée entre des processus communiquant dans un système réparti, en respectant la propriété FIFO des canaux de communication.
- ✓ Assigner une estampille scalaire modifiée à chaque événement et n'afficher que les messages permettant de simuler une délivrance causale.
- ✓ Analyser et visualiser des coupures cohérentes dans un système distribué.

Quelques prérequis :

- ✓ Avoir des connaissances de base en Java.
- ✓ Savoir manipuler les structures de données notamment les tableaux et les listes.
- ✓ Comprendre les concepts fondamentaux des systèmes distribués.
- ✓ Maitriser les algorithmes de datation et leur déroulement.

Outils et environnement de travail

- Java Development Kit (JDK) : pour exécuter les programmes Java.
- Un compilateur C (GCC) : pour compiler et exécuter le code C.
- Un IDE (IntelliJ IDEA , VS Code , Eclipse ou NetBeans) : pour développer le code.
- Graphviz (optionnel) : pour générer automatiquement et visualiser les graphes de causalité.

Bibliothèques Java utilisées

- java.util.List : pour la gestion des listes de messages.

- java.util.Arrays : pour manipuler les tableaux.

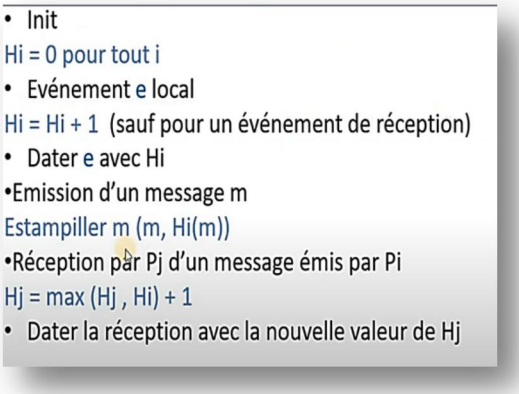
I. Horloges de Lamport

Rappel

L'algorithme de Lamport pour la **datation des événements** permet **d'ordonner partiellement** les événements dans un système distribué. Il est utilisé pour éviter les incohérences dans des systèmes répartis comme les BDD distribuées ou la gestion des transactions. Il repose sur l'utilisation d'une horloge logique, représentée par un **compteur entier** sur chaque site, servant **d'estampille temporelle** pour chaque événement.

Algorithme :

1. *Initialiser toutes les horloges à 0 Hi=0;*
2. *Événement local :*
   - *Incrémenter l'horloge locale : $Hi = Hi + 1$;*
   - *Associer l'estampille temporelle Hi à l'événement;*
3. *Émission d'un message m:*
   - *Estampiller le message avec la valeur actuelle de l'horloge Hi;*
4. *Réception d'un message m par un processus Pj:*
   - *Mettre à jour l'horloge de Pj: $Hj = max(Hj, Hi) + 1$;*
   - *Associer la nouvelle valeur de Hj à l'événement de réception;*



Travail à réaliser :

L'objectif étant de **créer trois processus** (P1, P2, P3), chacun doté d'une horloge de **Lamport initialisée à 0,** afin d'observer et **d'afficher l'évolution des horloges** lors des événements internes et des échanges de messages entre les processus. Et sachant que le scénario d'échange de messages entre les processus est le suivant:

- **Événement 1 :** P1 émit un message à P2
- **Événement 2 :** P2 reçoit le message de P1 et met à jour son horloge
- **Événement 3 :** P2 émit un message à P3
- **Événement 4 :** P3 reçoit le message de P2 et met à jour son horloge
- **Événement 5 :** P1 effectue un événement interne

- **Événement 6 :** P3 envoie un message à P1
- **Événement 7 :** P1 reçoit le message de P3 et met à jour son horloge.

Créer une classe **LamportClock** et implémenter les trois processus. Simuler les évènements (internes/locaux, émission d'un message, et la réception) en implémentant **trois méthodes** pour traiter chaque cas, et afficher dans un message l'état des horloges des trois processus à chaque événement, pour observer leur évolution.

II. Historique d'un évènement

Rappel

L'historique d'un événement **e'**, noté **hist(e')= {e} | e------> e',** est l'ensemble des événements qui le précèdent causalement, *i.e.* tous les événements qui ont influencé e' **directement ou indirectemnt,** permettant de **suiivre les dépendances causales** et de comprendre l'origine des événements dans un système distribué..
Un événement e précède causalement un événement e' ($e \rightarrow e'$) si l'un des trois cas suivants est vérifié :

1. **Précédence locale : Si e et e' appartiennent au même processus et que e s'est produit avant e'.**
2. **Échange de message : Si e est l'envoi d'un message et e' est la réception de ce message.**
3. **Transitivité** : **S'il existe un événement e" tel que e précède e" et e" précède e'.**

Travail à réaliser :

Modifier le code afin de capturer l'historique des évènements dans chaque processus :

1. Conserver et stocker les événements causalement précédents un évènement *e* grâce à une **liste** d'historique dans chaque processus.
2. **Afficher l'historique** des dépendances causales pour chaque événement à la fin de l'exécution du programme.

Bon Courage!