Projet de Groupe : "Eco-Sensing" – Simulateur de Réseau de Capteurs Contraints

Durée : 1 semaine

Thématiques abordées :

1. **Structures de données dynamiques** (Listes chaînées & Pointeurs)
2. **Gestion de fichiers binaires** (Persistance des données)
3. **Algorithmique & Simulation physique** (Modélisation énergétique)

1. Contexte (Mise en situation)

Vous êtes ingénieurs en systèmes embarqués. Vous devez développer le logiciel de simulation pour un nouveau type de capteur agricole autonome (IoT). Ce capteur mesure des données (température/humidité), les stocke dans une mémoire tampon limitée (buffer), et tente de les transmettre à une station de base.

Le défi : La batterie est critique et la mémoire RAM est minuscule.

2. Cahier des Charges Technique

Thématique 1 : Gestion de la Mémoire et Structures (Pointeurs)

Contrairement aux exercices simples, vous n'avez pas le droit d'utiliser de tableaux statiques (int tab[100]) pour stocker les données. Vous devez simuler une mémoire dynamique.

- **Structure Paquet :** Chaque mesure est un paquet contenant :
  - int id : Identifiant unique.
  - float valeur : La mesure relevée.
  - long timestamp : Le temps de la mesure.
  - struct Paquet *suivant : Pointeur vers le paquet suivant.
- **Structure Capteur :**
  - float batterie : Niveau d'énergie (en Joules).
  - float x, y : Position du capteur (pour le calcul de distance).
  - struct Paquet *buffer_tete : Tête de la liste chaînée des paquets en attente.
  - int buffer_usage : Nombre actuel de paquets dans la file.

- **Contrainte "Hardware" :** Le capteur a une mémoire limitée à **5 paquets maximum**. Si un 6ème paquet est généré, le paquet le plus ancien (en tête de liste) doit être supprimé et sa mémoire libérée (free) avant d'ajouter le nouveau.

Thématique 2 : Simulation Physique (Maths & Algorithmique)

Vous devez implémenter une boucle de simulation (ex: while(batterie > 0)) qui représente le temps qui passe.

- Transmission et Coût Énergétique :
  À chaque cycle, le capteur tente d'envoyer les paquets de son buffer vers la station de base (située en 0,0).
  L'énergie consommée pour l'envoi d'un paquet suit cette formule précise (modèle de dissipation d'énergie radio) :

$$E_{tx} = E_{elec} + E_{amp} \times d^2$$

  - d : Distance euclidienne entre le capteur (x,y) et la station (0,0), calculée avec sqrt et pow de <math.h>.
  - E_{elec} = 0.05 Joules (Coût circuit).
  - E_{amp} = 0.01 Joules/m² (Coût amplification).
- **Condition d'arrêt :** Si la batterie tombe à 0, le capteur est considéré comme "mort". Le programme doit afficher le nombre total de paquets transmis avant la panne.

Thématique 3 : Persistance des Données (Fichiers Binaires)

Les systèmes embarqués n'utilisent pas de fichiers texte (trop lourds à traiter). Vous devez sauvegarder l'état du capteur.

- Implémentez une fonction sauvegarder_etat(Capteur *c) qui écrit la structure du capteur et tout son buffer (liste chaînée) dans un fichier binaire save.bin en utilisant fwrite.
- Implémentez une fonction charger_etat(Capteur *c) qui lit ce fichier avec fread pour restaurer la simulation exactement là où elle s'était arrêtée.
- *Note :* La sauvegarde doit permettre de reprendre la simulation même si on ferme le programme.

Gestion de la Production des Données

Vous devez implémenter une fonction clé nommée void produire_paquet(Capteur *c) qui sera appelée à chaque étape de votre boucle de simulation. Cette fonction doit

respecter scrupuleusement le comportement mémoire d'un microcontrôleur réel selon les règles suivantes :

1. Génération Aléatoire et Allocation Dynamique

- Le capteur ne doit pas stocker des données pré-enregistrées. Vous devez générer une valeur de mesure (ex: humidité entre 0.0 et 100.0) aléatoirement à l'aide de rand().
- **Interdiction formelle** d'utiliser un tableau statique (ex: Paquet buffer[5]) pour stocker les mesures. Chaque nouveau paquet doit être un objet indépendant créé dynamiquement dans le tas (Heap) via malloc.

2. Gestion de la Saturation Mémoire (Le "Sliding Window") Votre capteur possède une mémoire physique limitée strictement à **5 paquets**. Avant d'ajouter le nouveau paquet créé à votre liste chaînée, vous devez vérifier l'état du buffer :

- **Si le buffer contient moins de 5 paquets :** Ajoutez simplement le nouveau paquet à la fin de la liste chaînée.
- **Si le buffer est plein (5 paquets) :** Vous devez simuler l'écrasement des anciennes données.
    1. Identifiez le paquet le plus ancien (celui en tête de liste).
    2. Détachez-le de la liste.
    3. **Impératif :** Libérez sa mémoire avec free(). *Tout oubli ici sera considéré comme une fuite mémoire grave.*
    4. Ajoutez ensuite le nouveau paquet à la fin de la liste.

3. Traçabilité (Preuve de fonctionnement) Pour valider que votre gestion de la mémoire fonctionne, votre programme doit afficher une alerte console spécifique à chaque fois qu'un écrasement a lieu :

> *"ALERTE : Mémoire saturée. Suppression du paquet ID [x] pour libérer de l'espace."*

3. Modalités de Rendu & Évaluation

Pour valider ce projet, le groupe doit fournir :

1. **Le Code Source (.c et .h) :**
    - Le code doit être modulaire (fichiers séparés pour la simulation, la gestion mémoire et les fichiers).

2. **La Preuve de Gestion Mémoire (Vidéo 5 min à publier sur youtube) :**
    - Une courte capture d'écran vidéo où vous montrez, via des logs ou un débogueur, le moment précis où le buffer est plein (5 paquets) et où le programme supprime le plus vieux paquet (free) pour faire de la place au nouveau.
3. **Le "Crash Test" :**
    - Votre programme doit générer automatiquement un fichier de log texte (log.txt) contenant l'évolution de la batterie ligne par ligne.
    - Exemple : Temps: 10s | Batterie: 45.2J | Paquets en attente: 3
4. Le code source est à mettre sur Github
5. Vous devez produire un rapport de 1 page maximum (doit contenir les difficultés, les choix techniques, les solutions apportées, les liens youtube et github)