MODULE 06
SÉANCE SYSTÈME 03
TP D'INFORMATIQUE
Durée 2h30

La classe IRSnifferCirpark : détection d'un capteur

BLOC DE COMPÉTENCES
U6 - VALORISATION DE LA DONNÉE ET CYBERSÉCURITÉ

COMPÉTENCE(S)
C08 - CODER

OBJECTIF PÉDAGOGIQUE
Dans cette partie, vous allez créer une nouvelle classe, envoyer des trames sur le réseau et recevoir des réponses et stocker les résultats dans un tableau de données.

CONNAISSANCES ISSUES DU RÉFÉRENTIEL
- Programmation Orientée Objet — Niveau 3

CONNAISSANCES OPÉRATIONNALISÉES
- Utiliser une classe — Niveau 2
- Créer une classe — Niveau 2

Gaëtan GEORGES
Pascal HAGOT
David IERVESE

Lycée Louis Armand
173 Bd de Strasbourg
Nogent sur Marne

Cahier des charges et analyse

Cahier des charges

L'objectif de cette partie est de mettre en place un sniffer Cirpark. Un sniffer est un outil qui permet de découvrir un réseau et de trouver par lui-même tous les éléments qui le compose.

Notre sniffer Cirpark aura pour mission de découvrir tous les capteurs présents dans le parking. Pour cela, il interrogera tous les capteurs possible et attendra une réponse. Si le capteur répond, il l'ajoutera à la liste des capteurs trouvés.

Analyse

> Expliquer le but de notre sniffer Cirpark.

Il permet de recuperer tout les elements cirparks tout seul

> Donner le nombre d'octets d'une adresse de capteurs SP.

4 à 5

> En déduire par un calcul le nombre d'adresse possible pour les capteurs SP.

65536 car il y a pour adresse max FFFF

> Supposons que la découverte d'un seul capteur prenne 200ms. Calculer le temps nécessaire pour les découvrir tous.

65536 * 0.2 = 13107 secondes
soit 3h et 38 minutes

> En déduire l'intérêt de faire une chercher des capteurs par plage d'adresse (une plage d'adresse est définie par une adresse de départ et une adresse de fin)

Pour eviter de perdre du tempsa a rechercher des adresse inexistante

Architecture matérielle

Votre sniffer Cirpark ne communiquera par directement sur le réseau RS485 des capteurs. Il passera par un TCPRS qui sert d'interface entre le réseau RS485 des capteurs et le réseau Ethernet. Le TCPRS possède un serveur UDP qui reçoit des trames provenant du réseau Ethernet. Ces trames contiennent un message pour un capteur. Le TCPRS transmet ce message sur le réseau RS485 des capteurs. Si le capteur existe et que le message est correct, le capteur répond. Ensuite, le TCPRS retransmet la réponse du capteur au client UDP.

Dessiner le diagramme de déploiement en indiquant les 3 éléments présents, leur multiplicité et les protocoles qui les relient.

Le protocole de communication

D'après la documentation du protocole des capteurs de type SP, quelle trame permet de connaître le type de capteur ? Quels sont les différents champs de cette trame ?

0x05 on type et version

Comment se calcule le champ bcc ?

Adrh + adrL + fonction = bcc

Donner la trame UDP à envoyer au capteur 1E3F pour connaître son type. Calculer le bcc.

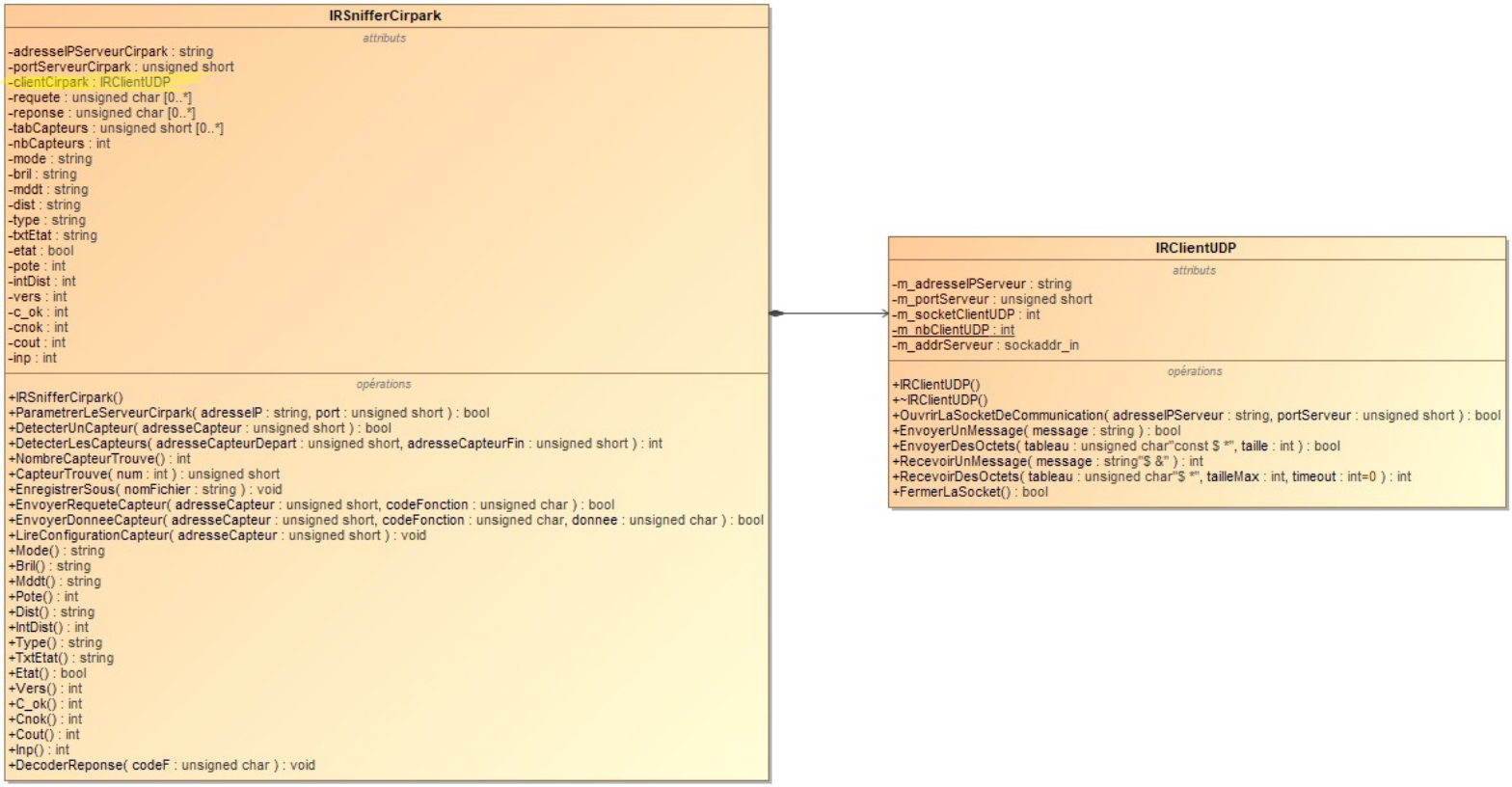
1E 3F 05 62

Ce sont ces trames là que nous allons envoyer pour découvrir les éléments présents sur le réseau Cirpark. Si nous obtenons une réponse, cela signifie que le capteur existe. Si nous n'avons pas de réponse, on en déduit que le capteur n'est pas présent.

Ecriture de la classe IRSnifferCirpark

Le diagramme de classe

Soit le diagramme de classe complet suivant :



La classe IRClientUDP vous est fournie.

> Donner le nom du lien qui relie les classes IRSnifferCirpark et IRClientUDP.

IRSniffer est composé de IRClientUDP

> Quelle est la conséquence de ce lien pour la classe IRSnifferCirpark ? Surligner la ligne correspondante.

On creer un objet de la classe IRClientTCP

Nous allons écrire la classe IRClientCirpark progressivement et la tester progressivement également.

Création d'un nouveau projet avec XE6

> Avec XE6, créer un projet en mode console. Enregistrer votre projet sous... dans un répertoire dédié.

> Ajouter dans votre répertoire de projet les sources de la classe IRClientUDP. Puis ajouter dans votre projet ces fichiers.

Ajout des fichiers pour la classe IRSnifferCirpark

Dans votre projet, créer une nouvelle Unité. Ceci à pour conséquence de créer 2 fichiers (.h et .cpp) que vous utiliserez pour créer la classe IRSnifferCirpark. Sauvegarder le fichier Unit1.h sous le nom « IRSnifferCirpark.h ». Le fichier Unit1.cpp est alors automatiquement renommé.

Déclaration partielle de la classe IRSnifferCirpark

Dans le fichier IRSnifferCirpark.h, déclarer la classe IRSnifferCirpark. On commencera avec seulement les attributs et méthodes suivantes :

- Les attributs : adresseIPServeurCirpark, portServeurCirpark, clientCirpark, requete (un tableau de 10 octets non signés), reponse (un tableau de 20 octets non signés), tabCapteurs (un tableau de 100 adresse de capteurs sur 16 bits), nbCapteurs.
- Les méthodes : Le constructeur, ParametrerLeServeurCirpark(), DetecterUnCapteur(), DetecterLesCapteurs(), NombreCapteurTrouve(), CapteurTrouve().

Définition des méthodes

Le constructeur

Le constructeur initialise les attributs de la classe.

Dans la définition du constructeur, attribuer la valeur « 0.0.0.0 » à adresseIPServeurCirpark, 10001 à portServeurCirpark, mettre des 0 dans toutes les cases des tableaux requete, reponse et tabCapteurs (vous pouvez faire une boucle for pour initialiser à 0 tous les éléments d'un tableau ou bien utiliser la fonction memset) et enfin 0 dans nbCapteurs.

Vérifier qu'il n'y pas d'erreur de compilation.

La méthode ParametrerLeServeurCirpark()

Cette méthode permet à l'utilisateur d'indiquer l'adresse IP et le port du module TCPRS avec lequel on souhaite communiquer. Cette méthode créer également la socket de communication.

On vous donne ci-dessous le code partiel de cette méthode :

```
1: bool IRSnifferCirpark::ParametrerLeServeurCirpark(string adresseIP, unsigned short port)
2: {
3:   adresseIPServeurCirpark = ___adresseip___;
4:   portServeurCirpark = ___port_____;
5:   ___bool___ ___retour___ = clientCirpark.OuvrirLaSocketDeCommunication(adresseIPServeurCirpark, portServeurCirpark);
6:   return ___retour___;
7: }
```

Compléter le code ci-dessus en vous aidant des indications qui suivent :

ligne 3 : l'attribut adresseIPServeurCirpark prend la valeur du 1er argument de la méthode (adresseIP).

ligne 4 : l'attribut portServeurCirpark prend la valeur du 2ème argument de la méthode (port).

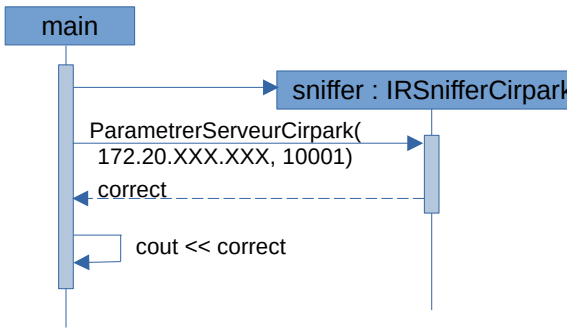
Ligne 5 : La valeur de retour de la méthode OuvrirLaSocketDeCommunication() est affectée à un booléen nommé « *retour* » qu'il faut créer.

Ligne 6 : La méthode renvoie la valeur du booléen « *retour* ».

Test du constructeur et de la méthode ParametrerLeServeurCirpark()

A partir du diagramme de séquence ci-contre, écrire le programme principal main.

Lors de l'exécution, vérifier que votre programme affiche « 1 ». Ceci indique que la méthode ParametrerServeurCirpark() a correctement fonctionné.



La méthode DetecterUnCapteur()

Maintenant que la socket de communication est créée avec la méthode ParametrerServeurCirpark(), il est temps d'envoyer une trame au serveur pour découvrir si un capteur est présent ou pas. Nous arrivons ici au coeur de notre projet et c'est le but de la méthode DetecterUnCapteur().

Pour savoir si un capteur est présent, nous allons utiliser la fonction 0x05.

D'après la documentation, de combien d'octet est composée la requête à envoyer aux capteurs SP3 ? Donner leur signification.

4 octets, 2 pour l'addresse du capteur une pour la fonction et un pour le bcc

En supposant que le capteur est 0x1EB9, donner la trame complète qu'il faut envoyer.

adrh	adrl	fonction	bcc
1E	B9	05	DC

Le capteur qui recevra cette requête renverra son type de capteur s'il est présent.

Donner le nom de l'argument de la méthode DetecterUnCapteur() et son type.

DetecterUnCapteur est un bool et en argument adressecapteur qui unsigned short

Décomposons maintenant les instructions de la méthode DetecterUnCapteur().

En C++, donner l'opération permettant d'extraire l'octet de poids fort de l'adresse du capteur passé en argument dans le 1er octet de la requête :

```
requete[0] = adresse & (0xFF << 8)
```

En C++, donner l'opération permettant d'extraire le l'octet de poids faible de l'adresse du capteur passé en argument dans le second octet de la requête :

```
requete[1] = adresse & 0xFF
```

En C++, compléter maintenant le troisième octet avec la valeur de la fonction :

```
requete[2] = 0x05
```

Pour terminer, compléter le quatrième octet avec la valeur du bcc (la somme algébrique des précédents octets :

```
requete[3] =   requete 0 + 1 +2
```

En C++, appeler la méthode EnvoyerDesOctets() avec l'objet clientCirpark. Passer en argument le tableau « requete » ainsi que le nombre d'octets à envoyer.

```
EnvoyerDesOctets(requete);
```

En C++, appeler la méthode RecevoirDesOctets(). Passer en argument le tableau « reponse » avec sa taille max de 20 octets. Stocker la valeur renvoyée dans un entier « nbOctets » que vous déclarerez.

```
char reponse[20];
nbOctets = RecevoirDesOctets(reponse,20)
```

Terminons la méthode. En C++, retourner « false » si nbOctets vaut 0 ou « true » sinon.

```
if(nbOctet > 1) return 1;

return 0;
```

Écrire la définition complète de la méthode DetecterUnCapteur() dans le fichier IRSnifferCirpark.cpp.

Testons maintenant cette méthode.

Dans le main(), appeler cette méthode avec l'objet « sniffer » et afficher la valeur que cette méthode renvoie. Pour que votre test soit complet, passer en argument un capteur qui existe puis un capteur qui n'existe pas. Vérifier avec Wireshark que la trame de requête est correcte et que vous obtenez une trame de réponse lorsque le capteur existe.

Arrivé à ce stade, nous avons écrit un programme qui est capable de savoir si un capteur est présent ou non. Passons maintenant à l'étape suivante : créer une méthode qui vérifie automatiquement si des capteurs sont présents dans une plage d'adresse.