BTS INFORMATIQUE ET RESEAUX
POUR L'INDUSTRIE ET LES SERVICES TECHNIQUES

Session 2007

Épreuve E.4
Étude d'un Système Informatisé

Prélèvements sur sites volcaniques

Sujet

Durée : 6h00 Coefficient 5

"Calculatrice autorisée (conformément à la circulaire n° 99-186 du 16 novembre 1999)."

Aucun document autorisé.

Ce document comprend 45 pages composées de :

Sujet	: pages 1 à 18	sur papier rose
Annexes	: pages 1 à 14	sur papier vert
Document réponse	: pages 1 à 13	sur papier blanc

à rendre obligatoirement (même vierge) .

Toutes les réponses sont à fournir sur le livret « Document Réponse » à l'exclusion de tout autre support.

Les réponses doivent être exclusivement fournies dans les emplacements prévus à cet effet. Si nécessaire le candidat a la possibilité de rectifier ses réponses sur la page non imprimée en regard.

On ne justifiera une réponse que si le document le demande.

Temps conseillés et barème indicatif :

Lecture du sujet	**30 minutes**		
B. Le robot mobile	**90 minutes**	**23 questions**	**40 points**
C. Contrôle-Commande	**90 minutes**	**8 questions**	**25 points**
D. Communication et réseau	**90 minutes**	**14 questions**	**35 points**
Relecture	**60 minutes**		

A. Présentation du Système

A.1. Expression du besoin

Depuis l'antiquité, l'activité volcanique a été examinée et répertoriée du fait même de l'impact des éruptions sur l'activité humaine. Les habitants des zones à risques vivaient dans la peur d'éruptions aussi soudaines qu'imprévisibles qui détruisaient leurs champs et leurs villages et causaient d'importantes pertes humaines.

Quelques personnes s'intéressèrent alors de plus près aux volcans et s'aventurèrent auprès des cratères dans le but de comprendre les phénomènes volcaniques. Dès lors, de nombreux scientifiques payèrent de leur vie des observations faites à proximité des zones d'éruptions.

La fin du XX^{ème} siècle notamment a coûté la vie à de nombreux vulcanologues du fait d'un regain en fréquence comme en intensité de l'activité volcanique.

1500 volcans potentiellement en activité sont répertoriés sur la planète, dont 500 sont effectivement rentrés en activité au cours du XX^{ème} siècle et 70 actifs et en phase éruptive à l'heure actuelle. 10% de la population terrestre vit sous la menace des volcans qui ont coûté la vie à 30.000 personnes au cours des 50 dernières années.

Des avancées en matière de prévision et de prédiction des éruptions ont été faites récemment grâce à l'étude des grandes éruptions. La surveillance des zones à risques est cependant insuffisante et une catastrophe volcanique peut intervenir à tout moment.

C'est ce constat, allié avec des avancées technologiques récentes dans le domaine de la robotique qui a conduit la Communauté Européenne à mener un nouveau projet nommé ROBOVOLC dont le but est l'étude et la réalisation d'un robot mobile pour l'exploration volcanique. Ce projet débuté en mars 2000 rassemble plusieurs partenaires dont des universités, des laboratoires de recherche et des entreprises privées.

L'objectif majeur du robot étudié est de minimiser les risques pris par les vulcanologues et les techniciens impliqués dans des activités à proximité des cratères en phase éruptive. Il est à noter que les observations les plus intéressantes sont faites au cours des phases paroxysmiques des éruptions, au cours desquelles le risque est bien entendu maximum.

A.2. Système technique

Le cahier des charges établi par l'ensemble des partenaires spécifie que le robot doit être capable de:

- S'approcher d'un cratère actif,
- Collecter des échantillons de rejets éruptifs,
- Collecter des échantillons gazeux,
- Collecter des données physiques et chimiques,
- Surveiller une bouche de cratère.

Lors de l'étude préliminaire, et pour atteindre ces objectifs, l'équipe de ROBOVOLC a décomposé le système en plusieurs sous-systèmes:

- Sous-système de déplacement,
- Sous-système de navigation,
- Sous-système de vision,
- Sous-système de prélèvement,
- Sous-système de communication,
- Sous-système de supervision.

Le sujet s'intéressera à la problématique du déplacement d'un point A à un point B, sur un site volcanique, ce qui l'amènera à aborder les sous-systèmes de déplacement, de navigation, de communication et de supervision du robot.

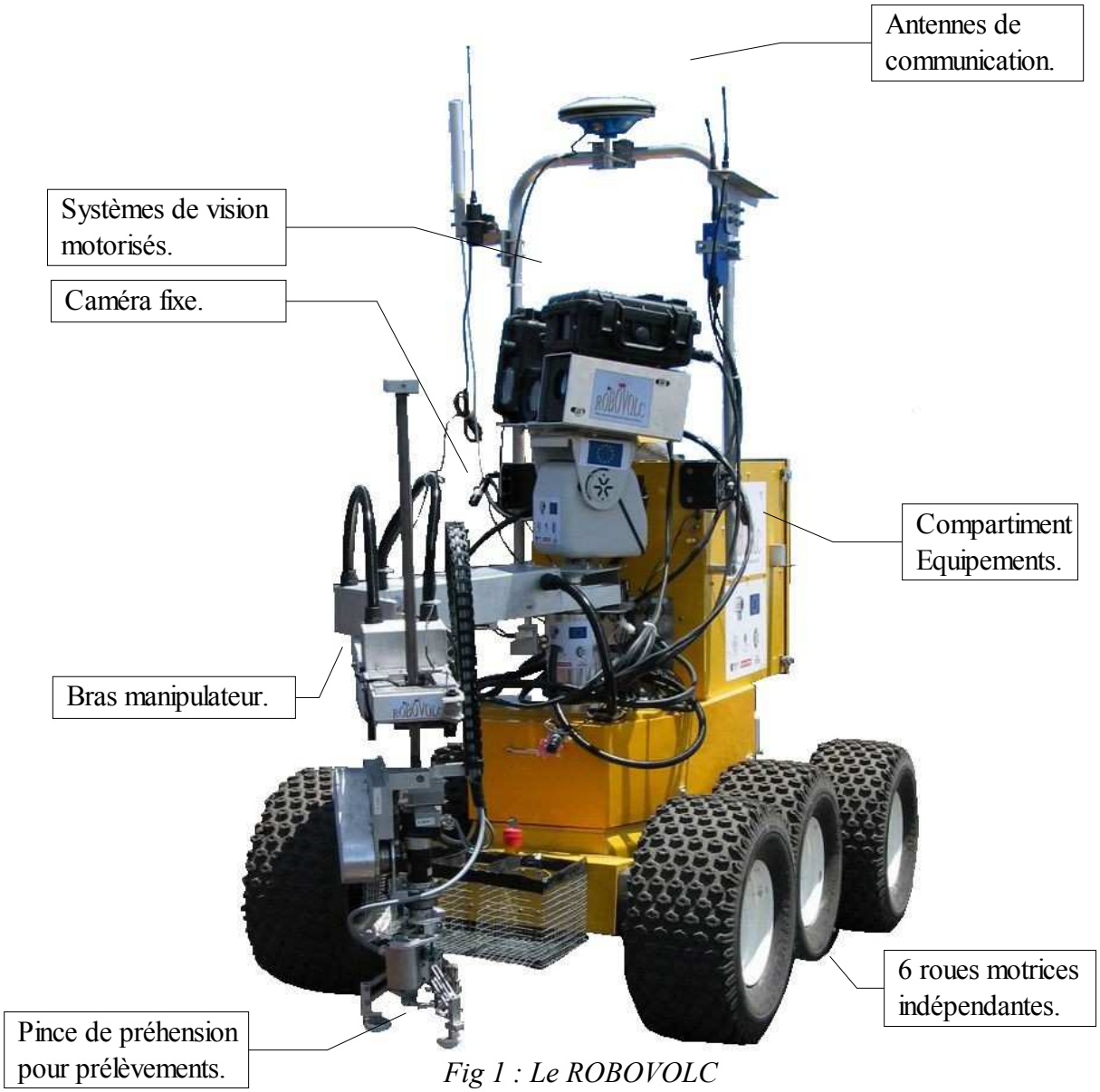


Fig 1 : Le ROBOVOLC

L'ensemble des éléments visibles sur cette photographie est repris dans le diagramme de déploiement ci-après.

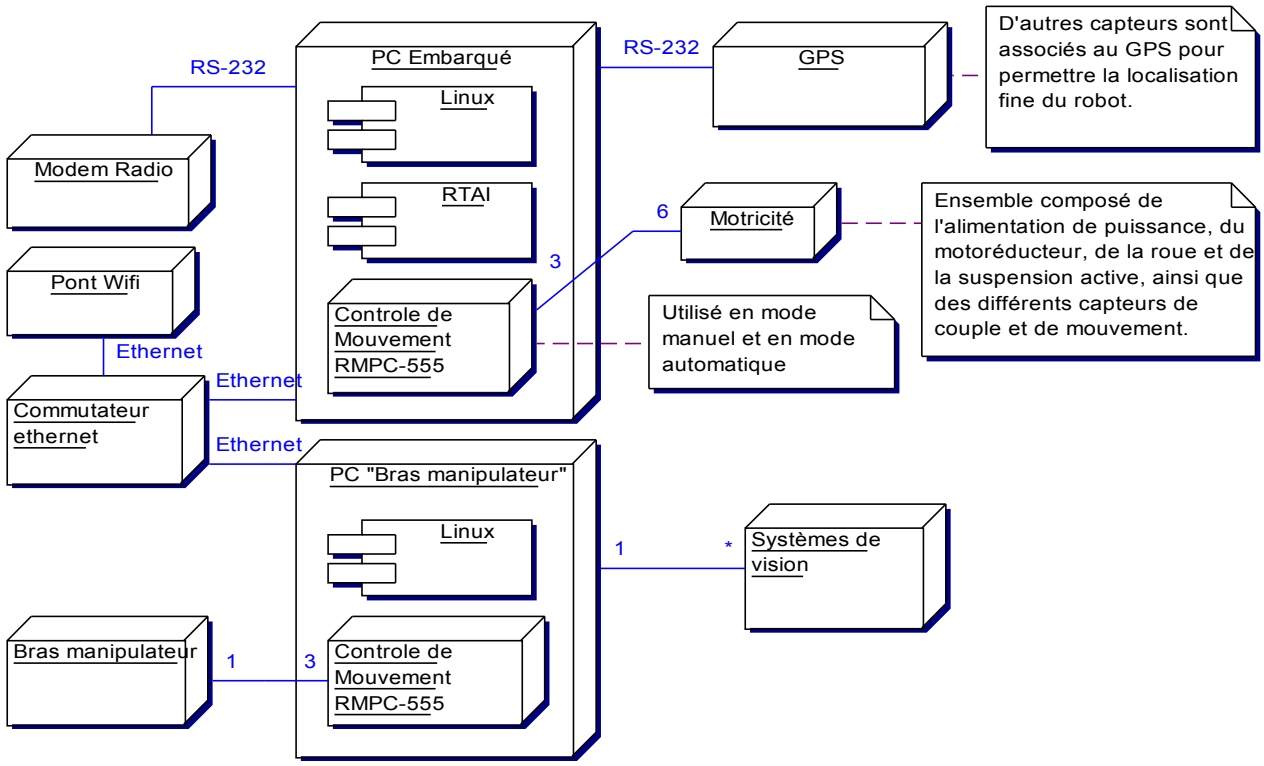


Fig 2 : Diagramme de déploiement du robot mobile.

La partie mobile est contrôlée à distance par un poste de contrôle localisé dans une zone sécurisée. Ci-dessous, le diagramme de déploiement de cette partie:

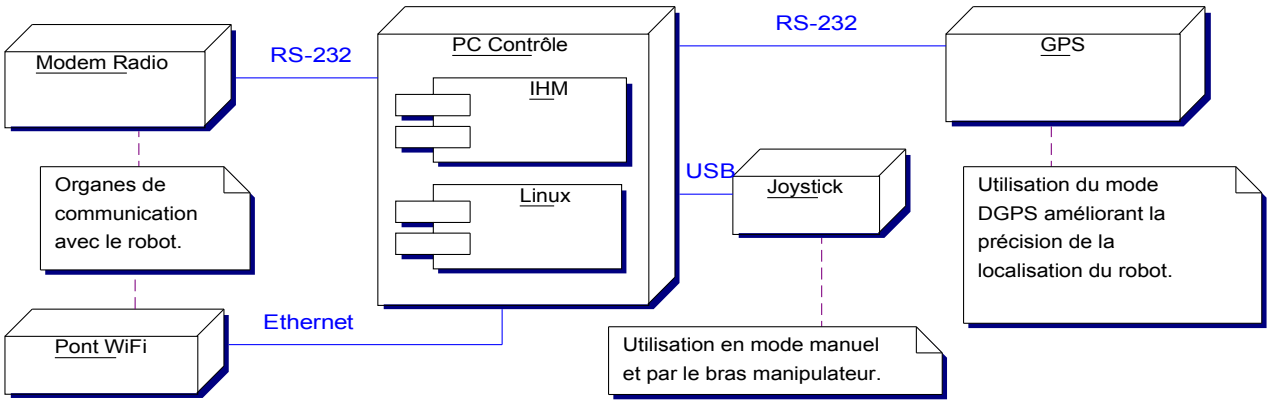


Fig 3: Diagramme de déploiement de la partie contrôle.

B. Le robot mobile

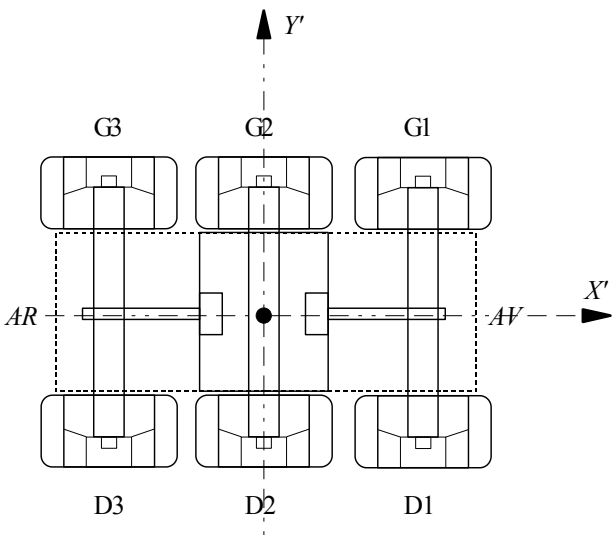
Le robot mobile est piloté à distance depuis le poste de contrôle. L'opérateur visualise en permanence les images transmises par la caméra embarquée, et reçoit cycliquement des informations sur la position géographique du robot.
Ces informations sont obtenues localement sur le robot par un système GPS (*Global Positioning System*) , et récupérées sur le poste de pilotage par l'intermédiaire de la liaison radio.

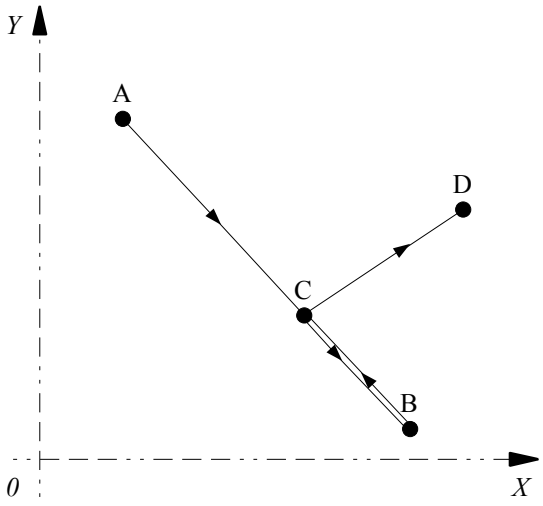
Pour ses déplacements, le robot est soit en mode automatique (il se dirige automatiquement vers un point géographique qui lui a été spécifié), soit en mode manuel (il est piloté manuellement, à distance, par l'opérateur).

B.1. Motorisation

Dans ce qui suit, nous nous intéressons exclusivement à la gestion des trajectoires du robot. Celui-ci est équipé de 3 essieux articulés. Chacune des six roues dispose d'une motorisation indépendante avec asservissement de vitesse.

Les figures suivantes montrent le robot en vue de dessus avec la dénomination des roues par rapport au sens normal d'avancement, et un exemple de trajectoire manuelle passant respectivement par les points A, B, C et D.





D'après les schémas ci-dessus, et sachant qu'une consigne positive fait tourner la roue dans le sens d'avancement du robot, on souhaite faire suivre au robot la trajectoire ABCD.

Question B.1.1.

Sur le document réponse, compléter le tableau exprimant les consignes à appliquer aux 6 roues à l'aide des symboles - (recule), 0 (arrêt), + (avance)

Dans la réalité, pour piloter le robot, il est nécessaire de contrôler finement la vitesse de rotation de chaque roue afin de minimiser les glissements, notamment en mode automatique, lorsque le robot doit suivre un cap de manière autonome...

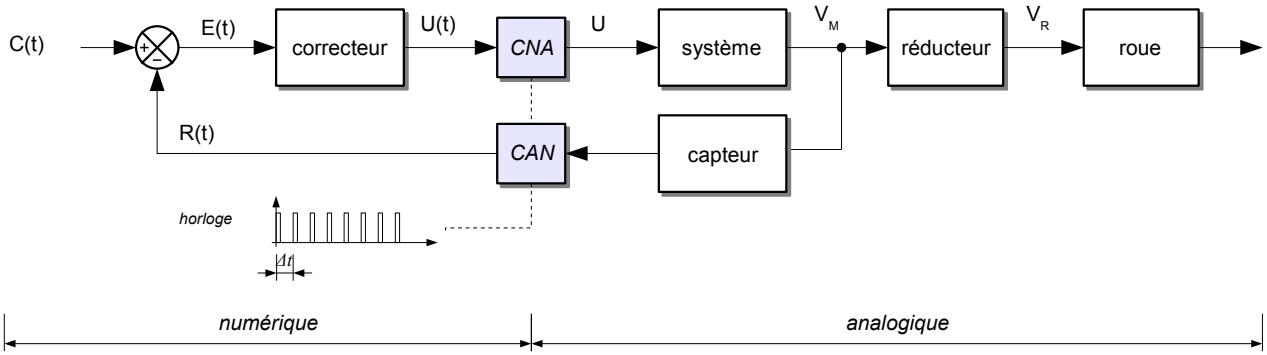
Les roues sont équipées de pneumatiques spéciaux dont le diamètre extérieur D est de 300 mm. On suppose un déplacement sans glissement ni patinage et on souhaite appliquer aux roues une consigne de vitesse Ω_R pour obtenir une vitesse de déplacement V_D en ligne droite égale à 0,2 mètres par secondes.

On rappelle que $V_D = \Omega_R * R$

Question B.1.2.

Exprimer Ω_R en fonction de D et de V_D. Puis faire l'application numérique en radian par secondes (rad/s) et convertir le résultat en tours par minute (trs/min).

Le système d'asservissement qui équipe chacune des roues est destiné à contrôler la vitesse de rotation de la roue, et doit permettre au système embarqué de détecter un glissement (manque d'adhérence) ou un patinage de celle-ci (comme par exemple quand la situation du robot fait que momentanément la roue ne touche plus le sol...).



Le bloc « système » représenté sur le schéma précédent est constitué d'un variateur et d'un moteur. Ce dernier est accouplé à un réducteur qui entraîne à son tour la roue. Le capteur permet d'obtenir une grandeur proportionnelle à la vitesse de rotation réelle de la roue, il est accouplé à l'axe de sortie du moteur.

La valeur U(t) en entrée du CNA (Convertisseur Numérique-Analogique) est codée sous forme d'un entier non signé sur 16 bits, elle est convertie en grandeur analogique U telle que -10V <= U <= +10V pour U(t) évoluant de 0x0000 à sa valeur maximale 0xFFFF.

Question B.1.3.

Quelle valeur numérique de U(t) faut-il appliquer pour obtenir une valeur nulle en sortie de CNA ?

Question B.1.4.

A quelle consigne U correspond la valeur hexadécimale d'entrée U(t) = 0xA000 ?

Le capteur utilisé pour mesurer la vitesse de rotation est de type dynamo tachymétrique, ce choix répond aux exigences de tenue en température et de robustesse. Ce capteur fournit une tension directement proportionnelle à la vitesse de rotation de la roue, cette tension varie au maximum entre -610mV et +650mV.

Le CAN (Convertisseur Analogique-Numérique) employé possède plusieurs canaux de conversion A/N 12 bits d'une linéarité de ± 1 bit. Le temps de conversion par canal est de 25µs.

- plage d'entrée réglable matériellement : 0-10V, ± 5V, ± 10V
- gain appliqué aux entrées configurables logiciellement : x 1 , x 10 , x 100

Question B.1.5.

Indiquer sur le tableau du document réponse les combinaisons valides de choix de plage et de gain cohérentes avec la tension retournée par la dynamo tachymétrique.

Question B.1.6.

Parmi les configurations valides ci-dessus, quelle est celle qui semble la mieux adaptée au problème ?

Question B.1.7.

Calculer la résolution en mV du CAN pour la configuration retenue.

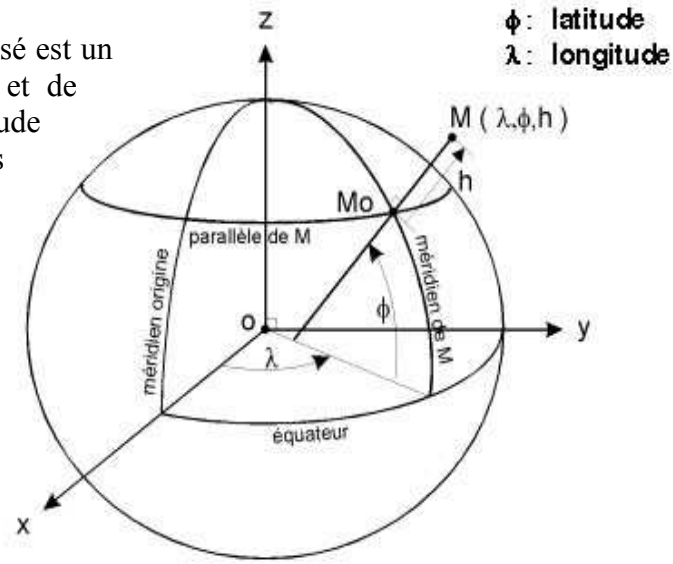
Question B.1.8.

Quel(s) autre(s) type(s) de capteur peut être employé pour mesurer une vitesse de rotation ?

B.2. Localisation

Les trajectoires du robot sont définies par des points cartographiques. Les positions à atteindre sont de la forme (x,y), les coordonnées retournées par le robot de la forme (x,y,z).

Le système de coordonnées géographiques utilisé est un quadrillage (non rectangulaire) de méridiens et de parallèles. Une position est définie par une latitude Nord-Sud et une longitude Est-Ouest exprimées en général en degrés par rapport à l'équateur et au méridien d'origine (Greenwich).



Les cartes intégrées au système distant de pilotage utilisent le système de coordonnées planes UTM (*Universal Transverse Mercator*) WGS84, en km. Le récepteur GPS implanté sur le robot utilise le standard NMEA (*National Marine & Electronics Association*) dans sa version NMEA-0183, pour l'émission de ses données.

Sous ce standard, toutes les données sont transmises en mode série asynchrone sous forme de trames de caractères ASCII. Les trames commencent par le caractère '$' et se terminent par les caractères CR (0x0D) et LF (0x0A).

Le protocole utilisé est 4800 bauds série asynchrone, sans parité, 8 bits de données et 1 bit de stop.

Question B.2.1.

Tracer le chronogramme de transmission des caractères CR et LF. Identifier clairement sous le chronogramme les bits de données sous la forme b7..b0.

Calculer en ms la durée de transmission d'un bit, notée Mt sur le document réponse.

Le récepteur GPS est interfacé avec le système informatique du robot par un circuit UART (*Universal Asynchronous Receiver / Transmitter*) de type 16550A à l'adresse 0x03F8. Le circuit est cadencé par un quartz de 1,8432 Mhz.

Question B.2.2.

A partir de l'annexe A.1, compléter le tableau réponse de manière à configurer l'UART pour communiquer avec le GPS. Pour chaque ligne, indiquer le nom du registre concerné, son adresse relative, et la valeur hexadécimale à y écrire.

L'annexe A.2 présente le format des trames NMEA transmises par le GPS sur la liaison série. Le système se doit de contrôler la validité des trames reçues en recalculant la somme de contrôle lorsqu'elle existe.

Question B.2.3.

Proposer le prototype en langage C d'une fonction `sommeDeControle()` qui reçoit en argument un pointeur sur le début d'une trame, et qui calcule puis retourne la somme de contrôle de cette trame sous forme d'un mot de 8 bits non signé.

Question B.2.4.

Proposer une implémentation de cette fonction, en considérant que la présence dans la trame des marqueurs '$' et '*' est vérifiée au préalable par le système utilisant la fonction.

Au cours d'une mission, le robot reçoit la trame GLL suivante en provenance de son récepteur GPS:

`$GPGLL,2114.35,S,05540.305,E,093518,A,*2F<CR><LF>`

Question B.2.5.

Exprimer en degrés, minutes, secondes les coordonnées géographiques du robot à cet instant.

Question B.2.6.

En tenant compte des caractéristiques du protocole série asynchrone utilisé, calculer le temps de transmission d'une trame constituée de N caractères.

Question B.2.7.

En déduire le temps de transmission de la trame GLL précédente.

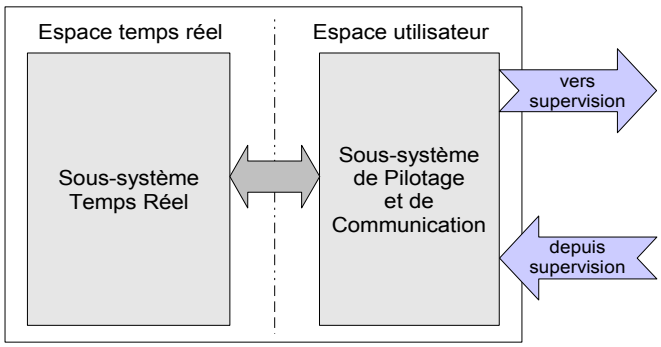
B.3. Communication entre tâches

RTAI (*Real Time Application Interface*) est une extension libre du noyau Linux lui apportant des fonctionnalités temps réel dures.
RTAI est basé sur la notion de « double noyau » :

- Un noyau temps réel à priorités fixes chargé de gérer les tâches temps-réel dur.
- Le noyau Linux est considéré comme une tâche de plus faible priorité par le noyau temps-réel et il conserve la gestion des tâches situées dans l'espace utilisateur.

Le robot possède pour son fonctionnement deux sous-systèmes distincts :

- Le sous-système temps réel : pilotage temps réel des six roues et du bras de manipulation.
- Le sous-système non temps réel : communication avec le monde extérieur et transmission d'ordre au sous-système temps réel.

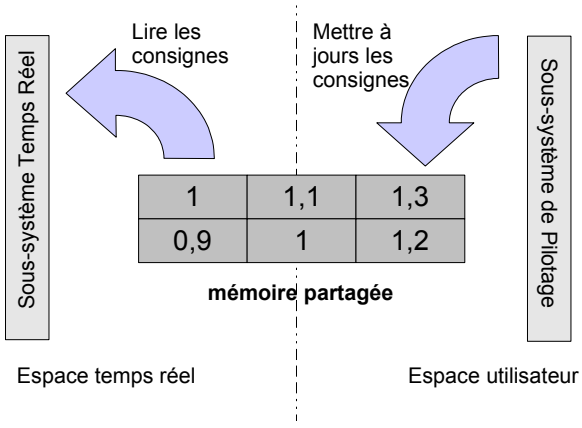


Robot

Le système de pilotage transmet des jeux de 6 valeurs de consigne des roues du robot. La communication entre le sous-système temps réel et le sous-système non temps réel de commande se fait grâce à l'utilisation d'une zone de mémoire partagée :

Deux tâches s'exécutent en parallèle :

- Le sous-système temps réel lit à période fixe les 6 consignes dans la mémoire partagée,
- Le sous-système de pilotage non temps réel écrit dans cette même mémoire partagée les 6 consignes à une fréquence voisine.



Le sous-système non temps réel écrit dans la mémoire partagée à partir de l'espace utilisateur, il est susceptible d'être interrompu au cours de cette écriture par le sous-système temps réel.

Question B.3.1.

En conséquence, quel est le risque encouru par l'application ?

L'écriture et la lecture de la mémoire partagée sont considérées comme des sections critiques, elles sont protégées par un sémaphore.

Question B.3.2.

Qu'appelle-t-on une section critique dans un système multitâche ?

L'annexe A3 présente un extrait de la bibliothèque RTAI relatif à la mise en oeuvre des sémaphores.

Question B.3.3.

Citer les fonctions permettant, depuis l'espace utilisateur, de réaliser les opérations présentées dans le tableau du document réponse.

Question B.3.4.

Implémenter une fonction de prototype `bool initSem(int numSem)` permettant d'initialiser le sémaphore `numSem` à la valeur 1, et retournant vrai en cas de succès ou faux en cas d'échec.

Soit la structure de la mémoire partagée suivante :

```
struct ShmMemWrite
{
      float v[6] ;
};
```

Forme C Ansi équivalente.

```
typedef struct
{
      float v[6] ;
} ShmMemWrite;
```

Question B.3.5.

Insérer dans la fonction

```
void envoyerConsignes( ShmMemWrite* shm, float* cons, int numSem )
```

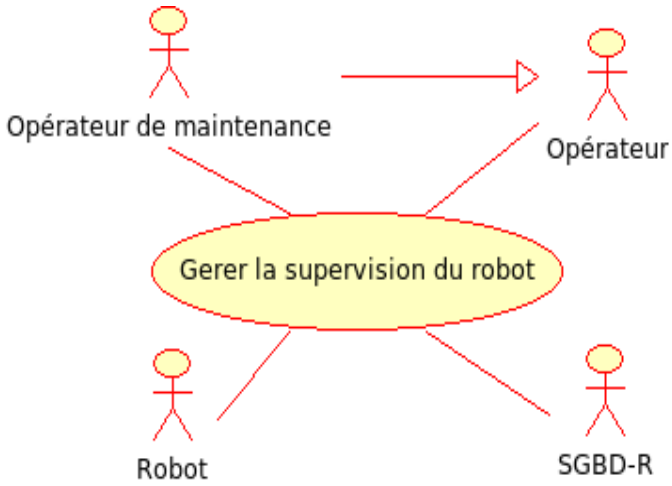
les lignes de code permettant de rendre la séquence d'écriture des six vitesses non interruptible.

C. Contrôle-Commande et Supervision

C.1. Analyse du système de supervision

Le système de supervision du robot interagit avec :

- le robot,
- un système de gestion de base de données permettant d'enregistrer les points de passage du robot,
- l'opérateur du système,
- et l'opérateur de maintenance.



Ceci est présenté dans le diagramme ci-contre.

Cette analyse des cas d'utilisation est sommaire, elle permet de définir les limites du système. Ceci implique, par exemple, que le robot est considéré comme acteur du système de supervision...

Question C.1.1.

Que signifie la flèche reliant Opérateur de maintenance à Opérateur ?

Question C.1.2.

Le document réponse propose quatre diagrammes de cas d'utilisation d'un niveau de détail supérieur à celui ci-dessus. Deux d'entre eux sont manifestement incohérents avec le diagramme ci-dessus. Indiquer lesquels et justifier la réponse.

C.2. Modes de fonctionnement du robot

Comme évoqué précédemment, le robot possède au moins deux modes de fonctionnement : un mode automatique dans lequel il gère seul son déplacement, et un mode de pilotage manuel se traduisant, vu de l'opérateur, par des envois d'ordres (position à atteindre, arrêt, prélèvement,...) et de commandes de déplacement (par l'intermédiaire d'un actionneur de type manette analogique à 2 axes : *joystick*...).

Procédure type d'utilisation lorsque le robot est en évolution :

- le robot est en pilotage mode manuel, il est localisé au point P1 sur le site à explorer, sa position est montrée à l'opérateur sur une carte du site ;
- Depuis ce mode manuel, l'opérateur indique au robot la nouvelle position à atteindre P2 en lui transmettant les coordonnées adéquates ;
- le robot passe en mode automatique et se dirige vers ce point.
- *le robot transmet sa position géographique (1) périodiquement, qu'il soit en mode manuel ou automatique ;*
- si nécessaire, l'opérateur peut en mode automatique modifier la vitesse d'évolution du robot en fixant un pourcentage de la vitesse nominale pré-réglée (ce pourcentage est par défaut égal à 50) ;
- lorsque un obstacle infranchissable se présente devant le robot, celui-ci le détecte (les capteurs utilisés pour cela ne sont pas concernés par la présente étude) et s'arrête. Il transmet alors au système distant un signal « obstacle détecté » ;
- Le mode arrêt permet à l'opérateur de reprendre le contrôle manuel du robot afin de lui faire contourner l'obstacle. Il est aidé pour cela par les images caméra ;
- *les consignes sont envoyées sous forme de couples direction/vitesse (2), elles sont traduites par le robot qui calcule les commandes de vitesse des roues , indépendante pour chacune d'elles ;*
- lorsque l'opérateur considère que la voie est à nouveau libre, il peut demander au robot de reprendre sa trajectoire en mode automatique vers le point P2 ;
- lorsque la position P2 est atteinte, le robot s'arrête et signale à l'opérateur l'évènement « position atteinte ». Ce dernier peut reprendre la main manuellement pour explorer localement le site, prélever d'éventuels échantillons, ou spécifier une nouvelle position à atteindre en automatique ;
- *l'opérateur peut à tout moment émettre l'ordre d'arrêt (3) du robot quel que soit son mode de fonctionnement.*

Le diagramme UML du document réponse (C.2.2) va être utilisé pour modéliser les différentes possibilités de mobilité du RoboVolc...

Question C.2.1.

Comment s'appelle le type de diagramme du document réponse C.2.2. dans la terminologie UML ?
Quel est son rôle ?

La phase d'initialisation, le mode arrêt du robot (4), et les *affirmations en italique* du texte précédent sont déjà modélisés sur le diagramme. Le signal (5) montre un exemple d'événement avec déclenchement d'action (événement / action).

Question C.2.2.

Après lecture attentive de la procédure type précédente, compléter le diagramme en faisant apparaître les **modes** « pilotage manuel » et « mode automatique », les **événements** et les **actions** de manière à couvrir tous les extraits soulignés dans le texte.

C.3. Centralisation et gestion des données

Les questions suivantes sont relatives à la gestion des trajectoires du robot sur le poste de supervision. L'ensemble des données manipulées est stocké en base de données. Une étude préalable a permis d'expliciter les termes suivants.

- Une campagne d'exploration est définie par une date de début et une durée prévisionnelle, une description du site, et des renseignements divers concernant l'objet de l'exploration, les chercheurs impliqués, le nom du manipulateur, la météo...

  Une campagne peut se traduire par plusieurs jours de travail. Chaque journée peut faire l'objet de plusieurs trajectoires et plusieurs prélèvements.

- Une trajectoire est définie par une succession de points de passage et par la distance totale parcourue. Le nombre de points n'étant pas limité, ils seront stockés dans une liste chaînée (en pratique, on utilisera la classe template **`list`** proposée par la STL).

  Lorsque le robot est piloté en manuel, la partie de trajectoire réalisée est modélisée de proche en proche sous forme d'une succession de points de passages.

  Une trajectoire se termine en fin de campagne ou lorsque l'opérateur en initialise une nouvelle.

- Un prélèvement est défini par un numéro d'ordre et par le point géographique où il a eu lieu. Il peut être de type prélèvement de gaz ou prélèvement de minéraux.

  Le robot peut stocker un seul prélèvement gazeux et possède 2 paniers distincts pour les prélèvements minéraux, la capacité de chaque panier est de 1000 cm^3.

- Un point de passage est caractérisé par ses coordonnées longitude-latitude-altitude, et une datation UTC (Temps Universel). Une information précise s'il s'agit d'un point programmé ou d'un point calculé suite à un pilotage manuel. Un point peut le cas échéant être référencé par un prélèvement qui y a été effectué.

  Un point programmé comme objectif, sera systématiquement mis à jour lorsque le robot l'aura atteint, ce sont alors les coordonnées fournis par le GPS qui permettent de connaître l'altitude...

Question C.3.1.

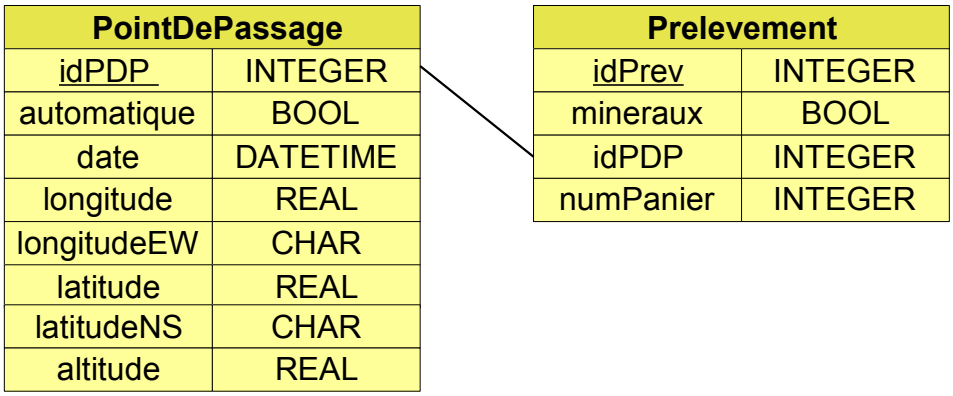
Faire apparaître sur le diagramme de classes fourni dans le document réponse les relations entre les différentes classes, et les multiplicités de ces relations.

Question C.3.2.

Proposer une déclaration en langage C++ de la classe Trajectoire restreinte à la description ci-dessus et au diagramme de classe.

Une partie des données est stockée en local sur le robot, c'est notamment le cas des trajectoires de la journée, des points de passages et des prélèvements. Le moteur de cette base de données est Sqlite3. Ce SGBD-R est conforme à la norme SQL 92.

Le modèle conceptuel de données (MCD) est global au système. La présente étude porte uniquement sur les points de passages et les prélèvements. Ci-après le MCD correspondant :



Il est à noter que les attributs longitudeEW et latitudeNS peuvent prendre respectivement 'E' ou 'W' et 'N' ou 'S' comme valeurs. Ceci correspond au codage des longitudes et latitudes (un nombre réel + une direction).

```
SELECT longitude, longitudeEW,
       latitude, latitudeNS,
       altitude, date
FROM PointDePassage
WHERE automatique = TRUE ;
```

La requête présentée permet de connaître tous les points de passage automatique ainsi que la date et l'heure de ce passage.

Question C.3.3.

En vous appuyant sur l'exemple ci-dessus, donner la requête SQL permettant de connaître le numéro de tous les prélèvements gazeux.

```
SELECT longitude, longitudeEW,
       latitude, latitudeNS,
       altitude, date
FROM PointDePassage, Prelevement
WHERE PointDePassage.idPDP = Prelevement.idPDP
  AND mineraux = FALSE
  AND altitude >= 500 AND altitude < 501 ;
```

Attention. Jointure indispensable puisque **mineraux** appartient à **Prélévement** et **altitude** à **PointDePassage**.

La requête présentée permet de connaître les lieux de tous les prélèvements gazeux qui ont eu lieu à 500m d'altitude. Attention, l'altitude étant un REAL nécessite un intervalle, ici **[500 ; 501[** mètres.

Question C.3.4.

Donner la requête SQL permettant de connaître le numéro des prélèvements, le type de prélèvements ainsi que le numéro de panier du point de coordonnées `4807.048,N,02131.324,E` à `545` m d'altitude.

D. Communication et réseau

D.1. Protocoles

Le réseau mis en œuvre entre le robot et le poste de supervision est de type WIFI 802.11g. Ce type de liaison radio peut s'assimiler pour le cas présent à une connexion Ethernet très peu fiable (perturbations dues à la conformation du terrain et à l'électromagnétisme).

Les données à transmettre sont des ordres pour donner une trajectoire ou pour piloter le robot en mode manuel. Ces ordres sont transmis via une transmission UDP.

Question D.1.1.

Quelle information des protocoles TCP et UDP identifie sans équivoque le processus destinataire du message ?

Cette information peut prendre des valeurs dont certaines sont réservées par la DARPA pour des services usuels. Le fichier `/etc/services` recense l'ensemble de ces services.

Question D.1.2.

Citer les trois informations définies par ce fichier pour chacun des services.

Question D.1.3.

Un même numéro de port peut-il être utilisé simultanément en TCP et en UDP par deux processus distincts ? Justifier votre réponse.

Question D.1.4.

Quels sont les particularités respectives des protocoles TCP et UDP ? Compléter le document réponse.

D.2. Sécurité des biens et des personnes

La communication WIFI étant très peu fiable, un mécanisme de détection de rupture de flux est mis en place.

Question D.2.1.

D'après le diagramme de déploiement, quelle solution matérielle peut être utilisée pour transmettre les informations de commande lors d'une rupture de flux WIFI ?

Question D.2.2.

Quelle action doit être envisagée en cas de perte totale du flux de données ?

L'équipe de RoboVolc accepte que le robot se déplace sur 0,25 m sans communication avec le superviseur.

La détection de rupture de flux est réalisée par le mécanisme suivant :

```
// coté robot
N <- valeur aléatoire
ERREUR <- 0
FAIRE
      Envoyer N au superviseur // via protocole UDP sur le port 3333
      Attendre la réception d'un entier R du superviseur pendant X ms
      SI pas de réception dans le délai imparti
            ERREUR <-  ERREUR + 1
      SINON
            SI (  R == N )
                  ERREUR <- 0
                  N <- N + 1
            SINON
                  ERREUR <-  ERREUR + 1
            FIN SI
      FIN SI
      SI ( ERREUR >= 5 )
            Appeler procédure Robot::enCasDeRuptureDeFlux()
            N <- valeur aléatoire
      FIN SI
TANT QUE ( VRAI )
```

```
// coté poste de supervision
MEM <- 0
ERREUR <- 0
FAIRE
      Attendre la réception d'un entier R du robot pendant X ms
      SI pas de réception dans le délai imparti
            ERREUR <-  ERREUR + 1
      SINON
            SI ( ( MEM + 1 ) == R )
                  ERREUR <- 0
            SINON
                  ERREUR <-  ERREUR + 1
            FINSI
            MEM <- R
            Envoyer R au robot
      FIN SI
      SI ( ERREUR >= 5 )
            Appeler procédure Supervision::enCasDeRuptureDeFlux()
      FIN SI
TANT QUE ( VRAI )
```

Remarque : Cet algorithme sera implémenté à la question D.3.2.

Question D.2.3.

Au bout de combien de tentatives infructueuses de réception de données les procédures `enCasDeRuptureDeFlux()` sont-elles appelées ?

Question D.2.4.

Calculer en millisecondes (ms) la valeur de X de l'algorithme précédent de manière à respecter les contraintes citées pour les deux vitesses suivantes du robot :

- 1 m/s ?
- 5 m/s ?

D.3. Programmation réseau

Soit la classe `DatagramSocket` permettant d'envoyer et de recevoir des paquets de données via le protocole UDP :

```
class DatagramSocket
{
private:
    unsigned short port ;
    int sock ;
    struct sockaddr_in source ;

public:
    DatagramSocket( unsigned short port = 0 ) ;
    ~DatagramSocket() ;

    // fixe le timeOut en ms

    void setTimeOut( unsigned int timeOut ) ;

    // La méthode readDatagram() attend la réception d'un datagramme
    //
    // Le résultat est stocké dans data, maxlen est la taille de data
    //
    // Valeur renvoyée : la longueur du datagramme lu,
    //                   ou -1 en cas d'erreur,
    //                   ou -2 en cas de timeout

    long readDatagram( void *data, long maxlen ) ;

    long writeDatagram(
                  const void *data,       // données à émettre
                  long len,               // taille des données à émettre
                  const char* host,       // machine cible
                  unsigned short port     // port destinataire
                ) ;
};
```

Fonction de lecture d'un datagramme.

Fonction d'écriture d'un datagramme.

Le constructeur de `DatagramSocket` prend en argument le numéro du port local, le sauvegarde, créer une socket et l'attache si nécessaire (port > 0).

Question D.3.1.

A l'aide de l'annexe 4, proposer une implémentation du constructeur

```
DatagramSocket( unsigned short port = 0 ) ;
```

Au bout de la durée X ms fixée par `setTimeOut()`, la méthode `readDatagram()` renvoie `-2` si aucun datagramme n'a été reçu.

Question D.3.2.

Compléter, d'après le pseudo-code des pages précédentes, l'implémentation de la méthode

```
void Superviseur::keepAlive(
    DatagramSocket *ds, // Instance de Datagram pour communication
    char* robotName,    // Nom (hostname) du destinataire
    int port );         // Port à utiliser
```

permettant la détection de rupture de flux coté poste de supervision.

On considère les modifications suivantes de la méthode **`readDatagram()`** :

```
// retourne : nombre d'octet lues
// exception: transmet une exception TimeOutException
//            au bout d'un timeOut de X ms
//            défini par la méthode setTimeOut(X)
long readDatagram(
                  char *data,
                  long maxlen )
                  throw (TimeOutException) ;
```

Voir annexe 5 « les exceptions en C++ » et l'exemple de mise en oeuvre qui y figure.

Question D.3.3.

Si la fonction membre `readDatagram()` de la classe `DatagramSocket` avait le prototype modifié proposé ci-dessus, comment pourrait-on intercepter l'émission de l'exception liée au timeout lancée par la méthode ?

D.4. Mise au point

En phase de mise au point des algorithmes précédents, le robot ouvre sa socket de communication sur le port 4444, tandis que le superviseur ouvre la sienne sur le port 5555. Les trois trames suivantes ont alors été capturées sur le réseau Wifi entre le superviseur et le robot :

Soit trois trames « décodées par Ethereal » :

- Trame 1

```
⊞ Frame 11 (42 bytes on wire, 42 bytes captured)
⊞ Ethernet II, Src: 192.168.0.16 (00:0f:b0:71:c4:c1), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊟ Address Resolution Protocol (request)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (0x0001)
    Sender MAC address: 192.168.0.16 (00:0f:b0:71:c4:c1)
    Sender IP address: 192.168.0.16 (192.168.0.16)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.0.219 (192.168.0.219)
```

- Trame 2

```
⊞ Frame 12 (64 bytes on wire, 64 bytes captured)
⊞ Ethernet II, Src: 192.168.0.219 (00:04:00:dc:1c:2d), Dst: 192.168.0.16 (00:0f:b0:71:c4:c1)
⊟ Address Resolution Protocol (reply)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (0x0002)
    Sender MAC address: 192.168.0.219 (00:04:00:dc:1c:2d)
    Sender IP address: 192.168.0.219 (192.168.0.219)
    Target MAC address: 192.168.0.16 (00:0f:b0:71:c4:c1)
    Target IP address: 192.168.0.16 (192.168.0.16)
```

- Trame 3

```
⊞ Frame 6 (44 bytes on wire, 44 bytes captured)
⊞ Ethernet II, Src: 192.168.0.16 (00:0f:b0:71:c4:c1), Dst: 192.168.0.219 (00:04:00:dc:1c:2d)
⊞ Internet Protocol, Src: 192.168.0.16 (192.168.0.16), Dst: 192.168.0.219 (192.168.0.219)
⊟ User Datagram Protocol, Src Port: 4444 (4444), Dst Port: 5555 (5555)
    Source port: 4444 (4444)
    Destination port: 5555 (5555)
    Length: 10
    Checksum: 0x89df [correct]
  Data (2 bytes): cc af
```

L'étude du décodage des trames précédents met en évidence l'utilisation du protocole ARP.

Question D.4.1.

Compléter le tableau du document réponse en indiquant si les propositions relatives au protocole ARP sont vraies ou fausses.

Question D.4.2.

Après avoir analysé le contenu de la trame 3, compléter le tableau du document réponse.

Question D.4.3.

Ce datagramme (trame 3) est-il émis du robot vers le poste de supervision ou du poste de supervision vers le robot ? Justifier.

Fin du questionnement.

BTS INFORMATIQUE ET RESEAUX

POUR L'INDUSTRIE ET LES SERVICES TECHNIQUES

Session 2007

Epreuve E.4
Etude d'un Système Informatisé

Prélèvements sur sites volcaniques

Annexes

Annexe A1 :	***Documentation de l'UART 16550A***	***3 pages***
Annexe A2 :	***Le protocole NMEA***	***1 page***
Annexe A3 :	***Documentation sémaphores RTAI***	***2 pages***
Annexe A4 :	***Programmation réseau***	***5 pages***
Annexe A5 :	***Les exceptions en C++***	***2 pages***

ANNEXE A1
Documentation de l'UART 16C550A (*extraits*)

16C550A UART
Récepteur / Transmetteur Asynchrone Universel

Caractéristiques

- Interface de communication asynchrone utilisant les signaux standards de contrôle Modem (CTS, RTS, DSR, DTR, RI et DCD).
- Interface série totalement programmable:
  - 5, 6, 7 ou 8 bits par caractères
  - Gestion de la parité
  - 1, 1.5 ou 2 bits d'arrêt
  - Vitesse jusqu'à 256kbds
- Gestion des interruptions de transmission, de réception, d'état de ligne et de données contrôlées indépendamment.
- FIFO activable et paramétrable de 16 caractères en transmission et réception de données.
- Générateur de Baud programmable pour une fréquence interne de 16*Baud.
- Logiciellement compatible avec le 16C450.

Description Générale

L'UART 16C550A est un circuit d'interface asynchrone série pour microprocesseur. Ces caractéristiques le rendent logiciellement compatible avec le 16C450 et le plus ancien 8250. Le circuit 16C550A totalement paramétrable et programmable est prévu pour l'interfaçage de liens de type série asynchrone. Il intègre deux FIFO pour diminuer le nombre d'interruptions du microprocesseur. Il inclus un générateur de fréquence interne de 16 coups d'horloge par bit de donnée qui autorise le transfert des données jusqu'à la vitesse de 256kbs.

Programmation

La sélection de l'un des 12 registres internes est effectuée à l'aide des 3 signaux d'adresse A0, A1 et A2 et d'un bit interne DLAB appartenant au registre de contrôle de ligne. Le tableau récapitulatif des registres internes par rapport à l'adresse de base et l'état de DLAB est donné au tableau A1.1.

Tableau A1.1. Plan d'adressage de l'UART 16C550A

Adresse relative	Nom	DLAB	Fonction en lecture	Fonction en écriture
0	THR/RBR	0	Tampon de réception	Tampon d'émission
	DLLB	1	Registre de poids faible du diviseur de fréquence (LSB)	
1	IER	0	Masque des interruptions *	
	DLHB	1	Registre de poids fort du diviseur de fréquence (MSB)	
2	IIR/FCR	X	Registre d'état des interruptions *	Contrôle des FIFO *
3	LCR	X	Registre de contrôle de ligne	
4	MCR	X	Registre de contrôle du modem	
5	LSR	X	Registre d'état de ligne	
6	MSR	X	Registre d'état du modem	
7	SR	X	Registre de personnalisation *	

* Non documenté dans l'annexe

Description des registres

- Tampon de réception (THR)
Ce registre contient le dernier caractère reçu.

- Tampon d'émission (RBR)
Ce registre contient le caractère qui sera émis par copie dans le registre de sérialisation.

- Registre du diviseur de fréquence (DLHB / DLLB)

Le registre du diviseur de fréquence sur 16 bits est accessible aux adresses relatives 0 et 1 à condition que le bit DLAB (*Divisor Latch Access Bit*) soit à 1. Ce registre permet de programmer le générateur de fréquence interne pour régler le débit binaire sériel. La fréquence du générateur interne est 16 fois plus grande que celle du débit binaire. Le calcul du diviseur est effectué soit à l'aide des tableaux A1.2 et A1.3 ou avec la relation :

Diviseur = Fréquence_horloge / (Débit_binaire x 16)

Où le débit binaire sériel est exprimé en Bauds et la fréquence d'horloge de l'UART 16C550A en Hz.

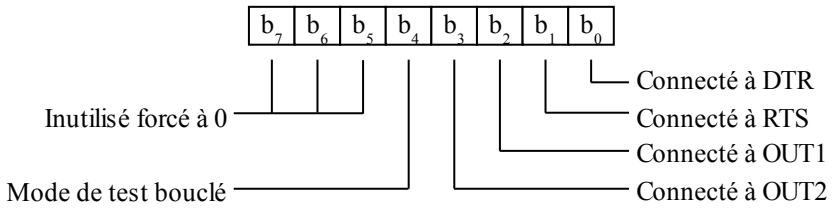
Tableau A1.2. Vitesse pour un quartz **1,8432MHz**

Vitesse en Bauds	Diviseur	Erreur %
75	1536	
110	147	0,026
150	768	
300	384	
600	192	
1200	96	
1800	64	
2000	58	0,69
2400	48	
3600	32	
4800	24	
7200	16	
9600	12	
19200	6	
38400	3	
57600	2	
115200	1	

Tableau A1.3. Vitesse pour un quartz **3,072MHz**

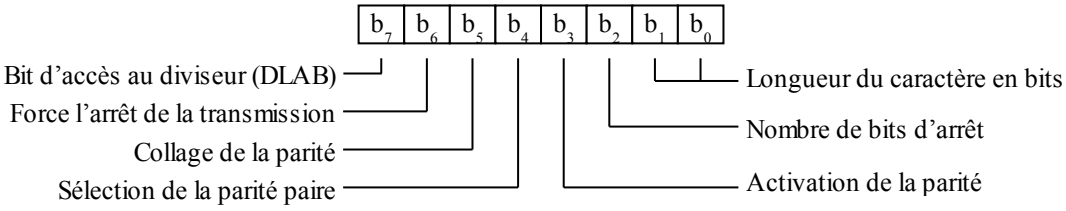
Vitesse en Bauds	Diviseur	Erreur %
75	2560	
110	1745	0,026
150	1280	
300	640	
600	320	
1200	160	
1800	107	0,312
2000	96	
2400	80	
3600	53	0,628
4800	40	
7200	27	1,23
9600	20	
19200	10	
38400	5	

- Registre de contrôle du modem (MCR)



Le bit de poids faible est connecté à la sortie DTR (Data Terminal Ready) et permet d'indiquer au Modem que le terminal est prêt à envoyer des données. Le positionnement à 1 de ce bit force à 0 la sortie DTR et inversement. Le bit 1 est connecté à la sortie RTS (Request To Send) et permet au terminal de demander au Modem une émission de données. Le positionnement à 1 de ce bit force à 0 la sortie RTS et inversement.
Les bits 2 et 3 permettent de contrôler respectivement les sorties utilisateur OUT1 et OUT2. Comme précédemment le contrôle de ces sorties est inversé.
Le bit 4 permet de mettre en œuvre une boucle interne afin de tester le transfert de données. Pendant cette phase de test, la sortie est forcée à 1 et l'entrée déconnectée extérieurement mais connectée en interne à la sortie du registre de sérialisation de l'émission de données. Les données émises seront ainsi transmises vers le registre de réception. Les bits 5 à 7 sont en permanence à 0.

- Registre de contrôle de ligne (LCR)



La longueur du caractère transmis est déterminé par les deux bits de poids faible et est réglable entre 5 et 8 bits. Le tableau A1.4 montre les diverses possibilités de réglage du nombre de bits. Le nombre de bits d'arrêt est aussi paramétrable et dépend de la valeur du bit 2 mais aussi de la taille du caractère transmis comme le montre le tableau A1.5.
La parité peut être activée à l'aide du bit 3. La mise à 1 du bit 4 permet de contrôler si la parité sera paire ou impaire pour la mise à 0. Le collage de parité, bit 5, force la parité à 0 si celle-ci est sélectionnée paire ou à 1 si celle-ci est sélectionnée impaire. Le collage de parité n'est possible que si la parité est activée.
Le bit 6 force un arrêt de l'émission en forçant la sortie à 0. Le bit 7 sélectionne l'accès au registre diviseur de fréquence (DLAB).

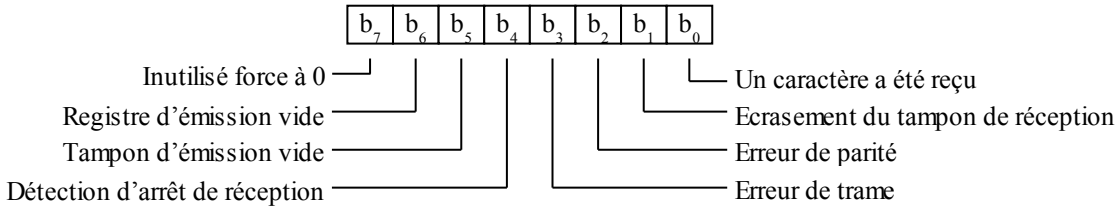
Tableau A1.4. Réglage du nombre de bits caractère

b1	b0	Taille
0	0	5
0	1	6
1	0	7
1	1	8

Tableau A1.5. Réglage du nombre de bits d'arrêt

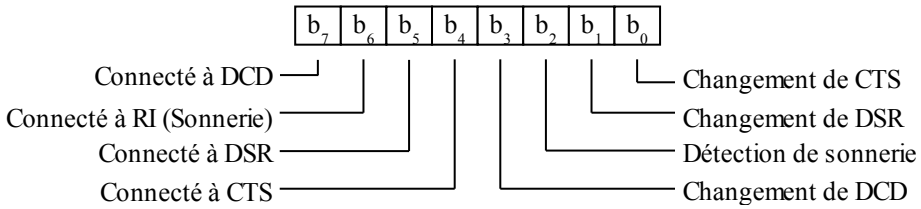
b2	Taille	Nb bits d'arrêt
0	X	1
1	5	1,5
1	6	2
1	7	2
1	8	2

- Registre d'état de ligne (LSR)



Le bit 0 indique qu'un caractère a été reçu et est disponible dans le tampon. Ce bit est remis à 0 lors de la lecture du tampon. Si le tampon n'est pas lu à temps et qu'un écrasement de la donnée est effectué par une nouvelle réception alors le bit 1 sera positionné. Le bit 2 sera positionné lors d'une détection d'erreur de parité sur une réception à condition que la parité soit activée. Si une réception indique une longueur incorrecte du bit d'arrêt correspondant à une erreur de trame alors le bit 3 sera positionné à 1. Le bit 4 indique le forçage d'un arrêt de réception par le modem.
L'émission d'un caractère est réalisée à l'aide du registre d'émission qui sera sérialisé. Dès que ce registre est vide le contenu du tampon d'émission y est transféré pour être à son tour émis. Si le tampon d'émission est vide l'indicateur binaire bit 5 est positionné à 1 pour indiquer qu'un nouveau caractère est demandé par le processus d'émission. D'autre part, si le registre d'émission et le tampon sont vides alors le bit 6 sera positionné à 1. En résumé, ce bit indique une rupture du flux d'émission. Ces bits, 5 et 6, sont mis à 0 dès qu'une nouvelle donnée a été écrite dans le tampon de réception par le microprocesseur. Les bits 1 à 5 sont remis à 0 lors d'une lecture du registre d'état par le microprocesseur.

- Registre d'état du modem (MSR)



Les bit 0,1 et 3 sont respectivement associés aux entrées CTS, DSR et DCD. Lors d'un changement de niveau sur une de ces trois entrées le bit correspondant est positionné à 1. Par contre le bit 2 qui est associé à l'entrée de détection de sonnerie du modem (entrée RI), n'est positionné à 1 que lors de l'apparition d'un front montant sur cette entrée. Ces quatre bits sont mis à 0 après la lecture du registre d'état du modem par le microprocesseur.
Les bits 4 à 7 sont respectivement connectés à travers un inverseur aux entrées CTS, DSR, RI, DCD.
Si le mode de test bouclé est activé par la mise à 1 du bit 4 du registre de contrôle du modem, ces bits 4 à 7 sont alors respectivement connectés en interne aux signaux de sorties RTS, DTR, OU1 et OUT2.

Annexe 2

Le protocole NMEA (*extraits*)

NMEA (*National Marine & Electronics Association*) est une association à but non lucratif fondée par un groupement de professionnels de l'industrie de l'électronique des instruments et périphériques marins, conjointement avec des fabricants, des distributeurs, des revendeurs, des institutions d'enseignements. L'association a pour but d'harmoniser et de standardiser les équipements de la marine.

NMEA est à l'origine de nombreux standards et en particulier du Standard NMEA-0183, utilisé dans les appareils GPS actuels.

Sous ce standard, toutes les données sont transmises sous la forme de trames ou phrases (*sentences*) constituées de caractères ASCII imprimables, elles commencent par le caractère '$' et sont terminées par les caractères [CR] Retour Chariot et [LF] Saut de ligne.

La longueur maximale d'une trame ne peut excéder 82 caractères.

Format simplifié des trames NMEA :

- les caractères GP indiquent qu'il s'agit d'une trame GPS (*Global Positioning System*)
- <id> est une suite de 3 caractères (GGA, GLL, GSV, ...) définissant le type de la trame
- les données sont séparées par des virgules, leur nombre dépend du type de trame
- la somme de contrôle (checksum) est précédée d'un caractère '*', elle est calculée par OU exclusif entre tous les caractères situés entre '$' et '*' . Ce champ est présenté sous forme hexadécimale, sa présence est en général facultative.
- Une trame NMEA ne comporte jamais de caractère Espace (0x20).

Détail d'une trame GGA avec checksum (CRLF non représentés) :

```
$GPGGA,124819,4807.048,N,02131.324,E,1,08,0.9,545.4,M,46.9,M,,*4C
```

GGA	= indicatif « Données d'acquisition Fix et Date - GPS »
124819	= acquisition du Fix à 12:48:19 UTC
4807.048,N	= Latitude 48° 7,048' Nord
02131.324,E	= Longitude 21° 31,324' Est
1	= Fix qualification (0 = non valide, 1 = Fix GPS, 2 = Fix DGPS, ...)
08	= nombre de satellites en poursuite
0.9	= dilution horizontale
545.4,M	= altitude en mètres au dessus du niveau moyen des océans (MSL : *Mean See Level*)
46.9,M	= correction de la hauteur de la géoïde en mètres par rapport à l'ellipsoïde WGS84
(champ vide)	= nombre de secondes écoulées depuis la dernière mise à jour DGPS
(champ vide)	= identification de la station DGPS
*4C	= préfixe + somme de contrôle

Détail d'une trame GLL sans checksum (CRLF non représentés) :

```
$GPGLL,4916.75,N,12311.12,W,225444,A
```

GLL	= indicatif « Positionnement Géographique Longitude / Latitude - GPS »
4916.75,N	= Latitude 49° 16,75' Nord soit 49° 16’ 45" Nord
12311.12,W	= Longitude 123° 11,12' Ouest soit 123° 11’ 7" Ouest
225444	= acquisition du Fix à 22:54:44 UTC
A	= indication de données valides

ANNEXE A3
Documentation sur les Sémaphores RTAI

RTAI (*Real Time Application Interface*) est une extension libre du noyau Linux lui apportant des fonctionnalités temps réel dures.

À proprement parler, RTAI n'est pas un système d'exploitation en temps réel, tel que Vxworks, OS-9 ou QNX. Linux n'est pas et ne sera jamais un système temps réel. RTAI est un noyau temps réel à priorité fixe qui exécute Linux en temps que tâche de faible priorité. Pour celà, le noyau RTAI s'intercale entre le matériel et le noyau Linux d'origine.

Les tâches temps réel développées sont donc concurrentes du noyau Linux.

Comme les autres OS temps-réel, RTAI offre aux applications au moins les services suivants :

- Une couche de gestion de matériel traitant les interruptions ou les polling de processeur/périphérique ;
- Un ordonnanceur programmable traitant l'activation de processus, priorités, par tranche de temps ;
- Des moyens de communications notamment avec le noyau Linux.

Parmi les moyens de communication disponibles sous RTAI, on trouve des mécanismes comme la gestion des files FIFO, les messages, la mémoire partagée et les sémaphores...

Sémaphores

Description

Les fonctions de gestion des sémaphores appartiennent à la librairie FIFO de RTAI. Le fichier d'entête de cette librairie est *rtai_fifos.h*. Les sémaphores binaires peuvent avoir de multiples utilisations. Ils permettent notamment la synchronisation des échanges de données avec la mémoire partagée. Les fonctions de gestion des sémaphores peuvent être appelée par les tâches temps réel (espace noyau) ou les processus linux (espace utilisateur). Cependant, seules les fonctions non bloquantes sont utilisables par les tâches temps réel. Les six fonctions sont listées ci-dessous :

rtf_sem_init(fd, init_val);	*Non bloquant*
rtf_sem_wait(fd);	*bloquant*
rtf_sem_trywait(fd);	*Non bloquant*
rtf_sem_timed_wait(fd, ms_delay);	*Non bloquant*
rtf_sem_post(fd);	*Non bloquant*
rtf_sem_destroy(fd);	*Non bloquant*

int rtf_sem_init (unsigned int minor, int value)

Initialise et crée un sémaphore binaire identifié par un descripteur de fichier ou un numéro de FIFO. Cette fonction peut être utilisée dans les espaces utilisateur ou noyau.

Paramètres : *minor* est le numéro du sémaphore. Les sémaphores doivent être associé à une FIFO pour des besoins d'identification.
value valeur initiale du sémaphore. Doit être choisie entre 0 et 1.

Valeur retournée : *0* en cas de succès, ou *EINVAL* si *minor* se réfère à un descripteur invalide.

int rtf_sem_destroy (unsigned int minor)

Efface un sémaphore binaire préalablement crée avec *rtf_sem_init()*. Cette fonction peut être utilisée dans les espaces utilisateur ou noyau. Une tâche bloquée sur ce sémaphore retournera une erreur s'il est détruit.

Paramètres : *minor* est le descripteur de sémaphore.

Valeur retournée : *0* en cas de succès, ou *EINVAL* si *minor* se réfère à un descripteur invalide.

int rtf_sem_post (unsigned int minor)

Rendre le sémaphore. Le sémaphore est activé et le premier processus qui attend la file démarrera. Cette fonction peut être utilisée dans les espaces utilisateur ou noyau.

Paramètres : *minor* est le descripteur de sémaphore.

Valeur retournée : *0* en cas de succès, ou *EINVAL* si *minor* se réfère à un descripteur invalide.

int rtf_sem_wait (unsigned int minor)

Prendre le sémaphore. Cette fonction attend qu'un évènement soit signalé par le sémaphore. La valeur du sémaphore est positionnée à 1 pour être testée et mise à 0. S'il y a 1, *rtf_sem_wait()* se termine immédiatement, sinon le processus appelant est bloqué et ajouté dans la file d'attente dans l'ordre de priorité déterminé par la priorité temps réelle POSIX. Le processus est débloqué s'il est en première place dans la file d'attente et que le sémaphore est positionné par une tâche ou si le sémaphore est détruit. Comme cette fonction peut être bloquante, elle ne doit être utilisée que dans l'espace utilisateur.

Paramètres : *minor* est le descripteur de sémaphore.

Valeur retournée : *0* en cas de succès, ou *EINVAL* si *minor* se réfère à un descripteur invalide.

int rtf_sem_timed_wait (unsigned int minor, int ms_delay)

Récupère le sémaphore avec un temps maximal. Cette fonction est presque identique à *rtf_sem_wait()* mais la libération du processus dans la file d'attente survient s'il est en première place dans la file d'attente et que le sémaphore est positionné par une tâche, si le temps imparti est écoulé ou si le sémaphore est détruit. Comme cette fonction peut être bloquante, elle ne doit être utilisée que dans l'espace utilisateur.

Paramètres : *minor* est le descripteur de sémaphore.
ms_delay est le délai d'attente en milliseconde.

Valeur retournée : *0* en cas de succès, ou *EINVAL* si *minor* se réfère à un descripteur invalide.

int rtf_sem_trywait (unsigned int minor)

Récupère le sémaphore sans blocage. Cette fonction est similaire à *rtf_sem_wait()* mais elle ne bloque jamais l'appelant. Si le sémaphore n'est pas libre *rtf_sem_trywait()* retourne immédiatement et le sémaphore demeure inchangé. Cette fonction n'est pas bloquante elle peut donc être utilisée dans les espaces utilisateur ou noyau.

Paramètres : *minor* est le descripteur de sémaphore.

Valeur retournée : *0* en cas de succès, ou *EINVAL* si *minor* se réfère à un descripteur invalide.

ANNEXE A4

Programmation Réseau (Syntaxe de quelques appels systèmes)

IP(4) Manuel du programmeur Linux IP(4)

NAME

ip - Implémentation Linux du protocole IPv4.

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

tcp_socket = socket(PF_INET, SOCK_STREAM, 0);
raw_socket = socket(PF_INET, SOCK_RAW, protocol);
udp_socket = socket(PF_INET, SOCK_DGRAM, protocol);
```

DESCRIPTION

Linux implémente le Protocole Internet (IP) version 4, décrit dans les RFC 791 et RFC 1122. ip contient une implémentation du multicasting niveau 2 conforme à la RFC 1112. Elle contient aussi un routeur IP comprenant un filtre de paquets.

L'interface de programmation est compatible avec les sockets BSD. Pour plus de renseignements sur les sockets, voir socket(7).

Une socket IP est créée par la fonction socket(2) invoquée sous la forme socket(PF_INET, type_socket, protocole). Les types valides des sockets sont SOCK_STREAM pour ouvrir une socket tcp(7), SOCK_DGRAM pour ouvrir une socket udp(7), ou SOCK_RAW pour ouvrir une socket raw(7) permettant d'accéder directement au protocole IP. Le protocole indiqué est celui inscrit dans les entêtes IP émis ou reçus. Les seules valeurs valides pour le protocole sont 0 et IPPROTO_TCP pour les sockets TCP, et 0 et IPPROTO_UDP pour les sockets UDP. Pour les sockets SOCK_RAW on peut indiquer un protocole IP IANA valide dont la RFC 1700 précise les numéros assignés.

Lorsqu'un processus veut recevoir de nouveaux paquets entrants ou connexions, il doit attacher une socket à une adresse d'interface locale en utilisant bind(2). Une seule socket IP peut être attachée à une paire (adresse, port) locale donnée. Lorsqu'on indique INADDR_ANY lors de l'attachement, la socket sera affectée à toutes les interfaces locales. Si on invoque listen(2) ou connect(2) sur une socket non affectée, elle est automatiquement attachée à un port libre aléatoire, avec l'adresse locale fixée sur INADDR_ANY.

L'adresse locale d'une socket TCP qui a été attachée est indisponible pendant quelques instants après sa fermeture, à moins que l'attribut SO_REUSEADDR ait été activé. Il faut être prudent en utilisant ce drapeau, car il rend le protocole TCP moins fiable.

FORMAT D'ADRESSE

Une adresse de socket IP est définie comme la combinaison d'une adresse IP d'interface et d'un numéro de port. Le protocole IP de base ne fournit pas de numéro de port, ils sont implémentés par les protocoles de plus haut-niveau comme udp(7) et tcp(7). Sur les sockets raw, le champ sin_port contient le protocole IP.

```
struct sockaddr_in {
  sa_family_t      sin_family;         /* famille d'adresses : AF_INET   */
  u_int16_t        sin_port;           /* port dans l'ordre d'octets réseau */
  struct in_addr   sin_addr;           /* adresse Internet   */
};


struct in_addr {
  u_int32_t        s_addr;             /* Adresse dans l'ordre d'octets réseau */
};
```

sin_family est toujours rempli avec AF_INET. C'est indispensable. Sous Linux 2.2, la plupart des fonctions réseau renvoient EINVAL lorsque cette configuration manque. sin_port contient le numéro de port, dans l'ordre des octets du réseau. Les numéros de ports inférieures à 1024 sont dits réservés. Seuls les processus avec un UID effectif nul ou la capacité CAP_NET_BIND_SERVICE peuvent appeler bind(2) pour ces ports.
Notez que le protocole IPv4 en tant que tel n'a pas de concept de ports, ils sont seulement implémentés par des protocoles de plus haut-niveau comme tcp(7) et udp(7).

sin_addr est l'adresse IP de l'hôte. le membre addr de la structure struct in_addr contient l'adresse de l'interface de l'hôte, dans l'ordre des octets du réseau. in_addr ne doit être manipulé qu'au travers des fonctions de bibliothèque inet_aton(3), inet_addr(3), inet_makeaddr(3) ou directement par le système de résolution des noms (voir gethostbyname(3)). Les adresses IPv4 sont divisées en adresses unicast, broadcast et multicast. Les adresses unicast décrivent une interface unique d'un hôte, les adresses broadcast correspondent à tous les hôtes d'un réseau, et les adresses multicast représentent tous les hôtes d'un groupe multicast.

Les datagrammes vers des adresses broadcast ne peuvent être émis et reçus que si l'attribut de socket SO_BROADCAST est activé. Dans l'implémentation actuelle, les sockets orientées connexion ne sont autorisées que sur des adresses unicast.

Remarquez que l'adresse et le port sont toujours stockés dans l'ordre des octets du réseau. Cela signifie qu'il faut invoquer htons(3) sur le numéro attribué à un port. Toutes les fonctions de manipulation d'adresse et port de la bibliothèque standard fonctionne dans l'ordre du réseau.

Il existe plusieurs adresses particulières :
INADDR_LOOPBACK (127.0.0.1) correspond toujours à l'hôte local via le périphérique loopback ;
INADDR_ANY (0.0.0.0) signifie un attachement à n'importe quelle adresse ;
INADDR_BROADCAST (255.255.255.255) signifie n'importe quel hôte et à le même effet que INADDR_ANY pour des raisons historiques.

VOIR AUSSI

sendmsg(2), recvmsg(2), socket(4), netlink(4), tcp(4), udp(4), raw(4), ipfw(4)

SOCKET(2) Manuel du programmeur Linux SOCKET(2)

NOM

socket - Créer un point de communication.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

DESCRIPTION

Socket crée un point de communication, et renvoie un descripteur.

Le paramètre domain indique un domaine de communication, à l'intérieur duquel s'établira le dialogue. Ceci permet de sélectionner la famille de protocole à employer. Ces familles sont définies dans le fichier linux/socket.h.

Les formats actuellement proposés sont :

AF_UNSPEC	Famille non spécifiée, laisser le système la déterminer.
AF_UNIX	Protocoles locaux internes UNIX (pipe,...)
AF_INET	Protocoles Internet (UDP, TCP, etc...)
…	
AF_INET6	Réservé pour le projet IP version 6

Les sockets ont le type indiqué, ce qui fixe la sémantique des communications. Les types définis actuellement sont :

SOCK_STREAM	Support de dialogue garantissant l'intégrité, fournissant un flux de données binaires, et intégrant un mécanisme pour les transmissions de données hors-bande. Les sockets de ce type sont des flux full-duplex, similaires à des tubes.
SOCK_DGRAM	Transmissions sans connexion, non garantie, de datagrammes de longueur fixe, généralement courte.
SOCK_RAW	Transmissions internes au système, le type SOCK_RAW, ne peut être utilisé que par le Super-User.
SOCK_RDM	Transmission garantie de datagrammes
SOCK_SEQPACKET	Dialogue garantissant l'intégrité, pour le transport de datagrammes de longueur fixe. Le lecteur peut avoir à lire le paquet de données complet à chaque appel système read.

Le protocole à utiliser sur la socket est indiqué par l'argument protocol. Normalement il n'y a qu'un seul protocole par type de socket pour une famille donnée. Néanmoins rien ne s'oppose à ce que plusieurs protocoles existent, auquel cas il est nécessaire de le spécifier.

Le numéro de protocole dépend du domaine de communication de la socket. Voir protocols(5).

Une socket de type stream doit être connectée avant que des données puisse y être lues ou écrites. Une connexion sur une autre socket est établie par l'appel système connect(2). Une fois connectée les données y sont transmises par read(2) et write(2) ou par des variantes de send(2) et recv(2).

Quand une session se termine, on referme la socket avec close(2).

Les données hors-bande sont envoyées ou reçues en utilisant send(2) et recv(2).

Le protocole de communication utilisé pour implémenter les sockets stream garantit qu'aucune donnée n'est perdue ou dupliquée. Si un bloc de données, pour lequel le correspondant a suffisamment de place dans son buffer, n'est pas transmis correctement dans un délai raisonnable, la connexion est considérée comme inutilisable, et les appels systèmes renverront une valeur -1 en indiquant une erreur ETIMEDOUT dans la variable globale errno.

Eventuellement les protocoles peuvent maintenir les sockets en service en forçant des transmissions directes toutes les minutes en l'absence de toute autre activité. Une erreur est indiquée si aucune réponse n'est reçue sur une socket inactive pendant une période prolongée (par exemple 5 minutes).

Un signal SIGPIPE est envoyé au processus tentant d'écrire sur une socket inutilisable, forçant les programmes ne gérant pas ce signal à se terminer.

Les sockets de type SOCK_SEQPACKET emploient les mêmes appels systèmes que celles de types SOCK_STREAM, à la différence que la fonction read(2) ne renverra que le nombre d'octets requis, et toute autre donnée restante sera éliminée.

Les sockets de type SOCK_DGRAM ou SOCK_RAW permettent l'émission de datagrammes à des correspondants indiqués au moment de l'appel système send(2). Les datagrammes sont généralement lus par la fonction recvfrom(2), qui fournit également l'adresse du correspondant.

Un appel à fcntl(2) permet de préciser un groupe de processus qui recevront un signal SIGURG lors de l'arrivée de données hors-bande. Cette fonction permet également de valider des entrées/sorties non bloquantes, et une notification asynchrone des évènements par le signal SIGIO.

Les opérations sur les sockets sont représentées par des options du niveau socket. Ces options sont définies dans sys/socket.h. Les fonctions setsockopt(2) et getsockopt(2) sont utilisées respectivement pour fixer ou lire les options.

VALEUR RENVOYÉE

socket retourne un descripteur référençant la socket créée en cas de réussite. En cas d'échec -1 est renvoyé, et errno contient le code d'erreur.

ERREURS

EPROTONOSUPPORT	Le type de protocole, ou le protocole lui-même n'est pas disponible dans ce domaine de communication.
EMFILE	La table des descripteurs par processus est pleine.
ENFILE	La table des fichiers est pleine.
EACCES	La création d'une telle socket n'est pas autorisée.
ENOBUFS	Pas suffisamment d'espace pour allouer les buffers nécessaires.

BIND(2) Manuel du programmeur Linux BIND(2)

NOM

bind - Fournir un nom à une socket.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int  bind(int  sockfd, struct sockaddr *my_addr, int addrlen);
```

DESCRIPTION

bind fournit à la socket sockfd l'adresse locale my_addr. my_addr est longue de addrlen octets. Traditionnellement cette operation est appelée "assignation d'un nom à une socket" (Quand une socket est créée, par l'appel-système socket(2),elle existe dans l'espace des noms mais n'a pas de nom assigné).

NOTES

Assigner un nom dans le domaine UNIX crée une socket dans le système de fichiers, qui devra être détruite par le créateur une fois qu'il n'en a plus besoin, en utilisant unlink(2).

Les règles d'assignation de nom varient suivant le domaine de communication. Consultez le manuel du programmeur Linux section 4 pour de plus amples informations.

VALEUR RENVOYÉE

bind renvoie 0 s'il réussit, ou -1 s'il échoue, auquel cas errno contient le code d'erreur.

ERREURS

EBADFF	sockfd n'est pas un descripteur valide.
EACCESS	L'adresse est protégée et l'utilisateur n'est pas le Super-User.
ENOTSOCK	L'argument est un descripteur de fichier, pas une socket.
EADDRINUSE	La socket a déjà une adresse assignée.

Les erreurs suivantes sont spécifiques au domaine UNIX(AF_UNIX):

EINVAL	La longueur addr_len est fausse, ou la socket n'est pas de la famille AF_UNIX.
EROFS	L'i-noeud de la socket se trouverait dans un système de fichiers en lecture seule.
EFAULT	my_addr pointe en dehors de l'espace d'adresse accessible.
ENAMETOOLONG	my_addr est trop long
ENOENT	Le fichier n'existe pas.
ENOMEM	pas assez de mémoire pour le noyau.
ENOTDIR	Un composant du chemin d'accès n'est pas un répertoire.
EACCES	L'accès à un composant du chemin d'accès n'est pas autorisé.
ELOOP	my_addr contient des références circulaires (à travers un lien symbolique).

SEND(2) Manuel du programmeur Linux SEND(2)

NOM

send, sendto, sendmsg - Envoyer un message sur une socket.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int send(int s, const void *msg, int len , unsigned int flags);
int sendto(int s, const void *msg, int len , unsigned int flags,
                const struct sockaddr *to, int tolen);
int sendmsg(int s, const struct msghdr *msg , unsigned int flags);
```

DESCRIPTION

Send, sendto, et sendmsg permettent de transmettre un message à destination d'une autre socket. Send ne peut être utilisé qu'avec les sockets connectée alors que sendto et sendmsg peuvent être utilisés tout le temps.

L'adresse de la cible est donnée par to avec la longueur tolen. La longueur du message est indiquée dans len. Si le message est trop long pour être transmis intégralement au protocole sous-jacent, l'erreur EMSGSIZE sera déclenchée et rien ne sera émis.

Aucune indication d'échec de distribution n'est fournie par send. Seules les erreurs locales sont détectées, et indiquées par une valeur de retour -1.

Si la socket ne dispose pas de la place suffisante pour le message, alors send va bloquer, à moins que la socket ait été configurée en mode d'entrées/sorties non-bloquantes. On peut utiliser l'appel système select(2) pour vérifier s'il est possible d'émettre des données.

Le paramètre flags peut contenir une ou plusieurs des options suivantes

```
#define  MSG_OOB          0x1 /* Traiter les données hors-bande */
#define  MSG_DONTROUTE    0x4 /* Contourner le routage      */
```

L'option MSG_OOB est utilisée pour émettre des données hors-bande sur une socket qui l'autorise (par ex : SOCK_STREAM). Le protocole sous-jacent doit également autoriser l'émission de données hors-bande.

MSG_DONTROUTE est utilisée par les programmes de diagnostique ou de routage.
Voir recv(2) pour une description de la structure msghdr.

VALEUR RENVOYÉE

Ces appels systèmes renvoient le nombre de caractères émis, ou -1 s'ils échouent, auquel cas errno contient le code d'erreur.

ERREURS

EBADF	Descripteur de socket invalide.
ENOTSOCK	L'argument s n'est pas une socket.
EFAULT	Un paramètre pointe en dehors de l'espace d'adressage accessible.
EMSGSIZE	La socket nécessite une émission intégrale du message mais la taille de celui-ci ne le permet pas.
EWOULDBLOCK	La socket est non-bloquante et l'opération demandée bloquerait.
EPIPE	L'écriture est impossible (correspondant absent), et le signal SIGPIPE a été ignoré par le processus.
ENOMEM	Pas assez de mémoire pour le noyau.
ENOBUFS	La file d'émission de l'interface réseau est pleine. Ceci indique généralement une panne de l'interface réseau, mais peut également être dû à un engorgement passager.

RECV(2) Manuel du programmeur Linux RECV(2)

NOM

recv, recvfrom, recvmsg - Recevoir un message sur une socket.

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(int s, void *buf, int len, unsigned int flags);
int recvfrom(int s, void *buf, int len, unsigned int flags,
                    struct sockaddr *from, int *fromlen);
int  recvmsg(int  s,  struct  msghdr  *msg,  unsigned  int flags);
```

DESCRIPTION

recvfrom et recvmsg sont utilisés pour recevoir des messages depuis une socket s, et peuvent servir à la lecture de données que la socket soit orientée connexion ou non.
Si from est non nul, et si la socket n'est pas orientée connexion, l'adresse de la source du message y est insérée. fromlen est un paramètre résultat, initialisé à la taille du buffer from, et modifié en retour pour indiquer la taille réelle de l'adresse enregistrée.

L'appel recv est normalement utilisé sur une socket connectée (voir connect(2)) et est équivalent à recvfrom avec un paramètre from nul. Comme ceci est redondant il est possible que recv ne soit plus maintenu dans le futur.

Ces trois routines renvoient la longueur du message si elles réussissent. Si un message est trop long pour tenir dans le buffer, les octets supplémentaires peuvent être abandonnés suivant le type de socket utilisé (voir socket(2)).

Si aucun message n'est disponible sur la socket, les fonctions de réception se mettent en attente, à moins que la socket soit non bloquante (voir fcntl(2)) auquel cas la valeur -1 est renvoyée, et errno est positionnée à EWOULDBLOCK.

Les fonctions de réception renvoient normalement les données disponibles dans la limite du paramètre len sans attendre d'avoir reçu le nombre exact réclamé. Ce comportement peut être modifié avec les options de socket SO_RCVLOWAT et SO_RCVTIMEO décrites dans getsockopt(2). L'appel select(2) peut permettre de déterminer si des données supplémentaires sont disponibles.

L'argument flags de l'appel recv est constitué par un OU binaire (|) entre les valeurs suivantes

MSG_OOB	traiter les données hors-bande
MSG_PEEK	lire sans enlever les données
MSG_WAITALL	attendre le nombre exact ou une erreur

L'option MSG_OOB permet la lecture des données hors-bande qui ne seraient autrement pas placées dans le flux de données normales. Certains protocoles placent ces données hors-bande en tête de la file normale, et cette option n'a pas lieu d'être dans ce cas.

L'option MSG_PEEK permet de lire les données en attente dans la file sans les enlever de cette file. Ainsi une lecture ultérieure renverra à nouveau les mêmes données.

L'option MSG_WAITALL demande que l'opération de lecture soit bloquée jusqu'à ce que la requête complète soit satisfaite. Toutefois la lecture peut renvoyer quand même moins de données que prévu si un signal est reçu, ou si une erreur ou une déconnexion se produisent.

VALEUR RENVOYÉE

Ces fonctions renvoient le nombre d'octets reçus si elles réussissent, ou -1 si elles échouent, auquel cas errno contient le code d'erreur.

ERREURS

EBADF	L'argument s n'est pas un descripteur valide.
ENOTCONN	La socket est associée à un protocole orienté connexion et n'a pas encore été connectée (voir connect(2) et accept(2)).
ENOTSOCK	L'argument s ne correspond pas à une socket.
EWOULDBLOCK	La socket est non-bloquante et aucune donnée n'est disponible, ou un délai de timeout a été indiqué, et il a expiré sans que l'on ait reçu quoi que ce soit.
EINTR	Un signal a interrompu la lecture avant que des données soient disponibles.
EFAULT	Un buffer pointe en dehors de l'espace d'adressage accessible.

Annexe 5

Les exceptions en C++

I – Le mécanisme try...catch...throw

La complexité de plus en plus grandes des applications informatiques, impliquant plusieurs développeurs, des bibliothèques de bas ou de haut niveau, des DLLs (ou équivalent) conduit parfois à des situations anormales ou exceptionnelles lors de l'utilisation (souvent par le client) de l'application. Les composants de bas niveau voient cette situation, mais ne savent pas à qui ni comment rendre compte, et ne peuvent ni ne doivent pas non plus décider de la conduite à tenir.

Il y a quelques années, ces situations se traduisaient par des plantages, des coups de téléphones et des dépannages d'urgence via des patches ou des versions « vite corrigées » des logiciels.

Pourtant, l'auteur des fonctions de haut niveau connaît à priori la manière de réagir aux situations anormales, mais ne peut pas toujours les anticiper ni les détecter. Le langage C++ (comme de nombreux autres langages modernes) met en place un « **mécanisme des exceptions** » permettant de manière générale à des fonctions appelées de notifier aux fonctions appelantes l'avènement d'une erreur interne.

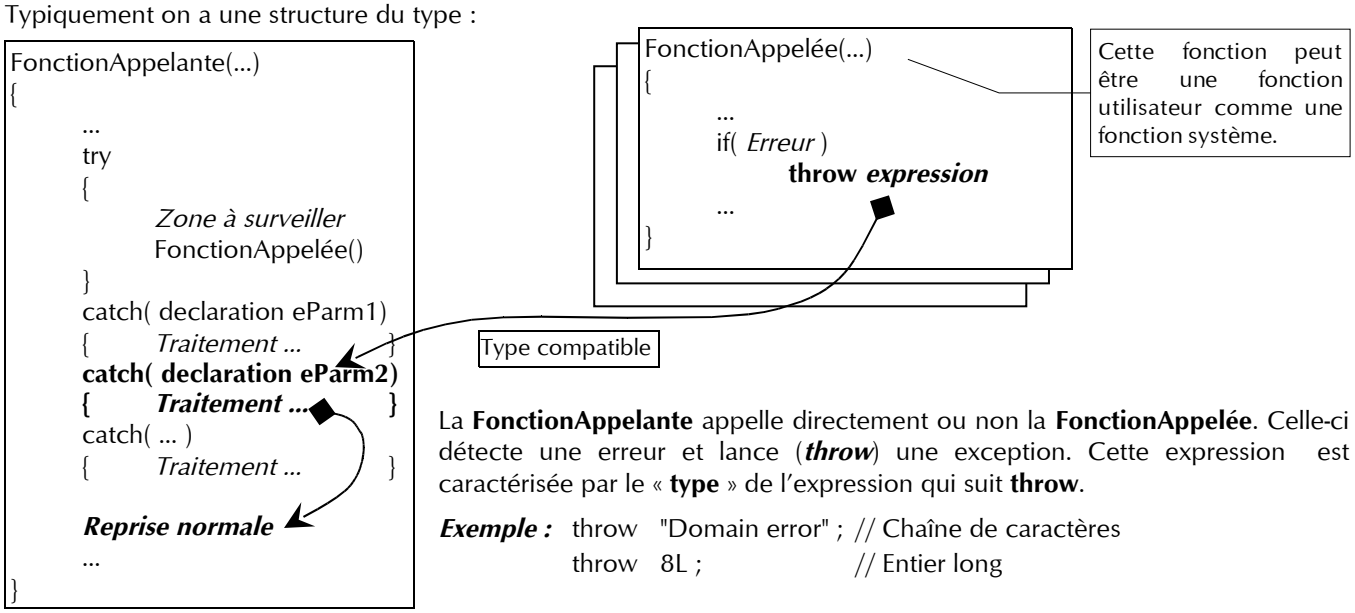
I-1 : Le fonctionnement du mécanisme

Les caractéristiques de ce mécanisme sont les suivantes :

- La fonction appelée qui détecte un événement exceptionnel construit une exception et la « lance » (*throw*) vers la fonction appelante ;
- Une exception est une instance (objet) d'une classe spécialement définie dans ce but (souvent une classe dérivée de la classe exception) comportant diverses informations utiles à la caractérisation de l'événement à signaler ; Ce peut être un nombre ou une chaîne.
- une fois lancée, l'exception traverse la fonction qui l'a lancée, sa fonction appelante, la fonction appelante de la fonction appelante, etc., jusqu'à atteindre une fonction active (c.-à-d. une fonction commencée et non encore terminée) qui a prévu d' « attraper » (*catch*) ce type d'exception ;
- lors du lancement d'une exception, la fonction qui l'a lancée et les fonctions que l'exception traverse sont immédiatement terminées : les instructions qui restaient à exécuter dans chacune de ces fonctions sont abandonnées ; malgré son caractère prématuré, cette terminaison prend le temps de détruire les objets locaux de chacune des fonctions ainsi avortées ;
- si une exception arrive à traverser toutes les fonctions actives, car aucune de ces fonctions n'a prévu de l'attraper, alors elle produit la terminaison du programme.

Pour intercepter des exceptions au sein d'un bloc d'instructions, une fonction doit délimiter cette séquence d'instructions au sein d'un bloc fonctionnel **try** d'une part, et d'autre part doit associer un traitement à chaque exception qu'elle souhaite intercepter à l'aide d'un gestionnaire **catch**.

Typiquement on a une structure du type :



La **FonctionAppelante** appelle directement ou non la **FonctionAppelée**. Celle-ci détecte une erreur et lance (***throw***) une exception. Cette expression est caractérisée par le « **type** » de l'expression qui suit **throw**.

Exemple :
```
throw "Domain error" ; // Chaîne de caractères
throw 8L ;             // Entier long
```

L'exécution de la fonction appelée s'interrompt immédiatement après le **throw**, mais proprement : les destructeurs des objets locaux sont appelés avant de quitter la fonction.

On remonte désormais dans la pile des appels jusqu'à ce que les conditions suivantes soient vérifiées :

- L'appel de fonction ayant généré l'erreur a été fait depuis un bloc surveillé (bloc **try**).
- Un des blocs de traitement **catch** possède un paramètre compatible avec le type de l'expression associée au **throw**, c'est-à-dire :
  - De même type strictement, sans casting implicite.
  - La classe du **throw** est une classe dérivée de celle du **catch**.
- La fonction possède un bloc de type **catch(...)** qui intercepte toutes les exceptions.

Nota : Si aucune des fonctions - jusqu'au **main()** - n'intercepte l'exception, le programme s'arrête.

L'exécution reprend alors normalement à la suite de la dernière zone **catch**. Le code non exécuté de toutes les fonctions traversées est ignoré. L'exception n'est pas transmise, une fois traitée, aux fonctions appelantes. Il est cependant possible de relancer cette exception en réutilisant le mot-clé **throw** au sein des blocs **catch**, sans même renommer l'exception (on écrit simplement **throw ;**).

I-2 : Préciser le prototype des fonctions et méthodes

Le prototype **void Fonction( ) throw(int) ;** amène une précision quand au fonctionnement de la fonction. Celle-ci ne peut émettre (laisser échapper) que des exceptions de type **int**. D'autres exemples :

- **void Fonction( ) throw(...) ;** est susceptible de laisser échapper n'importe quelle exception.
- **void Fonction( ) throw( ) ;** ne laisse normalement échapper aucune exception.
- **void Fonction( ) throw(int , float) ;** peut laisser échapper des exceptions de type **int** ou **float**.

II – La classe « exception »

Le langage C++ définit l'existence d'une classe de base dont la déclaration est :

```
class exception
{
  public:
       exception() throw();
       exception(const exception &e) throw();
       exception &operator=(const exception &e) throw();
       virtual ~exception() throw();
       virtual const char *what() const throw();
};
```

Note : La réalité est malheureusement un tout petit peu plus compliquée...
Cette classe est bien une classe de base, et ne devrait jamais être utilisée hors du mécanisme de dérivation. En particulier, la méthode **what()** de la classe **exception** renvoie le message « **Unknown exception** ». La librairie standard dérive elle-même cette classe en de nombreuses versions.

III - Exemple

La classe Vector possède un constructeur qui prend en paramètre le nombre de composantes du vecteur. 0 est une valeur interdite qui lève une exception.

```
class Vector
{
  public :
      int *m_Datas ;
      ...
      Vector ( int Size) throw (exception)
      {
            if( Size ) m_Datas = new int [Size] ;
            else throw new exception ;
      }
      ...
};
```

Un programme appelant pourrait inclure les quelques lignes :

```
int main()
{
    try                  { Vector V( rand() % 100) ; }
    catch(exception e)   { cout << e.what() << " : pas de bol, hein?" ; }
}
```