

Section : Génie électrique

Option : Informatique et télématique

Etude d'un système et/ou d'un processus technique

Durée : 8 heures

Corrigé

Gestion de contrôle d'accès de marchandises

A) ÉTUDE SYSTÈMES OS9 : ARCHITECTURE PHYSIQUE

Remarque : Le système UC OS9 Mourepiane est en fait composé de deux systèmes :

- **UC OS9 (maître) : unité de contrôle d'accès.**
- **UD OS9 (esclave) : unité déportée d'application.**

Description du système (RACK UD-OS9) :

COMPOSANTS MATERIELS :

Le matériel se présente sous la forme d'un rack 19 pouces 3U.

L'unité centrale GMI-CPU10 (68030) a été conçue autour du Bus G96.

Cette démarche permet d'utiliser toutes les cartes au bus G64/G96, qui respectent cette norme.

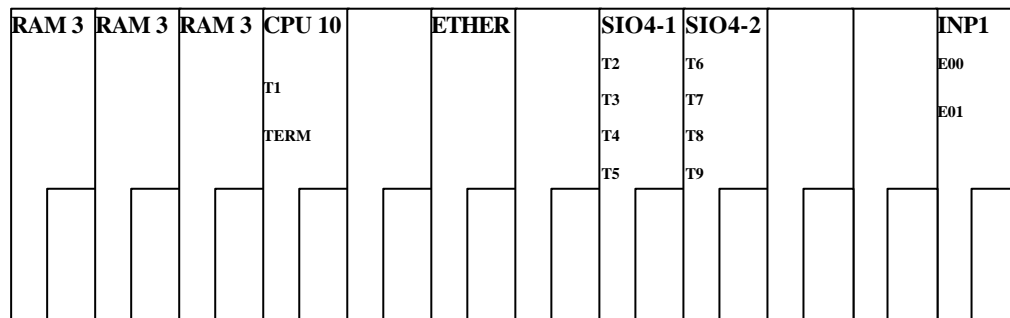
- Mémoire cachée interne
- MMU intégré
- Demande de DMA gérée par le Bus
- Interruptions auto-vectorisées,
- Interruptions vectorisées (191) avec protocole de lecture du vecteur,
- Zone VPA pour les périphériques en mode synchrone,
- Zone VPA pour les périphériques en mode asynchrone,
- Mémoire centrale extensible à 14 Moctets

L'intégration d'un chien de garde, de ressources internes (RAM, EPROM, Horloge et 2 liaisons de communication) sur cette carte permet de réaliser des applications industrielles.

Les cartes suivantes sont implantées :

- 1 Carte GMI-CPU10 avec processeur 68030
- 3 Cartes RAM 3
- 1 Carte Ethernet
- 2 Cartes de communication série (SIO4-1, SIO4-2)
- 1 Carte TOR (INP1)

Schéma du rack UD OS9



COMPOSANTS LOGICIELS :

La machine est livrée avec le système d'exploitation OS9 V2.4 et le progiciel TCP/IP installés dans des EPROMS spécifiques.

Dans le répertoire des descripteurs (/h0/cmds/bootobjs) figurent les descripteurs des liaisons séries de la carte CPU (Term, t1) et des cartes SIO4 (t2, t3, t4, t5, t6, t7, t8, t9).

Première partie : Carte CPU 10

La carte CPU10 est équipée d'un microprocesseur MC68030

Question A-1.1

Quelle est la capacité d'adressage physique du 68030 ?

Quelle est la capacité d'adressage maximale de la carte GMI-CPU10 ?

Abstraction faite des lignes FC0,FC1,FC2:

Le 68030 a 32 fils d'adresses, il gère $2^{32} = 4.294.967.296$ octets soit 4 Gigaoctets

Dans le module, seuls 25 fils sont connectés, donc le module gère $2^{25} = 33.554.432$ octets soit 32 Mégaoctets.

Question A-1.2

Le bus G96 ne peut gérer que 24 lignes d'adresses (A0-A23). Comment le fabricant de la carte CPU10 a-t-il procédé pour gérer 32 Mégaoctets ?

Il a décalé de un les fils d'adresses A1->A0 et A24->A23 et se sert de /LDS et /UDS pour doubler la zone.

Question A-1.3

Quelles sont les positions des cavaliers permettant d'utiliser des EPROM de 512K, configurer la carte CPU ?

	J18	J20	J21	J22	J24
	o	o	o	o	o
274001					
(512Ko)	o	o	o	o	o
	o	o	o	o	o

La carte GMI-CPU10 permet de choisir des interruptions vectorisées et/ou auto vectorisées

Question A-1.4

Présenter sous forme de séquences le déroulement du traitement d'une interruption logicielle ?

1) Le registre d'état est copié dans un registre tampon interne au processeur.

Le bit S est mis à un pour effectuer le passage en mode superviseur.

Les bits relatifs à la fonction Trace sont mis à zéro, puisque cette fonction génère elle-même une exception.

S'il s'agit d'une interruption matérielle, les bits I2, I1, I0 du masque sont mis au niveau de l'interruption.

2) Il faut trouver l'adresse du sous-programme d'exception et donc faire l'acquisition du numéro de vecteur. Ce numéro est trouvé soit en interne pour la plupart des exceptions, soit en externe pour certains types d'interruptions

3) Il faut sauvegarder le contexte. Au minimum : le format de cette sauvegarde ainsi que le déplacement existant entre l'adresse du vecteur et le début de la table des vecteurs, le compteur programme, le registre d'état qui a été copié dans le registre tampon.

4) Le vecteur d'exception est chargé dans le compteur programme. Le déroulement du sous-programme d'exception peut commencer.

Question A-1.5

Pour accéder aux différents octets, le 68030 possède deux signaux SIZ0 et SIZ1

SIZ1	SIZ0	Nombre d'octets
1	1	3
1	0	2
0	1	1
0	0	4

Donner les équations de la Pal 10 permettant de sélectionner chaque octet (ne pas tenir compte de FC0, FC1, FC2) sachant que :

UUD2 valide D24-D31

LMD2 valide D8-D15

UMD2 valide D16-D23

LLD2 valide D0-D7

$$UUD2 = A0 * A1 * SIZ1 * SIZ0 * AS /$$

$UMD2 = A0 * A1 * SIZ1 * SIZ0 * AS /$
 $LMD2 = A0 * A1 * SIZ1 * SIZ0 * AS /$
 $LLD2 = A0 * A1 * SIZ1 * SIZ0 * AS /$

Question A-1.6

La carte GMI-CPU10 utilise des PALs pour réaliser le décodage d'adresses. En vous servant du découpage mémoire décrit en Annexe GMI-CPU10 A-1 et du schéma de la carte GMI-CPU10 A-1-2, complétez (avec des 0, 1 ou x) les tables ci-dessous afin d'assurer le décodage des zones RAM.

A24	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0							x	x	x	x	x	x	x	x	x	x	x	x				

000000/3FFFFFF Si bascule RAM/EPROM = 1 → RAM interne à la carte

Question A-1.7

Complétez (avec des 0, 1 ou x) les tables ci-dessous afin d'assurer le décodage des périphériques suivants: MC68681 Registre du circuit

Watch-Dog D0=1 → Déclenchement

A24	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1					1	1	1															1

MC68681 FB8001 Registre du circuit

A24	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1					1	1	0											1	1	1	1	1

Watch-Dog FB001F D0=1 → Déclenchement

Dans le cadre du décodage d'adresse des zones déterminées par le constructeur (voir Annexe GMI-CPU10 A-1) au travers de la Pal 10

Question A-1.8

Donnez les équations utilisant le minimum de lignes pour décodé chaque zone (uniquement les lignes d'adresse).

	A24	A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
VMA(Réservé)	1	1	1																						
VMA	1	1	0																						
RAM/EPROM	1	0	1																						
EPROM	1	0	0																						
VPA Asynch	0	1	1					1	1	1															
VPA Synch	0	1	1					1	1	0															
Zone redond	0	1	1					1	0				1												
VPA Ext Asy	0	1	1					1	0				0	1											
VPA Ext Syn	0	1	1					1	0				0	0											
VMA	0	1	1					0																	
Ext RAM	0	1	0																						
Ext RAM	0	0	1																						
RAM	0	0	0																						

Equations:

VMA(réservé) 1C00000/1EFFFFFF A24*A23*A22
VMA 1800000/1BFFFFFF A24*A23*A22/
RAM/EPROM 1400000/17FFFFFF A24*A23/*A22
EPROM 1000000/13FFFFFF A24*A23/*A22/

VPA Asynch	FB8000/ FBFFFF	A24/*A23*A22*A17*A16*A15
VPA Synch	FB0000/ FB7FFF	A24/*A23*A22*A17*A16*A15/
Zone redond	FA1000/ FAFFFF	A24/*A23*A22*A17*A16/*A12
VPA Ext Asyn	FA0800/ FA0FFF	A24/*A23*A22*A17*A16/*A12/*A11
VPA Ext Syn	FA0000/ FA07FF	A24/*A23*A22*A17*A16/*A12/*A11/
VMA	C00000/ F9FFFF	A24/*A23*A22*A17/
Ext RAM	800000/ BFFFFFF	A24/*A23*A22/
Ext RAM	400000/7FFFFFF	A24/*A23/*A22
RAM(interne)	000000/3FFFFFF	A24/*A23/*A22/

Deuxième partie : Carte RAM

Les cartes mémoires vives peuvent être équipées de différents types de mémoires. La machine doit avoir une capacité de 32 Mo de mémoire (60, 70 ou 80 ns) à 25 MHz avec contrôle de parité.

Question A-2.1

Quel type de barrettes SIMM faut-il choisir? Indiquer quels cavaliers utiliser et comment les configurer?

Carte équipée de 32 Mo de mémoire (60, 70 ou 80 ns) à 25 MHz

Soient 8 barrettes SIMM de 4Mo x 9 installées

Cavalier J1 position 2-3

Cavalier J4 installé

Cavalier J5 enlevé

Cavalier J2 et J3 enlevé

Le contrôle de parité est installé.

Question A-2.2

Que se passe-t-il lorsqu'une erreur de parité survient?

Une erreur bus est générée.

Troisième partie : Carte Ethernet

Les cartes ethernet sont caractérisées par un numéro unique.

Question A-3.1

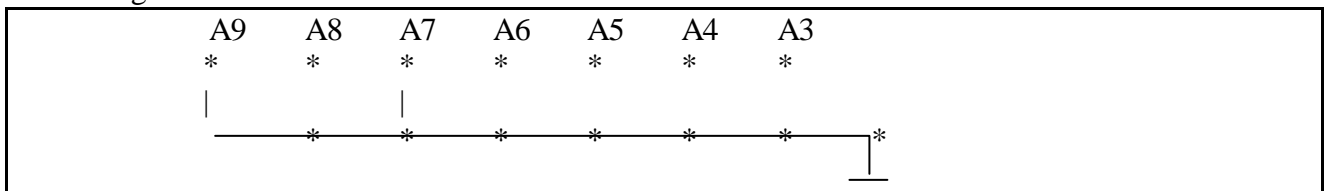
Comment est constitué ce numéro ?

Ce numéro est composé de 6 octets, trois sont caractéristiques du fabricant, trois correspondent à un numéro de carte.

La Carte Ethernet GMI ETHER (Annexe A-3 : GMI ETHER) est installée à l'offset \$1B8, comment configurer les cavaliers de J1 en sachant que l'adresse de base est VPA+0

Question A-3.2

Configurer les cavaliers de J1



Question A-3.2

Quelle est l'adresse physique du premier registre de la carte ethernet ?

\$FA0B71

(ligne Ai du 68030 reliée à ligne Ai-1 du Bus G96 + adresse impaire)

Question A-3.3

Quelles sont les couches du modèle OSI correspondant aux fonctionnalités d'une carte interface réseau?

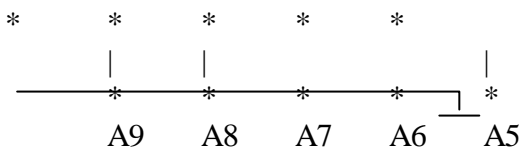
Les fonctionnalités relèvent des couches 1 (Physique) et 2 (liaison) du modèle OSI

Quatrième partie : Carte communication Série

La carte de communication série est dans la zone VPA (Annexe A- 4 : CARTE GMI SIO4)

Question A-4.1

Quelle est l'adresse physique du premier registre de la carte si les cavaliers sont configurés de la manière suivante?



Adresse \$FA0801 (adresse de base VPA asynchrone)

\$FA0981 (ligne Ai du 68030 reliée à ligne Ai-1 du Bus G96 + adresse impaire)

Question A-4.2

Quelle est l'adresse du DUART1?

\$FA09A1 (\$FA0981 + \$20)

Question A-4.3

Citer plusieurs protocoles de communication série ainsi que leur type (matériel, logiciel)

Logiciel: XON-XOFF
 Matériel: RTS-CTS
 DTR-DSR

Le tableau suivant décrit les caractéristiques des liaisons séries

Port	Interface	Format	Type de liaison
/term			
/t1			
/t2	RS485	9600,8,1,sans	RTS-CTS
/t3	RS485	9600,8,1,sans	RTS-CTS
/t4	Modem	9600,8,1,sans	XON-XOFF
/t5	Modem	9600,8,1,sans	XON-XOFF
/t6		9600,8,1,sans	XON-XOFF
/t7		9600,8,1,sans	XON-XOFF
/t8		9600,8,1,sans	XON-XOFF
/t9	Modem	4800,8,1,sans	XON-XOFF

Les ports /t2 et /t3 ont une interface RS485

Question A-4.4

Quels sont les critères qui permettent de choisir entre les liaisons RS485, RS232 et une liaison par modem ?

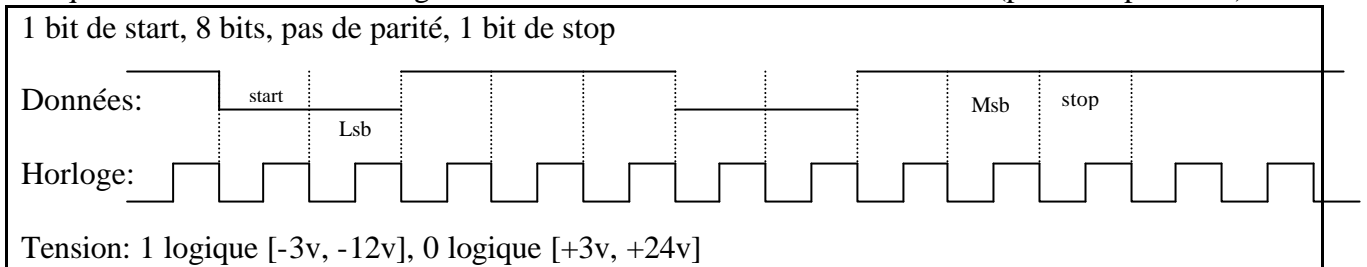
La distance entre les terminaux et l'unité centrale (ordre croissant) : RS232, RS485, Modem

La sensibilité au bruit (ordre croissant) : RS485, RS232.

L'interface RS232, modem sont point à point, la RS485 peut être multipoints.

Question A-4.5 (communication série RS232)

Précisez la nature des signaux échangés (tension). Tracez un chronogramme qui représente uniquement l'évolution de la ligne d'émission lors de l'envoi d'un caractère (par exemple 0x31).



Question A-4.6 (communication série RS232)

Sur une DB25 la voie 1 est associée à la masse châssis. A quoi sert-elle? Est-elle identique à la masse électrique? Comment la câble-t-on?

Elle permet le « blindage » du câble. Elle ne doit être câblée que d'un seul coté. Et ce n'est en aucun cas la masse électrique 0v.

Question A-4.7 (communication série RS232)

Expliquez le rôle des protocoles matériels et logiciels (par exemple RTS-CTS et XON-XOFF).

XON_XOFF

Le récepteur quand 80% de son buffer est plein, génère un XOFF (émission d'un caractère de contrôle) en direction de l'émetteur.

L'émetteur prendra en compte cette demande au bout d'un temps de propagation et de traitement, il arrête son émission.

Dès que le récepteur a vidé son buffer (taux de remplissage 20 %), il génère un XON (émission d'un caractère de contrôle) en direction de l'émetteur.

L'émetteur prendra en compte cette demande au bout d'un temps de propagation et de traitement, il reprend (continue) son émission.

B) Étude du système UC. Première partie : Spécification

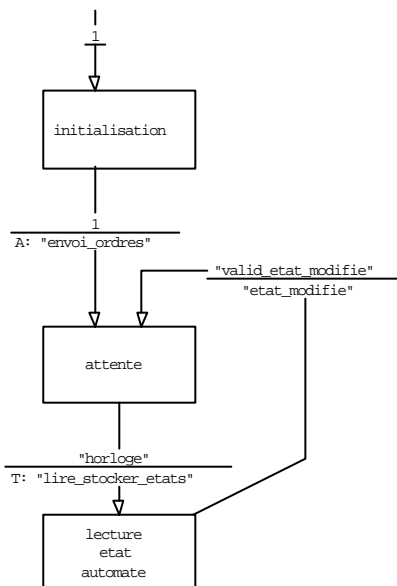
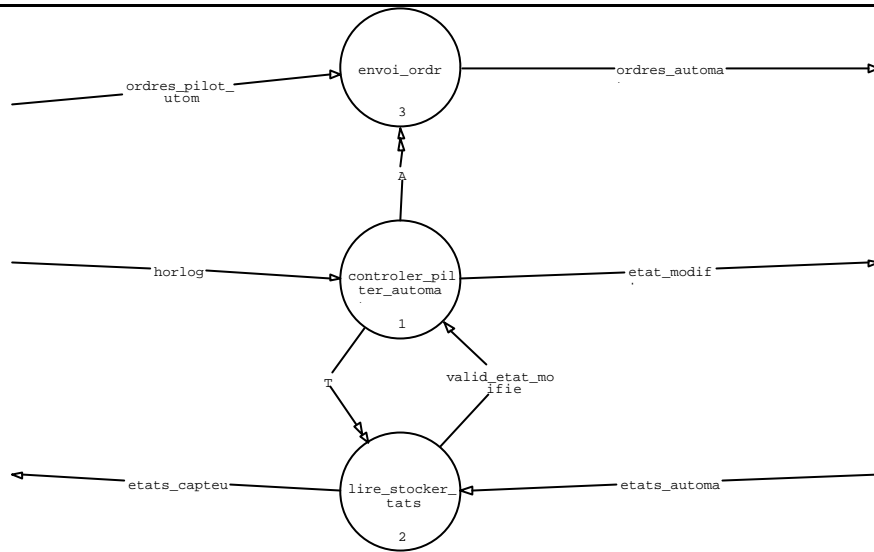
Question B-1

En cohérence avec les documents suivants :

Description fonctionnelle des systèmes, cf Annexe B-1

Dossier Spécification (méthode Ward Mellor) du sous système «système UC OS9» , cf Annexe B-1-1.

Réaliser la description détaillée (DFD, DFC et CSPEC) du processus «pilot_automate» assurant les échanges entre le «système UC OS9» et l'automate principal. Les processus de la décomposition, les flots de contrôles ou données que vous identifierez devront avoir des noms évocateurs.



Name: valid_etat_modifie
Type: Controlflow

Name: lire_stocker_etats
Type: Process

Name: attente
Type: State

Name: initialisation
Type: State

Name: lecture etat automate
Type: State

Name: controleur_piloteur_automate
Type: Control process

Name: envoi_ordres
Type: Process

@IN = ordres_pilot_autom
@OUT = ordres_automate

```
@PSPEC envoi_ordres
begin
  Lire ordres_pilot_autom
  Generer ordres_automate
end
@
```

```
@IN = etats_automate
@OUT = etats_capteurs
@OUT = valid_etat_modifie
```

```
@PSPEC lire_stocker_etats
begin
  Lire etats_automate
  Stocker etats_capteurs
  Generer valid_etat_modifie
end
@
```


Question B-2.3

En cohérence avec les choix précisés ci-dessus; en langage C, quelle représentation des données proposez-vous pour la zone de stockage "états capteurs».

La zone de stockage "etats_capteurs" doit contenir 76 informations binaires. On peut envisager, en cohérence avec l'annexe A_B1 les regroupements d'informations suivants qui conduisent à une structure de taille minimisée.

```
typedef struct Etats_capteurs {
    int  bar_VL_ondul:6 ;           /* 4 bits : dédiés Barriere VL + 2 bits dédiés Onduleur */
    int  bar_PL_sas:8 ;           /* 6 bits : dédiés Barriere PL + 2 bits dédiés Sas */
    int  port_nord:8 ;           /* 8 bits : dédiés Portail SNCF nord */
    int  port_centre:8 ;         /* 8 bits : dédiés Portail SNCF centre */
    int  port_sud_ecm:16 ;        /* 16 bits : dédiés Portail SNCF sud + ECM */
    int  port_eng_parc:8 ;        /* 8 bits : dédiés Portail SNCF engins de parc */
    int  info_reinj: 7;          /* 7 bits : dédiés informations Portail de réinjection */
    int  info_ES:15 ;           /* 7 bits : dédiés informations Portail Entrées terminal
                                7 bits : dédiés informations Portail Sorties terminal
                                1 bit dédié défaut dialogue automate local portail
                                ES */
} ETATS_CAPTEURS;
```

Question B-3.1

Justifier le choix technologique d'un pipe nommé.

Le système d'exploitation OS9 propose le concept de pipe nommé qui intègre les facilités suivantes:

- ♦ transfert purement séquentiel et synchronisation faite automatiquement par OS9
- ♦ peut être ouvert par plusieurs process indépendants
- ♦ la taille allouée au pipe est fixée lors de sa création
- ♦ pas de désallocation tant qu'il existe de l'information à consommer dans le pipe

Dans cette application:

- ♦ Les taches sont indépendantes
- ♦ Un tube de faible taille sera suffisant (flot « ordres_pilot_autom » = 4 octets -28 bits nécessaires-);

donc, en envisageant un très improbable engorgement, une taille de 32 octets semble très suffisante.

Question B-3.2

Proposer, une fonction creat_pipe(nom_pipe, ...) écrite en langage C, qui assure la création d'un pipe en cohérence avec l'application.

```
int creat_pipe(char *nom_pipe, int taille)
{
    /* Création d'un pipe nommé: avec droits d'accès en lecture et écriture owner et public de "taille" octets */
    return( create(nom_pipe, _READ + _WRITE + S_ISIZE,
                  S_IWRITE + S_IWRITE + S_IWRITE + S_IWRITE + S_ISIZE , taille)
    );
}
```

Question B-4.1

Quelle(s) solution(s) proposez-vous afin de fiabiliser l'échange d'informations à travers la zone de mémoire partagée entre la tâche «producteur» et la tâche «consommateur»

Identifier les concepts existants sous OS9 68 K permettant de mettre en œuvre ce fonctionnement.

Zone de mémoire partagée : concept de datamodule.

Accès au datamodule via un sémaphore d'exclusion mutuelle : concept d'événement.

Question B-4.2

Ecrire, en langage C, la fonction "créer_datamodule(...)" conforme à la description suivante :

```
/* nom          : créer_datamodule
   fonction     : créer un data module (sur lequel d'autres tâches pourront se lier via
modlink() )
   entrées : nom du data module
           : taille de la zone de données
   retourne  : pointeur sur corps du data module (début de l'espace réservé aux données)
*/
char * créer_datamodule(char * nom, unsigned int taille)
{
    mod_exec *ptr_modul;
    char      *ptr_data;
    if ( (ptr_data = _mkdata_module(nom, taille, mkattreves(MA_REENT, 1),
                                MP_OWNER_READ | MP_OWNER_WRITE) ) != (char *) -1 )
    { /* init de ptr_modul: pointe sur début d'1 module OS9 -type struct mod_exec- */
        ptr_modul = (mod_exec *) ptr_data;
        ptr_data += ptr_modul->_mexec; /* ajout offset du 1er octet du corps du module =>
                                        ptr_data pointe sur 1er octet de la zone de
données*/
    }
    return( ptr_data );
}
```

Question B-4.3

En cohérence avec la description ci-dessus:

Proposer les séquences d'initialisation qui permettront à la tâche «consommateur» de lire, de façon fiable, l'information contenue dans la zone de mémoire commune via l'usage d'une fonction "lit_datamodule(...)".

Préciser les paramètres à passer à la fonction "lit_datamodule(...)".

Une conception correcte suppose que :

Un process d'initialisation ait créé :

- Le data module.
- L'événement, permettant la gestion de l'accès au datamodule, avec des valeurs cohérentes permettant une gestion par exclusion mutuelle; par exemple : ev_value_initiale= 0, wait_increment= 1, signal_increment= -1.

Le process « consommateur » qui réalise la lecture dans le datamodule se soit :

- Lié avec le data module déjà créé (modlink()), ce qui induit la connaissance d'un pointeur "ptr_data" sur la zone des données.
- Lié avec l'événement, permettant la gestion de l'accès au datamodule (_ev_link()), identifié par son ID : "id_event"

La primitive "lit_datamodule" aura pour paramètres :

- l'adresse de la zone de données, l'identificateur d'événement, adresse du buffer de stockage des données lues, le nombre d'octets à lire

Question B-4.4

Ecrire, en langage C, la fonction "lit_datamodule(...)"

```
/* prototype minimal : sans gestion d'erreur */
```

```
lit_datamodule(char *ptr_data, int id_event, char *buffer, int nb_octets)
```

```
{/* demande de prise de ressource (attente libération ressource si déjà occupée) */
```

```

ev_wait( id_event, 0, 0);      /* fenetre d'accès : ev_value ∈ [0 , 0] */
for (i=0; i<nb_octets; i++)   *buffer++ = *ptr_data++; /* lecture de la zone de données */
/* libération de la ressource */
ev_signal(id_event, 0);      /* le 1er process en attente de la ressource sera activé */
}

```

C) Étude de la partie réseau.

Première partie : Architecture réseau.

Question C-1

Identifier et décrire les caractéristiques principales des bus de terrain :

- ◆ Mis en œuvre au sein de cette application, s'il(s) existe(nt)?
- ◆ Que vous connaissez ?

Deux bus de terrain sont mis en œuvre :

modbus
unitelway

Autres bus de terrain : CAN, FIP, JBUS, ...

Question C-2

Quelle est la fonction d'un routeur?

Identifier, s'il(s) existe(nt) le(s) routeur(s) présent(s) sur le schéma d'architecture fourni en Annexe générale G-1.

Dans le cadre des caractéristiques et composants des routeurs, préciser :

- ◆ des interfaces LAN et WAN
- ◆ des protocoles réseaux
- ◆ des protocoles de routage

Les routeurs relient sur la couche 3 du modèle ISO des réseaux de topologies différentes.

Pas de routeur présent.

Interfaces LAN : Token ring, Ethernet, FDDI ou encore ATM. Liaisons par ports (ex: Ethernet AUI, BNC, RJ45) ou sous forme d'insertion.

Interfaces WAN : Liaisons à 64Kbits/s ,..., 2048 Kbits/s (liaison E1 en Europe). Interface courants en mode synchrone : RS449, V35, X21; en mode asynchrone : RS232; RNIS : BRI.

Protocoles réseaux : les routeurs routent un ou plusieurs protocoles réseaux. Les plus courants sont IP, IPX; mais on peut également citer DECnet, ApppleTalk, XNS, VINES.

Protocoles de routage : RIP (Router Information Protocole), OSPF (Open Shortest Path First), NLSP (Netware Link State Protocol), IGRP (produit Cisco), ...

Deuxième partie : communication réseau Unix - Windows

Afin d'autoriser la sortie de conteneurs, les sociétés de manutention dont les sièges sociaux sont situés hors du site de Mourepiane :

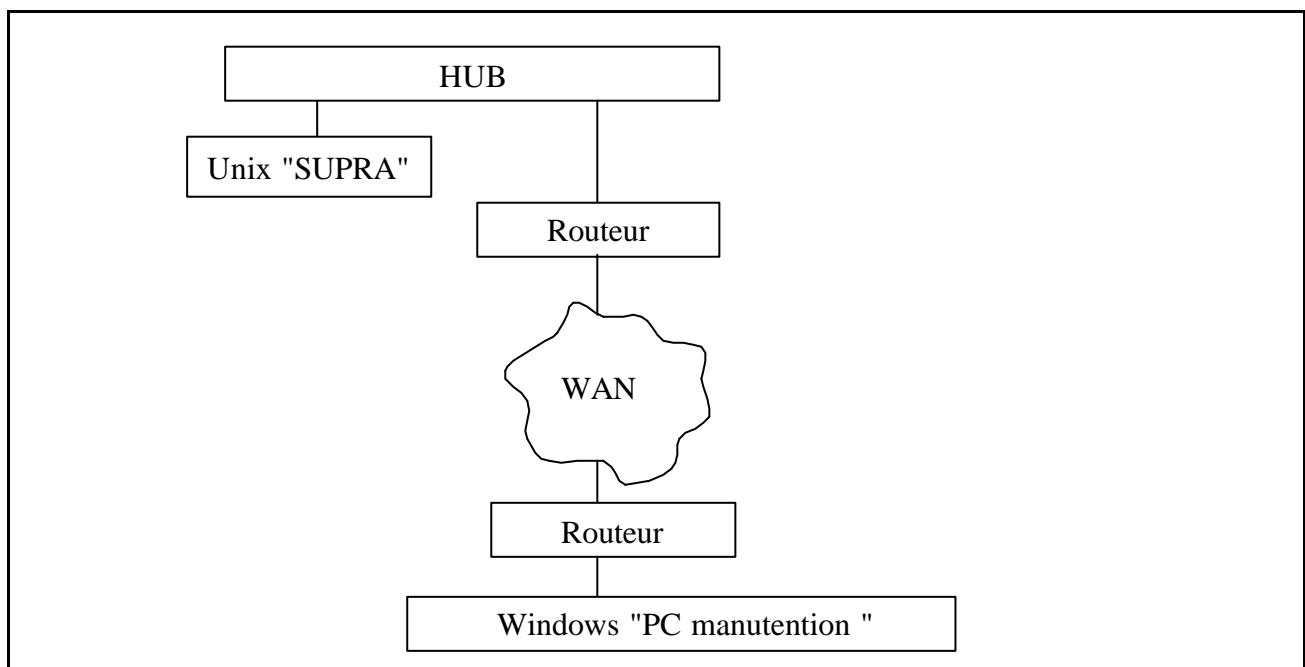
Transmettent, via un PC Windows, vers la machine Unix "SUPRA" du site de Mourepiane , un datagramme contenant :

- ◆ le login de la société de manutention (nom d'identification et password)
- ◆ le type de l'opération demandée:

- ◆ TYPE_LOGIN : uniquement contrôle du login (nom d'identification et password de la société de manutention)
- ◆ TYPE_CONVOI_ENTREE : contrôle du mot de passe + informations complémentaires (optionnelles).
- ◆ TYPE_CONVOI_SORTIE : contrôle du mot de passe + informations complémentaires.
- ◆ Des informations complémentaires :
 - ◆ la date et l'heure prévisionnelle du transfert des conteneurs
 - ◆ le numéro de badge
 - ◆ le numéro d'immatriculation du tracteur PL
 - ◆ le nombre de colis (0 : tracteur seul, 1..2)
 - ◆ des informations sur le colis 1 :
 - la nature (0 : conteneur, 1 : autre)
 - le mode (0 : normal, 1 : transit, 2 : douane)
 - le nombre de conteneurs (0..4)
 - la liste des conteneurs (tableau statique)
 - des commentaires additionnels
 - ◆ des informations sur le colis 2 : (idem colis 1)
- ◆ le crc

Question C-3

Compléter le schéma représentant l'architecture physique du réseau, compatible avec le schéma d'architecture fourni en Annexe générale G-1 permettant aux sociétés de manutention de communiquer avec le PC Unix "SUPRA".



Question C-4

Proposer, en langage C, au niveau de la machine "Unix SUPRA" une solution commentée qui gèrera la communication réseau par datagramme (PC(s) Windows manutention – Unix SUPRA). Vous disposez d'une «documentation Socket Unix» cf Annexe C-1 + «Windows Sockets» cf Annexe D-3

Ce **serveur** sera :

- Compatible avec le fonctionnement décrit ci-dessus.
- Compatible avec le programme source partiel fourni en Annexe D-1 (fichier manut.cpp).
- Lancé à partir du shell Unix en background.

Hypothèses : (pour les types à manipuler se référer à l'Annexe D-2 fichier types.h)

Vous disposez des primitives suivantes :

```
/* parcourt la BD : si nom du «user» non trouvé alors rend ERR_NOM_INCONNU
   sinon si password associe à nom user NON VALIDE alors rend ERR_PASSWD
   sinon rend OK */
```

unsigned short verifie_login(TRAME_CMD_UNIX *trame);

```
/* parcourt la BD : si numéro carte non trouvé ou carte invalide
   alors rend ERR_CARTE_NON_VALIDE
   sinon rend OK */
```

unsigned short verifie_carte(TRAME_CMD_UNIX *trame);

```
/* parcourt la BD : si numero tracteur non trouvé ou tracteur interdit
   alors rend ERR_TRACTEUR_INTERDIT
   sinon rend OK */
```

unsigned short verifie_tracteur(TRAME_CMD_UNIX *trame);

```
/* vérifie la cohérence des listes des 2 colis, si incohérence : rend ERR_LISTE_COLIS_1 et (ou)
   ERR_LISTE_COLIS_2 sinon rend OK */
```

unsigned short verifie_colis(TRAME_CMD_UNIX *trame);

```
/* stocke dans la BD toutes les informations associées à une trame validée issue de la société de
   manutention + la date et l'heure de cet enregistrement. Si pas d'erreur rend OK */
```

unsigned short stock_bd_trame(TRAME_CMD_UNIX *trame);

```
/* calcule le crc d'une trame: rend somme non signée sur 16 bits des octets d'une trame qcq –modulo
   0xFFFF- */
```

unsigned short calcule_crc(unsigned char *trame, unsigned short size)

```
/* vérifie le crc de la trame de commande reçue, si crc non valide: rend ERR_CRC
   sinon rend OK */
```

unsigned short verifie_crc(TRAME_CMD_UNIX *trame);

La solution partielle proposée valide le fonctionnement suivant :

- la société manutentionnaire émet 1 trame
- Si taille trame réceptionne par le serveur == OK alors
si crc == OK faire test cohérence : login + password, type trame + ...

```
char addr_ip_serv [] = {128, 6, 1, 107, 0}; /* serveur */
/* ===== */
void main ()
{
  struct sockaddr_in ad_serv, ad_client;
  int i, taille = sizeof(struct sockaddr_in);
  TRAME_CMD_UNIX trame_recv;
  TRAME_REP trame_rep;
```

```

.....
if ( ( soc_service = socket ( AF_INET, SOCK_DGRAM, 0 ) ) == -1 ) // Creation de la socket de service
{ printf ( "Erreur création socket \n" ); exit(1); }
/* Remplissage des adresses au sens Internet du serveur */
ad_serv.sin_family    = AF_INET;
ad_serv.sin_port      = htons ( SERV_PORT_UNIX );
ad_serv.sin_addr.s_addr = addr_ip_serv ? * (( u_long* ) addr_ip_serv) : 0;
/* une solution plus "finie" utiliserait gethostbyname() */
// ou : ad_serv.sin_addr.s_addr = htonl(INADDR_ANY); toutes les adresses possibles de cette station
/* Nommage de la socket du serveur*/
if ( bind ( soc_service, ( struct sockaddr* ) &ad_serv, sizeof ( ad_serv ) ) == -1 )
    { printf ( "Erreur nommage socket \n" );    clos_canal(soc_service); exit(2); }
for ( ; ; ) /* Boucle d'attente de requête */
{
    if (recvfrom(soc_service,(char *) &trame_rcv,sizeof(TRAME_CMD_UNIX),0,
                (struct sockaddr *) &ad_client, &taille) == sizeof(TRAME_CMD_UNIX))
    { /* Bonne réception de la trame : */
        trame_rep.reponse = REP_OK; /* initialisation : pas d'erreur */
        /* contrôle du CRC, si non OK => fin */
        if ( (trame_rep.err_type = verifie_crc(&trame_rcv)) == OK)
        {
            switch(trame_rcv.type)
            {
                case TYPE_LOGIN: // trame uniquement destinée a contrôler validité du couple: nom user - password
                    if ((trame_rep.err_type = verifie_login(&trame_rcv.login)) != OK)
                        trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
                    break;
                case TYPE_CONVOI_SORTIE: /* trame associée à une requête de sortie de convoi */
                    /* la vérification de la validité de tous les champs est nécessaire */
                    /* Contrôle du login : si OK => contrôle des autres champs sinon fin */
                    if ( (trame_rep.err_type = verifie_login(&trame_rcv.login)) != OK)
                        trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
                    else /* login = OK => contrôle de tous les autres champs : */
                    { /* contrôle numéro carte */
                        if ((trame_rep.err_type += verifie_carte(trame_rcv.carte)) != OK)
                            trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
                        /* contrôle numéro tracteur */
                        if ((trame_rep.err_type += verifie_tracteur(trame_rcv.tracteur)) != OK)
                            trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
                        /* contrôle listes des colis */
                        if ((trame_rep.err_type += verifie_colis(&trame_rcv)) != OK)
                            trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
                    }
                    break;
                case TYPE_CONVOI_ENTREE: // la vérification de la validité de tous les champs n'est nécessaire
                    /* le traitement est semblable à TYPE_CONVOI_SORTIE : non traite */
                    break;
                default: /* type de trame inconnu */
                    trame_rep.reponse = REP_ERR;
                    trame_rep.err_type = ERR_TYP_TRAME_INCONNU;
            }
        }
    }
}

```

```

        break;
} /* fin switch */
} /* fin traitement associe a crc OK */
else /* crc trame réceptionnée est incorrect */
    trame_rep.reponse = REP_ERR; /* erreur precisee ds champ err_type */
} /* fin if recvfrom(..) */
else /* si mauvaise réception de trame : */
    { trame_rep.reponse = REP_ERR; trame_rep.err_type = ERR_INDEFINIE ; }
/* si champs trame == OK alors stocker et dater informations contenues dans trame dans la Base de Données: */
if (trame_rep.reponse == REP_OK) stock_bd_trame( &trame_recv);
/* Retourner réponse a société manutentionnaire */
trame_rep.crc = calcul_crc( (unsigned char *) &trame_rep, sizeof(TRAME_REP));
sendto(soc_service,(char *)&trame_rep,sizeof(TRAME_REP),0,(const struct sockaddr *) &ad_client,taille);
} /* fin for */
}

```

Question C-5

On désire réaliser un contrôle de trame à chaque envoi et réception de trame réseau de type «struct Trame_Cmd_UNIX» (voir Annexe D-1, types.h):

Citez quelques méthodes de contrôle de trame et précisez leurs avantages ou inconvénients.

On vous demande de réaliser la fonction suivante *unsigned short calculer_crc (unsigned char *trame, int size)* où le résultat renvoyé est l'addition non signée (modulo 0XFFFF) de tous les octets de la trame de manière à satisfaire l'appel suivant:

```

if(trame.crc == calculer_crc((unsigned char *) trame,
sizeof(trame))

```

Cette fonction est utilisée par la fonction *verifie_crc()*.

Le but est de détecter une erreur lors de la transmission, il faudra donc comparer le message transmis à une valeur qui sera obtenue par un calcul sur le message. Cette valeur ne sera pas unique (des messages différents peuvent avoir la même valeur de contrôle), en plus le cas très rare où plusieurs erreurs sur un même message peuvent s'annuler et produire la même valeur (dans ce cas l'erreur ne sera pas détectée).

L'exemple le plus courant est le contrôle de parité dans la liaison série (parité paire, impaire).

```

unsigned short calculer_crc(unsigned char *trame, int size)
{
    unsigned short result = 0;
    for(size-=sizeof(unsigned int); size >= 0; size--) result+=trame[size];
    return(result);
}

```

Question C-6

On désire lancer automatiquement le processus **serveur** au démarrage de inetd (cf Annexe C-2), fournir des extraits des fichiers de configuration permettant ce fonctionnement

```

/etc/services :
    nom_prog    5002/udp

/etc/inetd.conf :
    nom_prog    dgram udp wait root /root/nom_prog    nom_prog

```

Question C-7

Fournir le code C du serveur :

- "Lancé" par inetd (cf Annexe C-2)
- Capable de gérer, à un instant t, qu'une seule requête issue de société manutentionnaire.

Vous disposez des primitives décrites à la question C-5.

La solution partielle proposée valide le même fonctionnement que celui de la question C5 :
Le processus serveur se termine dès qu'il a rendu son service

```

char addr_ip_serv [] = {128, 6, 1, 107, 0}; /* serveur */
/* ===== */
void main ()
{
struct sockaddr_in  ad_serv, ad_client;
int                i, taille = sizeof(struct sockaddr_in),
                  soc_service = 0; /* numero canal dedie a socket de service, cf doc inetd */
TRAME_CMD_UNIX trame_rcv;
TRAME_REP      trame_rep;
.....
if (recvfrom(soc_service,(char *) &trame_rcv,sizeof(TRAME_CMD_UNIX),0,
            (struct sockaddr *) &ad_client, &taille) == sizeof(TRAME_CMD_UNIX))
{ /* Bonne réception de la trame : */
trame_rep.reponse = REP_OK; /* initialisation : pas d'erreur */
if ( (trame_rep.err_type = verifie_crc(&trame_rcv)) == OK) /* contrôle du CRC, si non OK => fin */
{
switch(trame_rcv.type)
{
case TYPE_LOGIN: // trame uniquement destinée a contrôler validité du couple: nom user - password
if ((trame_rep.err_type = verifie_login(&trame_rcv.login)) != OK)
trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
break;
case TYPE_CONVOI_SORTIE: /* trame associée à une requête de sortie de convoi */
/* la vérification de la validité de tous les champs est nécessaire */
/* Contrôle du login : si OK => controle des autres champs sinon fin */
if ( (trame_rep.err_type = verifie_login(&trame_rcv.login)) != OK)
trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
else /* login = OK => contrôle de tous les autres champs : */
{ /* contrôle numéro carte */
if ((trame_rep.err_type += verifie_carte(trame_rcv.carte)) != OK)
trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
/* contrôle numéro tracteur */
if ((trame_rep.err_type += verifie_tracteur(trame_rcv.tracteur)) != OK)
trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
/* contrôle listes des colis */
if ((trame_rep.err_type += verifie_colis(&trame_rcv)) != OK)
trame_rep.reponse = REP_ERR; /* erreur précisée ds champ err_type */
}
break;
case TYPE_CONVOI_ENTREE: // la vérification de la validité de tous les champs n'est nécessaire
/* le traitement est semblable à TYPE_CONVOI_SORTIE : non traite */
break;
default: /* type de trame inconnu */

```

```

        trame_rep.reponse = REP_ERR;
        trame_rep.err_type = ERR_TYP_TRAME_INCONNU;
        break;
    } /* fin switch */
} /* fin traitement associe a crc OK */
else /* crc trame réceptionnée est incorrect */
    trame_rep.reponse = REP_ERR; /* erreur precisee ds champ err_type */
} /* fin if recvfrom(..) */
else /* si mauvaise réception de trame : */
    { trame_rep.reponse = REP_ERR; trame_rep.err_type = ERR_INDEFINIE ; }
// si champs trame == OK alors stocker et dater informations contenues dans trame dans la Base de Données:
if (trame_rep.reponse == REP_OK) stock_bd_trame( &trame_recv);
/* Retourner réponse a société manutentionnaire */
trame_rep.crc = calcul_crc( (unsigned char *) &trame_rep, sizeof(TRAME_REP));
sendto(soc_service,(char *) &trame_rep, sizeof(TRAME_REP),0,
        (const struct sockaddr *) &ad_client,taille);
exit(0);
}

```

Question C-8

Est-il possible d'écrire un serveur :

- "Lancé" par inetd
- Capable de gérer "simultanément" plusieurs requêtes datagrammes issues de sociétés manutentionnaires .

Dans le cadre d'une réponse affirmative:

- Les fichiers de configuration décrits par vos soins en réponse à la Question C-7 doivent-ils être adaptés.
- Exposer les grandes lignes de votre solution.
-

Ce fonctionnement est possible, les fichiers de configuration restent inchangés : cf doc. inetd.

Principe:

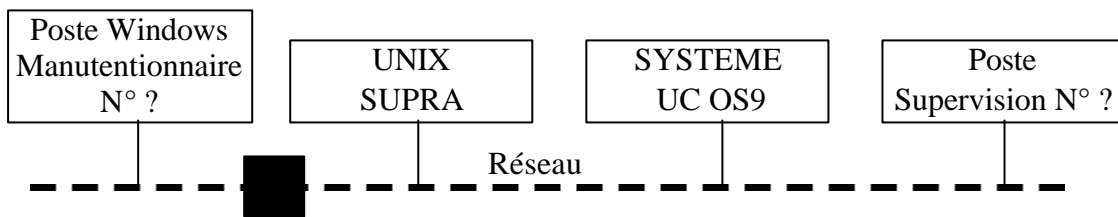
Processus serveur :

Si (Réception trame issue société manutention == OK)	(recvfrom())
Créer process fils qui assurera le traitement de la requête	(fork())
Exit	

Processus fils :

Traiter trame réceptionnée	
Créer socket de service	(socket())
Retourner trame réponse via socket de service	(sendto())
Détruire socket de service	(close())
Exit	

Troisième partie : programme demandes manutentionnaires



Question C-9 (analyse programme poste manutentionnaire).

En sachant que le programme est écrit en C++ Borland 3.1 pour Windows (programmation Objet 16 bits, avec des bibliothèques objets fournies), proposez un cycle de développement adapté à ce type de programmation et justifiez votre proposition ?

Il existe deux grandes familles de cycle de développement :

- Le cycle en V (cascade) qui est très bien adapté à la programmation classique.
- Le cycle par itération qui est plus adapté à la programmation objet car il réduit les risques.

Le cycle de développement par itération est perçu comme une suite d'itération où le logiciel évolue par incrément, donnant à chaque fois un nouveau prototype.

Le premier prototype est souvent une maquette qui permet de visualiser les concepts de l'application et un certain nombre de faisabilités.

Chaque itération est basée sur un nombre réduit de scénarios qui s'attaquent d'abord aux risques importants et détermine les blocs à construire dans l'itération.

Question C-10 (analyse programme poste manutentionnaire).

Donnez le rôle d'un atelier de génie logiciel (AGL) dans la conduite d'un projet en informatique industrielle.

L'atelier de génie logiciel va permettre de formaliser l'analyse dans un langage, de tester la structure, et de sortir un canevas de codage.

Question C-11 (analyse UML pour un programme en C++ sur Windows prog: manutentionnaire).

On vous demande de réaliser uniquement les vues : Use Case View et Logical View ; de l'analyse UML avec les diagrammes nécessaires afin de préciser la phase de «login».

Le cahier des charges imposé par les manutentionnaires et le gestionnaire de service est le suivant :

- ♦ Lancement du programme.



- ◆ Demande de login

- ◆ Teste si ce login est correct quand on valide avec envoi vers le serveur UNIX.
- ◆ Stockage pour toute la session du login (nom, passwd) si réponse du serveur OK avec nouveau menu.



- ◆ Saisie de convoi voie P.L., ...

- ◆ Validation avec envoi vers le serveur UNIX
- ◆ Réponse du serveur
- ◆ Possibilité de nouvelles saisies, ...

La classe Com_Trame_Unix gère entièrement toute la communication réseau (envoi de trame de commande : TRAME_CMD_UNIX ; réception de trame de validation : TRAME_REP), le stockage de la trame de commande à envoyer à la machine UNIX, le stockage pour toute la session du login (nom, passwd).

La construction de cette classe constituera une évolution du programme manutentionnaire (dont les déclarations sont fournies en Annexe D-2 programmes Windows : «manut.cpp»). Les attributs wasadata, sock_init, sock_client, ip_local, ip_serv, login seront transférés à cette classe.

Vue des cas d'utilisations

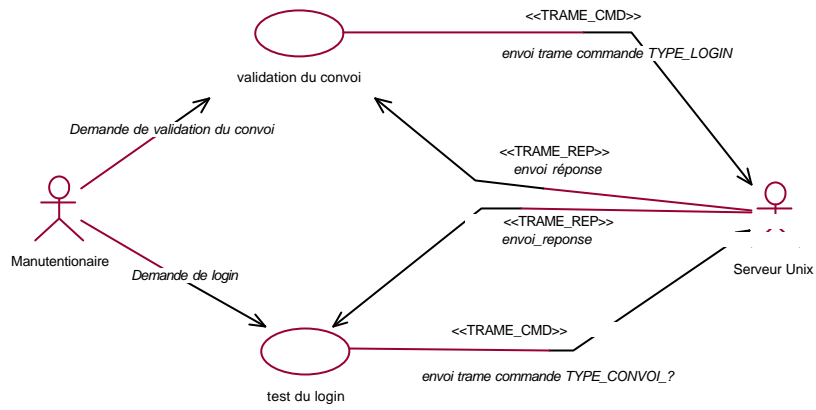
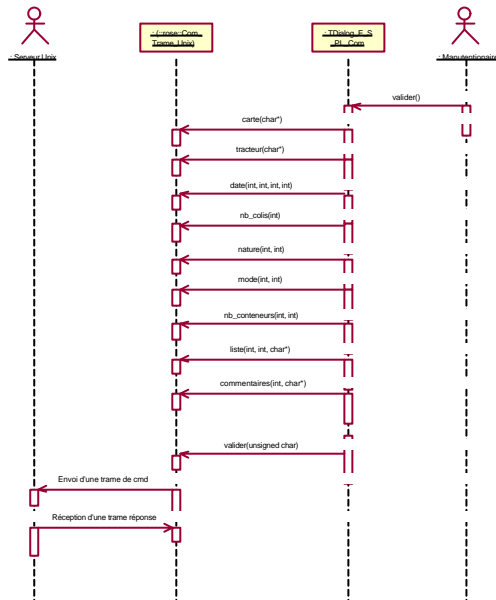
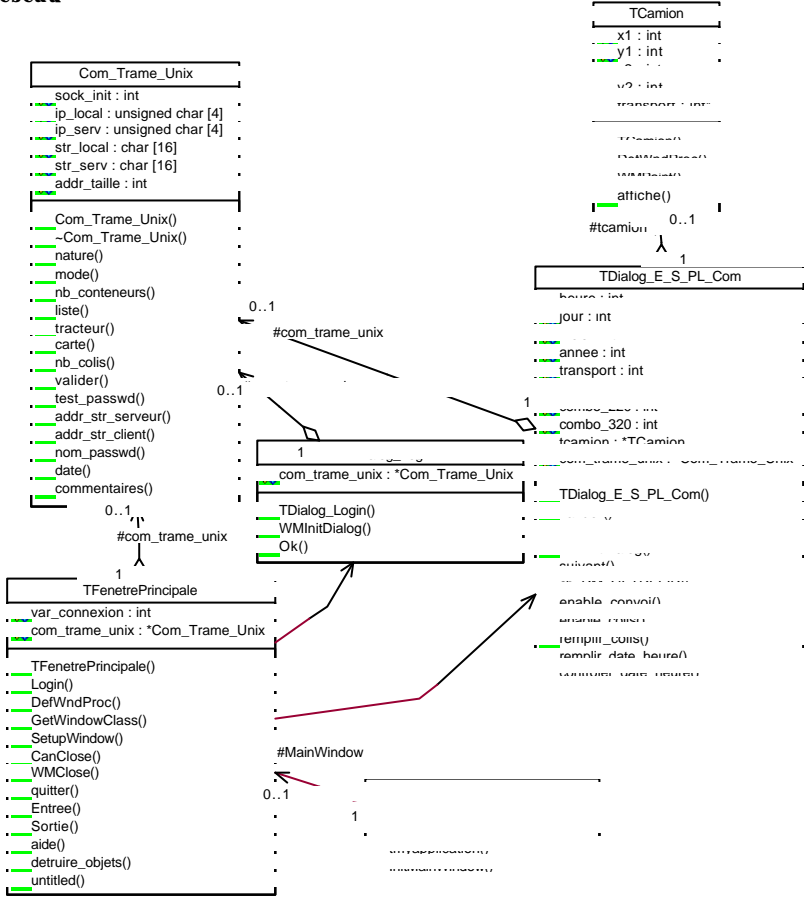


Diagramme de séquence du cas d'utilisation « test du login »



Logical View

reseau



Question C-12 (Programmation : recherche si une année est bissextile).

On vous demande d'écrire en C la fonction *int bissextile(int annee)* qui renvoie 0 (année non bissextile) ou 1 (année bissextile).

Nota: Une année est bissextile si :

- Elle est multiple de 4.
- Sauf si elle est multiple de 100 ; elle l'est quand même si elle est multiple de 400.

```
int bissextile(int annee)
{
    return(((annee % 4) == 0)&&((annee % 100) != 0))|((annee % 400) == 0));
}
```

Quatrième partie : Réseau Ethernet.

L'environnement TCP/IP a été configuré comme suit:

Dans le fichier /h0/internet/ISP/driver/GMI_ETH1/gmo.a

```
Broadcast_addr
dc.b 128.6.255.255 broadcast address (internet)
interface_address
dc.b 128.6.1.101 direct address (internet)
```

Dans le fichier /h0:internet/ISP/ipconfig/ipconfig.a

```
*Internet configuration
dc.l 0x0001 gateway flag
dc.b 128.6.0.1 default internet destination
```

Dans le fichier /h0/internet/etc/hosts

```
127.0.0.1      localhost
128.1.10.126   UC_MAR
128.1.4.83     UC_GARE
128.1.10.121   SUPRAMAR
128.1.2.1      SUPRAFOS
128.6.1.101    UC_MOUREPIANE
128.6.1.102    UD_MOUREPIANE
128.6.1.103    PC_OP1
128.6.1.104    PC_OP2
128.6.1.105    PC_OP3
128.6.1.106    PC_CARTE
128.6.1.107    SUPRA_MOUREPIANE
```

Question C-13

Citer les classes de réseaux ainsi que les plages d'adresses IP existantes au sein du monde IP.

		IP 0 réservé		
Classe A	0	IP 1 à 126	S.R	255.0.0.0
		IP 127 réservé	fonction loopback	
Classe B	10	IP 128 à 191	S.R	255.255.0.0
Classe C	110	IP 192 à 223	S.R	255.255.255.0
Classe D	1110	IP 224 à 239		
Classe E	1111	IP 240		
		IP 255 réservé		

Question C-14

Déterminer la classe utilisée, ainsi que les réseaux dans cette application. Comment choisir le(s) masque(s) de sous réseaux afin de distinguer les sous réseaux 128.1.2.0, 128.1.4.0, 128.1.10.0.

Classe B

Réseaux 128.1.0.0
 128.6.0.0

Masques :

Par défaut : 255.255.255.0

Possibles : 255.255.254.0
 255.255.14.0

D) Étude de la partie Supervision.

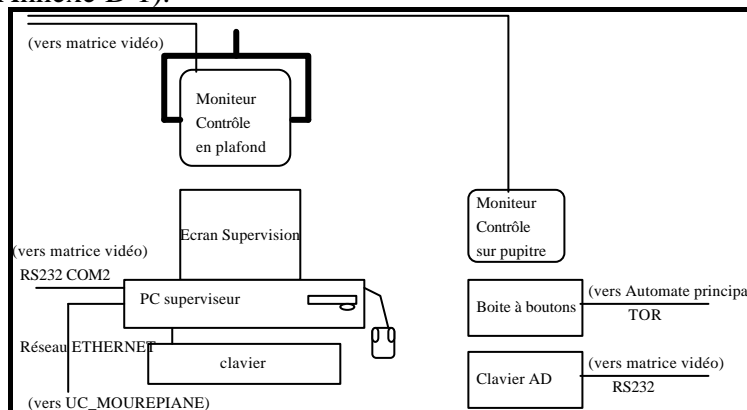
Les 3 postes superviseurs implantés dans le poste de contrôle de Mourepiane, sont conformes au schéma ci-dessous. Chaque poste superviseur est constitué : d'un ordinateur type PC avec un système d'exploitation Windows 3.1 (+ module gestion réseau TCP/IP), d'un logiciel de supervision (superviseur), de moniteurs vidéo de contrôle (moniteur en plafond + moniteur sur pupitre), d'un clavier de pilotage de la matrice vidéo (clavier AD) et d'un pupitre de commande (boîte à boutons). Lors de la conception de l'application le SE choisi fut Windows 3.1.

Les PC supervisions ont 2 moyens de communiquer avec l'extérieur voir Annexe générale G-1:

- par l'intermédiaire d'une carte réseau ethernet (vers Hub)
- par l'intermédiaire d'une RS232 COM2 (vers matrice vidéo)

Le logiciel de supervision « superv_pl.cpp » (fourni partiellement en Annexe D-2 programmation Windows superviseur) permet le dialogue avec l'unité centrale Mourepiane (système UC_0S9 : UC_MOURPIANE) par l'intermédiaire du réseau (communication socket en mode connecté), et il permet de piloter la matrice vidéo par l'intermédiaire de la liaison COM2 grâce à une DLL fournie par le fabricant de la matrice.

Ce logiciel fut développé en 92 avec le compilateur Borland C++ 3.1 (16 bits) pour Windows (bibliothèque OWL 1.0, Annexe D-1).



Les principales fenêtres de l'interface homme-machine du superviseur sont les suivantes :

Fenêtre principale au départ :



Fenêtre de login : (Connexion)

Fenêtre principale après connexion :



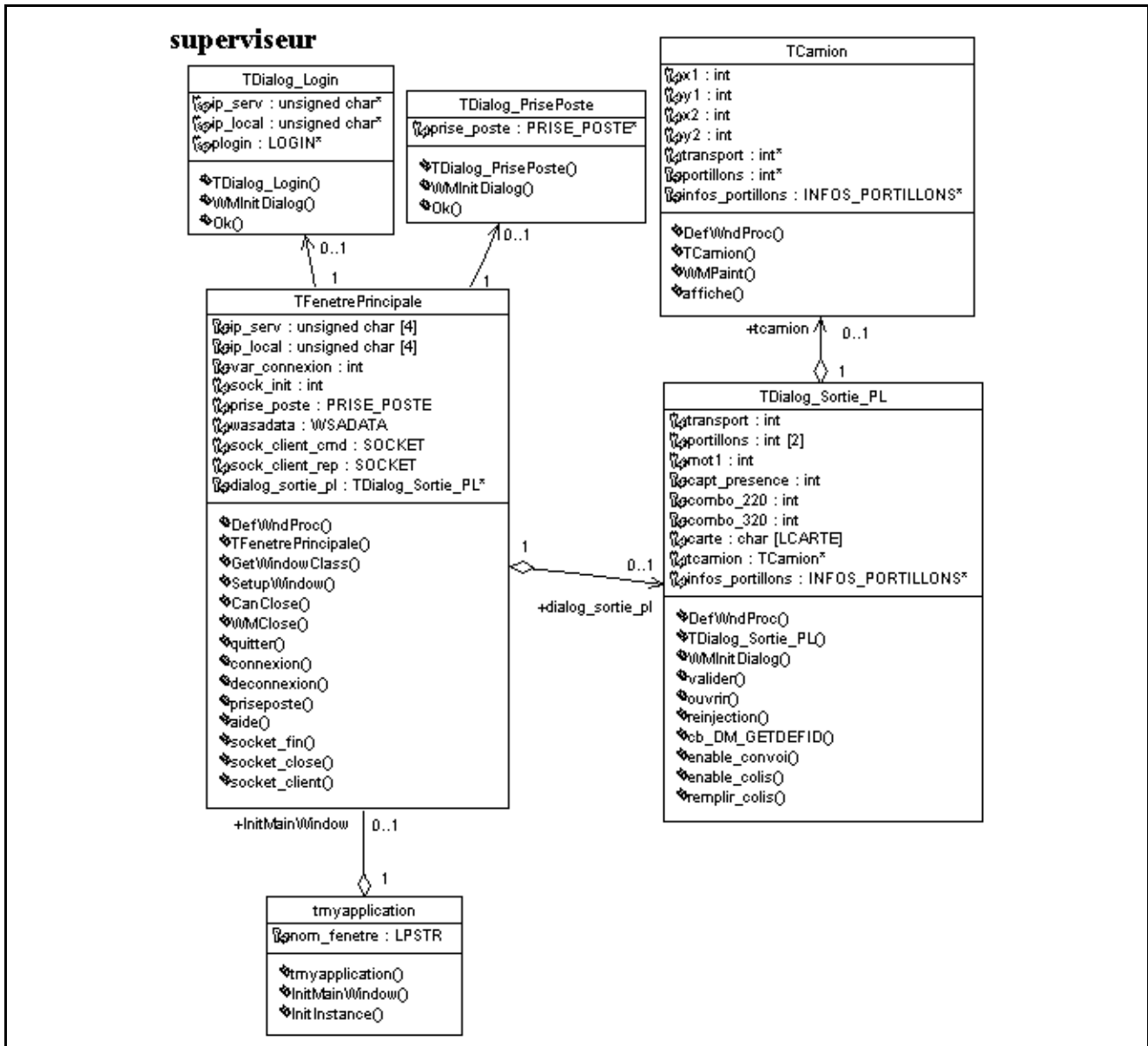
Fenêtre de Prise de Poste : (Après Connexion)

Fenêtre d'Entrée P.L. 1

Première partie : Programmation Windows en C++

Question D-1 (UML)

En partant du programme source partiel «superv_pl.cpp» fourni en Annexe D-2, établissez le diagramme des classes simplifié (reverse ingeneering) de la vue logique où vous montrerez toutes les classes (les méthodes et les variables ne seront pas indiquées) et leurs relations (donnez les noms des associations). Les héritages nécessaires à la compréhension seront indiqués.



Question D-2 (programmation Windows)

En vous aidant du programme source «superv_pl.cpp» en Annexe D-2, des documents sur les classes OWL 1.0 en Annexe D-1 et du diagramme que vous avez réalisé à la question D-1, expliquez en quelques lignes le fonctionnement du programme (uniquement la phase de démarrage, jusqu'à l'affichage de la fenêtre principale).

Quel est son point d'entrée?

Quels sont les objets créés?

Quels sont leurs rôles?

- Le point d'entrée est la fonction int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow).

Cette fonction crée et instancie l'objet tmyapplication myapp(...). Puis elle lance la méthode Run() de cet objet. Et elle renvoie le résultat de cet objet.

- L'objet myapp est dérivée de TApplication lui-même dérivé de TModule et sert de remplacement orienté-objets à un module applicatif Windows classique.

TApplication et TModule fournissent les capacités de base d'une application Windows. Les méthodes de TApplication assurent l'initialisation des instances et le traitement des messages.

- La méthode Run() réalise les initialisations nécessaires pour toutes instances exécutives de l'application :

Appelle InitApplication(),

Appelle InitInstance()

Qui appelle InitMainWindow().

Qui crée puis affiche la fenêtre principale en appelant :

TModule::MakeWindow

TWindowsObject::Show

Rend l'application active en appelant MessageLoop si l'initialisation a réussi.

Incarné la boucle de traitement de message de l'application qui s'exécute pendant toute la durée d'exécution.

Elle demande des messages au noyau Windows. Lorsqu'un message est reçu, elle le traite.

- tmyapplication::InitMainWindow() réalise le lien entre l'application Windows et la fenêtre principale :

MainWindow = new TFenetrePrincipale(NULL, nom_fenetre, "MENU_CLIENT_DEPART", NULL);

Question D-3 (programmation sous Windows)

Expliquez succinctement la programmation par messages de Windows? Sur Unix le principe est-il le même au niveau de la gestion de l'interface graphique XWindow?

Chaque événement extérieur (click souris, clavier,...) est caractérisé par l'envoi d'un message vers la fenêtre concerné.

Un message Windows est composé de :

```
struct TMessage {
    HWND Receiver;
    WORD Message;
    union { WORD WParam;
            struct tagWP { BYTE Lo; BYTE Hi; } WP; };
    union { DWORD LParam;
            struct tagLP { WORD Lo; WORD Hi; } LP; };
    long Result;};
```

Une structure TMessage contient des informations au sujet d'un message en entrée depuis Windows. La structure est renseignée par ObjectWindows et transmise aux fonctions de réponse à message.

Receiver contient le handle de la fenêtre à laquelle le message est destiné.

Message contient l'identificateur de message.

WParam et LParam contiennent des informations spécifiques aux messages provenant de Windows.

LP et WP sont des structures qui sont définies pour permettre un accès aisé à la partie haute et basse de WParam et de LParam.

Result contient la valeur à renvoyer au noyau Windows.

L'inconvénient majeur de cette gestion par message est le blocage de la file de message (qui grandit très vite) dès qu'une fonction de traitement de message rentre dans une boucle de traitement un peu longue. Ceci était surtout vrai pour Windows 3.1 où il n'y avait qu'une seule file de message système.

Question D-4 (programmation Windows)

Comment une fenêtre fille peut-elle dialoguer avec sa fenêtre mère (dans les deux sens) ?

Sens fille mère :

Soit directement en appelant des méthodes de la fenêtre mère car elle connaît son pointeur.

Surtout en lui envoyant des messages :

SendMessage() envoi avec attente de retour

PostMessage() envoi sans attente de retour

Ex: SendMessage(Parent->HWindow,WM_USER+1,TYPE_DEMANDE_VALIDATION,(long) &message_a_valider);

Sens mère fille :

Soit directement en appelant des méthodes de la fenêtre fille (si elle a mémorisé son pointeur à la création).

Surtout en lui envoyant des messages :

SendMessage() envoi avec attente de retour

PostMessage() envoi sans attente de retour

Deuxième partie : Programmation Socket sous Windows en C++

Question D-5 (programmation socket sous Windows)

A partir du programme source «superv_pl.cpp» en Annexe D-2 et de l'Annexe D-3 documentation «Windows Sockets», expliquez la méthode *int TFenetrePrincipale::socket_client()*.

Pourquoi y a-t-il 2 sockets?

Précisez le rôle de la fonction *WSAAsyncSelect(...)*?

Une socket est reliée à la commande `WSAAsyncSelect(sock_client_cmd,HWindow, WM_USER, FD_READ|FD_CLOSE)`. Le gestionnaire de réseau de Windows génère, à chaque envoi d'une trame par le serveur, un message Windows de type `WM_USER` à la fenêtre principale. La fenêtre principale n'a plus qu'à lire ce message. Cela permet pour Windows 3.1 de ne pas bloquer toutes les applications en attendant une information (trame) sur un `recv()`. L'autre socket permet de transmettre ou d'attendre des informations complémentaires (validation,...). Pour transmettre on pourrait utiliser la même socket. Mais pas pour recevoir, car le gestionnaire de réseau intercepterait la communication et générerait un nouveau message `WM_USER`.

Question D-6 (programmation socket sous Windows)

Que se passe-t-il quand le serveur implanté sur le système UC_OS9 se déconnecte ?

Un message de déconnexion automatique est envoyé à la socket client. Le gestionnaire de socket génère un message Windows (`WM_USER,FD_CLOSE`) qui prévient le client de la déconnexion par le serveur.

La fonction `WSAAsyncSelect(WSAAsyncSelect(sock_client_cmd,HWindow,WM_USER,FD_READ|FD_CLOSE)` grâce au flag `FD_CLOSE` permet au gestionnaire de socket d'envoyer ce message.

Un message Windows (`WM_USER,FD_READ`) avec une trame vide est envoyé avant.

Question D-7 (programmation socket sous Windows)

Que se passe-t-il quand le serveur implanté sur le système UC_OS9 envoie une trame de commande ?

Le gestionnaire de socket génère un message Windows (`WM_USER,FD_READ`) qui prévient le client qu'une trame en provenance du serveur est à lire.

La fonction `WSAAsyncSelect(WSAAsyncSelect(sock_client_cmd,HWindow,WM_USER,FD_READ|FD_CLOSE)` grâce au flag `FD_READ` permet au gestionnaire de socket d'envoyer ce message.

Question D-8 (programmation socket sous Windows)

En sachant que le programme source a été créé pour le système d'exploitation Windows 3.1 avec un compilateur Borland 3.1 (16 bits); avec les compilateurs et les systèmes d'exploitation (Windows 95,...) plus récents pourrait-on se passer de la fonction `WSAAsyncSelect(...)` et dans ce cas quelle forme aurait le programme.

On pourrait imaginer qu'un thread gèrerait les réceptions des trames de commandes en provenance du serveur.

Question D-9 (programmation socket sous Windows)

En sachant que le programme superviseur est implanté sur un PC avec Windows 3.1 (ce programme a été compilé avec le compilateur Borland C++ 3.1 pour Windows 16 bits), que le programme de réception est implanté sur une machine 68000 avec OS9 68K et un compilateur C++ 32 bits, les trames seront elles directement utilisables sur la machine 68000? Sinon donnez les principes d'une solution.

Elles ne seront pas directement utilisables car :

Les entiers (int) sont sur 16 bits sur PC et de 32 bits sur 68000.

En plus à l'intérieur de chaque mot (16 bits, ou 32 bits...) les poids forts et les poids faibles seront inversés entre les deux types de plate-forme.

Il faudra donc faire un traitement de conversion de trame sur la machine OS9 qui tiendra compte de cette spécificité. Ce programme de conversion servira aussi partiellement pour son dialogue avec la machine UNIX qui sera aussi un PC (mais sur UNIX les entiers sont sur 32 bits).

Question D-10 (programmation socket sous Windows)

Est-il nécessaire de réaliser un contrôle de trame en mode connecté?

Non en mode connecté il y a déjà un contrôle qui est réalisé au niveau de l'envoi de la trame TCP.