NOTE : I 8 + 4 + 4 + 4 + 2

II m 4 + 4 + 2 + 4 + 4   r< 8 + 8 + 4 + 4

Numéro à reporter sur les intercalaires : 133054   Nombre d'intercalaires : 2

I. ACL ldap

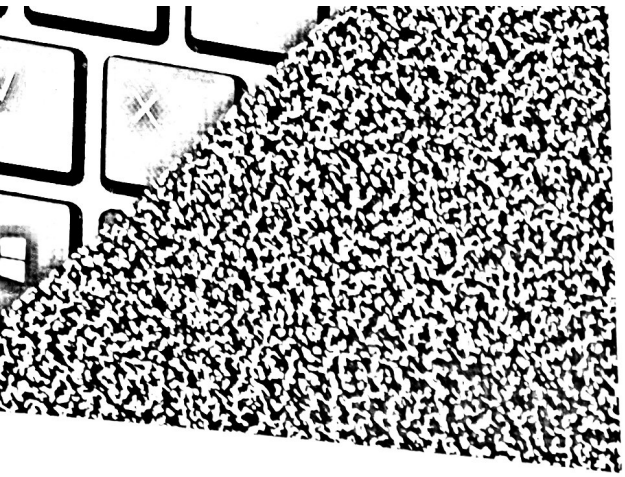
Q.I.2) Pour qu'un utilisateur puisse modifier toutes ses données, il faut échanger "self" et "Boss" dans la première directive. En effet, TdP est "Boss" donc il ne peut que lire ses propres données.

Ensuite, pour que les membres de "RH" modifient les données de tous, excepté le mot de passe, il faut ajouter dans la 2ème directive le droit d'écriture pour les "RH"

exemple :

1	*	Administrateur	manage	stop
		Self	write	stop
		Boss	read	stop
		*	none	break
2	telNumber, displayName, Name	Administrateur	manage	stop
		RH	write	stop
		*	read	stop

La suite est inchangée.



Q.I.3) Afin de donner aux machines le droit de lire uniquement "compte informatique", il faut supprimer "machine" de la directive 4 et ajouter une directive 2 pour gérer les informations 'compte informatique'

1	*	Admin	manage	stop
		Self	write	stop
		Boss	read	stop
		*	none	break
2	"compte info"	"machines"	read	stop
		*	none	break
3	telephoneNumber, displayName, Name	Admin	manage	stop
		RH	write	stop
		*	read	stop
4	*.userPassword	anonymous	auth	stop
		*		stop
5	"info employés"	RH	write	stop
		consultants	read	stop
		*	none	break
6	*	*	none	break

Q. I. 4) Par la directive 1, l'administrateur manage déjà tout. De même, le "Boss" peut tout lire. Les membres de RH peuvent modifier les données des employés, ainsi que l'utilisateur.
Il faut donc faire en sorte que Hal $ ait un droit de lecture et interdire aux consultants, machines et autres.
Il faut donc ajouter une directive sous la première :

2	"info perso"	"Hal $"	read	stop
		*	none	break

non !

II - Serveur de jeu mobile

Q. II. 1 et Q. II. 3

```
int main () {
    int sin ; vector<pid_t> pidTab;
    int pid_t;
    int s = socklib:: CreerSocketServeur ("8080");
    while (1) {
        sin = socklib::AcceptConnexion (s); // on attend un client
        pid = fork();
        if (pid == 0) {
            // FILS
            traitement_client (sin);
            close (sin);
        }
        // PERE
        pidTab. push_back (pid);
    } // fin while
    int status; pid_t res;
    for (unsigned int i=0; i<pidTab.size(); ++i) { // libération des processus fils
        res = waitpid ( pidTab[i], &status, 0);
        if (res < 0) { cerr << "Erreur"; exit(1); }
    }
    return 0;
}
```

Q-II-2-

```
void traitement_client ( int sin) {
    BufferedReaderWriter in(sin);
    // lecture de l'id
    vector <char> id(128);
    id = in.read_all (128 * sizeof(char));
    string mess, commande;
    while (1) {
        mess = in.read_line();
        commande = mess.substr(0, 3); // les commandes ont 4 char

        if ( commande == "next") {
            image im = get_next_image(id); // image suivante
            uint size = im.size();
            const char* data = im.data();
            // envoi de l'entier
            in.write((char*)&size, 0, sizeof(uint));
            in.write ( "\n");
            // envoi de l'image
            in.write ( data, 0, size);
        }
        else if ( commande == "prop") {
            string solution = mess.substr (4, mess.size() - 2); // on enlève \n
            string reponse = get_reponse ( id, solution);
            if (reponse == "") reponse = "Error";
            reponse += "\n";
            in.write (reponse);
        }
        else if ( commande == "quit") {
            string bye = "Bye\n";
            in.write( bye);
            break; // c'est fini
        }
        else { string reponse = "Error\n"; in.write(reponse); }
```

— int **write**(int fd, char *buf, size_t count);
write() écrit count octets depuis le buffer buff. Retourne le nombre d'octets qui ont été effectivement écrits (ou -1 en cas d'erreur).

Attention, vous ne pouvez pas mélanger l'utilisation de la fonction de lecture de la librairie standard (**read**) avec celle de la librairie socklib.

Pour la librairie socklib

— Dans l'espace de nom socklib :
  — int CreeSocketServeur (const std::string &p) Cree une socket d'attente pour le serveur sur le port p.
  — int CreeSocketClient (const std::string &serveur, const std::string &port)Crée une socket de client en se connectant à un serveur.
  — int AcceptConnexion (int s) Accepte un nouveau client qui s'est connecté à la socket d'attente s.

  Ces 3 fonctions renvoient une socket comme résultat.
— Dans la classe BufferedReaderWriter :
  — BufferedReaderWriter (int fd) le constructeur.
  — void **write**(char *data, int offset, int taille) envoie des données situées entre &data[offset] et &data[offset+taille].
  — void **write**(const vector<char> &data) envoie des données du vector data.
  — void **write**(const string &msg) envoie la chaine de caractère msg.
  — vector<char> **read**() lit des octets et les renvoie sous la forme d'un vecteur.
  — vector<char> read_all(int taille) lit taille octets et les renvoie sous la forme d'un vecteur.
  — vector<char> read_until(char end='\n') lit tous les octets arrivés sur la socket jusqu'au prochain caractère end et les renvoie sous la forme d'un tableau de caractères.
  — int read_data(char *buff, int offset, int len) lit len octets exactement et les stocke dans le tableau buff à partir de la case offset.
  — string read_line(char end='\n') lit tous les octets arrivé sur la socket jusqu'au prochain caractère end et les renvoie sous la forme d'une chaine de caractères.

III Feuille de réponse

Numéro de copie (**Ne pas mettre votre nom**) :

13 30 54

			Droits de				
			Alice	Dogbert	Catbert	TdP	Hal$
Droits sur	Alice	Name	W	ℛ	ℛ	ℛ	ℛ
		userPassword	W	none	none	ℛ	none
		uidNumber	W	ℛ	W	ℛ	ℛ
		homePhone	W	ℛ	W	ℛ	ℛ
	TdP	Name	ℛ	ℛ	ℛ	ℛ	ℛ
		userPassword	none	none	none	ℛ	none
		uidNumber	none	ℛ	W	ℛ	ℛ
		homePhone	none	ℛ	W	ℛ	ℛ
	Catbert	Name	ℛ	ℛ	W	ℛ	ℛ
		userPassword	none	none	W	ℛ	none
		uidNumber	none	ℛ	W	ℛ	ℛ
		homePhone	none	ℛ	W	ℛ	ℛ