

TD 4 - LIF12 système d'exploitation

Passage d'informations sur les pipes

3 mai 2017

I Droits unix

Sur le système considéré, il y a 4 utilisateurs :

- fontaine qui fait partie du groupe prof et user ;
- elise qui fait partie des groupes etu et user ;
- hippolyte qui fait partie du groupe prof.
- root qui est l'administrateur et fait parti du groupe root

Q.I.1) - Représentez les possibilités d'accès des 4 utilisateurs aux fichiers visionneurPDF, sujet.pdf, correction.pdf et notes.ods.

```
drwxr-x--x 27      root  user  /bin/
-rwsr-xr-x  1      root  etu   /bin/visionneurPDF
drwxr-xr-x  80      root  user  /home/
drwxr-x--x  10     fontaine prof  /home/fontaine/
drwx--x---  4     fontaine prof  /home/fontaine/prive/
-rw-r-x---  1     fontaine prof  /home/fontaine/sujet.pdf
-rw-r--r--  1     hippolyte prof  /home/fontaine/prive/correction.pdf
-rw-rw----  1     fontaine prof  /home/fontaine/prive/notes.ods
```

Attention, pour les fichiers vous devez tenir compte des droits des répertoires et sous répertoires.

D'après les droits simples :

	visionneurPDF	sujet.pdf	correction.pdf	notes.ods
fontaine	r-x	rw-	r--	rw-
elise	r-x	---	---	---
hyppolyte	r-x	r-x	rw-	rw-
root	rwX	rwX	rw-	rw-

Il faut noter plusieurs choses :

- root a forcément le droit read et write. De plus, il a le droit d'exécution sur tous les fichiers exécutables (c'est à dire qu'il a le droit d'exécution si l'un des groupes a le droit d'exécution).
- Pour accéder à un fichier, un utilisateur doit avoir le droit d'accéder à tous les répertoires du chemin qui permet d'aller à ce fichier. Cela signifie avoir le droit d'exécution sur les répertoires. Donc par exemple Élise ne peut pas lire le fichier correction.pdf car ce dernier se trouve dans le répertoire /home/fontaine/prive/ qui ne lui est pas accessible.
- hippolyte a accès aux fichiers correction.pdf et notes.ods car il a accès au répertoire /home/fontaine/prive/. Mais, il ne peut ni faire de ls (qui demande le droit de lecture du répertoire), ni changer le nom ou effacer le fichier (qui demande le droit d'écrire dans le répertoire).
- Comme le *setuid* bit du fichier visionneurPDF est actif (le « s » dans la colonne du propriétaire), cela signifie qu'un utilisateur qui exécute ce fichier le fait comme s'il était le propriétaire (donc root). Cela signifie que les droits utilisés seront ceux de root et que par exemple, cette visionneusePDF pourra lire les fichiers correction.pdf et sujet.pdf même si c'est elise qui l'utilise.

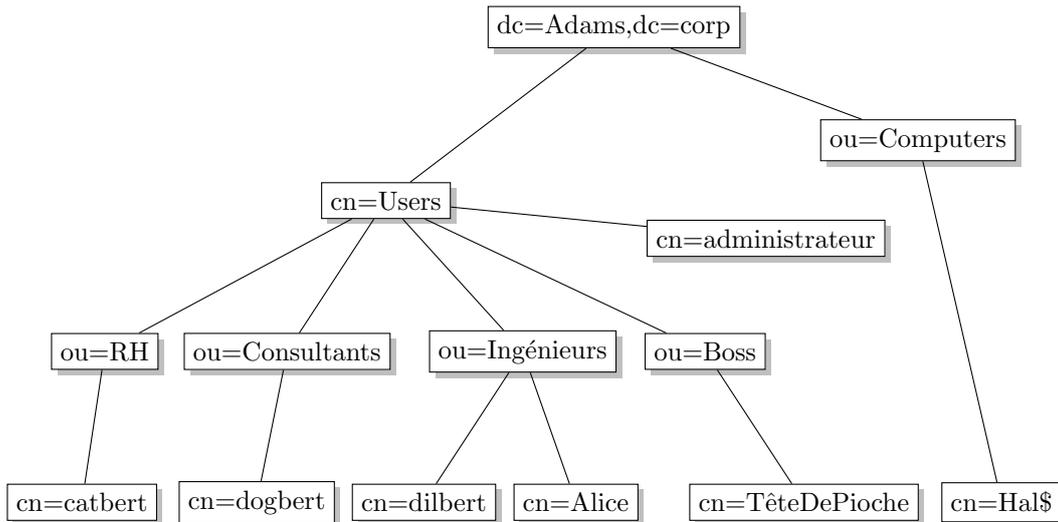


FIGURE 1 – Organisation de l'annuaire LDAP dans notre exemple.

II ACL ldap

On rappelle que dans l'annuaire ldap, les informations sont stockées dans un arbre (voir par exemple la figure 1). Les ACLs se composent de 4 parties :

- **<what>** spécifie l'objet cible ;
- **<who>** l'objet auquel le droit est attribué ;
- **<access>** le droit attribué (`none` | `auth` | `compare` | `search` | `read` | `write` | `manage`) ;
- **<control>** l'action à faire ensuite. Si rien n'est précisé, c'est `stop` (arrêter d'évaluer les ACLs), mais il est possible de choisir `continue` (évalue la suite de la même directive) ou `break` (passe à la directive suivante).

Attention, les parties **<who>** et **<what>** ne peuvent être définies que de manière positive en utilisant l'arbre. On peut par exemple définir « tous les employés » c'est à dire tout les descendant du noeud `cn=Users,dc=Adams,dc=Corp`, mais pas « tous les employés sauf l'administrateur ». La partie **<what>** contient une description de l'utilisateur concerné et une liste du ou des champs demandés. On peut définir :

- une liste de champs concernant tout le monde, par exemple « le mot de passe » (`*.userPassword`) ;
- les champs d'un groupe de personnes ou d'une personne, par exemple « les information des employés » (descendant de `"cn=Users,dc=Adams,dc=Corp".*`) ;
- une liste de champs pour un groupe ou une personne, par exemple les « informations personnelles » (descendant de `"cn=Users,dc=Adams,dc=Corp"."homePostalAddress, homePhone, street"`) ;

Pour simplifier nous allons utiliser les abréviations suivantes :

- *Administrateur* est l'utilisateur `cn=Administrateur,cn=Users,dc=Adams,dc=Corp` ;
- *machines*, descendants de `ou=Computer,dc=Adams,dc=Corp` ;
- *employés*, tous ceux qui sont sous `cn=Users,dc=Adams,dc=Corp` ;
- *RH*, tous ceux qui sont sous `ou=RH,cn=Users,dc=Adams,dc=Corp` ;
- *Ingénieurs*, tous ceux qui sont sous `ou=Ingénieurs,cn=Users,dc=Adams,dc=Corp` ;
- *Consultant*, tous ceux qui sont sous `ou=Consultant,cn=Users,dc=Adams,dc=Corp` ;
- *Boss*, tous ceux qui sont sous `ou=Boss,cn=Users,dc=Adams,dc=Corp` ;
- *infos employés*, tous les attributs des employés (descendants de `"cn=Users,dc=Adams,dc=Corp".*`) : `jpegPhoto`, `telephoneNumber`, `displayName`, `homePostalAddress`, `homePhone`, `street`, `userPassword`,... ;
- *infos perso*, les attributs représentant les informations personnelles descendants de `"cn=Users,dc=Adams,dc=Corp"."homePostalAddress, homePhone, street"`.

nous avons le tableau de directives suivant :

directive	what	who	access	control
1	userPassword	self Administrateur anonymous *	write manage auth none	<i>stop</i> <i>stop</i> <i>stop</i> <i>stop</i>
2	*	“Administrateur” “consultants” self *	write read write none	<i>stop</i> break <i>stop</i> break
3	“infos employés”	“RH” Administrateur *	write manage none	<i>break</i> <i>stop</i> break
4	telephoneNumber, displayName	* Administrateur	read manage	<i>stop</i> <i>stop</i>
5	“infos perso”	“Boss” “RH” “Administrateur” *	read manage write none	<i>stop</i> <i>stop</i> <i>stop</i> break
6	*	*	<i>search</i>	<i>stop</i>

Q.II.1) - Sous la forme d'un tableau, donner les droits obtenus par Catbert, Dilbert, Dogbert, Administrateur, Hal\$ sur les champs jpegPhoto, userPassword, displayName et homePhone de Dilbert et TêteDePioche.

Solution:			Droits de				
			Catbert	Dilbert	Dogbert	Administrateur	Hal\$
Droits sur	Dilbert	jpegPhoto	write	write	read	write	search
		userPassword	none	write	none	manage	none
		displayName	write	write	read	write	read
		homePhone	manage	write	read	write	search
	Tête De Pioche	jpegPhoto	write	search	read	write	search
		userPassword	none	none	none	manage	none
		displayName	write	read	read	write	read
		homePhone	manage	search	read	write	search

Q.II.2) - Pour utiliser l'annuaire ldap, afin d'authentifier les utilisateurs qui essaient de se connecter, une machine se connecte grâce à son compte dédié (exemple hal\$) et fait 2 requêtes :

- À partir du nom de login fourni par l'utilisateur, elle recherche l'objet de l'arbre dont l'un des champs `samAccountName`, `uid` ou `displayName` correspond. S'il y a un resultat elle lit la valeur du champs `dn` (identifiant unique).
- À partir du `dn` et du mot de passe elle essaye de s'authentifier.

Cela est-il possible avec la configuration actuelle ? Sinon, que faut-il changer ? Si oui quelles directives sont concernées ?

Solution: Une machine a le droit de `search` sur les champs `samAccountName`, `uid` ou `displayName`. Donc elle pourra faire la première partie (rechercher un utilisateur dont le login correspond à l'un des trois champs). Mais pour l'authntification, elle doit avoir le droit `auth` sur le mot de passe ce qui n'est pas le cas. Il faut donc rajouter une ligne à la première directive :

directive	what	who	access	control
1	userPassword	self Administrateur anonymous « machine » *	write manage auth auth none	<i>stop</i> <i>stop</i> <i>stop</i> <i>stop</i> <i>stop</i>

Q.II.3) - Que faut-il modifier si vous souhaitez que tous les « employés » aient accès aux données de tous les autres employés en lecture à l'exception des « informations personnelles ».

Solution: En regardant la solution de la question, on peut voir que les employés ont le droit **search** sur les données des autres employés grâce à la directive 6. Cela signifie que lorsque le système résout les droits, des employés vis à vis des données des autres employés, il n'est pas bloqué avant, mais descend jusqu'à cette directive. On peut donc la modifier pour changer le **search** en **read**. L'intérêt de le faire le plus bas possible est que cela n'a pas de conséquences sur les autres droits. Par exemple, le cas des « Données personnelles » sera déjà réglé par la directive 5. Une des solutions est donc de modifier uniquement la directive 6 :

...
6	*	"employees" *	read <i>search</i>	<i>stop</i> <i>stop</i>

III Signal de message (CC mars 2015)

Vous devez faire un programme qui se duplique (**fork**). Le fils doit envoyer un certain nombre de messages au père via un pipe. Pour compliquer la chose, il y a 2 types de messages :

- un entier (4 octets) ;
- ou une chaîne de caractères (contenue dans un tableau d'exactly 100 octets).

Il faut donc utiliser 2 façons de lire ces messages. Et pour choisir la bonne méthode, le fils doit prévenir le père. Pour cela, il utilise 2 signaux différents SIGUSR1 pour annoncer un message de type entier, SIGUSR2 pour annoncer un tableau de 100 octets.

Vous devez donc écrire un programme :

- qui ouvre un pipe de discussion ;
- qui crée un processus fils ;
- le fils doit envoyer un nombre déterminé de messages en choisissant au hasard le type du message (tableau de 100 caractères ou entier de 4 octets) ;
- le père doit attendre un signal du fils (fonction pause), à chaque message et en fonction du type, il le lit de la manière correcte et l'affiche ;
- à la fin, le fils se termine, le père doit reconnaître cette terminaison et s'arrêter lui-même en affichant « fin du programme ».

Vous veillerez à bien terminer le programme, par exemple, toutes les extrémités du pipe doivent être fermées et le fils doit être attendu par un **wait**.

III.1 Manuel des fonctions

Vous pouvez utiliser les fonctions standards :

- **pid_t getppid(void)** ; retourne l'identifiant de processus du père du processus courant.
- **int kill(pid_t p, int sig)** envoie le signal sig au processus de pid p, la fonction renvoie 0 en cas de succès et -1 en cas d'échec.

- **int pipe(int pipefd [2]);**
Crée un canal de communication unidirectionnel. Si la fonction ne renvoie pas `-1` (signe d'une erreur), le résultat de la fonction est le tableau `pipefd`. Il contient alors 2 descripteurs de fichiers `pipefd[0]` pour la lecture, `pipefd[1]` pour l'écriture.
- **int pause(void);** pause force le processus (ou le thread) appelant à s'endormir jusqu'à ce qu'un signal soit distribué.
- **sighandler_t signal(int signum, sighandler_t handler);** met en place `handler` comme fonction de gestion du signal `signum`. Renvoie la fonction utilisée précédemment ou `SIGERR` en cas d'erreur.
- **int rand(void);** renvoie un nombre entier pseudo aléatoire. Par exemple `rand()%5` donne un nombre entre 0 et 4.

Pour les communications, vous pouvez au choix utiliser les fonctions standards :

- **int read(int fd, char *buf, size_t count);**
`read()` demande à lire au plus `count` octets sur `fd`, et à placer le résultat dans le tampon `buf`. Retourne le nombre d'octets qui ont été effectivement lus, qui peut être inférieur à la limite donnée pour cause de non-disponibilité (`-1` en cas d'erreur, `0` en fin de fichier).
- **int write(int fd, char *buf, size_t count);**
`write()` écrit `count` octets depuis le buffer `buf`. Retourne le nombre d'octets qui ont été effectivement écrits (ou `-1` en cas d'erreur).

Ou alors utiliser les `BufferedReaderWriter` qui fonctionnent avec les pipes comme avec les sockets (attention cependant à ne pas écrire dans un `BufferedReaderWriter` construit à partir de l'extrémité de sortie du pipe et inversement).

- `BufferedReaderWriter::write(char *data, int offset, int taille)` envoie des données situées entre `&data[offset]` et `&data[offset+taille]`.
- `BufferedReaderWriter::write(vector<char> data)` envoie des données du vector `data`.
- `vector<char> BufferedReaderWriter::read_all(int taille)` lit `taille` octets et les renvoie sous la forme d'un vecteur.
- `BufferedReaderWriter::read_(char *buf, int offset, int len)` lit `len` octets et les stocke dans le tableau `buf` à partir de la case `offset`.