TP 2
Docker

Exercice 1 : installation et prise en main

Installez docker. Cela peut être fait avec apt-get. Il faut suivre la procédure du site Docker. Pour la distribution Ubuntu, voici le lien :

https://docs.docker.com/engine/install/ubuntu/

1. Affichez la version de Docker installé :

   ```
   sudo docker version
   ```

2. Lancez votre premier conteneur (hello-world image):

   ```
   sudo docker run hello-world
   ```

   Docker va automatiquement chercher l'image sur Internet si elle n'est pas déjà installée.

3. Vérifiez que le serveur docker (the docker daemon) est bien en cours d'exécution

   ```
   ps aux | grep dockerd
   ```

4. Listez les conteneurs en cours d'exécution et les exécutions passés :

   ```
   docker ps
   docker ps -a
   ```

5. Listez les images installées sur votre système :

   ```
   docker images
   ```

6. Téléchargez l'image ubuntu sans l'exécuter :

   ```
   docker pull ubuntu
   ```

7. Lancez le conteneur ubuntu

   ```
   docker run ubuntu
   ```

   Le conteneur termine immédiatement car aucun processus n'est en cours d'exécution.

8. Lancez plusieurs commandes dans le conteneur ubuntu (sleep, ls, etc.) :

   ```
   docker run ubuntu ls
   docker run ubuntu sleep 10
   ```

9. Pour lancer des processus/commandes dans un conteneur en cours d'exécution, on utilise la commande docker exec. Ci-dessous, vous lancez le conteneur ubuntu avec un sleep suffisamment long pour que vous ayez le temps de lancer d'autres commandes puis vous lancez des commandes grâce au nom de votre conteneur :

   ```
   docker run ubuntu sleep 300 //
   ```
   Le conteneur est partie pour 300sec

`docker ps //`on visualise le nom du conteneur en cours d'exécution (c'est le dernier champ)

`docker exec <nom_du_conteneur>  ls`

10. Lancez votre conteneur en background (option –d)

    `docker run –d ubuntu sleep 200` //Effectuée plusieurs fois vous aurez plusieurs instances du conteneur

11. Remettez un de vos conteneurs en foreground (commande docker ps pour récupérer le nom):

    `docker attach <nom_du_conteneur>`

Exercice 2 : vie et environnement du conteneur

1. Lancez le conteneur « bash »

   `docker run bash`

   Le conteneur lance la commande bash puis s'arrête.

2. Pour interagir avec lui, il faut que celui-ci accède à l'entrée standard (option –i) et qu'il accède au terminal (option –t) :

   `docker run –it bash`

3. Affichez les pid des processus de votre conteneurs (commande ps usuel). Vous pouvez aussi voir ceux du namespace root (`ps au` dans une autre console).
4. Affichez les points de montage de votre conteneur (commande mount). Parmi les dossiers de votre racine (celle du conteneur) lesquels sont commun avec la racine hôte ? Vous pouvez utiliser la commande ls –i pour afficher les numéros d'inode par exemple. Vous pouvez créer un fichier dans le dossier tmp (touch tmp/test) dans votre conteneur et vérifiez que ce fichier n'existe pas sur l'hôte.
5. Affichez aussi les namespace des processus de pid 1 (celui du namespace root – le vrai processus init, et celui de votre namespace le bash). Quelles sont les namespaces qui sont différents ?
6. Observez la filiation de votre conteneur bash.

   `docker run –it bash` //si pas déjà en cours d'exécution

   Dans une autre console, récupérez le pid de votre bash (commande ps a) puis tapez la commande ci-dessous.

   `pstree –s –p <pid>`

7. Affichez les cartes réseaux et les adresses IP (commandes `ip link` et `ip addr`) dans le conteneur et à l'extérieur. Les interfaces/adresses sont-elles les mêmes ?
8. Le conteneur peut-il accéder à l'interface de l'host (testez avec un ping <@IP>) et sur l'Internet (ping www.google.fr pour vérifier).
9. Le conteneur bash doit toujours être en cours d'exécution. Vous allez voir si les processus de votre conteneur appartiennent à un cgroup particulier. Pour ce faire, après un « ps a »

permettant de récupérer le pid de votre bash/conteneur affichez le contenu du fichier suivant et expliquez (sur l'host):

```
cat  /sys/fs/cgroup/memory/docker/<votre conteneur id en hexa>/crgoup.procs
```

10. Arrêter votre conteneur et relancez en un avec des capacités CPU limitées :

```
docker run -it --cpu-shares=512 bash
```

11. Vérifiez dans le fichier cpu-shares que cette limite est bien affectée au cgroup (sur l'host) :

```
cat /sys/fs/cgroup/cpu/docker/<votre conteneur id en hexa>/cpu.shares
```

Exercice 3 : utilisateurs et privilèges.

1. Lancez le conteneur bash avec la commande sudo :

```
sudo docker run –it bash
```

2. Que vous donne la commande id ? Etes-vous root sur le container ? Etes-vous le root de la machine host ?

3. Pour lancer un conteneur en simple utilisateur il faut rajouter un utilisateur au groupe docker :

```
usermod -aG docker <votreLogin>
```

Vérifiez que vous avez bien été rajouté au groupe (fichier /etc/group).

La prise en compte effective de ce changement de groupe peut requérir un redémarrage de votre Linux. Donc n'hésitez pas à redémarrer si la commande suivante ne fonctionne pas.

```
docker run –it bash
```
//Ne pas le faire en sudo !

4. Vérifiez avec la commande `ps au` dans une autre console à quel utilisateur est associé le bash et la commande docker quand c'est lancé en sudo ou non.

5. Pour lancer un conteneur pour un utilisateur donné, il faut rajouter l'option --user. Récupérez l'uid de votre utilisateur et lancez un shell bash :

```
docker run -it  --user=<user UID>  bash
```
//Sans le sudo!!!

Faites de même en sudo:

```
sudo docker run -it  --user=<user UID>  bash
```

Le propriétaire du bash est il le même dans les deux (commande `ps au` dans une autre console) ?

6. Lancez le bash une nouvelle fois :

```
docker run  -it bash
```

L'utilisateur associé est donc le root. Testez les privilèges de ce root avec quelques commandes qui demandent les droits privilégiés (chown, chmod, hostname, etc.).

Le root est le même que sur la machine host. Cependant, il a des « capabilities » moindre. Pour tester, vous pouvez tapez les commandes suivantes.

Dans le bash/conteneur :

```
bash# cat /proc/1/status
```

Récupérez la ligne commençant par « CapEff ». Celle-ci vous donne la liste des « capabilities » autorisées par le SE pour votre processus. Pour l'obtenir en format plus lisible :

```
capsh  --decode=<valeurHexa>
```

7. Faites de même un avec une console root classique (`sudo -s ; cat /proc/$$/status ; capsh --decode=…`) et observez que la liste des capabilities/appels systèmes autorisés pour le root est nettement inférieur dans le conteneur.

Note : un conteneur peut avoir les droits root et plus de capabilities. Il faut la lancer en mode priviligié avec l'option --priviliged.

Exercice 4 : service web

Pour cet exercice un accès au réseau est nécessaire.

1. Listez les images disponibles sur docker-hub correspondant à un service web de votre choix (nginx, apache2, etc.).

   ```
   docker search <mot_cle>
   ```

   `docker search --help` //pour les options (nombre de résultats, etc.)

   Les conteneurs contenant un « / » sont des conteneurs non officiels, mis à disposition par la communauté sur docker hub.

2. Téléchargez et lancez un de ces conteneurs. Accédez à la page web via votre navigateur. Pour que cela fonctionne, il faudra probablement effectuer un port mapping avec l'option « `-p 80:80` » lors du lancement du conteneur et accéder à la page web : `http://localhost`.
3. Vous pouvez accéder au conteneur de deux manières. Il s'agira d'ouvrira un bash ici. Testez les deux approches.
   Pour un conteneur en cours d'exécution (le nom est à récupérer avec `docker ps`):

   ```
   docker exec -it <nom du conteneur> bash
   ```

   Sinon sur un conteneur au démarrage (par ce qu'il a justement du mal à démarrer ou que vous souhaitez changer un élément de configuration interne) :

   ```
   docker run -it <nom de l'image> bash
   ```

4. Modifiez l'un des paramètres de votre serveur web (port à l'écoute, page web, etc.). Rechargez votre serveur web si nécessaire (pas votre conteneur). Testez avec votre navigateur.

5. Arrêtez l'exécution de votre conteneur.

   `docker stop <nom du conteneur>` //Les signaux SIGTERM puis SIGKILL seront envoyés aux processus de votre conteneur.

6. Relancez votre conteneur. Les modifications apportées tout à l'heure sont-elles toujours effectives ?
7. Démarrez votre conteneur une nouvelle fois. Mettez en pause puis relancez votre conteneur. Cela met en pause l'ensemble des processus du conteneur.

   `docker pause/unpause <nom du conteneur>`

   `docker ps` //A faire entre la pause et le unpause

8. Démarrez plusieurs fois votre conteneur avec des ports côté host différents (port mapping entre le conteneur et l'hôte). Testez avec votre navigateur sur ces différents ports.

   `docker run –p 5000:80 <nom de l'image>`

   `docker run –p 5001:80 <nom de l'image>`

   `docker run –p 5002:80 <nom de l'image>`

   Listez les conteneurs en cours d'exécution avec `docker ps`. Puis, testez avec votre navigateur : `http://localhost:500x`.

9. Arrêtez vos conteneurs puis supprimez l'image.

   `docker rmi -f <nom de l'image>`

   Le –f est optionnel et force la suppression si un conteneur est en cours d'exécution (arrêt des processus correspondants).

Exercice 5 : persistance des données et logs

Les modifications opérées dans le conteneur sont détruites à la fin de l'exécution de celui-ci. Il est cependant possible de conserver ces données au travers des volumes et de `mount bind`.

1. Lancez un conteneur avec un répertoire commun avec l'hôte :

```
docker run –it --mount type=bind,source=chemin1,target=chemin2 bash
```

   Dans la commande ci-dessus `chemin1` devra être le chemin absolue d'un dossier du système de fichiers host, et `chemin2` le chemin absolu dans le conteneur. Lancez cette commande et testez en créant des fichiers (dans le bon dossier !). Vérifiez que ces fichiers existent bien sur le point de montage spécifié sur l'host.

2. Il est aussi possible d'utiliser des volumes **nommés** (système de fichiers géré par docker), vous pouvez créer un volume :

   `docker volume create volume1`

   `docker volume ls` //pour vérifier son existence

   `docker volume inspect volume1` //pour voir ses propriétés

3. Il y a deux moyens équivalents de monter ce volume dans votre conteneur lors du lancement :

   ```
   docker run -it --mount source=volume1,target=/var/newFolder2 bash
   ```

   ```
   docker run -it --volume volume1:/var/newFolder2 bash
   ```
   //ou `-v`

   Notez que le volume est créé s'il n'existe pas.

   Testez l'une des commandes ci-dessus, et vérifiez que les écritures sont persistantes d'un lancement de votre conteneur à l'autre.

4. Testez un même volume avec deux conteneurs en même temps. Vérifiez que les écritures du premier sont bien visible sur le second.
5. Testez aussi un volume en lecture seule (il faudra mettre des choses dedans avant pour que cela ait un intérêt).

   ```
   docker run -it --volume volume1:/var/newFolder2:ro bash
   ```

6. Effacez vos volumes:

   ```
   docker volume rm volume1
   ```

7. Enfin vous pouvez créer des volumes anonymes (non nommés). Ils sont effacés à la fin de l'exécution du conteneur.

   ```
   docker run -it --volume /var/newFolder bash
   ```

   Créez quelques fichiers dans ce dossier, puis relancez votre conteneur. Les fichiers sont-ils toujours là ?

8. Lancez un ou deux conteneurs qui affiche quelques éléments sur leur sortie standard. Affichez les logs pour ces conteneurs. Les logs sont ici équivalent à la sortie standard de vos conteneurs.

   ```
   docker logs <nom_du_conteneur>
   ```
   //le nom est obtenu avec docker ps

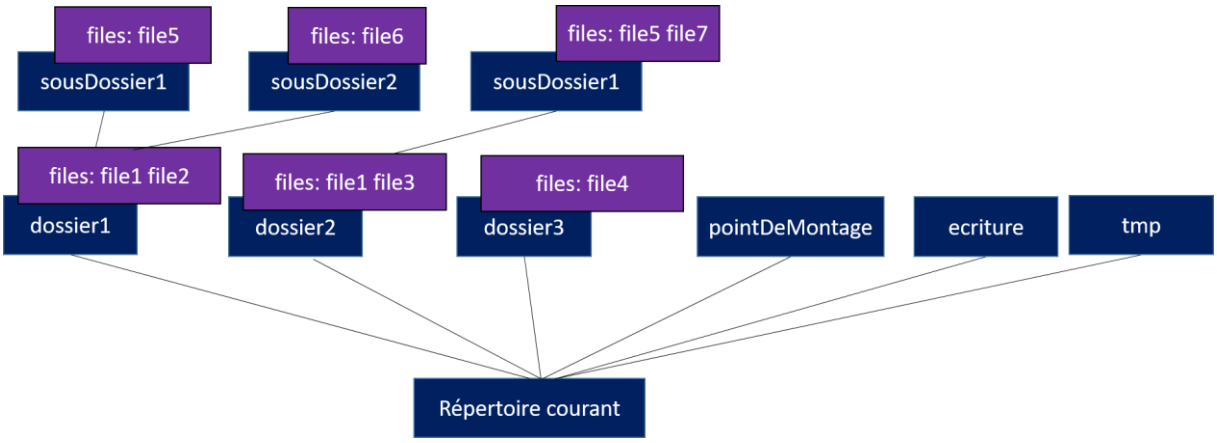
2ème partie

Attention : le nom des images, des logins, des volumes ne doivent pas comporter de majuscules dans docker.

Exercice 1 : système de fichiers overlay

Système de fichiers en lecture

1. Dans un répertoire créer les dossiers et fichiers suivants :



Il faut bien respecter les noms de dossiers et de fichiers. Il faut que les fichiers file5 et file1 aient des contenus différents dans le dossier1 et dossier2.

2. Monter sur un format « overlay » l'ensemble de ces dossiers sur le point de montage « pointDeMontage » :

```
mount -t overlay overlay -o lowerdir=dossier1:dossier2:dossier3 pointDeMontage
```

3. Observer l'arborescence et les fichiers obtenus (dans pointDeMontage). Il faut observer que c'est l'union qui est effectuée. Observez aussi que les fichiers qui apparaissaient plusieurs fois (file1 et file5) ont été écrasés. Ce sont les couches qui se superposent, la couche de niveau supérieure écrasant les fichiers de même nom. Déduisez-en l'ordre des couches (inférieur-supérieur) entre les dossiers dossier1, dossier2, dossier3.
4. Peut-on modifier les fichiers ou en créer ? Testez.
5. Démontez le système de fichiers

```
umount pointDeMontage
```

<u>Système de fichiers en lecture/écriture</u>

Vous allez maintenant créer un système de fichiers overlay qui puisse être modifié. Les couches inférieures sont toujours en lecture uniquement. C'est uniquement la couche de plus haut niveau (the upper) qui est en écriture et dans laquelle toutes les modifications seront enregistrées. Le système overlay a aussi besoin d'un répertoire de travail utilisé pour gérer les fichiers d'une couche à l'autre.

1. Montez de nouveau un système de fichiers overlay :

```
mount -t overlay overlay -o
lowerdir=dossier1:dossier2:dossier3,upperdir=ecriture,workdir=tmp pointDeMontage
```

2. Modifiez `file1`, et ajoutez deux nouveaux fichiers : un sur votre racine (`pointDeMontage`), et dans le sous-dossier `sousDossier2`.
3. Démontez le système de fichiers et observez les modifications dans `dossier1`, et surtout dans `ecriture`.

Toutes les modifications sont apportées dans le dossier écriture (the upper). Lorsqu'un fichier est modifié, aucune modification n'est apportée aux couches inférieurs mais comme la dernière couche écrase les précédentes, les modifications sont visibles/effectives lors de la fusion.

Exercice 2 : création de votre propre image

Première image : un bash de base avec la commande ip en plus. Nécessite un accès Internet.

1. Lancez le conteneur ubuntu :

   ```
   docker run –it ubuntu bash
   ```

   Dans ce bash, tester la commande « ip addr ». Celle-ci n'est pas installée.

2. Vous souhaitez donc créer une image ubuntu mais avec la commande ip installée. Dans un nouveau répertoire créer un fichier « dockerfile » partant de l'image « ubuntu », installant la commande ip en plus. La commande ip s'installe avec «`apt-get install iproute2` ». Vous aurez probablement un « apt-get update ») à faire avant. La commande par défaut sera `echo « Hello world ! »`.
3. Si la question précédente ne fonctionne pas essayer « apt-get install » avec l'option « –y » pour l'installation (cela évite la question sur l'installation).
4. Pour tester :

   `docker build -t <votre image> .` //construit votre image (Dockerfile du répertoire courant) – ne pas oublier le « . » à la fin de la commande – pas de majuscule dans les noms.

   `docker run <votre image>` //doit afficher Hello world !

   `docker run –it <votre image> bash` //puis tester la commande ip addr

Troubleshooting :

Remarque : Si votre VM est en bridge et pas en NAT, l'accès au réseau par le conteneur (apt-get update et apt-get install peuvent être problématique).

Remarque : Si vous êtes sur une VM openstack de l'université mettre à jour la variable d'environnement `http_proxy` (l'affichez dans la machine hôte pour connaître sa valeur).
Remarque : pour la directive « COPY » dans votre image, la source doit se trouver sur le dossier du Dockerfile ou un des sous dossiers.

Deuxième image : ubuntu avec votre propre commande, un tar, et une variable d'environnement.

Vous devez créer une image à partir d'une image ubuntu. L'image devra avoir les spécificités suivantes :

- contenir dans le dossier /bin un nouveau binaire
- décompresser un fichier tar (d'un dossier avec quelques fichiers) qui sera décompressé dans le dossier /home
- contenir une directive CMD correspondant à « Hello world ! »
- contenir une directive ENTRYPOINT correspondant à la commande echo
- contenir une variable d'environnement MYVARENV=toto

Pour tester :

- `docker run votre_image` //doit afficher Hello world
- `docker run votre_image toto` //doit afficher toto (vous pouvez aussi tester avec titi)
- `docker run --entrypoit= « ls » votre_image` //doit aficher le résultat du ls
- `docker run --entrypoint= « myCommand » votre_image`//doit effectuer votre commande
- Pour finir les tests :
  - `docker run -it --entrypoint="bash" votreImage`
  - puis vérifiez que le dossier home contient bien votre archive décompressé, et que la variable MAVARENV est bien définie (avec env ou echo).

Exercice 3 : Docker-compose

Afin de bien comprendre comment docker compose fonctionne, nous considérons deux applications (un client et un serveur) très simple. Libre à vous de mettre en place des services plus complexes (une base de données, un serveur web, un interpréteur php, etc.).

1. Installez docker-compose (apt-get installe docker-compose sous Ubuntu sinon voir le site docker).
2. Téléchargez les codes serveur.c et client.c.
3. Créez un dossier (« compose » par exemple et deux dossiers à l'intérieur dossierClient et dossierServeur). Compilez les deux fichiers dans leurs dossiers respectifs. Testez les dans deux consoles (lancez le serveur en premier).
4. Dans chacun des deux dossiers créez un docker file :

   Pour le client :

   ```
   FROM ubuntu

   COPY  client  /bin

   RUN   apt-get update && apt-get install -y iproute2 iputils-ping
   dnsutils net-tools

   ENTRYPOINT ["client"]

   CMD  ["serveur"]
   ```

   Pour le serveur :

   ```
   FROM ubuntu

   COPY  serveur /bin

   RUN apt-get update && apt-get install -y iproute2 iputils-ping dnsutils net-tools

   CMD  ["serveur"]
   ```

5. Créez vos images : `docker build -t nomImage .`
6. Revenez dans le dossier parent (« compose » dans mon exemple). Et créez un fichier docker-compose.yml avec le contenu suivant:

```
version: "3"
services:
  serveur:
    image: "serveur:latest" #serveur c'est le nom de l'image lors du
build
  client:
    image: "client:latest"
```

7. Lancez les conteneurs :

   `docker-compose up` #Vous devez vous trouver dans le dossier de votre .yml

8. Vérifiez que la connexion fonctionne bien entre le client et le serveur (en fonction des logs).
9. Quand le client est encore en cours d'exécution, lancez un bash dans celui-ci

   `sudo docker exec -it nom_du_conteneur  "bash"` #le nom est la dernière colonne de docker ps

10. Dans le conteneur, vérifiez qu'une adresse IP lui a bien été donné (ip address show).
11. Observez comment le client obtient l'adresse de l'autre conteneur pour se connecter : nslookup serveur. Expliquez.

Code du client :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <arpa/inet.h>

void erreur(char* string, int exit_status);

int main(int argc, char* argv[])
```

```
{
	int sockfd, error, i, nbOct;
	//struct sockaddr_in serverAddr, clientAddr;
	struct addrinfo hints, *res, *lecture;
	void *addr;
	char buffer[50], port[10]="5000";


	if(argc<2)
	{
		fprintf(stderr,"Erreur: l'adresse/nom du destinataire est
manquant.\n");
		fprintf(stderr,"Usage: %s dest\n",argv[0]);
		exit(1);
	}

	sleep(3);

	memset(&hints,0,sizeof(hints));
	hints.ai_family = AF_INET;
	hints.ai_socktype = SOCK_STREAM;

	if((error=getaddrinfo(argv[1],port,&hints,&res))!=0)
gai_strerror(error);

	if((sockfd=socket(res->ai_family, res->ai_socktype, res-
>ai_protocol))<0) erreur("Erreur socket():",2);

	printf("Client: connexion sur %s sur le port %s\n",argv[1],port);
	if(connect(sockfd, res->ai_addr, res->ai_addrlen)==0)
	{
		for(i=0;i<100;i++)
		{
			if(i%2) strcpy(buffer,"toto");
			else strcpy(buffer,"titi");

			if( (nbOct=send(sockfd,buffer,strlen(buffer),0)) < 0 )
erreur("Error: send failed",2);
			else{
				printf("Le client a envoyÃ©: %s\n",buffer);
				sleep(3);
			}
		}
	} else {
		erreur("Error: connect failed",2);
	}

	return(0);
}

void erreur(char* string, int exit_status)
{
	perror(string);
	exit(exit_status);
}
```

Code du serveur :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <arpa/inet.h>

int sockfd;
void erreur(char* string, int exit_status);
void zombie(int signal);
void controleC(int signal);
int manageConnection(int confd);

int main()
{
      int confd, pid, size, error, tr=1;
      struct sockaddr_in serverAddr, clientAddr;
      struct addrinfo hints, *res, *lecture;
      void *addr;
      char clientAddrASCII[50];

      if(signal(SIGCHLD,zombie)<0) erreur("Erreur: signal() SIGCHLD",4);
      if(signal(SIGINT,controleC)<0) erreur("Erreur: signal()
SIGCHLD",4);

      memset(&hints,0,sizeof(hints));
      hints.ai_family = AF_INET;
      hints.ai_socktype = SOCK_STREAM;
      hints.ai_flags = AI_PASSIVE;

      if((error=getaddrinfo(NULL,"5000",&hints,&res)) !=0 )
gai_strerror(error);

      if((sockfd=socket(res->ai_family, res->ai_socktype, res-
>ai_protocol))<0) erreur("Erreur socket():",2);
      if(setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&tr,sizeof(int)) == -
1) erreur("Erreur setsockopt() SO_REUSEADDR",2);
      if(bind(sockfd, (struct sockaddr *) res->ai_addr, res-
>ai_addrlen)<0) erreur("Erreur bind()",2);
      if(listen(sockfd, SOMAXCONN)<0) erreur("Erreur listen()",2);

      size=sizeof(clientAddr);
```

```
	while((confd=accept(sockfd, (struct sockaddr *) &clientAddr,
(socklen_t *) &size))>0)
	{
		pid=fork();

		switch(pid)
		{
			case 0:	fprintf(stderr,"Nouvelle connexion addr:%s
port client:%d port
local:6000\n",inet_ntop(AF_INET,&clientAddr.sin_addr,clientAddrASCII,si
zeof(clientAddrASCII)),ntohs(clientAddr.sin_port));
				manageConnection(confd);
				exit(0);
				break;
			case -1: erreur("Erreur: fork()",4);
				 break;
			default: close(confd);
				 break;
		}

	}

	erreur("Erreur accept()",2);
	return(0);
}

void erreur(char* string, int exit_status)
{
	perror(string);
	exit(exit_status);
}

void zombie(int signal)
{
	int pid;

	while((pid=waitpid(-1,NULL,WNOHANG))>0)
	{
		//fprintf(stderr,"DEBUG: terminaison fils de pid %d\n",pid);
	}
}

void controleC(int signal)
{
	printf("Fin du serveur\n");
	close(sockfd);
	exit(0);
}

int manageConnection(int confd)
{
	int retourRead;
	char bufferRecv[500];

	while( (retourRead=read(confd,bufferRecv,sizeof(bufferRecv)))>0)
	{
		bufferRecv[retourRead]='\0';
		printf("The serveur has received: %s\n",bufferRecv);
```

```
        }

        close(confd);
}
```