CONSERVATOIRE NATIONAL DES ARTS ET METIERS

Samedi 8 septembre 2007
2 heures

METHODES DE PROGRAMMATION SYSTEME

UE NSY103

CORRIGE EXAMEN

Le sujet doit impérativement être remis dans la copie

CONSERVATOIRE NATIONAL
DES ARTS ET METIERS
CENTRE RÉGIONAL
CHAMPAGNE-ARDENNE

Centres Régionaux du PTI Nord-Est

Vrai / Faux (2 points)

1. Un objet IPC disparaît lorsque le programme qui l'a mis en place se termine. (0,5 point)
Faux

2. Plusieurs processus peuvent accéder au même espace d'adressage. (0,5 point)
Faux

3. Un processus fils partage avec son père toutes les variables que son père a déclaré avant sa création. (0,5 point)
Faux

4. La valeur d'un sémaphore ne pas être négative. (0,5 point)
Faux

Questions (3 points)

1. Expliquez le principe de redirection des entrées/sorties standards. (1 point)

Ce mécanisme permet de réutiliser un programme binaire qui attend des données de l'utilisateur et qui affiche ses résultats à l'écran. On redirige l'entrée standard pour que ce programme prenne les données dans un fichier, par exemple. On redirige ensuite la sortie standard pour que les résultats de ce programme ne soient pas afficher sur l'écran mais envoyés, par exemple, dans un autre fichier.

2. Expliquez pourquoi la taille du quantum dans une politique d'ordonnancement de type « tourniquet » doit être choisie avec soin. (1 point)

Si le quantum utilisé est trop grand les processus ayant une durée d'exécution longue vont monopoliser le processeur durant toute la durée du quantum. Ce phénomène va entraîner une augmentation des temps de réponse du système. A l'inverse, l'utilisation d'un quantum trop petit va provoquer une augmentation des commutations de contexte ce qui aura pour effet, une dégradation des performances du système. En effet, une partie non négligeable du temps offert par le processeur sera utilisée pour réaliser ces commutations de contexte.

3. Quelle est la différence entre un serveur sans état et un serveur avec états ? (1 point)

Le serveur sans état ne conserve aucune information concernant les requêtes en cours de traitement. En cas de panne, les traitements en cours ne pourront pas être restaurés, le travail déjà effectué sera perdu. Avec un serveur à états, des informations concernant le traitement des requêtes courantes sont stockées sur un support non volatile. En cas de panne, le serveur peut alors récupérer une partie du travail qui a déjà été effectué. Ce serveur va « se souvenir » des requêtes qu'il été en train de traiter au moment de la panne.

Exercice 1 (5 points)

On dispose du programme en langage C suivant :

```
#include<stdio.h>

void Fils1(void)
{
	/* Traitements */
}

void Fils2(void)
{
	/* Traitements */
}

void Fils3(void)
{
	/* Traitements */
}

int main(void)
{
	if(fork()==0)
	{
		printf("Création du processus 1…\n");
		Fils1();
	}
	if(fork()==0)
	{
		printf("Création du processus 2…\n");
		Fils2();
	}
	if(fork()==0)
	{
		printf("Création du processus 3…\n");
		Fils3();
	}
	/* Traitements */
	return 0;
}
```

1. Déterminez le nombre total de processus qui vont être créés lors du lancement de ce programme. (1,5 point)

Le lancement du programme principal va créer un processus. Au premier appel à la fonction fork on va avoir une duplication du code et on va se retrouver avec un processus P et un processus P1. Le processus P va continuer sont exécution et produire un processus P2 mais le processus P1 va également créer un processus P2. On se retrouve donc avec deux processus de type P2. Les processus P, P1, P2 et P2' vont ensuite créer chacun un processus de type P3. On se retrouve donc avec 4 processus de type P3, 2 processus de type P2, un processus de type P1 et un processus de type P. Le lancement de ce programme a donc donné naissance à ***8 processus****.*

2. Il semble évident que le programmeur a réalisé une erreur de conception dans son programme. Proposez une solution simple. (0,5 point)

Il suffit de rajouter l'appel à la fonction exit à la fin de chacune des trois fonctions Fils1, Fils2 et Fils3 pour être bien sûr que l'exécution du processus créé va se terminer.

3. Les différents processus créés par ce programme vont-ils s'exécuter de manière concurrente ? (0,5 point)

Oui, les processus vont s'exécuter de manière concurrente car aucun mécanisme de synchronisation n'est présent pour empêcher cette concurrence.

4. Modifiez ce programme de manière à ce que le processus principal puisse préciser dans quel ordre l'exécution de ses fils va se terminer. (2 points)

```
#include<stdio.h>

void Fils1(void)
{
	/* Traitements */
	exit(0);
}

void Fils2(void)
{
	/* Traitements */
	exit(0);
}

void Fils3(void)
{
	/* Traitements */
	exit(0);
}

int main(void)
{
	int i;
	pid_t f1,f2,f3,w;

	if((f1=fork())==0)
	{
		printf("Création du processus 1…\n");
		Fils1();
	}
	if((f2=fork())==0)
	{
		printf("Création du processus 2…\n");
		Fils2();
	}
	if((f3=fork())==0)
	{
		printf("Création du processus 3…\n");
		Fils3();
	}
	/* Traitements */
	for(i=0;i<3;i++)
	{
		w=wait();
		if(w==f1)
			printf("Mort du fils1…\n");
		else if(w==f2)
			printf("Mort du fils2…\n");
		else if(w==f3)
			printf("Mort du fils3…\n");
	}
	return 0;
}
```

5. Quelle supposition faut-il faire sur l'architecture du système informatique pour que ce programme soit plus performant que sa version avec un seul processus. (0,5 point)

Pour que ce programme soit plus performant que sa version avec un seul processus, il faut l'utiliser sur un système informatique qui comporte plusieurs processeurs de manière à ce que plusieurs processus puissent réellement travailler de manière simultanée.

Exercice 2 (2,5 points)

Considérons les 6 processus suivants :

- P1 : date de départ = 0, durée = 6, priorité = 6
- P2 : date de départ = 1, durée = 2, priorité = 5
- P3 : date de départ = 2, durée = 4, priorité = 3
- P4 : date de départ = 3, durée = 3, priorité = 4
- P5 : date de départ = 4, durée = 6, priorité = 1
- P6 : date de départ = 5, durée = 3, priorité = 2

Dessinez le schéma d'ordonnancement dans le cas où l'on est sur un système non préemptif puis dans le cas où le système est préemptif.

[…]

Exercice 3 (5 points)

On veut écrire un service de conversion en minuscules. On transmet des caractères et le service réalise la conversion des majuscules en minuscules et retourne les caractères au client.

1. Quel type de socket est-il préférable d'utiliser ? (0,5 point)

Il vaut mieux utiliser les sockets SOCK_STREAM.

2. Ecrire le code du serveur qui offre ce service en version parallèle ainsi que celui du client. (4,5 points)

Serveur

```
#include<stdio.h>
#include<sys/types.h>
#include<netdb.h>
#include<netinet/in.h>
#include<arpa/inet.h>

int main(void)
{
	bool fin;
	char tampon[256];
	int s,sc,i,nbr;
	struct sockaddr_in myaddr,clientaddr;

	s=socket(AF_INET,SOCK_STREAM,0);
	myaddr.sin_family=AF_INET;
	myaddr.sin_s_addr=htonl(INADDR_ANY);
	myaddr.sin_port=htons(10000); // numéro de port
	bind(s,&myaddr,sizeof(myaddr));
	while(1)
	{
		sc=accept(s,&clientaddr,sizeof(clientaddr));
		fin=false;
		while(!fin)
		{
			nbr=read(sc,tampon,256);
			if(tampon[nbr-1]='\0')
				fin=true;
			else
				for(i=0;i<nbr;i++)
					if(tampon[i]>='A' && tampon[i]<='Z')
						tampon[i]+='A'-'a';
			write(sc,tampon,nbr);
		}
		close(sc);
	}
}
```

Client

```
#include<stdio.h>
#include<sys/types.h>
#include<netdb.h>
#include<netinet/in.h>
#include<arpa/inet.h>

int main(void)
{
	char chaine[256]="Bonjour Tout Le Monde",chaine2[256];
	int s,i,nbr;
	struct sockaddr_in myaddr,serveraddr;

	s=socket(AF_INET,SOCK_STREAM,0);
	myaddr.sin_family=AF_INET;
	myaddr.sin_s_addr=htonl(INADDR_ANY);
	myaddr.sin_port=htons(10020); // numéro de port
	bind(s,&myaddr,sizeof(myaddr));
	serveraddr.sin_family=AF_INET;
	serveraddr.sin_s_addr=inet("xxx.yyy.zzz.ttt"); // Adresse IP du serveur
	serveraddr.sin_port=htons(10000); // Numéro de port du service
	connect(s,&serveraddr,sizeof(serveraddr));
	for(i=0;i<strlen(chaine);i++)
	{
		write(s,chaine[i],1);
		read(s,chaine2[i],1);
	}
	close(s);
}
```

Exercice 4 (2,5 points)

On utilise un système qui possède une méthode d'allocation mémoire par zone unique avec un algorithme d'allocation de type Best Fit. La mémoire, qui a une taille de 200 Ko, contient déjà les espaces d'adressage des processus suivants :
Processus A : début en 20, taille de 40 Ko
Processus B : début en 80, taille de 50 Ko
Processus C : début en 160, taille de 40 Ko

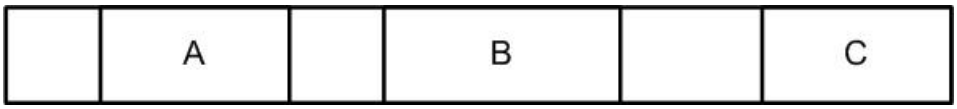
1. Quel est le principal problème de cette méthode d'allocation mémoire ? (0,5 point)

Cette méthode pose un problème car elle induit un phénomène de fragmentation. C'est-à-dire que le nombre de zones augmente mais que leur taille diminue. On se retrouve donc avec plein de zones libres mais qui ne sont pas contiguës. Il devient alors difficile d'allouer de la mémoire à d'autres processus.

2. Quelle est la solution à ce problème ? (0,5 point)

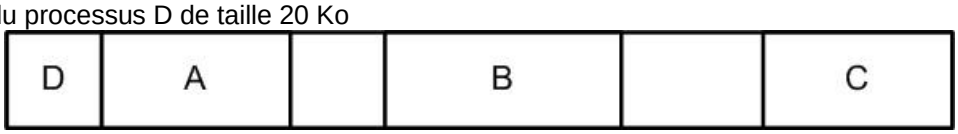
Pour résoudre ce problème on utilise le compactage qui se charge de déplacer les espaces d'adressage de tous les processus présents en mémoire de manière à rendre contigu l'espace libre. Cette méthode est très coûteuse en temps.

3. Réalisez une représentation de l'état de la mémoire. (0,5 point)

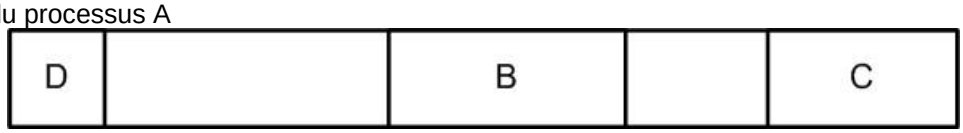


4. Représentez l'état de la mémoire après chacune des opérations suivantes : (1 point)

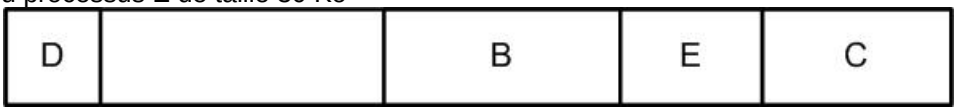
- Arrivée du processus D de taille 20 Ko



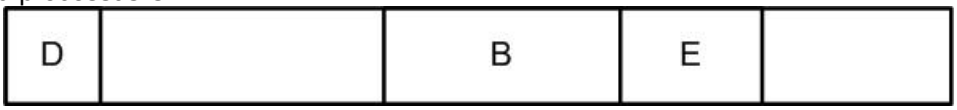
- Départ du processus A



- Arrivée du processus E de taille 30 Ko



- Départ du processus C



- Arrivée du processus F de taille 70 Ko

Fonctions C

Accept
Description : attente de demande de connexion sur une socket d'écoute
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int accept(int sock,struct sockaddr_in *p_adr,int *p_lgadr)
Paramètre 1 : identifiant de la socket
Paramètre 2 : adresse de la socket connectée
Paramètre 3 : taille de la structure adresse en octets

Bind
Description : permet d'attacher une adresse à une socket
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int bind(int s,struct sock_addr_in *adresse,int lg)
Paramètre 1 : Socket concernée
Paramètre 2 : Adresse concernée
Paramètre 3 : Taille de la structure d'adresse
Valeur de retour : -1 si problème

Connect
Description : demande de connexion sur une socket d'écoute
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int connect(int sock,struct sockaddr_in *p_adr,int *p_lgadr)
Paramètre 1 : identifiant de la socket
Paramètre 2 : adresse de la socket connectée
Paramètre 3 : taille de la structure adresse en octets

Exit
Description : termine le processus courant
Bibliothèques : stdlib.h
Syntaxe : void exit(int retour)
Paramètre 1 : valeur de retour du processus

Fork
Description : création d'un nouveau processus
Bibliothèques : unistd.h
Syntaxe : pid_t fork(void)
Valeur de retour : 0 dans le processus fils et pid du fils dans le processus père

Ftok
Bibliothèques : sys/ipc.h
Description : Création d'une clé en fonction d'un fichier
Syntaxe : key_t ftok (const char *fichier,int num)
Paramètre 1 : nom du fichier concerné
Paramètre 2 : valeur numérique quelconque
Valeur de retour : valeur d'une clé

Getpid
Description : récupère le pid du processus courant
Bibliothèques : unistd.h
Syntaxe : pid_t getpid(void)
Valeur de retour : pid du processus appelant

Getppid
Description : récupère le pid du processus courant
Bibliothèques : unistd.h
Syntaxe : pid_t getpid(void)
Valeur de retour : pid du processus appelant

Kill
Description : envoyer un signal à un processus
Bibliothèques : sys/types.h, signal.h
Syntaxe : int kill(pid_t pid,int sig)
Paramètre 1 : pid du processus destinataire
Paramètre 2 : signal à envoyer
Valeur de retour : 0 en cas de succès ou -1 en cas d'échec

Listen
Description : mise en écoute sur une socket
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int listen(int sock,int nb)
Paramètre 1 : identifiant de la socket
Paramètre 2 : nombre de connexions pendantes maximum

Mmap
Description : projection d'un fichier en mémoire
Bibliothèques : ?
Syntaxe : void* mmap(void *adr,size_t taille,int prot,int option,int desc,off_t off)
Paramètre 1 : NULL
Paramètre 2 : taille de la projection
Paramètre 3 : PROT_READ, PROT_WRITE, PROT_NONE, …
Paramètre 4 : MAP_SHARDE ou MAP_PRIVATE
Paramètre 5 : descripteur du fichier
Paramètre 6 : début de la zone à projeter
Valeur de retour : pointeur sur la zone de mémoire

Msgctl
Description : gestion d'une file de messages
Bibliothèques : sys/ipc.h, sys/msg.h
Syntaxe : int msgctl(int msgid,int cmd, struct msgid_ds *buf)
Paramètre 1 : identifiant de la file de messages
Paramètre 2 : IPC_STAT, IPC_SET ou IPC_RMID
Paramètre 3 : définition des droits
Valeur de retour : résultat de l'opération

Msgget
Description : création ou récupération de l'identifiant
Bibliothèques : sys/ipc.h, sys/msg.h
Syntaxe : int msgget(key_t cle,int options)
Paramètre 1 : clé du mécanisme IPC (valeur numérique ou IPC_PRIVATE)
Paramètre 2 : si création IPC_CREAT|[droits Unix]
Valeur de retour : identifiant du mécanisme IPC

Msgrcv
Description : recevoir un message
Bibliothèques : sys/ipc.h, sys/msg.h
Syntaxe : int msgrcv(int msgid,struct msgbuf *buf, int taille,long type,int options)
Paramètre 1: identifiant de la file de messages
Paramètre 2 : tampon pour recevoir le message
Paramètre 3 : taille du message
Paramètre 4 :

- 0 : extraire le premier message
- >0 : extraire le premier message de type type
- <0 : extraire le premier message de type <= |type|

Paramètre 5 : IPC_NOWAIT pour un envoi non bloquant
Valeur de retour : résultat de l'opération

Msgsnd
Description : envoyer un message
Bibliothèques : sys/ipc.h, sys/msg.h
Syntaxe : int msgsnd(int msgid,struct msgbuf *buf, int taille,int options)
Paramètre 1 : identifiant de la file de messages
Paramètre 2 : message à envoyer
Paramètre 3 : taille du message
Paramètre 4 : IPC_NOWAIT pour un envoi non bloquant
Valeur de retour : résultat de l'opération

Nice
Description : ajouter une valeur à la priorité dynamique du processus courant
Bibliothèques : unistd.h
Syntaxe : int nice(int i)
Paramètre 1 : valeur à ajouter à la priorité dynamique
Valeur de retour : 0 en cas de succès ou -1 en cas d'échec

Pipe
Description : création d'un tube
Bibliothèques : unistd.h
Syntaxe : int pipe(int tube[2])
Paramètre 1 : tableau de deux entiers pour recevoir les descripteurs d'entrée et de sortie du tube
Valeur de retour : 0 en cas de succès ou -1 en cas d'échec

Pthread_create
Description : création d'un thread
Bibliothèques : pthread.h
Syntaxe : int pthread_create (pthread_t *thread,const pthread_attr_t *attr,
void*(*start_routine)(void),void *arg)
Paramètre 1 : identificateur du thread
Paramètre 2 : attributs pour la création du thread (NULL pour les attributs par défaut)
Paramètre 3 : fonction que le thread doit exécuter
Paramètre 4 : arguments de la fonction à exécuter
Valeur de retour : 0 en cas de succès ou code d'erreur en cas d'échec

Pthread_exit
Description : terminer un thread
Bibliothèques : pthread.h
Syntaxe : void pthread_exit(void *valeur)
Paramètre 1 : valeur de retour du thread

Pthread_join
Description : synchroniser deux threads
Bibliothèques : pthread.h
Syntaxe : int pthread_join(pthread_t thread, void **value_ptr)
Paramètre 1 : identificateur du thread
Paramètre 2 : valeur de retour du thread mort
Valeur de retour : 0 en cas de succès ou code erreur

Pthread_self
Description : récupérer l'identificateur du thread courant
Bibliothèques : pthread.h
Syntaxe : pthread_t pthread_self(void)
Valeur de retour : identificateur du thread courant

Read
Description : lit des données dans une socket flux
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int read(int sock,char *msg,int lg)
Paramètre 1 : identifiant de la socket
Paramètre 2 : adresse de la zone mémoire pour stocker les données
Paramètre 3 : taille de la zone mémoire

Recvfrom
Description : réception de données en mode datagramme
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int recvfrom(int sock,char *msg,int lg,int option,struct sockaddr_in *p_dest,int lgdest)
Paramètre 1 : identifiant de la socket
Paramètre 2 : adresse de stockage du message
Paramètre 3 : taille de la zone en octets
Paramètre 4 : options à utiliser // 0 : message lu et retiré de la socket, MSG_PEEK : message lu et non retiré
Paramètre 5 : adresse du stockage de l'adresse de l'expéditeur
Paramètre 6 : taille de la structure d'adresse en octets

Semctl
Description : gestion d'un ensemble de sémaphores
Bibliothèques : sys/ipc.h, sys/sem.h
Syntaxe : int semctl(int semid,int semnum,int op,union semun arg)
Paramètre 1 : identifiant de l'ensemble de sémaphores
Paramètre 2 : semaphores concerné
Paramètre 3 : operation à réaliser (SETVALL, SETALL, GETVAL, GETALL, GETNCNT, GETZCNT, IPC_STAT, IPC_SET ou IPC_RMID)
Paramètre 4 : valeur associée
Valeur de retour : résultat de l'opération

Semget
Description : création ou récupération de l'identifiant
Bibliothèques : sys/ipc.h, sys/sem.h
Syntaxe : int semget(key_t cle,int options)
Paramètre 1 : clé du mécanisme IPC (valeur numérique ou IPC_PRIVATE)
Paramètre 2 : si création IPC_CREAT|[droits Unix]
Valeur de retour : identifiant du mécanisme IPC

Semop
Description : réalisation d'une opération sur un ensemble de sémaphores
Bibliothèques : sys/ipc.h, sys/sem.h
Syntaxe : int semop(int semid,struct sem_buf *p,unsigned int nbop)
Paramètre 1 : identifiant de l'ensemble de sémaphores
Paramètre 2 : tableau d'opérations
Paramètre 3 : nombre d'opérations à réaliser
Valeur de retour : résultat de l'opération

Sendto
Description : envoi de données en mode datagramme
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int sendto(int sock,char *msg,int lg,int option,struct sockaddr_in *p_dest,int lgdest)
Paramètre 1 : identifiant de la socket
Paramètre 2 : message à émettre
Paramètre 3 : taille du message en octets
Paramètre 4 : options à utiliser
Paramètre 5 : adresse du destinataire
Paramètre 6 : taille de la structure d’adresse en octets

Shmat
Description : attachement d'un segment de mémoire
Bibliothèques : sys/ipc.h, sys/shm.h
Syntaxe : void* shmat(int shmid,void* adrmem,int option)
Paramètre 1 : identifiant du segment partagé
Paramètre 2 : NULL
Paramètre 3 :
- 0 : lecture/écriture
- SHM_READONLY : lecture seule

Valeur de retour : pointeur sur la zone de mémoire

Shmctl
Description : gestion d'un segment de mémoire partagé
Bibliothèques : sys/ipc.h, sys/shm.h
Syntaxe : int shmctl(int msgid,int cmd, struct shmid_ds *buf)
Paramètre 1 : identifiant du segment de mémoire partagé
Paramètre 2 : IPC_STAT, IPC_SET ou IPC_RMID
Paramètre 3 : définition des droits
Valeur de retour : résultat de l'opération

Shmdt
Description : détacher un segment de mémoire
Bibliothèques : sys/ipc.h, sys/shm.h
Syntaxe : int shmdt(void* adr)
Paramètre 1 : adresse du segment mémoire
Valeur de retour : résultat de l'opération

Shmget
Description : création ou récupération de l'identifiant
Bibliothèques : sys/ipc.h, sys/shm.h
Syntaxe : int shmget(key_t cle,int taille,int options)
Paramètre 1 : clé du mécanisme IPC (valeur numérique ou IPC_PRIVATE)
Paramètre 2 : taille du segment de mémoire en octets
Paramètre 3 : si création IPC_CREAT|[droits Unix]
Valeur de retour : identifiant du mécanisme IPC

Signal
Description : définir la comportement à adopter lors de la réception d'un signal
Bibliothèques : signal.h
Syntaxe : void* signal(int sig,void(*fonction)(int)
Paramètre 1 : Signal concerné
Paramètre 2 : Fonction définie pour le signal ou SIG_DFL (action par défaut) ou SIG_IGN (signal ignoré)

Socket
Description : permet la création d'une socket
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int socket(int domaine,int type,int protocole)
Paramètre 1 : Domaine concerné (AF_INET ou AF_UNIX)
Paramètre 2 : Type de la socket (SOCK_DGRAM ou SOCK_STREAM)
Paramètre 3 : Protocole concerné

System
Description : exécuter une commande shell
Bibliothèques : stdlib.h
Syntaxe : int system(const char *commande)
Paramètre 1 : commande shell
Valeur de retour : valeur de retour de la commande shell

Wait
Description : suspend l'exécution du processus pour attendre la fin d'un processus fils
Bibliothèques : sys/types.h, sys/wait.h
Syntaxe : pid_t wait(int *statut)
Paramètre 1 : pointeur sur un entier pour stocker la valeur de retour du processus
Valeur de retour : pid du processus fils mort

Write
Description : écrit des données dans une socket flux
Bibliothèques : sys/types.h, netdb.h, netinet/in.h, arpa/inet.h
Syntaxe : int write(int sock,char *msg,int lg)
Paramètre 1 : identifiant de la socket
Paramètre 2 : adresse des données à envoyer
Paramètre 3 : taille des données

Structures et unions

```
struct sem_buf
{
        ushort sem_num;
        short sem_op;
        short sem_flg;
}

union semun
{
        int val;
        struct semid_ds *buf;
        unsigned short *array;
}

struct in_addr
{
        u_long s_addr;
}

struct sockaddr_in
{
        short sin_family; // AF_INET
        ushort sin_port;
        struct in_addr sin_addr;
        char sin_zero[8]; // 8 zéros
}
```