Création d'un nouveau processus

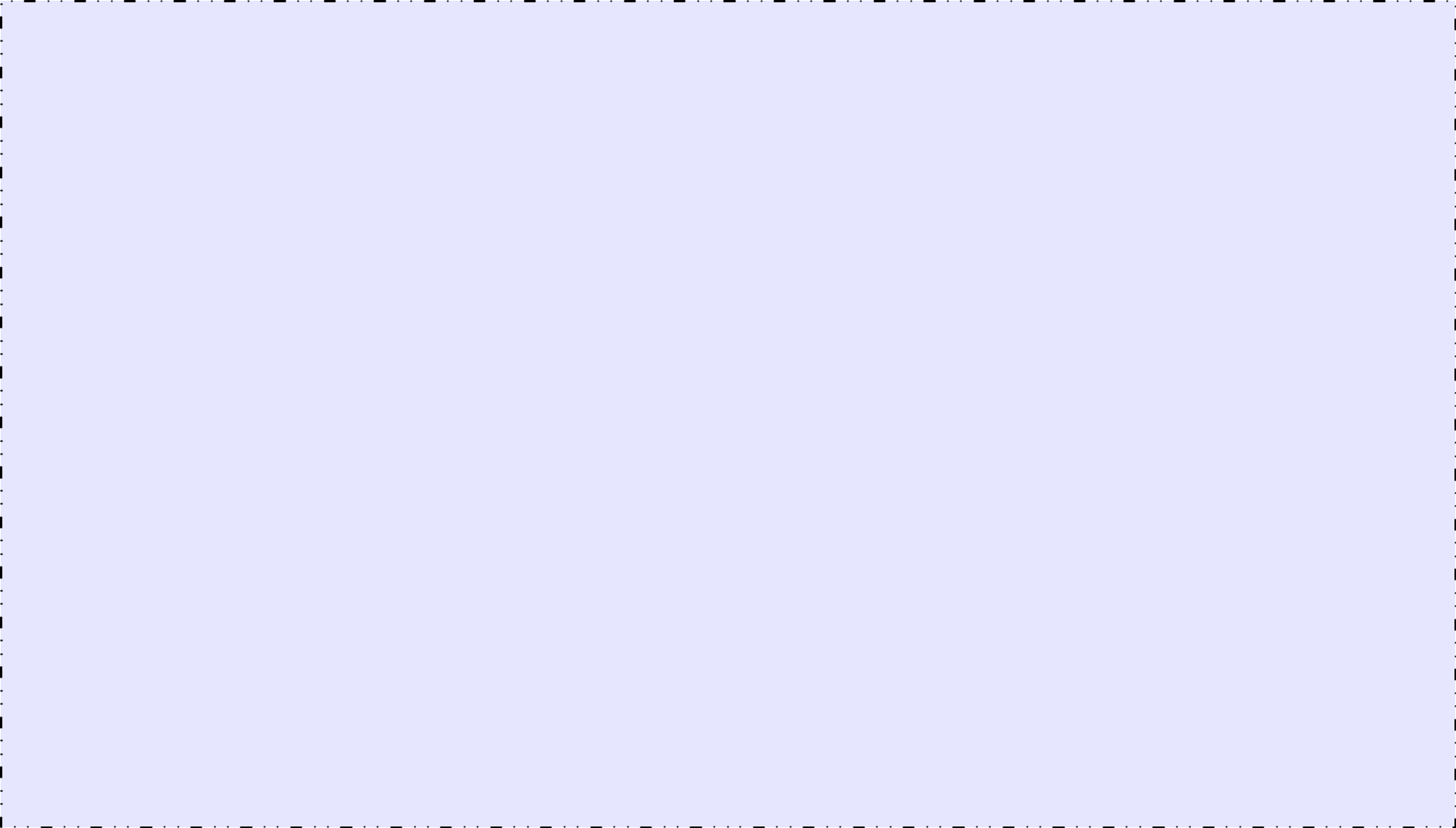
* En C, on peut créer un nouveau processus grâce à la fonction fork :
  + **pid\_t fork(void);**
* Cette fonction :
  + Crée un processus fils
  + Renvoie un identifiant de processus PID
  + On peut tester si on est le processus père ou le processus père grâce au PID renvoyé
    - fork renvoie 0 pour le fils
    - fork renvoie l'identifiant du fils au père
* Autres fonctions utiles :
  + **pid\_t getpid(void);** pour avoir l'id du processus courant
  + **pid\_t getppid(void);** pour avoir l'id du processus père

PR - Concurrence C 6

Création d'un nouveau processus (2)

* En pratique :
  + Tout se passe dans le même code
  + On teste le retour de fork pour savoir si on est le père ou le fils
    - **if(fork()==0) {...}**
  + Cela permet de distinguer ensuite les exécutions du processus fils et du père
  + Points sur lesquels il faut être attentif
    - Les variables ne sont pas partagées
    - À la création d'un processus fils, l'espace d'adressage du père est copiée

PR - Concurrence C 7

Exemple

**int main() {**

**int id=getpid();**

**printf("Je suis le processus %d\n",id); printf("Je fais un fork\n");**

**int idfork=fork();**

**if(idfork==0){**

**int idfils=getpid();**

**int idpere=getppid();**

**printf("Je suis %d fils de %d\n",idfils,idpere);**

**} else {**

**printf("Mon fils est %d\n",idfork); int etat;**

**waitpid(idfork, &etat, 0);**

**}**

**return 0;**

**}**

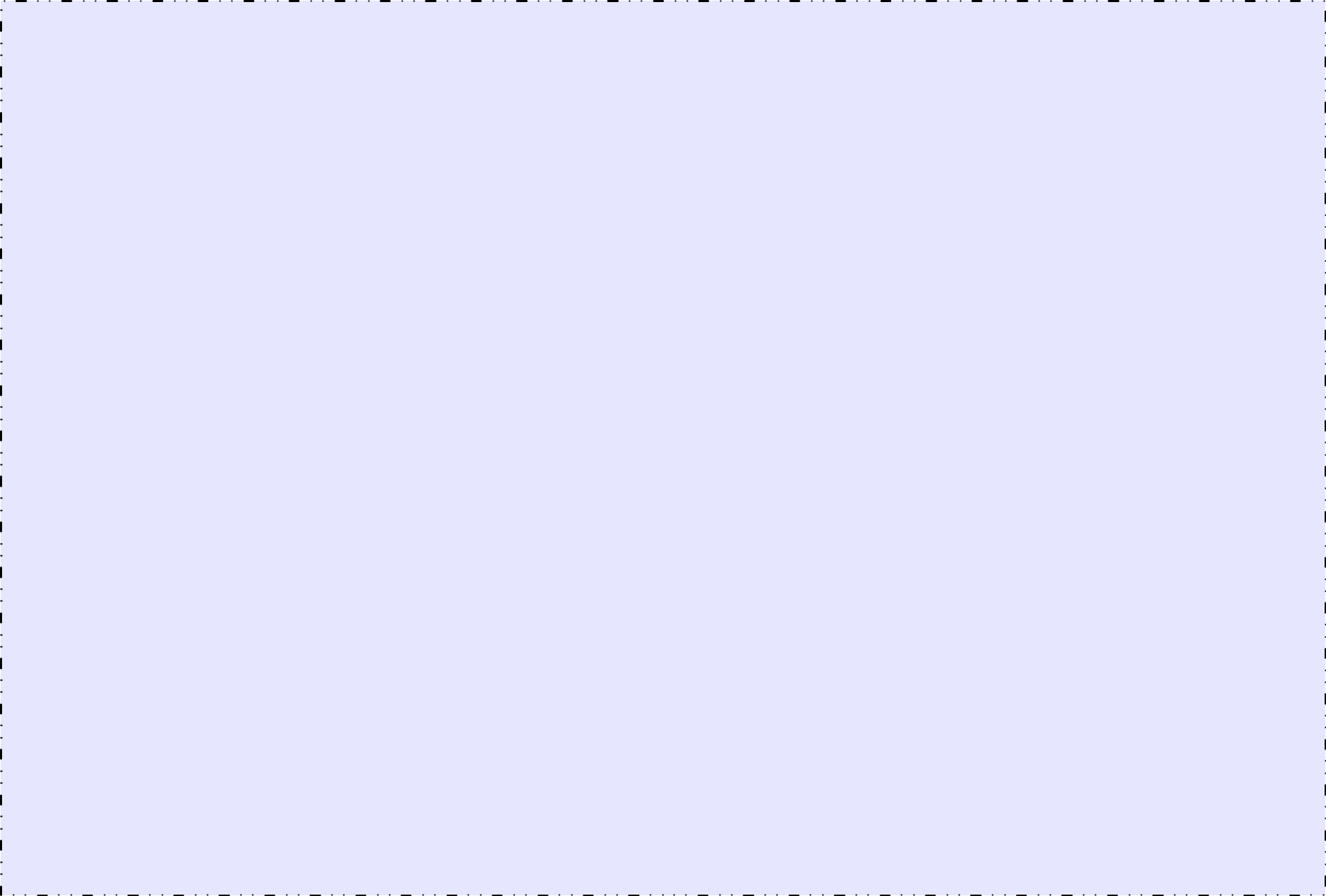
PR - Concurrence C 8

Serveur concurrent avec fork

* Au niveau du serveur, après
  + **int sock2=accept(sock,(struct sockaddr \*)&caller,&size);**
* Le serveur créera un processus fils avec **fork**
* Le processus fils devra
  + Fermer son descripteur **sock**
  + Communiquer avec le client via son descripteur **sock2**
  + Fermer **sock2** à la fin de la communication
  + Faire **exit** pour terminer son exécution
* Le processus père devra
  + Fermer son descripteur **sock2**
  + Retourner sur l'**accept** pour attendre une nouvelle connexion
* **Vu que les espaces d'adressage sont copiés, si le père ferme son sock2, il ne ferme pas celui du fils**

PR - Concurrence C 10

Exemple



|  |  |
| --- | --- |
| **int main() {** |  |
| **int sock=socket(PF\_INET,SOCK\_STREAM,0);** |  |
| **struct sockaddr\_in sockaddress;** |  |
| **sockaddress.sin\_family=AF\_INET;** |  |
| **sockaddress.sin\_port=htons(4244);** |  |
| **sockaddress.sin\_addr.s\_addr=htonl(INADDR\_ANY);** |  |
| **int r=bind(sock,(struct sockaddr \*)&sockaddress,sizeof(struct sockaddr\_in));** |  |
| **if(r==0){** |  |
| **r=listen(sock,0);** |  |
| **if(r==0){** |  |
| **struct sockaddr\_in caller;** |  |
| **socklen\_t si=sizeof(caller);** |  |
| **while(1){** |  |
| **int sock2=accept(sock,(struct sockaddr \*)&caller,&si);** |  |
| **if(sock2>=0){** |  |
| **int idfork=fork();** |  |
| **if(idfork==0){** |  |
| **close(sock);** |  |
| **/\*Code de communication\*/** |  |
| **close(sock2);** |  |
| **exit(0) ;** |  |
| **}** |  |
| **else{** |  |
| **close(sock2);** |  |
| **}** |  |
| **}** |  |
| **}** |  |
| **}** |  |
| **return 0;** | 11 |
| **}** |  |

Création de thread

* Pour créer un thread, on utilise la fonction de création de thread :
* **int pthread\_create(**

**pthread\_t \*thread, //On stockera les données du thread créé const pthread\_attr\_t \*attr, // Attributs du thread**

**void \*(\*start\_routine) (void \*), // Fonctions à exécuter void \*arg); // Arguments de la fonction à exécuter**

* En pratique pour les attributs, on mettra **NULL** pour avoir les attributs par défauts
* Ces attributs permettent par exemple de donner une politique pour l'ordonnancer
* Le code qu'exécutera est donné par la fonction **start\_routine**
* Dès que la fonction **pthread\_create** a fini, le thread est créé et s'exécute

PR - Concurrence C 17

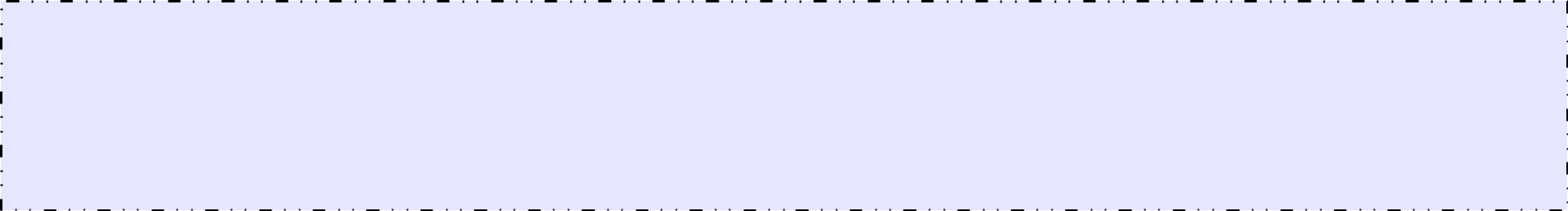
Création de thread en pratique

* On créé une fonction qui contient le code a exécuté par chacun des threads, de signature
  + **void \*function(void \*)**
* Si on veut donner des arguments à cette fonction au moment de l'appel de thread
  + on les passe comme dernière argument de la fonction **pthread\_create**
* Il faut parfois attendre la fin d'exécution des thread créé :
  + si le programme principale termine avant, les threads sont détruits
  + on peut utiliser pour cela la fonction
    - **int pthread\_join(pthread\_t thread, void \*\*value\_ptr);**
  + le deuxième argument est celui créé par pthread\_create
  + value\_ptr est la valeur renvoyé par le thread en appelant
    - **void pthread\_exit(void \*value\_ptr);**
  + Attention : un appel à exit fait terminer le processus !!!!

PR - Concurrence C 18

Exemple

* On commence par donner le code que va exécuter



**void \*affiche(void \* arg){**

**char \*s=(char \*) arg;**

**printf("Mon message est :%s",s);**

**return NULL;**

**}**

* Ensuite pour créer un thread qui exécute cette fonction, on procède ainsi

**pthread\_t th1;**



**char \*s1="Je suis thread 1\n";**

**int r1=pthread\_create(&th1,NULL,affiche,s1);**

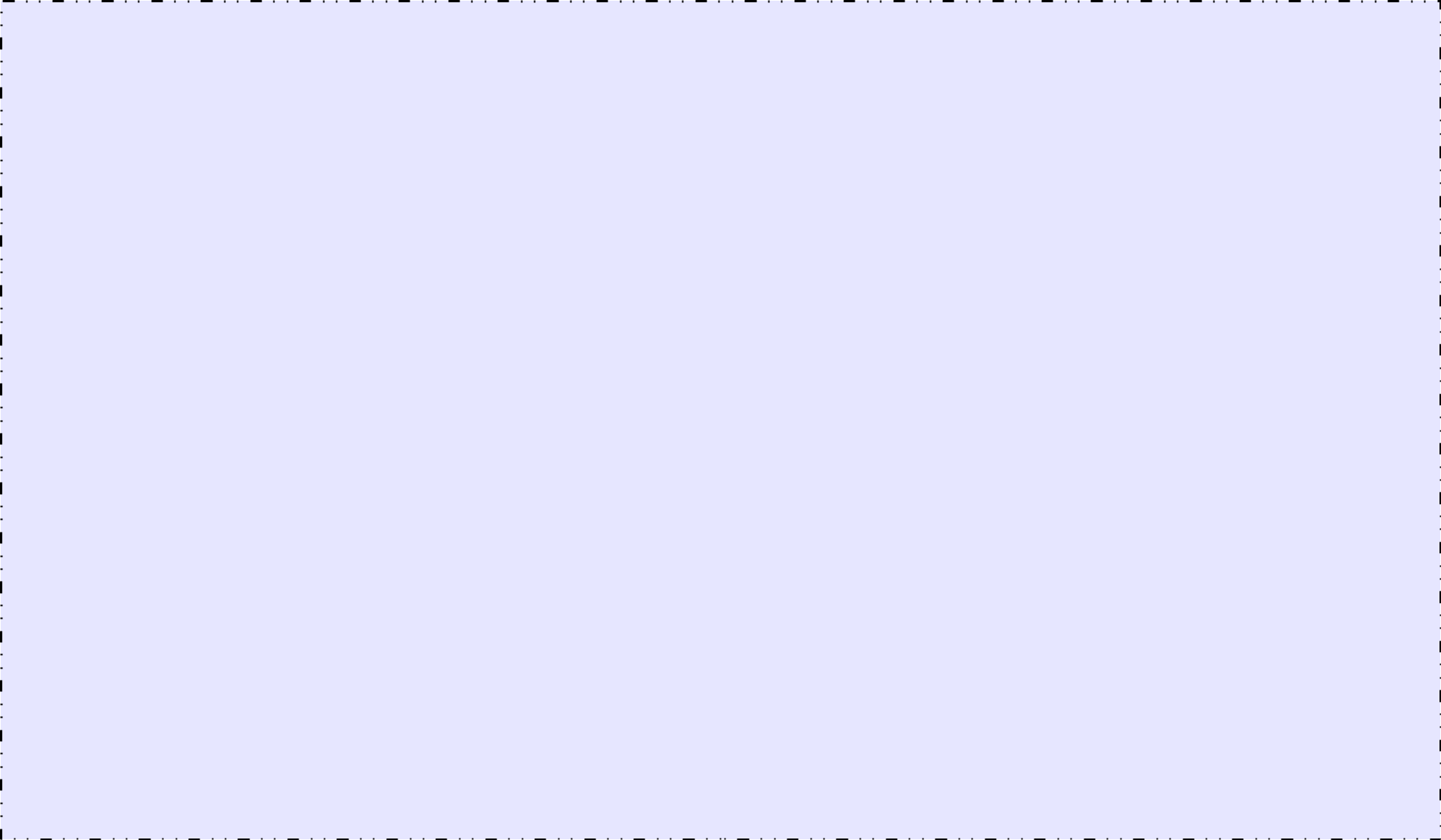
* Finalement on peut faire un join pour attendre la fin du processus



**pthread\_join(th1,NULL);**

PR - Concurrence C 19

Exemple avec retour de fonction



**int main(){**

**pthread\_t th1, th2;**

**char \*s1="Je suis thread 1\n";**

**char \*s2="Je suis thread 2\n";**

**pthread\_create(&th1,NULL,affiche,s1);**

**pthread\_create(&th2,NULL,affiche,s2);**

**char \*r1;**

**char \*r2;**

**pthread\_join(th1,(void \*\*)&r1);**

**pthread\_join(th2,(void \*\*)&r2);**

**printf("%s",r1);**

**printf("%s",r2);**

**return 0;**

**}**

**void \*affiche(void \*ptr){**

**char \*s=(char \*)ptr;**

**printf("Mon message est : %s",s);**

**char \*mess=(char\*)malloc(100\*sizeof(char)); strcat(mess,s);**

**strcat(mess," Fini\n");**

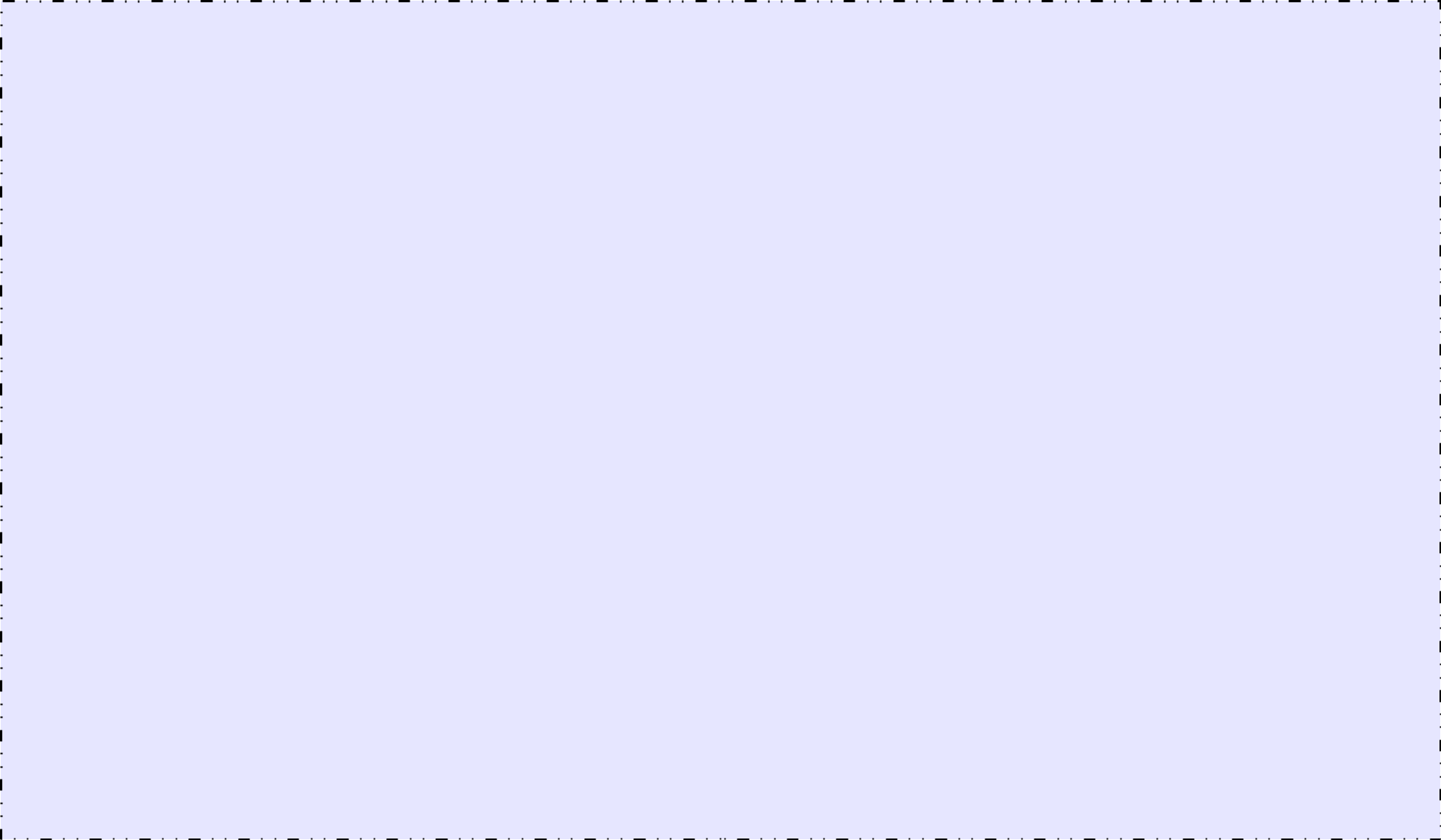
**return mess;**

**}**

PR - Concurrence C 21

Exemple avec pthread\_exit

**int main(){**



**pthread\_t th1, th2;**

**char \*s1="Je suis thread 1\n";**

**char \*s2="Je suis thread 2\n";**

**pthread\_create(&th1,NULL,affiche,s1);**

**pthread\_create(&th2,NULL,affiche,s2);**

**char \*r1;**

**char \*r2;**

**pthread\_join(th1,(void \*\*)&r1);**

**pthread\_join(th2,(void \*\*)&r2);**

**printf("%s",r1);**

**printf("%s",r2);**

**return 0;**

**}**

**void \*affiche(void \*ptr){**

**char \*s=(char \*)ptr;**

**printf("Mon message est : %s",s);**

**char \*mess=(char \*)malloc(100\*sizeof(char)); strcat(mess,s);**

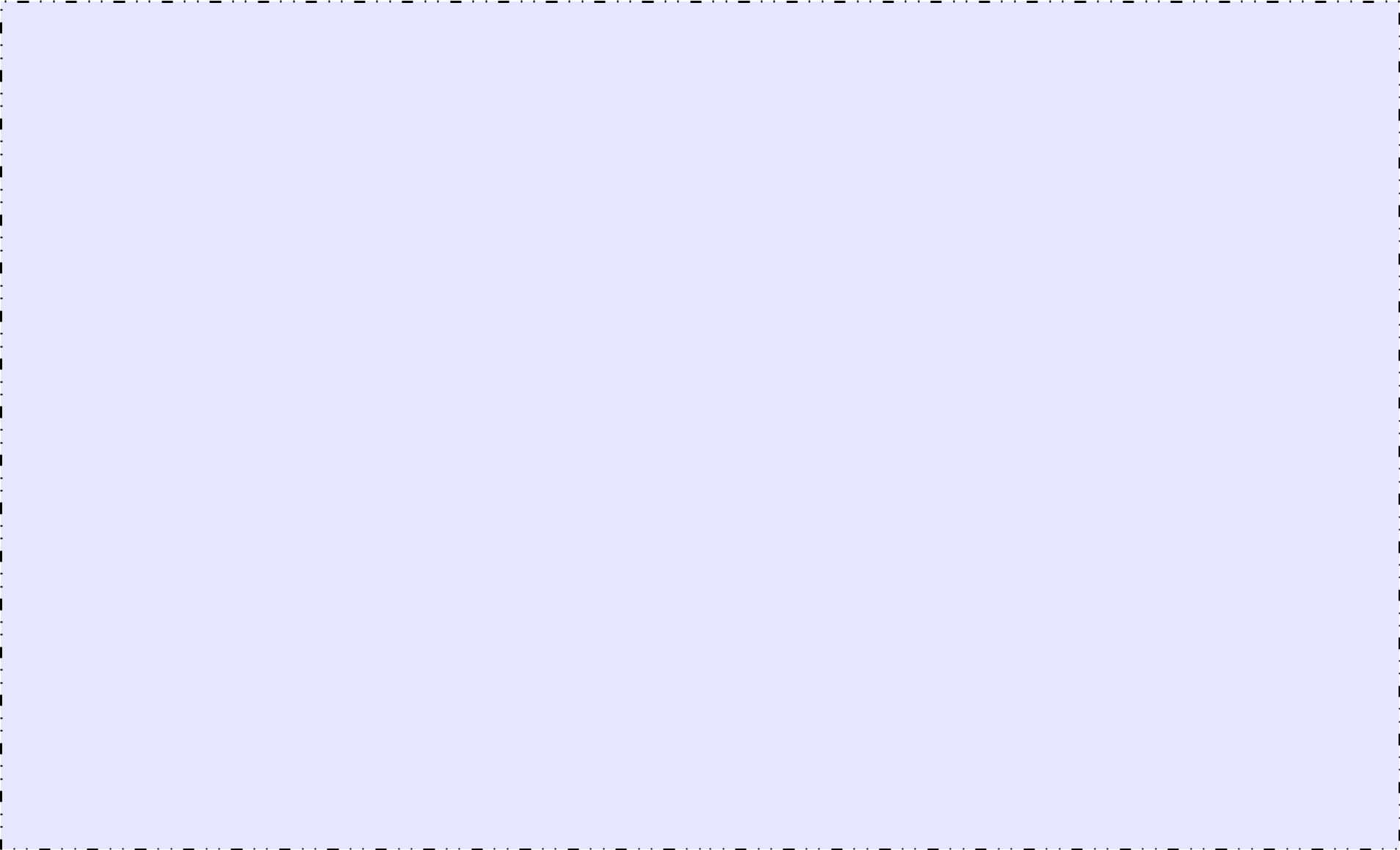
**strcat(mess," Fini\n");**

**pthread\_exit(mess);**

**}**

PR - Concurrence C 22

Partage de variables – pthread\_create



**void \*inc(void \*ptr);**

**int a=0;**

**int main(){**

**pthread\_t th1, th2;**

**pthread\_create(&th1,NULL,inc,NULL);**

**pthread\_create(&th2,NULL,inc,NULL);**

**pthread\_join(th1,NULL);**

**pthread\_join(th2,NULL);**

**printf("a vaut %d\n",a);**

**return 0;**

**}**

**void \*inc(void \*ptr){**

**a=a+1;**

**return NULL;**

**}**

PR - Concurrence C 24

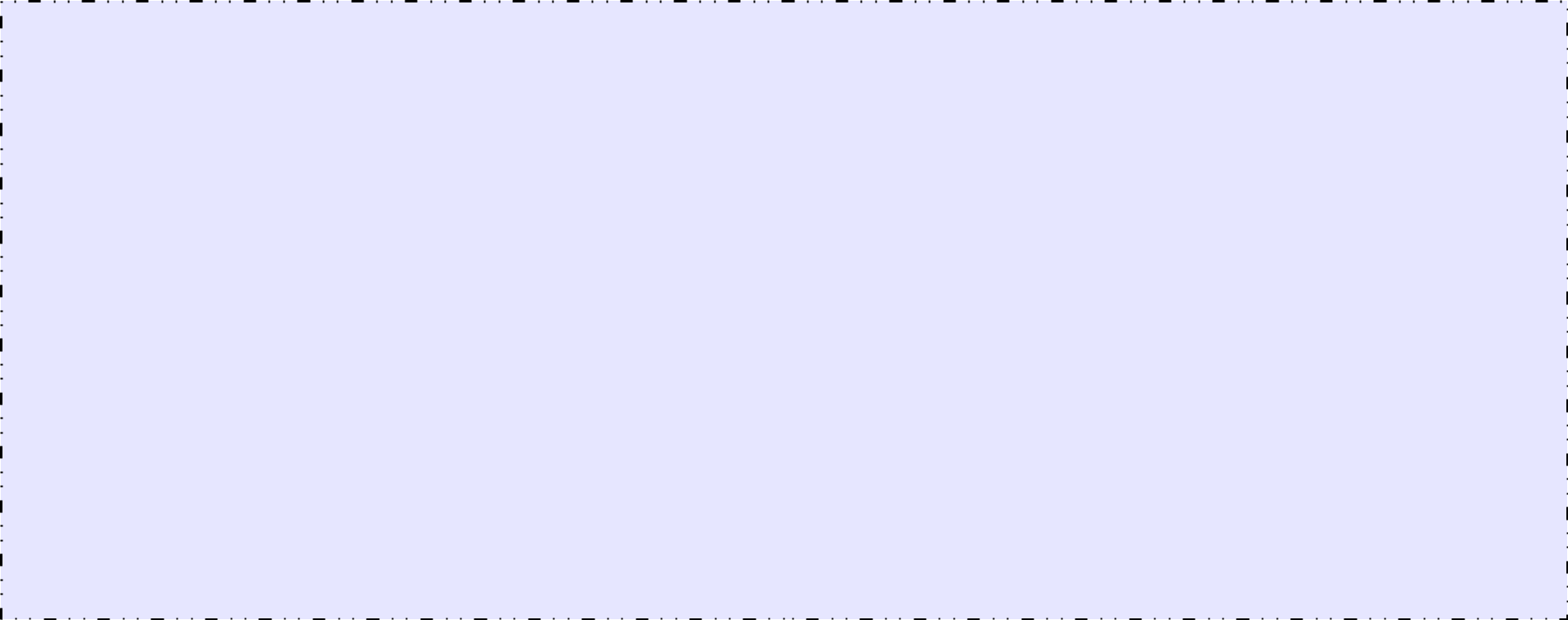
Retour sur notre serveur concurrent

* Pour programmer le serveur concurrent avec des threads, on va :
  + Créer un thread après chaque **accept**
  + La fonction qui sera exécuté par chaque thread sera en charge de la communication
  + Il faudra passer en argument de cette fonction, le descripteur de socket
    - En faisant attention à comment on le passe
  + Le thread devra uniquement fermer sa socket de communication lorsque sa conversation avec le client prendra fin
  + Le programme principal qui est censé ne jamais terminé (boucle **while(1){...}**) n'aura pas besoin de faire des **pthread\_ join** pour attendrela fin d'exécution des thread communiquant avec les clients.

PR - Concurrence C 2

Serveur concurrent (1)

**void \*comm(void \*arg){**



**int sockcomm=\*((int \*)arg); char \*mess="Hi\n";**

**send(sockcomm,mess,strlen(mess)\*sizeof(char),0); char buff[100];**

**int recu=recv(sockcomm,buff,99\*sizeof(char),0); buff[recu]='\0';**

**printf("Message recu : %s\n",buff);**

**close(sockcomm);**

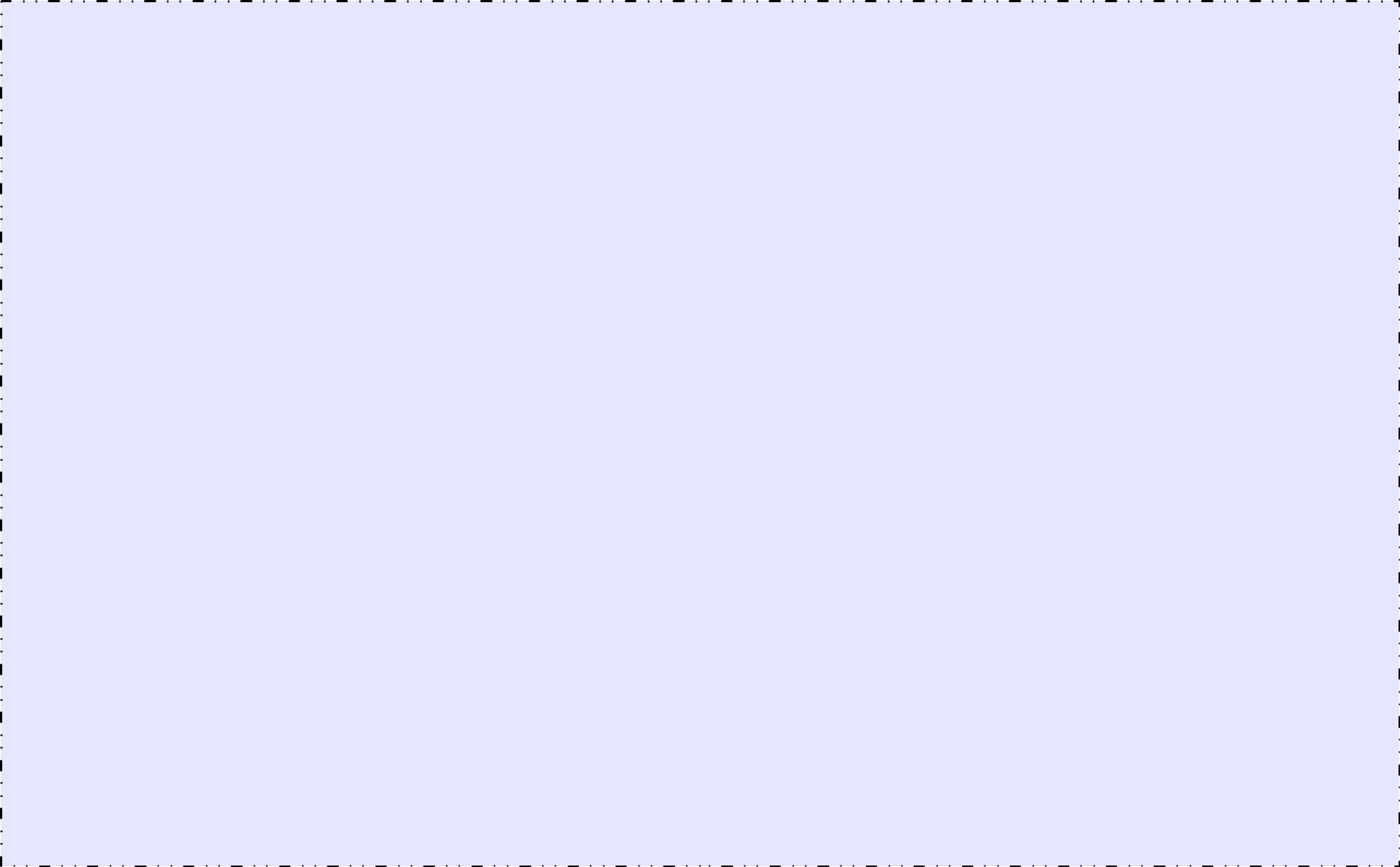
**return NULL;**

**}**

PR - Concurrence C 26

Serveur concurrent (2)

**int main() {**



**int sock=socket(PF\_INET,SOCK\_STREAM,0);**

**struct sockaddr\_in sockaddress; sockaddress.sin\_family=AF\_INET; sockaddress.sin\_port=htons(4244); sockaddress.sin\_addr.s\_addr=htonl(INADDR\_ANY);**

**int r=bind(sock,(struct sockaddr \*)&sockaddress,sizeof(struct**

**sockaddr\_in));**

**if(r==0){**

**r=listen(sock,0);**

**if(r==0){**

**struct sockaddr\_in caller;**

**socklen\_t si=sizeof(caller);**

**while(1){**

**int sock2=accept(sock,(struct sockaddr\*)&caller,&si); if(sock2>=0){**

**pthread\_t th;**

**pthread\_create(&th,NULL,comm,&sock2);**

**}**

**}**

**}**

**}**

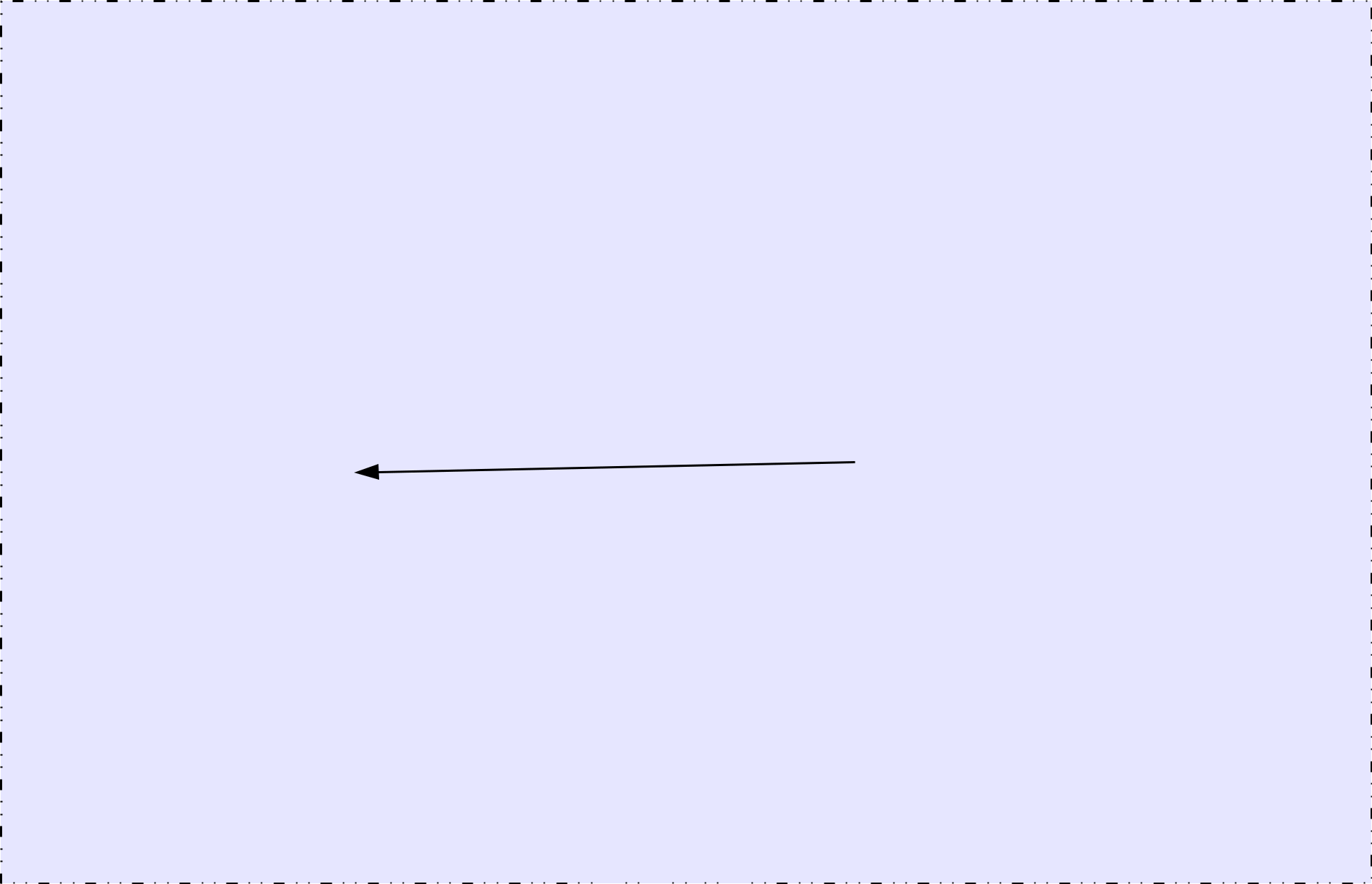
**return 0;**

**}**

PR - Concurrence C 27

Mauvais exemple serveur

**int main() {**



**int sock=socket(PF\_INET,SOCK\_STREAM,0);**

**struct sockaddr\_in sockaddress; sockaddress.sin\_family=AF\_INET; sockaddress.sin\_port=htons(4244); sockaddress.sin\_addr.s\_addr=htonl(INADDR\_ANY);**

**int r=bind(sock,(struct sockaddr \*)&sockaddress,sizeof(struct**

|  |  |  |
| --- | --- | --- |
| **if(r==0){** | **sockaddr\_in));** |  |
| **r=listen(sock,0);** |  |  |
| **if(r==0){** |  |  |
| **struct sockaddr\_in caller;** |  |  |
| **socklen\_t si=sizeof(caller);** | **NON !!!!** |  |
| **int sock2 ;** |  |
| **while(1){** |  |  |

**sock2=accept(sock,(struct sockaddr\*)&caller,&si); if(sock2>=0){**

**pthread\_t th;**

**pthread\_create(&th,NULL,comm,&sock2);**

**}**

**}**

**}**

**}**

**return 0;**

**}**

PR - Concurrence C 28

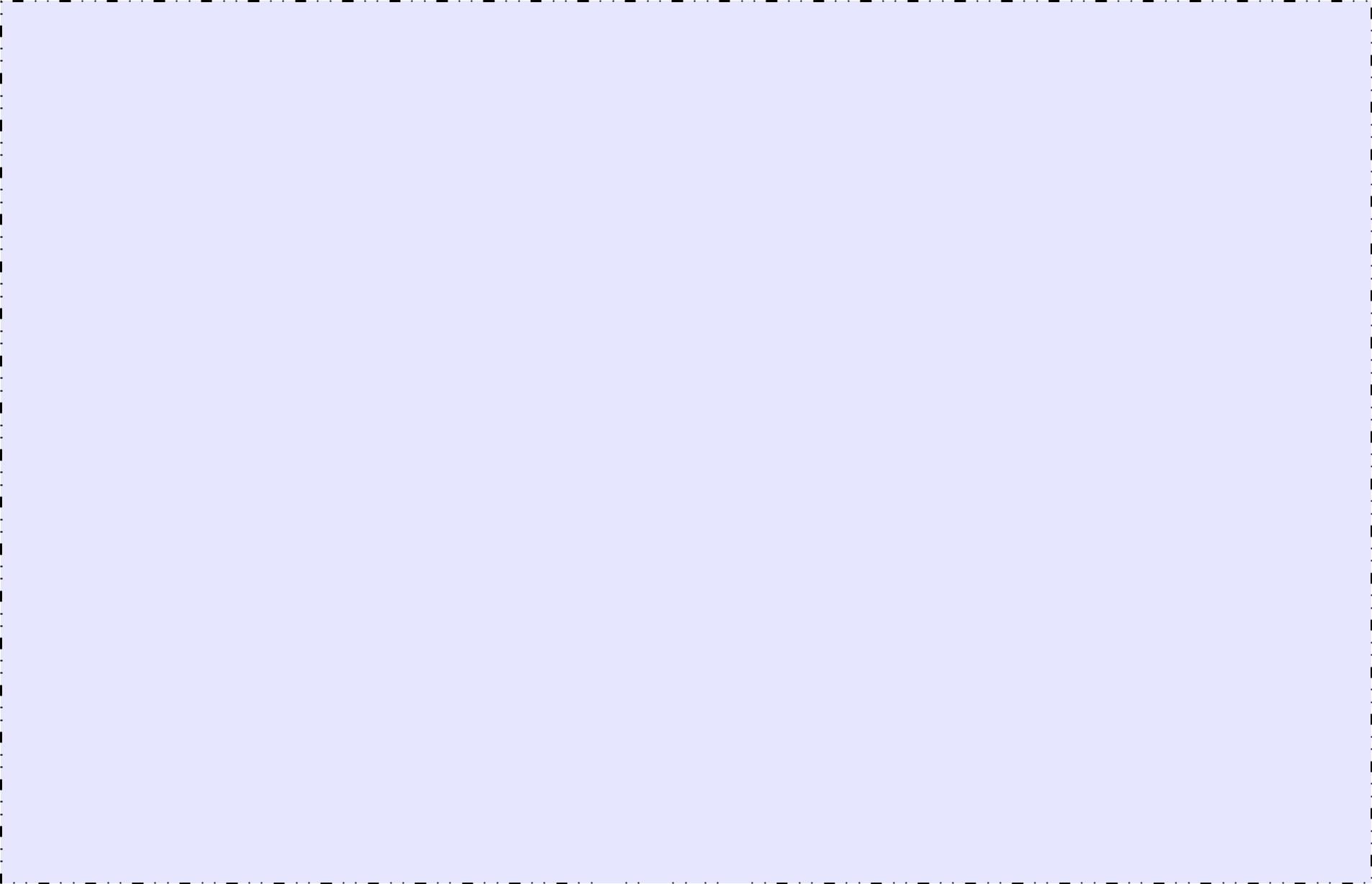
Explication

* Dans l'exemple précédent :
  + la variable sock2 sera partagée entre tous les threads de communication
* Si un nouveau client arrive et que le accept est exécuté avant que le thread précédent ait pu stocké la valeur de sock2 concernant sa communication, il y aura des problèmes
* Pour cela il suffit que le client soit lent avant de stocker la valeur de sock2 pour sa communication
* En fait le problème arrive aussi dans l'exemple précédent
* **Bonne pratique de programmation : allouez dynamiquement la mémoire pour stocker les descripteurs**
* Et bien entendu il faut faire les free correspondant.

PR - Concurrence C 29

Exemple serveur correct

**int main() {**



**int sock=socket(PF\_INET,SOCK\_STREAM,0);**

**struct sockaddr\_in sockaddress; sockaddress.sin\_family=AF\_INET; sockaddress.sin\_port=htons(4244); sockaddress.sin\_addr.s\_addr=htonl(INADDR\_ANY);**

**int r=bind(sock,(struct sockaddr \*)&sockaddress,sizeof(struct**

**sockaddr\_in));**

**if(r==0){**

**r=listen(sock,0);**

**if(r==0){**

**struct sockaddr\_in caller;**

**socklen\_t size=sizeof(caller);**

**while(1){**

**int \*sock2=(int \*)malloc(sizeof(int));**

**\*sock2=accept(sock,(struct sockaddr \*)&caller,&si); if(sock2>=0){**

**pthread\_t th;**

**pthread\_create(&th,NULL,comm,sock2);**

**}**

**}**

**}**

**}**

**return 0;**

**}**

PR - Concurrence C 30

Attention au partage des données

●

●

●

Les threads s'exécutent en parallèle

Vous ne pouvez faire aucune hypothèse sur l'ordonnanceur

Si on n'est pas attentif sur la manipulation des variables partagées, on peut observer des comportements étranges

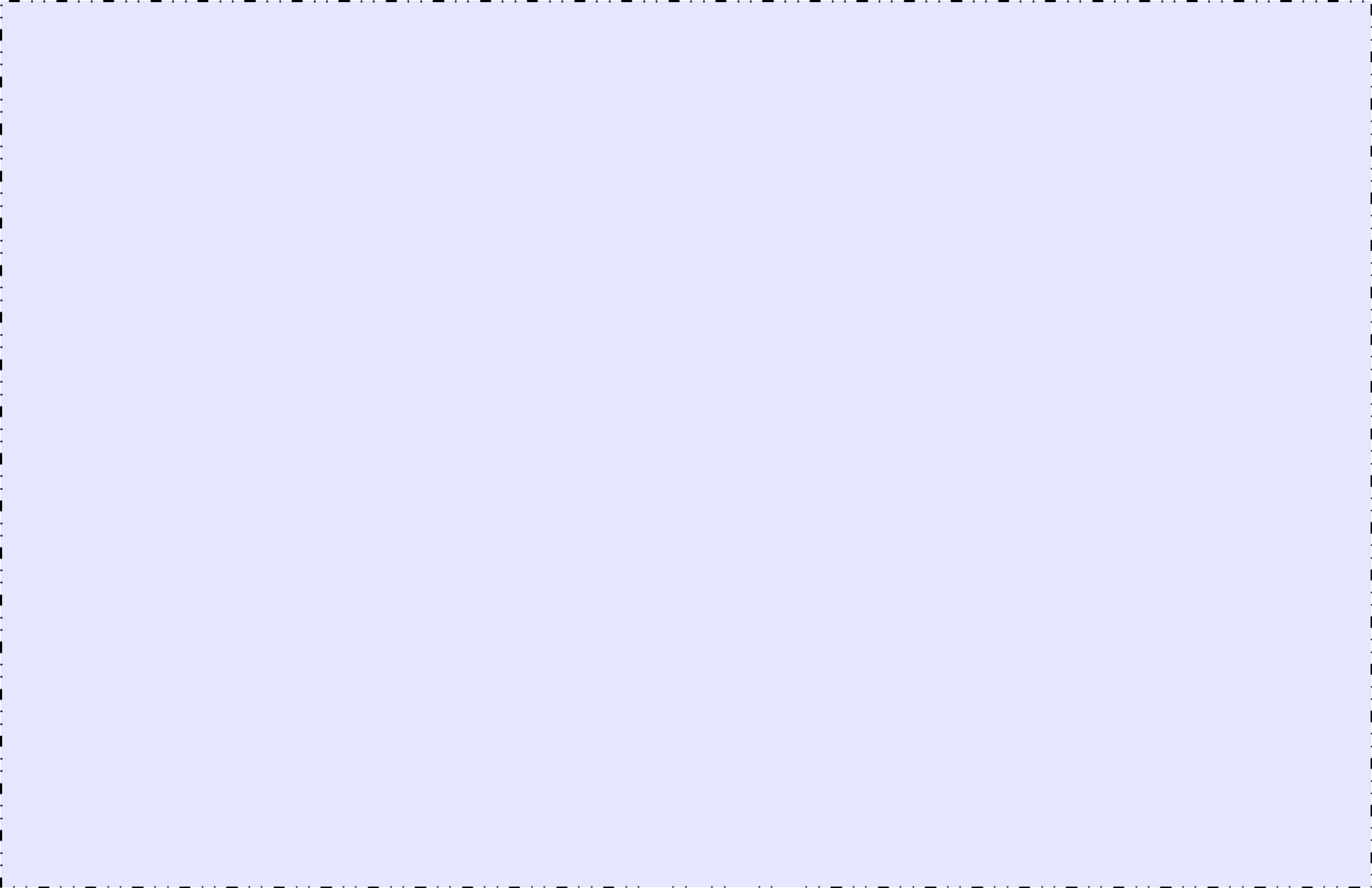
* Par exemple :
  + Un thread teste si une variable entière est positive avant de la décrémenter
  + Il est interrompu entre le test et la décrémentation
  + Un autre thread met la variable partagée à 0
  + Le premier thread met la variable à -1 ... ce qu'on ne voulait pas

PR - Concurrence C 31

Les verrous en C

* La librairie POSIX a des verrous de type :
  + **pthread\_mutex\_t verrou ;**
* La première chose à faire est initialisée le verrou
* Le plus simple consiste à faire :
  + **verrou=PTHREAD\_MUTEX\_INITIALIZER ;**
* Pour prendre le verrou, on utilise la fonction :
  + **int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);**
* Et pour libérer le verrou on utilise :
  + **int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);**
* Ces deux fonctions renvoient 0 si tout se passe bien

PR - Concurrence C 34

Exemple utilisation de verrou

**pthread\_mutex\_t verrou= PTHREAD\_MUTEX\_INITIALIZER;**

**int a=0;**

**int main(){**

**pthread\_t th1, th2,th3;**

**pthread\_create(&th1,NULL,inc,NULL);**

**pthread\_create(&th2,NULL,dec,NULL);**

**pthread\_create(&th3,NULL,dec,NULL);**

**pthread\_join(th1,NULL);**

**pthread\_join(th2,NULL);**

**pthread\_join(th3,NULL);**

**printf("a vaut %d\n",a);**

**return 0;**

**}**

**void \*inc(void \*ptr){**

**pthread\_mutex\_lock(&verrou);**

**a=a+1;**

**pthread\_mutex\_unlock(&verrou);**

**return NULL;**

**}**

**void \*dec(void \*ptr){**

**pthread\_mutex\_lock(&verrou);**

**if(a>0){**

**a=a-1;**

**}**

**pthread\_mutex\_unlock(&verrou);**

**return NULL;**

**}**

PR - Concurrence C 35