NSY103

JANVIER 2018

ABEHCERA NISSIM

Projet NSY103

Application Client / Serveur

Consultation/Réservation de spectacles

Question 1 : Processus lourds.

Tout d’abord, mon code compile et s’exécute normalement.

Nous somme en présence d’une application de type client → serveur, sur une même machine.

Pour démarrer le programme, il faut exécuter tout d’abord le fichier serveur_fork.c ou le fichier serveur_thread.c, puis le fichier client.c.

Deux lignes de commande (au minimum) sont donc nécessaires, pour tester le projet. J’ai opté pour cette solution, car de cette manière, on peut exécuter autant de clients que l’on souhaite et vérifier la concurrence d’accès, ce qui reflète beaucoup plus la réalité d’une application client → serveur.

De plus, vu que j’utilise une sémaphore dans le projet, il est primordiale de pouvoir exécuter plusieurs clients en parallèle pour vérifier que la concurrence d’accès se fait correctement.

Bien entendu, le processus serveur doit être mis hors service après la mort de tous les processus clients, a l’aide du signal SIGINT(ctrl+c) avec son handler associé, sinon les appels systèmes comme « msgsnd » et «msgrcv» se bloqueront a l’infinie en attente de messages, qui ne viendront bien

entendu jamais.

Pour une raison que j’ignore, le handler associé au signal SIGINT ne fonctionne pas correctement lorsque j’exécute le serveur avec GDB.

Dans un soucis de modularité et de concision, j’ai crée un header «header_serveur.h » qui répertorie toutes les prototypes des appels systèmes et fonctions appelés par les serveurs(fork et thread).

Avec aussi un fichier outil.c qui fait office de bibliothèque pour le projet, avec les implémentations de tous les prototypes présents dans le .h .

De plus, j’ai ajouté au projet un makefile qui facilitera l’étape de la compilation.

Le fichier client.c a l’aide d’une boucle while, affiche un menu :

→ soit pour consulter un nombre de places libres pour un spectacle donné,

→ soit pour réserver des places,

→ soit pour quitter l’application.

Une fonction est définie pour une demande de réservation ou de consultation.Ces mêmes fonctions remplissent une structure qui sera envoyée au serveur, qui traitera la demande et renverra une réponse, qui sera traitée par le client a l’aide d’une fonction d’affichage(printf).

Le fichier serveur_fork.c, dés la réception d’une requête créera un

processus fils, consultation.c ou réservation.c en fonction de la demande du client.

Dans le cas d'une réservation, le processus fils réservation (du père serveur_fork) créera a son tour un fils qui lui-même fera la réservation des places et renverra l'acquittement ou pas au client.

Quand au serveur_thread.c , lui créera une deuxième thread, qui exécutera une fonction de réservation ou de consultation en fonction de la demande du client.

Les deux fichiers serveurs ont dans leurs boucle while, l'appel système «msgrcv» pour être a l'écoute de façon permanente des demandes du client.

Le fichier client.c est composé d'une boucle while qui contient un 1er menu:

1) Consultation
2) Réservation
3) Arrêt du programme.

En fonction du choix de l'utilisateur, des requêtes sont envoyées au serveur qui les traitent et le client par la primitive msgrcv reçoit les réponses.

Communications inter-processus.

J’ai opté pour faire communiquer les processus serveur et client a l’aide d’une MSQ, car je trouve ce moyen de communiquer beaucoup plus pratique que des tubes. Car il faudrait créer un tube a chaque fois qu’il est question de faire transiter un message d’un processus a l’autre, ce qui alourdirait et complexifierait le code inutilement.

J’ai choisi de donner la même clé externe a tous les processus mis en jeu, car je me dis que le champ requête_type étant différent pour chaque communication, le risque qu’un processus intercepte un message qui ne lui est pas destiné est nul et aussi pour éviter de se retrouver avec beaucoup de clés différentes, ce qui pourrait embrouiller le code.

J’ajouterais que le champ requete_type pour le client étant son propre PID, étant unique pour chaque processus, le risque de collision est nul.

La requête_type de la fonction msgrcv, du fichier serveur_fork.c est fixée a 10 et reçoit tous les messages provenant du client. De même pour le serveur_thread.c, car les 2 serveurs ne pouvant pas fonctionner en même temps, cela ne pose donc aucun problème structurel.

Structure des messages envoyés

J’utilise pour l’échange de messages une structure composée de 7 champs qui sont :

long requete_type; //type de requête.

char nom_spectacle[TAILLE_SPECTACLE]; //nom du spectacle

unsigned int requete_id; //requete_id=1 →consultation
//requête_id=2→ réservation

int pid_client ; le pid du client.

unsigned int nbr_places_disponible; //nombre places dispos pour 1 spectacle demandé.

unsigned int nbr_places_souhaite; //nombres places souhaites

unsigned int ack; // si ack=1 → réservation réussie
si ack=0 → réservation echouée

Cette structure est utilisée dans tous les échanges fais entre processus.

Que ce soit entre le client et le processus serveur, ou entre le processus serveur et ses fils consultation et réservation.

Ce qui pourrait amener a penser que certains champs sont inutiles lors d'une transaction, ce qui est effectivement le cas, mais d'une part, toujours dans un soucis de clarté du code, se retrouver avec des structures différentes pour chaque échange, seraient lourd a gérer et compliqueraient le code inutilement.De plus, ce n'est que l'adresse de la structure qui est

passée en paramètre des fonctions, ce qui optimise grandement les traitements et revient au final moins chère en terme de mémoire ou de calculs, car moins de déclarations de structures.

Au final, sur les 7 champs que compte la structure, les 5 premiers champs sont primordiales que ce soit pour une consultation ou pour une réservation. Seuls les 2 derniers champs sont utiles uniquement pour la réservation.

De plus, dans le fichier header_serveur.h , je crée la structure avec un typedef et ainsi elle est utilisable dans chaque fichiers qui inclut le .h, ce qui élimine du code trop redondant. Mis a part du coté client, car j'ai souhaité que le client n'inclut pas le header, car ce serait un non-sens, si le client pourrait utiliser des fonctions propres a un serveur.

Segment de mémoire partagée (SHM)

J'ai choisis de placer le tableau de structures contenant les noms des spectacles et leurs nombres de places respectives dans une SHM, car j'ai voulu simuler un vrai serveur ou les données seraient stockées dans un espace qui serait accessible a tous les processus concernés comme le processus serveur et ses fils, consultation et réservation.

La clé externe est la même pour tous les processus concernés encore dans

un souci de clarté et le pointeur qui pointe sur la zone mémoire, donc la structure, est accessible aux processus qui en ont le droit.

Avant de confirmer le nombre de places libres ou avant de réserver des places pour un client donné, le programme fait un accès a la zone mémoire partagée et renvoi un résultat sous forme numérique,

disant le nombre de places libres et si c'est possible ou non de réserver des places, a l'aide de fonctions dédiées.

Cette structure placée dans la SHM a 2 champs, un pour le nom du spectacle et l'autre pour le nombre respectif de places pour chaque spectacle.

Du coté client, un simple tableau a 2 dimensions, avec les noms de chaque spectacle.

Sémaphore :

Dans le fichier outil.c , dans la fonction«update_nbr_places_libre_serveur» j'ai inséré un sémaphore avant de modifier la variable qui mémorise le nombre de places libres d'un spectacle. Cette variable étant une ressource critique, j'ai initialisé le sémaphore a un seul jeton, pour éviter lors d'une transaction entre plusieurs clients, une anomalie qui fausserait toutes les transactions futures.

Je redonne le jeton a un autre client potentiel, qui voudrait accéder a son tour a cette ressource.

Mon projet, de la manière dont il est fait, exclu l'intersection entre lecteurs et rédacteurs pour cette ressource critique, car dans le cas d'une consultation de places, j'utilise une fonction «nbr_places_dispo_spectacle» totalement différente de la fonction pour réserver.

Donc l'incohérence de la ressource qui peut apparaître ne peut se faire qu'entre rédacteurs.

Question 2

Pour la question 2, j'ai réécris le serveur qui utilise la fonction pthread_create pour créer un 2eme file d'exécution, avec 2 fonctions.

L'une pour envoyer une réponse de consultation et l'autre pour une réponse de réservation.

Ces mèmes fonctions :

- envoient une structure au client avec le nombre de places disponibles a un spectacle donné,
- ou un acquittement de réservation pour un spectacle
- ou un refus de réservation si il n' y a plus de places disponible.
-ou un refus de réservation si le nombre de places demandées est supérieur

au nombre de places disponibles.

J’ai de plus crée une structure qui sera envoyée par référence a la thread et qui contient 4 champs :

char nom_spectacle[TAILLE_SPECTACLE]; le nom du spectacle

unsigned int nbr_places_souhaite; le nombre de places souhaitées

spectacles *ptr_shm; le pointeur vers la shm

int msq_id; l’id de la msq venant d’être crée.
Int pid_client; pid du client qui envoi la requête.

Le pointeur vers la shm et l’id de la msq étant crées dans la fonction «main» et que je passe en paramètre a la thread ,me servant d’éviter de recréer ces moyens de communications a l’intérieur de la thread.

```c
                                //serveur_fork.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <errno.h>
#include <stdint.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <signal.h>
#include "header_serveur.h"

int shmid,msqId;
spectacles *ptr_mem_partagee;
void handler(int a){

    if(a==SIGINT){//si signal SIGINT recu
        shmdt(ptr_mem_partagee);//detachement du processus a la shm
        shmctl(shmid,IPC_RMID,NULL);//supprimer la shm
        msgctl(msqId,IPC_RMID,NULL);//supprimer msq
        wait(0);//attendre mort des fils
        //exit(0);//fin du processus
    }

}
int main(int argc, char **argv)
{
    signal(SIGINT,handler);//associer signal SIGINT a son handler
    ptr_mem_partagee=NULL;//pointeur type spectacles initialisé a null
    reponse_vers_client reponse_vers_client;//objet contenant réponse du serveur
    memset(&reponse_vers_client,0,sizeof reponse_vers_client);
    ptr_mem_partagee=init_shm(ptr_mem_partagee);//initialisation shm, recupere l'id
    init_tab_spectacle_shm(ptr_mem_partagee);//initialisation tableau dans shm(nom
spectacles + nombre places)
    msqId=create_msq();//creation msq
    do{

        receive_msg(msqId,&reponse_vers_client,10);//recoit messages de clients
        //declaration tableau de char qui stocke le nombre de places souhaitèes
        char nbr_places_souhaite_tab[4];
        char pid_client_tab[12];//tableau char  pid client
        int pid_client=reponse_vers_client.pid_client;
        printf("pid_client%d\n",pid_client );
        sprintf(pid_client_tab,"%d",pid_client);
        if(reponse_vers_client.requete_id==2){//id=2 =>reservation
            int nbr_places_souhaite=reponse_vers_client.nbr_places_souhaite;//
récupere dans la variable nbr_places_souhaite le contenu du champ
nbr_places_souhaite de la structure
            sprintf(nbr_places_souhaite_tab,"%d",nbr_places_souhaite);//copie
valeur nbr_places_souhaite dans tableau de char tab2
            fork_child_serveur
(reponse_vers_client.nom_spectacle,"serveur_reservation.o",nbr_places_souhaite_tab,pid_client_tab
            //fork processus reservation
        }else if (reponse_vers_client.requete_id==1){//id=1 =>consultation
            fork_child_serveur
(reponse_vers_client.nom_spectacle,"serveur_consultation.o",nbr_places_souhaite_tab,pid_client_ta
            //fork processus consultation

        }
        wait(0);//attente de la mort du fils (consultation ou reservation)
}while(1);
    return 0;
}
```

```c
                                //serveur_consultation.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include "header_serveur.h"

void consultation_nbr_places_spectacle(const char nom_spectacle[],const char
pid_client[]){

        printf("\nconsultation\n");
        reponse_vers_client reponse_vers_client;//création objet de type
reponse_vers_client
        //qui contiendra les infos utiles pour le serveur
        spectacles *ptr_mem_partagee=acceder_shm();//acceder a la zone de mémoire
partagée
        reponse_vers_client.requete_type=atol(pid_client);//requete type=pid client
        strcpy(reponse_vers_client.nom_spectacle,nom_spectacle);//placer nom
spectacle dans l'objet
        reponse_vers_client.nbr_places_disponible=nbr_places_dispo_spectacle(
        nom_spectacle,ptr_mem_partagee);//remplir champ nombre places disponibles,
qui est stocké dans la shm
        int msqId=acceder_msq();//acceder msq
        envoi_msg(msqId,&reponse_vers_client);//envoi objet au serveur
        shmdt(ptr_mem_partagee);//détacher le processus de la shm
        exit(0);//mort du fils
}

int main(int argc, char **argv){

    consultation_nbr_places_spectacle(argv[0],argv[2]);//appel fonction avec comme
parametre le nom du spectacle et pid client
    return 0;
}
```

```c
//serveur_reservation.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <errno.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include "header_serveur.h"

void
reservation_places(const char nom_spectacle[],const char nbr_places_souhaite
[],const char pid_client[]){

    printf("\nreservation\n");
    pid_t retour=fork();//creation processus fils reservation
    if(retour==0){
        reponse_vers_client reponse_serveur;//creation objet qui sera envoye au
client
        int ack=0;//variable ==1 si reservation effectuée 0 sinon
        spectacles *ptr_mem_partagee=acceder_shm();//pointeur vers la shm
        int msqId=acceder_msq();//id de la msq
        strcpy(reponse_serveur.nom_spectacle,nom_spectacle);//copie du nom de
spectacle dans l'objet
        reponse_serveur.nbr_places_souhaite=atoi(nbr_places_souhaite);//cast et
copie du nombre de places souhaitees dans l'objet
        reponse_serveur.nbr_places_disponible=update_nbr_places_libres_serveur
(nom_spectacle,ptr_mem_partagee,reponse_serveur.nbr_places_souhaite,&ack);
        //mise a jour du nombre de places libres après reservation
        reponse_serveur.requete_type=atol(pid_client);//requete type=pid client

        reponse_serveur.ack=ack;
        envoi_msg(msqId,&reponse_serveur);//envoi structure au client
        shmdt(ptr_mem_partagee);//detachement de la shm
        exit(0);//fin du processus fils
    }else{
        wait(0);
        exit(0);
    }

}

int main(int argc, char **argv)
{
    reservation_places(argv[0],argv[1],argv[2]);
    return 0;
}
```

```c
                                //serveur_thread.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <errno.h>
#include <stdint.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <signal.h>
#include <pthread.h>
#include "header_serveur.h"

spectacles *ptr_mem_partagee;
int shmid,msqId;
pthread_t thread_serveur;
void handler_thread(int a){

    if(a==SIGINT){//si signal SIGINT recu
        shmdt(ptr_mem_partagee);//detachement du processus a la shm
        shmctl(shmid,IPC_RMID,NULL);//supprimer shm
        msgctl(msqId,IPC_RMID,NULL);
        pthread_join(thread_serveur,NULL);
        exit(0);//fin du processus

    }

}
void reservation_places(parametre_fonction *parametre_fonction){

    printf("\nreservation\n");
    reponse_vers_client reponse_serveur;//objet envoyé au client
    int ack=0;//ack=1 si reservation effectuée 0 sinon
    strcpy(reponse_serveur.nom_spectacle,parametre_fonction->nom_spectacle);//
copie nom spectacle dans objet
    reponse_serveur.nbr_places_souhaite=parametre_fonction->nbr_places_souhaite;//
copie dans objet nombre de places souhaitee
    reponse_serveur.nbr_places_disponible=update_nbr_places_libres_serveur
(parametre_fonction->nom_spectacle,parametre_fonction->ptr_shm,parametre_fonction-
>nbr_places_souhaite,&ack);
    reponse_serveur.requete_type=parametre_fonction->pid_client;//requete_type=pid
client

    reponse_serveur.ack=ack;
    envoi_msg(parametre_fonction->msq_id,&reponse_serveur);//envoi objet au client
    pthread_exit(NULL);//fin de la thread
}
void consultation_nb_places_spectacle(parametre_fonction *parametre_fonction){

        printf("\nconsultation\n");
        reponse_vers_client reponse_vers_client;
        reponse_vers_client.requete_type=parametre_fonction->pid_client;//
requete_type=pid client
        strcpy(reponse_vers_client.nom_spectacle,parametre_fonction-
>nom_spectacle);
        reponse_vers_client.nbr_places_disponible=nbr_places_dispo_spectacle(
        parametre_fonction->nom_spectacle,parametre_fonction->ptr_shm);//renvoi le
nombre de places disponible
        envoi_msg(parametre_fonction->msq_id,&reponse_vers_client);//envoi objet
au client
        pthread_exit(NULL);//fin de la thread
}
void
start_pthread(pthread_t *thread, void *fonction,parametre_fonction
```

```
*parametre_fonction){
        if(pthread_create(thread,NULL,fonction,(void *)parametre_fonction)!=0){
                perror("thread create");
                exit(EXIT_FAILURE);
        }//creation de la thread, si erreur arret immediat de la fonction
}

int main(int argc, char **argv)
{
    signal(SIGINT,handler_thread);//associe un handler au signal sigint
    ptr_mem_partagee=NULL;//pointeur pour shm initialisé a null
    //pthread_t thread_serveur;//declaration thread
    reponse_vers_client requete_client;//objet renvoye au client
        parametre_fonction parametre_fonction;//objet passé en parametre de
pthread_create
    ptr_mem_partagee=init_shm(ptr_mem_partagee);//initialisation shm
    init_tab_spectacle_shm(ptr_mem_partagee);//creation liste spectacle (nom +
nombre places) dans shm
    msqId=create_msq();//id msq
    do{

        receive_msg(msqId,&requete_client,10);//recevoir messages depuis client
        parametre_fonction.ptr_shm=ptr_mem_partagee;//copie pointeur shm en
parametre thread
        parametre_fonction.msq_id=msqId;//envoyer pointeur shm et id msq a la
thread
        parametre_fonction.pid_client=requete_client.pid_client;
        if(requete_client.requete_id==2){//si requete_id==2 =>demande de
reservation
            strcpy(parametre_fonction.nom_spectacle,requete_client.nom_spectacle);

parametre_fonction.nbr_places_souhaite=requete_client.nbr_places_souhaite;
            start_pthread
(&thread_serveur,&reservation_places,&parametre_fonction);//creation thread
        }else{//sinon demande de consultation
                strcpy
(parametre_fonction.nom_spectacle,requete_client.nom_spectacle);
          //creation thread avec objet parametre_fonction comme parametre
          start_pthread
(&thread_serveur,&consultation_nb_places_spectacle,&parametre_fonction);
        }
        pthread_join(thread_serveur,NULL); //synchronisation threads
    }while(1);

        return 0;
}
```

```c
                                    //client.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <sys/wait.h>
#define CLE_MSQ 1990
#define TAILLE 5
#define TAILLE_TAB 11

typedef struct{//structure acheminant les informations (consultation/
reservation)entre client et
    //serveur
    long requete_type;
    char nom_spectacle[TAILLE_TAB];
    unsigned int requete_id;
    unsigned int pid_client;
    unsigned int nbr_places_disponible;
    unsigned int nbr_places_souhaite;
    unsigned int ack;

}requete;

//retourne 1 si nom spectacle present dans liste spectacle, 0 sinon
int choix_spectacle(char indice_spectacle[]){

    int i,indice_num;
    char indice[3];
    for(i=0; i<TAILLE;i++){
        indice_num=i+1;
        sprintf(indice,"%d",indice_num);//cast int =>char []
        if(strcmp(indice_spectacle,indice)==0)return 1;
    }
    return 0;
}
//envoi un message dans la file, au serveur, si erreur exit()
void envoi_msg(int msqId,requete *requete_vers_serveur){

    int rep=msgsnd(msqId,requete_vers_serveur,sizeof(*requete_vers_serveur),0);
    if(rep==-1){
        perror("msgsnd process consultation");
        exit(EXIT_FAILURE);
    }
}
//recupere un message provenant du serveur de la file de msg
void receive_msg(int msqId,requete *reponse_serveur){

    while(1){
        int taille=msgrcv(msqId,reponse_serveur,sizeof(*reponse_serveur),getpid
(),0);
        if(taille==-1){//si erreur
            perror("msgrcv process consultation 1");
            exit(EXIT_FAILURE);//arret immediat de la fonction
        }
        if(taille>0)break;//si reception de données, arret de la boucle
    }
}
//affiche les spectacles present dans la liste
void affiche_spectacles(char spectacles[][11]){
    int i;
    printf("\n");
    for(i=0; i<TAILLE;i++){
```

```
        if(i<TAILLE-1)
            printf("%d)%s\n",(i+1),spectacles[i]);
        else
            printf("%d)%s ",(i+1),spectacles[i]);
    }
}
//nettoie le buffer
static void purger(void){
    int c;
    while ((c = getchar()) != '\n' && c != EOF){}

}

static void clean (char *chaine)

{
    char *p = strchr(chaine, '\n');//si saut de ligne rencontre
    if(p){
        *p = 0;//supprimer saut de ligne
    }
    else{
        purger();
    }
}
void requete_consultation(char indice_spectacle[],int msqId,requete *requete){

    char nom_spectacle[11];//stocke le nom du spectacle a consulter
    requete->requete_type=10;//requete type serveur=10
    strcpy(nom_spectacle,"spectacle");//copie nom spectacle
    strcat(nom_spectacle,indice_spectacle);//rajoute numero spectacle
    strcpy(requete->nom_spectacle,nom_spectacle);//copie le nom du spectacle a
consulter
    requete->requete_id=1;//1 =>demande de consultation
    requete->pid_client=getpid();//rajoute pid client
    envoi_msg(msqId,requete);//envoi structure vers serveur
    receive_msg(msqId,requete);//retour structure avec nombre de places dispo
    printf("\nLe nombre de places disponible est de :  %u\n",requete-
>nbr_places_disponible);
    //affiche nombre de places disponible
}
void requete_reservation(char indice_spectacle[],int msqId,int nbr_places,requete
*requete){

    char nom_spectacle[11];//stocke nom spectacle
    requete->requete_type=10;//requete type serveur =10
    requete->nbr_places_souhaite=nbr_places;//affectation du nombre de places
souhaitees
    strcpy(nom_spectacle,"spectacle");//copie mot spectacle
    strcat(nom_spectacle,indice_spectacle);//rajoute numero spectacle
    strcpy(requete->nom_spectacle,nom_spectacle);//copie du nom du spectacle
concerne
    requete->requete_id=2;// id=2 =>demande reservation
    requete->pid_client=getpid();//rajoute pid client
    envoi_msg(msqId,requete);//envoi structure au serveur
    receive_msg(msqId,requete);//reponse du serveur
    if(requete->ack==1){//ack=1 =>reservation effectuée
        printf("\nReservation réussie\nIl reste %u place(s) libre(s)\n",requete-
>nbr_places_disponible);
    }
    else{//reservation rejetée
        printf("\nEchec de la réservation\nLe nombre de place(s) disponible(s)
est  :  %u\n",requete->nbr_places_disponible);
    }
}
int acceder_msq(){
    int msqId=msgget(CLE_MSQ,0);//acceder a la msq
    if(msqId==-1){//si erreur
        perror("msgget client");//affiche erreur
        exit(EXIT_FAILURE);//fin de la fonction
```

```c
    }
    return msqId;
}

int main(void)
{
    int msqId=acceder_msq();//variable stockant l'id msq
    int choix_menu=0;//variable stockant le choix utilisateur
    requete requete_vers_serveur;//objet qu'on envoi au serveur
    memset(&requete_vers_serveur,0,sizeof requete_vers_serveur);//mettre a 0 tous
les champs
    int nbr_places=0;
    char spectacles[TAILLE][11]={"spectacle1","spectacle2","spectacle3",
                            "spectacle4","spectacle5"};//liste nom spectacles
    char choix_user[3],choix_user_nbr_places[5];
    char choix_spectacle_indice[3];

    do{
        do{
        printf("\n1)Consultation\n2)Reservation\n3)Arret du programme \n");//menu
programme
        fgets(choix_user,3,stdin);//saisie 1 ou 2 ou 3
        clean(choix_user);//nettoyer le buffer
        }while(strcmp(choix_user,"1")!=0 && strcmp(choix_user,"2")!=0 && strcmp
(choix_user,"3")!=0);//boucle tant que le choix utilisateur different de 1 ou 2 ou
3
        choix_menu=atoi(choix_user);//conversion char =>int du choix utilisateur
        if(choix_menu==3){//si arret du programme

            printf("\nAu revoir\n");
            break;//sortie de boucle
        }
        else if(choix_menu==1){//si consultation souhaitée
            do{
                printf("\nQuel spectacle voulez-vous consulter ? : \n");
                affiche_spectacles(spectacles);//affiche liste spectacles
                fgets(choix_spectacle_indice,3,stdin);//saisie du numero de
spectacle
                clean(choix_spectacle_indice);//nettoie le buffer
            }while(choix_spectacle(choix_spectacle_indice)!=1);
            requete_consultation
(choix_spectacle_indice,msqId,&requete_vers_serveur);
        }
        else if(choix_menu==2){//si reservation souhaitée
            do{
                printf("\nPour quel spectacle voulez-vous reserver des places ? :
\n");
                affiche_spectacles(spectacles);//affiche liste spectacles
                fgets(choix_spectacle_indice,3,stdin);//saisie numero spectacle
                clean(choix_spectacle_indice);//nettoie le buffer
            }while(choix_spectacle(choix_spectacle_indice)!=1);
            do{
                printf("\nCombien de places voulez-vous reserver pour le spectacle%
s : ",choix_spectacle_indice);
                fgets(choix_user_nbr_places,5,stdin);//saisie nombre places
spectacle
                clean(choix_user_nbr_places);//nettoie le buffer
                nbr_places=atoi(choix_user_nbr_places);//cast char =>int
            }while(nbr_places==0);
            requete_reservation
(choix_spectacle_indice,msqId,nbr_places,&requete_vers_serveur);
        }

    }while(choix_menu!=3);//tant que l'utilisateur ne tape pas 3
    return 0;
}
```

```c
                                    //outil.c
#include "header_serveur.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <pthread.h>
#include <signal.h>
#include <sys/sem.h>
#define TAILLE_TAB 11
#define CLE_MSQ 1990
#define CLE_SHM 1975
#define NBR_SPECTACLES_TAB_SHM 5
int shmid,msqId;
spectacles *ptr_mem_partagee=NULL;

/*fonction qui initialise la structure spectacle,
avec le nom et le nombre de chaque spectacles
et insertion de cette structure dans la mémoire partagée*/
void init_tab_spectacle_shm(spectacles *spectacle){
    int i,j=0;
    char num_spectacle[2];//tableau contenant numero spectacle
    for(i=0; i<NBR_SPECTACLES_TAB_SHM;i++){
        sprintf(num_spectacle,"%d",(++j));//cast int => tableau de char
        //copie mot "spectacle" dans structure
        strcpy(spectacle[i].nom_spectacle,"spectacle");
        //concatenation "spectacle" avec numero//copie mot "spectacle" dans
structure
        strcat(spectacle[i].nom_spectacle,num_spectacle);
        spectacle[i].nbr_places_dispo=100*(i+1);//stocke nombre places spectacle

    }
}

void fork_child_serveur(const char nom_spectacle_souhaite[],const char
*path_to_executable,const char nbr_places_souhaite[],const char pid_client[]){
    pid_t valeur=fork();//fork
        if(valeur==-1){//si erreur
            perror("fork process serveur");//affichage erreur
            exit(EXIT_FAILURE);//arret immediat de la fonction
        }
        if(valeur==0){//si coté processus fils
            execl
(path_to_executable,nom_spectacle_souhaite,nbr_places_souhaite,pid_client,NULL);
            /*recouvrement avec chemin vers executable,
            plus comme parametres: nom spectacle
            et nombre de places souhaites et pid du client*/

        }
}

spectacles *init_shm(spectacles *ptr_mem_partagee){//creation d'une shm
    shmid=shmget((key_t)CLE_SHM,sizeof(spectacle),0750|IPC_CREAT|IPC_EXCL);
    if(shmid==-1){//si erreur
        perror("shmid");//affichage erreur
        exit(EXIT_FAILURE);//arret immediat
    }
    //attachement de la shm
    if((ptr_mem_partagee=shmat(shmid,NULL,0))==(spectacles *)-1){
        perror("shmat");
        exit(EXIT_FAILURE);
```

```
    }
    return ptr_mem_partagee;
}
int acceder_msq(){//acceder file message existante, grace a une cle externe
    int msqId=msgget(CLE_MSQ,0);
    if(msqId==-1){
        perror("msgget process serveur");
        exit(EXIT_FAILURE);
    }
    return msqId;
}
int create_msq(){//creation msq a l'aide de la clé externe
    int msqId=msgget(CLE_MSQ,IPC_CREAT|IPC_EXCL|0666);
    if(msqId==-1){
        perror("msqid process client");
        exit(EXIT_FAILURE);
    }
    return msqId;
}
void
receive_msg(int msqId,reponse_vers_client *requete_client,long requete_type){
    while(1){
        //fonction qui permet de recevoir des messages a partir d'une file
        int taille=msgrcv(msqId,requete_client,sizeof
(*requete_client),requete_type,0);
        if(taille==-1){//si erreur
            perror("msgrcv process serveur");//affichage erreur
            exit(EXIT_FAILURE);//arret fonction
        }
        if(taille>0)break;//si données recues ,quitte la boucle
    }
}

spectacles *acceder_shm(){
    int shmid=shmget((key_t)CLE_SHM,0,0);//acceder a une shm deja cree
    spectacles *ptr_mem_partagee;//pointeur de type spectacles
    if((ptr_mem_partagee=shmat(shmid,NULL,0))==(spectacles *)-1){
      /*attachement du pointeur
      ptr_mem_partagee a la shm, si erreur fin de la fonction*/
        perror("shmat serveur consultation");
        exit(EXIT_FAILURE);
    }

    return ptr_mem_partagee;
}

int
update_nbr_places_libres_serveur(const char nom_spectacle[],spectacles *ptr,const
int nbr_places_souhaite,int *ack){
    int i;
    for(i=0; i<NBR_SPECTACLES_TAB_SHM;i++){
      //si nom_spectacle est dans la liste de la shm
        if(strcmp(nom_spectacle,ptr[i].nom_spectacle)==0){
          /*si nombre de places disponible est superieur ou egale
           au nombre de places souhaitees*/
            if(ptr[i].nbr_places_dispo>=nbr_places_souhaite){
                *ack=1;//si reservation effectuée => ack=1 sinon ack=0
                //semaphore, avec cle externe 15
                int sem_id=semget(15,1,IPC_CREAT|IPC_EXCL|0600);
                struct sembuf operation;
                semctl(sem_id,0,SETVAL,1);
                operation.sem_num=0;
                operation.sem_op=-1;//opration p
                operation.sem_flg=0;//flag=0
                semop(sem_id,&operation,1);//verrouillage ressource critique a 1
jeton
                //le nombre de places dispo apres reservation
                ptr[i].nbr_places_dispo-=nbr_places_souhaite;
                operation.sem_num=0;
```

```
                operation.sem_op=1;//operation v
                operation.sem_flg=0;//flag=0
                semop(sem_id,&operation,1);//liberation jetton
                semctl(sem_id,0,IPC_RMID,0);//destruction semaphore
                return ptr[i].nbr_places_dispo;
            }else{
                *ack=0;//reservation non effectuée
                return ptr[i].nbr_places_dispo;//retourne nombre places dispo
            }
        }
    }
    return -1;
}

void envoi_msg(int msqId,reponse_vers_client *reponse_vers_client){
    //fonction servant a envoyer des messages dans la file de messages
    int rep=msgsnd(msqId,reponse_vers_client,sizeof(*reponse_vers_client),0);
    if(rep==-1){//si erreur
        perror("msgsnd process consultation");//affichage erreur
        exit(EXIT_FAILURE);//fin fonction
    }
}

unsigned int
nbr_places_dispo_spectacle(const char nom_spectacle[],const spectacles *ptr){
    int i;
    for(i=0; i<NBR_SPECTACLES_TAB_SHM;i++){
      //si "nom_spectacle" est dans la shm
        if(strcmp(nom_spectacle,ptr[i].nom_spectacle)==0)
            return ptr[i].nbr_places_dispo;//retourne nombre places dispo
    }
    return 0;
}
```

```c
                                //header_serveur.h
#ifndef HEADER_SERVEUR_H_INCLUDED
#define HEADER_SERVEUR_H_INCLUDED
#define TAILLE_SPECTACLE 11
#define TAILLE 5

typedef struct{//structure dans shm
    char nom_spectacle[TAILLE_SPECTACLE];
    unsigned int nbr_places_dispo;
}spectacles;

typedef struct{//structure entre client et serveur
    long requete_type;
    char nom_spectacle[TAILLE_SPECTACLE];
    unsigned int requete_id;
    unsigned int pid_client;
    unsigned int nbr_places_disponible;
    unsigned int nbr_places_souhaite;
    unsigned int ack;

}reponse_vers_client;

typedef struct {//structure en parametre de la thread

    char nom_spectacle[TAILLE_SPECTACLE];
          unsigned int nbr_places_souhaite;
    spectacles *ptr_shm;
    int msq_id;
    int pid_client;
}parametre_fonction;

extern spectacles spectacle[TAILLE];
spectacles *acceder_shm();
int acceder_msq();
void envoi_msg(int msqId,reponse_vers_client *reponse_serveur);
spectacles *init_shm(spectacles *ptr_mem_partagee);
void init_tab_spectacle_shm(spectacles *spectacle);
int update_nbr_places_libres_serveur(const char nom_spectacle[],spectacles
*ptr,const int nbr_places_souhaite,int *ack);
//void handler(int a);
void fork_child_serveur(const char nom_spectacle_souhaite[],const char
*path_to_executable,const char nbr_places_souhaite[],const char pid_client[]);
unsigned int nbr_places_dispo_spectacle(const char nom_spectacle[],const
spectacles *ptr);
int create_msq();
void receive_msg(int msqId,reponse_vers_client *reponse_serveur,long type_requete);
void handler_thread(int a);
#endif // HEADER_SERVEUR_H_INCLUDED
```