

ANNEXES
---------

**Clientjeu.c**

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee  */
/*  Richeton Nicolas et Romelard Fabrice                */
/*  FAB-NICO BATAILLE NAVALE                           */
/*-----*/

#include <signal.h>
#include "socket.h"
#include <netdb.h>

main( int argc, char * argv[])
{
    int port;
    int socket_client;
    struct sockaddr_in adresse_client;
    struct sockaddr_in adresse_serveur;

    struct hostent *hp;

    if ( argc != 3 )
    {
        fprintf( stderr, "FAB-NICO BATAILLE NAVALE\n");
        fprintf( stderr, "Syntaxe de la connexion : %s 'machine' 'port' (Exemple : %s localhost 6666) \n", argv[ 0 ], argv[ 0 ] );
        return( 1 );
    }

    hp=gethostbyname( argv[ 1 ] );
    if ( hp == NULL )
    {
        fprintf( stderr, "FAB-NICO BATAILLE NAVALE\n");
        fprintf( stderr, "La machine %s est inconnue\n", argv[ 1 ] );
        return( 1 );
    }

    port = 0;

    socket_client=creation_socket( SOCK_STREAM, &port, &adresse_client );
    if ( socket_client == -1 )
    {
        fprintf( stderr, "FAB-NICO BATAILLE NAVALE\n");
        fprintf( stderr, "Création du socket client impossible\n");
        return( 1 );
    }

    printf( "Client sur le port %d\n", ntohs( adresse_client.sin_port ) );

    adresse_serveur.sin_family = AF_INET;
    adresse_serveur.sin_port = htons( atoi( argv[ 2 ] ) );

    memcpy( &adresse_serveur.sin_addr.s_addr, hp->h_addr, hp->h_length );

    if ( connect( socket_client, &adresse_serveur, sizeof( adresse_serveur ) ) == -1 )
    {
        perror( "Erreur de connexion" );
        return( 1 );
    }

    printf("FAB-NICO BATAILLE NAVALE\n");
    printf( "La connexion est acceptée\n" );
    Client( socket_client );

    return( 0 );
}
```

**Comclient.c**

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                    */
/*  FAB-NICO BATAILLE NAVALE                               */
/*-----*/

#include "protoserv.h"
#include <stdio.h>
#include "partie.h"
#include <errno.h>
#include <string.h>

int socket_client;
char nom[ 80 ];
extern Map_t * map;
int Client( int socket )
{
    char password[ 80 ];
    char choice[ 80 ];
    msg_ident_t ident;
    msg_rep_t rep;
    socket_client = socket;
    printf( "Votre Login : " );
    fgets( nom, 80, stdin );
    printf( "Mot de Passe : " );
    fgets( password, 80, stdin );
    if ( EnvoieldMsg( MSG_IDENT ) != 0 )
        return( 1 );
    nom[ strlen( nom ) - 1 ] = '\0';
    strcpy( ident.nom, nom );
    ident.nom[ strlen( nom ) ] = '\0';
    strcpy( ident.password, password );
    ident.password[ strlen( password ) - 1 ] = '\0';
    if ( send( socket_client, &ident, sizeof( msg_ident_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission msg_identification\n" );
        return( 1 );
    }
    if ( receptionRep( &rep ) != 0 )
        return( 1 );
    if ( rep.status != '0' )
    {
        switch( rep.status )
        {
            case '1' :
                printf( "Login inconnu\n" );
                break;
            case '2' :
                printf( "Mauvais password\n" );
                break;
            case '3' :
                printf( "Déjà loggé\n" );
                break;
            case '4' :
                printf( "Pb technique\n" );
                break;
            default :
                printf( "Problème inconnu\n" );
        }
        return( 1 );
    }
    if ( strcmp( ident.nom, "administrateur" ) == 0 )
        gestionihmAdmin();
    else
        gestionihmClient();
    return( 0 );
}

int receptionRep( msg_rep_t * rep )
{
    if ( recv( socket_client, rep, sizeof( msg_rep_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission msg_rep\n" );
        return( 1 );
    }
    return( 0 );
}
```

```
}

int EnvoieIdMsg( int idMsg )
{
    if ( send( socket_client, &idMsg, sizeof( int ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission idMsg\n" );
        return( 1 );
    }
    return( 0 );
}

int EnvoiePositionJeu( int x, int y )
{
    msg_rep_t rep;
    msg_play_t m;
    m.x = x;
    m.y = y;
    if ( EnvoieIdMsg( MSG_PLAY ) != 0 )
        return( 5 );
    if ( send( socket_client, &m, sizeof( msg_play_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission msg_play\n" );
        return( 5 );
    }
    if ( receptionRep( &rep ) != 0 )
        return( 5 );
    return( rep.status - '0' );
}

int EnvoieCreationPartie( int x, int y, int nbJoueurs, int nbBateaux )
{
    msg_create_t create;
    msg_rep_t rep;
    create.x = x;
    create.y = y;
    create.nbJoueurs = nbJoueurs;
    create.nbBateaux = nbBateaux;

    if ( EnvoieIdMsg( MSG_CREATE ) != 0 )
        return( 5 );
    if ( send( socket_client, &create, sizeof( msg_create_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission msg_create\n" );
        return( 5 );
    }
    if ( receptionRep( &rep ) != 0 )
        return( 3 );
    return( rep.status - '0' );
}

int EnvoieDebutPartie()
{
    if ( EnvoieIdMsg( MSG_DEBUT ) != 0 )
        return( 1 );
    return( 0 );
}

int EnvoieAjoutJoueur( char * nom, char * password )
{
    char * filename = "listejoueurs.cfg";
    FILE * io;
    io = fopen( filename, "a+" );

    if ( io == NULL )
    {
        printf( "Erreur d'ouverture du fichier %s\n", filename );
        return( -1 );
    }

    printf( "\nJoueur Ajouté : %s | Password : %s\n", nom, password );
    fprintf( io, "%s;%s\n", nom, password );
    fclose( io ); return( 0 );
}

int EnvoieSupJoueur( char * nom )
{
    char * filename = "listejoueurs.cfg";
    FILE * io;
    char buffer[ 80 ];
    char pointVirgule = ',';
    char listeUsers[20][80];
    int i, j;
```

```
int NbUsers;
int position, point;
char * testlecture;

io = fopen( filename, "rt" );
if ( io == NULL )
{
    printf( "Erreur d'ouverture du fichier %s\n", filename );
    return( -1 );
}

NbUsers = 0;
i = 0;
while( fgets( buffer, 80, io ) != NULL )
{
    strcpy( listeUsers[i], buffer );
    NbUsers++;
    i++;
}
fclose(io);
for (i=0;i<80;i++)
point = -1; for (j=0;j<NbUsers;j++)
{
    position = 0;
    for (i=0;i<strlen(nom);i++)
    {
        if (nom[i]==listeUsers[j][i])
            position ++;
    }
    if (position == strlen(nom))
    {
        if (listeUsers[j][position]==pointVirgule)
        {
            point = j;
        }
    }
    if ( point == -1)
    {
        return (1);
    }
}
else
{
    io = fopen( filename, "w" );
    if ( io == NULL )
    {
        printf( "Erreur d'ouverture du fichier %s\n", filename );
        return( -1 );
    }
    for (j=0;j<NbUsers;j++)
    {
        if (j != point)
        {
            fprintf (io, "%s", listeUsers[j]);
        }
    }
    printf ("n Joueur supprime dans le fichier : %s\n", nom);
    fclose(io);
    return( 0 );
}

}

int EnvoieBateau( int x, int y, int lg, char * dir, int id )
{
    msg_bateau_t bateau;
    msg_rep_t rep;
    bateau.x = x;
    bateau.y = y;
    bateau.lg = lg;
    bateau.id = id;
    strcpy( bateau.dir, dir );
    if ( EnvoieIdMsg( MSG_BATEAU ) != 0 )
        return( 3 );
    if ( send( socket_client, &bateau, sizeof( msg_bateau_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission msg_bateau\n" );
        return( 1 );
    }
    if ( receptionRep( &rep ) != 0 )
        return( 3 );
    return( rep.status - '0' );
}

int EnvoieFinPartie()
{
    return( EnvoieIdMsg( MSG_END ) );
}
```

```
int EnvoieDecnx()
{
    return( EnvoieIdMsg( MSG_DECNX ) );
}

int EnvoieInscription()
{
    msg_rep_t rep;
    if ( EnvoieIdMsg( MSG_INSCRIPTION ) != 0 )
        return( 6 );
    if ( receptionRep( &rep ) != 0 )
        return( 6 );
    return( rep.status - '0' );
}

int AttenteInitPartie()
{
    int idMsg;
    printf( "Attente Recv INIT\n" );
    if ( recv( socket_client, &idMsg, sizeof( int ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur réception INIT\n" );
        return( -1 );
    }
    if ( idMsg != MSG_INIT )
        return( idMsg );
    return( DemandeMap() );
}

int DemandeMap()
{
    if ( EnvoieIdMsg( MSG_GETMAP ) != 0 )
        return( 1 );
    return( receptionMap() );
}

int receptionMap()
{
    msg_map_t m;
    if ( recv( socket_client, &m, sizeof( msg_map_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur réception msg_sizemap\n" );
        return( 1 );
    }
    if ( m.x == 0 )
        return( 2 );
    map = allocationMap( m.x, m.y );
    if ( map == NULL )
    {
        printf( "Erreur malloc map = %d\n", errno );
        return( 3 );
    }

    map->sizeX = m.x;
    map->sizeY = m.y;

    /* Si admin alors on recoit le map complet */

    if ( strcmp( nom, "administrateur" ) == 0 )
    {
        if ( recv( socket_client, map->data, map->sizeX * map->sizeY, 0 ) == -1 )
        {
            fprintf( stderr, "Erreur réception map\n" );
            return( 4 );
        }
    }
    else
        initialisationMap( map );
    return( 0 );
}

int AttendDebutPartie()
{
    int idMsg;
    if ( recv( socket_client, &idMsg, sizeof( int ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur réception go_play\n" );
        return( -1 );
    }
    if ( idMsg == MSG_GO_PLAY )
        return( 0 );
    if ( idMsg == MSG_HIT )
    {
        majMap();
        return( 1 );
    }
}
```

```
    }
    if ( idMsg == MSG_END )
        return( 2 );
    return( -1 );
}

int majMap()
{
    msg_hit_t hit;
    Pos_t origine;
    Pos_t * listePos;
    if ( recv( socket_client, &hit, sizeof( msg_hit_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur réception msg_hit\n" );
        return( -1 );
    }
    origine.x = hit.x;
    origine.y = hit.y;
    listePos = CalculPosition( &origine, hit.dir, hit.lg );

    ModifieListePositionMap( map, listePos, hit.lg, hit.valeur );
    free( listePos );
    printf( "Le joueur %s a touché...\n", hit.nom );
    return( 0 );
}
```

### Compile.sh

```
#!/bin/sh
##-----*/
##      Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
##      Richeton Nicolas et Romelard Fabrice                  */
##      FAB-NICO BATAILLE NAVALE                              */
##-----*/

cc -o servjeu socket.c servjeu.c comserv.c gestpart.c partie.c joueurs.c connectjoueurs.c
cc -o clientjeu socket.c clientjeu.c comclient.c ihm.c partie.c
chmod 755 servjeu
chmod 755 clientjeu
chmod 666 listejoueurs.cfg
```

### Comserv.c

```
/*-----*/
/*      Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*      Richeton Nicolas et Romelard Fabrice                  */
/*      FAB-NICO BATAILLE NAVALE                              */
/*-----*/

#include "protoserv.h"
#include "gestpart.h"
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <sys/socket.h>
#include <sys/types.h>

int socket_service;
char nom[ 30 ];

int LanceRep( char res );
extern Map_t * map;
extern Com_t * com;
extern ListeConnectUsers_t * listeJoueurs;

void handlersigusr1( int sig )
{
    switch ( com->typeFlux )
    {
        case 1 :
            LanceLeJeu();
            break;

        case 2 :
```

## ANNEXES BATAILLE NAVALE PROJET UNIX

```
        EnvoieHit();
        break;
    case 4 :
        EnvoieldMsg( MSG_INIT );
        break;
    case 5 :
        EnvoieldMsg( MSG_END );
        break;
    default :
        printf( "Type de flux inconnu\n" );
    }
}

int CreationFils( int socket )
{
    int idMsg;
    int flagExit = 0;
    msg_ident_t ident;
    msg_bateau_t msgbateau;
    msg_create_t create;
    msg_play_t tour;
    int nbOctets;
    int res;
    socket_service = socket;

    while( flagExit == 0 )
    {
        signal( SIGUSR1, handlersigusr1 );

        printf( "----- Attente PID=%d USER=%s -----\\n", getpid(), nom );
        nbOctets = recv( socket_service, &idMsg, sizeof( int ), MSG_WAITALL );

        printf( "----- PID=%d USER=%s -----\\n", getpid(), nom );

        if ( nbOctets == 0 )
        {
            fprintf( stderr, "Fils Erreur de réception idMsg\\n" );
            return( 1 );
        }

        switch( idMsg )
        {
            case MSG_DATAS:
                ImprimeDatas();
                break;
            case MSG_IDENT:
                nbOctets = recv( socket_service, &ident, sizeof( msg_ident_t ), 0 );
                if ( nbOctets == 0 )
                {
                    fprintf( stderr, "Erreur de réception de l'identification\\n" );
                    return( 2 );
                }
                else
                {
                    printf( "Identification : %s/%s\\n", ident.nom, ident.password );
                    strcpy( nom, ident.nom );

                    res = identificationJoueur( &ident );
                    if ( LanceRep( res + '0' ) != 0 )
                    {
                        return( -1 );
                    }
                    break;
                }
            case MSG_DEBUT:
                diffuseSignalTousJoueurs( 4 );
                com->joueurCourant=0;
                com->typeFlux = 1;
                printf( "Envoi du kill vers %d\\n", listeJoueurs->liste[ com->joueurCourant ].pid );
                if ( kill( listeJoueurs->liste[ com->joueurCourant ].pid, SIGUSR1 ) == -1 )
                {
                    perror( "Erreur envoi du kill" );
                    return( 1 );
                }
                break;
            case MSG_PLAY:

```

## ANNEXES BATAILLE NAVALE PROJET UNIX

```
        nbOctets = recv( socket_service, &tour, sizeof(msg_play_t), 0 );
        if ( nbOctets == 0 )
        {
            fprintf( stderr, "Erreur dans la réception du tour de jeu\n" );
            return( 2 );
        }
        res = JoueJeu( &tour );
        if ( LanceRep( res + '0' ) != 0 )
        {
            return( -1 );
        }
        if ( res == 2 || res == 3 || res == 4 )
        {
            strcpy( com->hit.nom, nom );
            diffuseSignalTousJoueurs( 2 );
        }

        if ( res == 4 )
        {
            /* Fin de partie */
            diffuseSignalTousJoueurs( 5 );
            FindePartie();
        }
        else
        {
            /* Demande au joueur suivant de joueur */
            printf( "Envoi du kill vers %d\n", listeJoueurs->liste[ com->joueurCourant ].pid );
            com->typeFlux = 1;
            kill( listeJoueurs->liste[ com->joueurCourant ].pid, SIGUSR1 );
        }
        break;
case MSG_CREATE:
    nbOctets = recv( socket_service, &create, sizeof( msg_create_t ), 0 );
    if ( nbOctets == 0 )
    {
        fprintf( stderr, "Erreur dans la réception du create\n" );
        return( 2 );
    }
    res = creationPartie( &create );
    if ( LanceRep( res + '0' ) != 0 )
    {
        return( -1 );
    }
    break;
case MSG_GETMAP:
    if ( strcmp( nom, "administrateur" ) == 0 )
        EnvoieMap();
    else
        EnvoieMapVide();
    break;
case MSG_INSCRIPTION:
    res = demandeInscription();
    if ( LanceRep( res + '0' ) != 0 )
    {
        return( -1 );
    }
    break;
case MSG_DIFFUSE_MAP:
    EnvoieMap();
    break;
case MSG_END:
    diffuseSignalTousJoueurs( 5 );
    FindePartie();
    break;
case MSG_DECNX:
    deconnexion( nom );
    flagExit = 1;
    break;
case MSG_BATEAU:
    nbOctets = recv( socket_service, &msgbateau, sizeof(msg_bateau_t), 0 );
    if ( nbOctets == 0 )
    {
        fprintf( stderr, "Erreur de réception du bateau\n" );
        return( 2 );
    }
}
```

## ANNEXES BATAILLE NAVALE PROJET UNIX

```
        }
        res = ajoutBateau( &msgbateau );
        if ( LanceRep( res + '0' ) != 0 )
        {
            return( -1 );
        }
        break;
default:
    printf( "Identification de message inconnue : %d\n", idMsg );
    return( 3 );
    }
}
return( 0 );
}

int LanceRep( char res )
{
    msg_rep_t rep;

    rep.status = res;
    if ( send( socket_service, &rep, sizeof( msg_rep_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission msg_rep\n" );
        return( 1 );
    }
    return( 0 );
}

int diffuseSignalTousJoueurs( int typeflux )
{
    int res;

    com->typeFlux = typeflux;

    for( res = 0; res < listeJoueurs->nbUsed ; res ++ )
    {
        printf( "Envoi du kill USR1, flux=%d vers %d\n",
                com->typeFlux, listeJoueurs->liste[ res ].pid );
        if ( kill( listeJoueurs->liste[ res ].pid, SIGUSR1 ) == -1 )
        {
            perror( "Erreur envoi du kill" );
            return( 1 );
        }
    }
    sleep( 1 );
    return( 0 );
}

/* Envoie la map vide (pour la taille) */
int EnvoieMapVide()
{
    msg_map_t m;
    if ( com->partie == 0 )
    {
        m.x = 0;
        m.y = 0;
    }
    else
    {
        m.x = map->sizeX;
        m.y = map->sizeY;
    }

    if ( send( socket_service, &m, sizeof( msg_map_t ), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission du msg_map\n" );
        return( 1 );
    }
    return( 0 );
}

/* Envoie map */
int EnvoieMap()
{

```

```
        if ( com->partie == 0 )
        {
            return( EnvoieMapVide() );
        }
        else
        {
            if ( send( socket_service, map, sizeof(int)*2+map->sizeX*map->sizeY, 0 ) == -1 )
            {
                fprintf( stderr, "Erreur émission de la map\n" );
                return( 1 );
            }
        }
        return( 0 );
    }

int EnvoieIdMsg( int idMsg )
{
    if ( send( socket_service, &idMsg, sizeof(int), 0 ) == -1 )
    {
        perror( "Erreur émission idMsg" );
        return( 1 );
    }
    return( 0 );
}

/* Lance le Jeu */
int LanceLeJeu()
{
    if ( EnvoieIdMsg( MSG_GO_PLAY ) != 0 )
        return( 1 );

    com->joueurCourant++;
    if ( com->joueurCourant >= listeJoueurs->nbUsed )
        com->joueurCourant = 0;

    return( 0 );
}

int EnvoieHit()
{
    if ( EnvoieIdMsg( MSG_HIT ) != 0 )
        return( 1 );

    if ( send( socket_service, &(com->hit), sizeof(msg_hit_t), 0 ) == -1 )
    {
        fprintf( stderr, "Erreur émission hit\n" );
        return( 1 );
    }
    return( 0 );
}
```

### Connectjoueurs.c

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                    */
/*  FAB-NICO BATAILLE NAVALE                                */
/*-----*/

#include "connectjoueurs.h"

/* Recherche d'un Joueur dans la liste */
/* return indice du Joueur ou -1 */

int ChercheConnectionUser( char * name, ListeConnectUsers_t * liste )
{
    int i;
    for ( i=0; i < liste->nbUsed && strcmp( name, liste->liste[ i ].nom ); i++ );
    if ( i < liste->nbUsed )
        return( i );
    else
        return( -1 );
}
```

```
/* Affichage d'une liste de Joueurs */

void AfficheListeConnect( ListeConnectUsers_t * liste )
{
    int i;
    printf( "NbMax : %d NbUsed = %d\n",
            liste->nbMax,
            liste->nbUsed );
    for ( i=0; i < liste->nbUsed ; i++ )
    {
        printf( "Nom : %-20s\tPid : %d\tSocket : %d\n",
                liste->liste[ i ].nom,
                liste->liste[ i ].pid,
                liste->liste[ i ].socket );
    }
}

/* Ajout d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=deja enregistré */

int AjouterConnectUser( char * name, int pid, int socket, ListeConnectUsers_t * liste )
{
    if ( ChercheConnectionUser( name, liste ) != -1 )
        return( -2 );

    strcpy( liste->liste[ liste->nbUsed ].nom, name );
    liste->liste[ liste->nbUsed ].pid = pid;
    liste->liste[ liste->nbUsed ].socket = socket;
    liste->nbUsed++;

    return( liste->nbUsed );
}

/* Suppression d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=inconnu */

int SupprimeConnectUser( char * name, ListeConnectUsers_t * liste )
{
    int i;

    i=ChercheConnectionUser( name, liste );
    if ( i == -1 )
        return( -2 );
    i++;
    for( ; i < liste->nbUsed; i++ )
    {
        strcpy( liste->liste[ i-1 ].nom, liste->liste[ i ].nom );
        liste->liste[ i-1 ].pid = liste->liste[ i ].pid;
    }
    liste->nbUsed--;

    return( liste->nbUsed );
}
```

### **Connectjoueurs.h**

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                    */
/*  FAB-NICO BATAILLE NAVALLE                               */
/*-----*/

#ifndef __CONNECTUSERS__
#define __CONNECTUSERS__

#include <stdio.h>
#include <string.h>

/* Structure d'un Joueur */

typedef struct connectusers
{
    char nom[ 20 ];
```

```
        int pid;
        int socket;
    } ConnectUser_t;

/* Structure de la liste des Joueurs */

typedef struct listeconnectusers
{
    int nbMax;
    int nbUsed;
    ConnectUser_t liste[ 1 ];
} ListeConnectUsers_t;

/* Recherche d'un Joueur dans la liste */
/* return indice du Joueur ou -1 */

int ChercheConnectionUser( char * name, ListeConnectUsers_t * liste );

/* Affichage d'une liste de users */

void AfficheListeConnect( ListeConnectUsers_t * liste );

/* Ajout d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=deja enregistré */

int AjouterConnectUser( char * name, int pid, int socket, ListeConnectUsers_t * liste );

/* Suppression d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=inconnu */

int SupprimeConnectUser( char * name, ListeConnectUsers_t * liste );

#endif
```

### **Gestpart.c**

```
/*-----*/
/*   Projet de Fin d'annee de C Unix - ESGI 3 eme Annee   */
/*   Richeton Nicolas et Romelard Fabrice                */
/*   FAB-NICO BATAILLE NAVALE                           */
/*-----*/

#include "gestpart.h"

Map_t * map = NULL;
Liste_Bateaux_t * listeBateaux = NULL;
ListeUsers_t * listeUsers = NULL;
ListeConnectUsers_t * listeConnectUsers = NULL;
ListeConnectUsers_t * listeJoueurs = NULL;
Com_t * com = NULL;
int mutex;
int semJoueurs[ NB_JOUEURS_MAX ];

struct sembuf sbuf={ 0,0,0 };

extern char nom[ 30 ];
extern int socket_service;

/* Fonction sur les Semaphores */

/* Recuperation d'une semaphore */
int getSem( int num )
{
    key_t key = KEY_SEM+num;
    int id;

    /* Creation de la semaphore */
    id = semget(key, 1, IPC_CREAT | 0700);
    if (id == -1)
    {
        printf("\nERREUR semget %d\n", num );
        return( -1 );
    }
    printf("Création réussie du sem %d key=%x id=%d\n", num, key, id);
}
```

```
        return( id );
    }

/* Initialisation d'une semaphore */
int initSem( int id, int val )
{
    arg.val = val;

    /* Initialisation de la semaphore */
    if ( semctl(id, 0, SETVAL, arg) == -1 )
    {
        printf("\nERREUR semctl %d\n", id );
        return( -1 );
    }

    printf("Initialisation réussie id=%d val=%d\n", id, val );
    return( 0 );
}

/* Destruction d'un semaphore */
int destructSem( int id )
{
    if( semctl( id, IPC_RMID, 0 ) == -1)
    {
        printf("\nERREUR destruction du sem %d\n", id);
        return( -1 );
    }
    printf("Destruction réussie du sem %d\n", id);
    return( 0 );
}

/* Creation d'une SHM et attachement d'un pointeur pour l'utilisation */
int initShm( int num, char ** adrpt, int lg )
{
    key_t key = KEY_SHM+num;
    int id;

    /* Creation/recuperation du segment de SHM */
    /* Segment de taille lg */
    id = shmget(key, lg, IPC_CREAT | 0700);
    if (id == -1)
    {
        printf("Erreur création du shm %d key=%x\n", num, key);
        perror( "shmget" );
        return( -1 );
    }

    /* Recuperation pointeur sur le segment de SHM */
    *adrpt = shmat(id, 0, 0);
    if ( (int)(*adrpt) == -1 )
    {
        printf("Erreur d'attachement du shm %d\n", num );
        return( -1 );
    }
    printf("Création et attachement réussie du shm %d key=%x id=%d\n", num, key, id);
    return( id );
}

/* Detachement et destruction du segment de SHM */
int destructShm( int id )
{
    /* Destruction de la SHM */
    if( shmctl( id, IPC_RMID, 0 ) == -1 )
    {
        printf( "Erreur suppression du shm %d", id );
        return( -1 );
    }

    printf("Suppression réussie du shm %d\n", id);
    return( 0 );
}

/* Detachement de la SHM */
int detache( char * ptShm )
```

```
{
    /* Detachement du pointeur */
    if( shmdt( ptShm ) != 0 )
    {
        printf( "Erreur détachement du pointeur shm" );
        return ( -1 );
    }
    return( 0 );
}

/*-----*/
/* Initialisation de l'application */

int initialisationSHM()
{
    if ( initShm( SHM_COM, (char**)&com, sizeof( Com_t ) ) == -1 )
        return( -1 );

    com->partie=0;

    if ( initShm( SHM_USERS, (char**)&listeUsers, sizeof( ListeUsers_t ) *(NB_USERS_MAX-1) ) == -1 )
        return( -1 );

    listeUsers->nbMax = NB_USERS_MAX;
    listeUsers->nbUsed = 0;
    initieListe( FICHER_USERS, listeUsers );

    if ( initShm( SHM_CONNECTUSERS, (char**)&listeConnectUsers, sizeof( ListeConnectUsers_t )
*(NB_CONNECTUSERS_MAX-1) ) == -1 )
        return( -1 );

    listeConnectUsers->nbMax = NB_CONNECTUSERS_MAX;
    listeConnectUsers->nbUsed = 0;

    mutex = getSem( SEM_LOCK );
    if ( mutex == -1 )
        return( -1 );

    if ( initSem( mutex, 1 ) == -1 )
        return( -1 );

    if ( initShm( SHM_MAP, (char **)&map, sizeof(int)*2 + MAX_X * MAX_Y ) == -1 )
        return( -1 );

    if ( initShm( SHM_JOUEURS, (char**)&listeJoueurs, sizeof( ListeConnectUsers_t ) * NB_CONNECTUSERS_MAX-1 ) == -1
)
        return( -1 );

    if ( initShm( SHM_BATEAUX, (char **)&listeBateaux, sizeof( Liste_Bateaux_t ) + sizeof( Bateau_t ) * NB_BATEAUX_MAX-
1 ) == -1 )
        return( -1 );

    printf("init OK\n");
}

/* Affichage des shared memory */
void ImprimeDatas( void )
{
    printf( "----- Map -----\\n" );
    AfficheMap( map );
    printf( "----- Liste des bateaux -----\\n" );
    AfficheListeBateaux( listeBateaux );
    printf( "----- Liste des users -----\\n" );
    AfficheListe( listeUsers );
    printf( "----- Liste des users connectés -----\\n" );
    AfficheListeConnect( listeConnectUsers );
    printf( "----- Liste des joueurs -----\\n" );
    AfficheListeConnect( listeJoueurs );
}

/* Identification */
int identificationJoueur( msg_ident_t * ident )
{
    int i;
```

```
/* User reference */
i = TrouverJoueur( ident->nom, listeUsers );
if ( i == -1 )
    return( 1 );
/* Password OK */
if ( strcmp( ident->password, listeUsers->liste[ i ].password ) != 0 )
    return( 2 );
/* Deja connecté */
if ( ChercheConnectionUser( ident->nom, listeConnectUsers ) != -1 )
    return( 3 );
/* Connection */
if ( AjouterConnectUser( ident->nom, getpid(), socket_service, listeConnectUsers ) == -1 )
    return( 4 );

return( 0 );
}

/* Deconnexion */
int deconnexion( char * nom )
{
    int i;
    i = ChercheConnectionUser( nom, listeConnectUsers );
    if ( i == -1 )
        return( 1 );
    SupprimeConnectUser( nom, listeConnectUsers );
    return( 0 );
}

/* Creation d'une partie */
int creationPartie( msg_create_t * create )
{
    int i, res;

    if ( com->partie == 1 )
        return( 1 );

    if ( create->nbJoueurs > NB_JOUEURS_MAX )
        return( 2 );

    if ( create->x > MAX_X || create->y > MAX_Y )
        return( 3 );

    if ( create->nbBateaux > NB_BATEAUX_MAX )
        return( 4 );

    map->sizeX = create->x;
    map->sizeY = create->y;

    initialisationMap( map );

    listeJoueurs->nbMax = create->nbJoueurs;
    listeJoueurs->nbUsed = 0;

    listeBateaux->nbMax = create->nbBateaux;
    listeBateaux->nbBateaux = 0;

    com->partie = 1;
    return( 0 );
}

/* Ajout bateau */
int ajoutBateau( msg_bateau_t * b )
{
    Pos_t pos;
    Bateau_t * bateau;
    int res;

    pos.x = b->x;
    pos.y = b->y;

    bateau = nouveauBateau( &res, b->id, map, &pos, b->dir, b->lg );
    if( bateau == NULL )
    {
        printf( "Erreur bateau = %d\n", res );
    }
}
```

```
        return( res );
    }
    else
    {
        AjoutBateau( listeBateaux, bateau );
        free( bateau );
    }
    return( 0 );
}

/* Fin de partie */
int FindePartie()
{
    int i, winner, max=0;

    for( i=0; i < listeJoueurs->nbUsed ; i++ )
    {
        if ( listeJoueurs->liste[ i ].socket >= max )
        {
            winner = i;
            max = listeJoueurs->liste[ i ].socket;
        }
    }

    printf( "#####\n" );
    printf( "##### G A G N A N T #####\n" );
    printf( "#####\n" );
    printf( "##### %s , SCORE = %d #####\n", listeJoueurs->liste[ winner ].nom,
        listeJoueurs->liste[ winner ].socket );
    printf( "#####\n" );
    printf( "#####\n" );

    com->partie = 0;
    listeJoueurs->nbUsed = 0;
}

/* Demande d'inscription */
/* Pas de validation par admin */
int demandeInscription()
{
    int i;
    if ( com->partie == 0 )
        return( 4 );
    i = ChercheConnectionUser( "administrateur", listeConnectUsers );
    if ( i == -1 )
        return( 5 );
    if ( listeJoueurs->nbUsed == listeJoueurs->nbMax )
        return( 3 );

    i = ChercheConnectionUser( nom, listeConnectUsers );

    if ( AjouterConnectUser( listeConnectUsers->liste[i].nom, listeConnectUsers->liste[i].pid, 0, listeJoueurs ) < 0 )
        return( 6 );

    return( 0 );
}

/* Tour de jeu */
int JoueJeu( msg_play_t * pos )
{
    Pos_t p;
    p.x = pos->x;
    p.y = pos->y;
    return( JoueAvecCom( map, listeBateaux, &p ) );
}

int JoueAvecCom( Map_t * map, Liste_Bateaux_t * liste, Pos_t * pos )
{
    int result;
    int i;
    char val;
    Pos_t * listePos;

    result = verifiePositionMap( map, pos );
}
```

```
switch( result )
{
    case 0: /* Dans l'eau */
        printf( "Dans l'eau\n" );
        break;
    case 1: /* Out of range */
        printf( "Hors des Limites définies\n" );
        break;
    case 2: /* Touch something */
        val = RecupMapPos( map, pos );
        if ( val != MAP_TOUCHE && val != MAP_COULE )
        {
            printf( "Touché.\n" );
            ModifiePositionMap( map, pos, MAP_TOUCHE );
            liste->bateaux[ val - 'A' ].nbTouche++;
            com->hit.valeur = MAP_TOUCHE;
            strcpy( com->hit.dir, liste->bateaux[ val - 'A' ].dir );
            com->hit.x = pos->x;
            com->hit.y = pos->y;
            com->hit.lg = 1;
            i = ChercheConnectionUser( nom, listeJoueurs );
            listeJoueurs->liste[ i ].socket += 1;
            if ( liste->bateaux[ val - 'A' ].nbTouche == liste->bateaux[ val - 'A' ].lg )
            {
                printf( "Coulé !!!\n" );
                listePos = CalculPosition( &(liste->bateaux[ val - 'A' ].origine),
                                            liste->bateaux[ val - 'A' ].dir,
                                            liste->bateaux[ val - 'A' ].lg );
                ModifieListePositionMap( map, listePos, liste->bateaux[ val - 'A' ].lg,
                                          MAP_COULE );

                free( listePos );
                result = 3;

                com->hit.valeur = MAP_COULE;
                com->hit.x = liste->bateaux[ val - 'A' ].origine.x;
                com->hit.y = liste->bateaux[ val - 'A' ].origine.y;
                com->hit.lg = liste->bateaux[ val - 'A' ].lg;
                listeJoueurs->liste[ i ].socket += 1;

                if ( verificationFinPartie( liste ) == 1 )
                {
                    printf( "Fin de partie\n" );
                    result = 4;
                }
            }
        }
    else
        result = 0;
        break;
    default:
        printf( "Code non géré !!!\n" );
        result = 5;
}
return( result );
}
```

### Gestpart.h

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee  */
/*  Richeton Nicolas et Romelard Fabrice                */
/*  FAB-NICO BATAILLE NAVALLE                          */
/*-----*/

#ifndef __GESTPART__
#define __GESTPART__

#include "partie.h"
#include "joueurs.h"
#include "connectjoueurs.h"
#include <stdio.h>
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <string.h>
#include "protoserv.h"

/* Cle de base pour les semaphores et les segments de memoire */
#define KEY_SEM 0x1300
#define KEY_SHM 0x1400

#define NB_BATEAUX_MAX 10
#define NB_USERS_MAX 20
#define NB_CONNECTUSERS_MAX 20
#define NB_JOUEURS_MAX 20

#define MAX_X 70
#define MAX_Y 30

#define FICHER_USERS "listejoueurs.cfg"

enum IDSEM
{
    SEM_LOCK=0,
    SEM_DEB_JOUEURS=1
};

enum IDSHM
{
    SHM_MAP=0,
    SHM_BATEAUX=1,
    SHM_USERS=2,
    SHM_CONNECTUSERS=3,
    SHM_JOUEURS=4,
    SHM_COM=5
};

/* Union pour les operations sur les semaphores */
union sem_union
{
    int val;
    struct semid_ds *buf;
    ushort * array;
} arg;

/* Zone de communication inter process serveur */
typedef struct com
{
    int partie;
    int typeFlux; /* 1=GO_PLAY, 2=HIT, 3=INSCRIPTION, 4=INIT, 5=END_PARTIE */
    msg_hit_t hit;
    int joueurCourant;

    int idMsg;
    int numConnecte;
    int pidAdmin;
    char nom[ 30 ];
} Com_t;

/* Prototypes des fonctions */
int getSem( int num );
int initSem( int num, int val );
int destructSem( int id );
int initShm( int num, char ** pt, int lg );
int destructShm( int id );
int detache( char * ptShm );

/* Initialisation des shared memory */
int initialisationSHM();

/* Affichage des shared memory */
void ImprimeDatas(void);

/* Identification */
```

```
int identificationJoueur( msg_ident_t * ident );
```

```
#endif
```

### Ihm.c

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                  */
/*  FAB-NICO BATAILLE NAVALE                             */
/*-----*/
#include "partie.h"
#include "protoserv.h"
Map_t * map = NULL;
extern char nom[ 30 ];
void gestionihmClient()
{
    char choice[ 80 ];
    int x;
    int y;
    printf( "IHM CLIENT : %s\n", nom );
    do
    {
        printf( "\n--- Menu ---\n\n" );
        printf( "i - Demande partie\n\n" );
        printf( "q - Quitter\n" );
        printf( "\nChoix ? " );
        scanf( "%s", &choice );
        switch( (int) choice[ 0 ] )
        {
            case 'i':
                switch ( EnvoieInscription() )
                {
                    case 0:
                        printf( "OK\n" );
                        LancementPartie();
                        break;
                    case 1:
                        printf( "Refus, trop de user\n" );
                        break;
                    case 2:
                        printf( "Refus par admin\n" );
                        break;
                    case 3:
                        printf( "Partie pleine\n" );
                        break;
                    case 4:
                        printf( "Pas de partie\n" );
                        break;
                    case 5:
                        printf( "Pas d'admin connecté\n" );
                        break;
                    case 6:
                        printf( "Problème technique\n" );
                        break;
                    default:
                        printf( "Problème inconnu !!!\n" );
                }
                break;
            case 'z':
                EnvoieIdMsg( MSG_DATAS );
                break;
            case 'q':
                switch( EnvoieDecnx() )
                {
                    case 0:
                        printf( "OK\n" );
                        break;
                    default :
                        printf( "Code retour inconnu !!!\n" );
                }
                break;
            default:
                printf( "Commande inconnue\n" );
        }
    }
}
```

```
        while( choice[ 0 ] != 'q' );
    }

void gestionIhmAdmin()
{
    char choice[ 80 ];
    char nom[ 80 ];
    char password[ 80 ];
    printf( "IHM ADMIN\n" );
    do
    {
        DemandeMap();
        if ( map != NULL )
            AfficheMap( map );
        printf( "\n--- Menu ----- \n\n" );
        printf( "c - Création d'une nouvelle partie\n" );
        printf( "g - Début de la partie\n" );
        printf( "f - Fin de la partie\n" );
        printf( "a - Ajouter un utilisateur\n" );
        printf( "d - Supprimer un utilisateur\n" );
        printf( "i - informations\n" );
        printf( "q - Quitter\n" );
        printf( "\nChoix ? " );
        scanf( "%s", &choice );
        switch( (int) choice[ 0 ] )
        {
            case 'z':
                EnvoieIdMsg( MSG_DATAS );
                break;

            case 'c':
                nouvellePartie();
                break;

            case 'g':
                EnvoieDebutPartie();
                break;

            case 'i':
                menuInformation();
                break;

            case 'f':
                EnvoieFinPartie();
                break;

            case 'a':
                printf( "Nom du joueur ? " );
                scanf( "%s", nom );
                printf( "Mot de Passe du joueur ? " );
                scanf( "%s", password );
                switch( EnvoieAjoutJoueur( nom, password ) )
                {
                    case 0:
                        printf( "\nOK\n" );
                        break;

                    default :
                        printf( "\nCode retour inconnu !!!\n" );
                }
                break;

            case 'd':
                printf( "Nom du joueur ? " );
                scanf( "%s", nom );
                switch( EnvoieSupJoueur( nom, password ) )
                {
                    case 0:
                        printf( "\nOK\n" );
                        break;

                    case 1:
                        printf( "\nUtilisateur non present dans le fichier\n");
                        break;

                    default :
                        printf( "\nCode retour inconnu !!!\n" );
                }
                break;

            case 'q':
                switch( EnvoieDecnx() )
                {
                    case 0:
                        printf( "OK\n" );
                        break;
```

## ANNEXES BATAILLE NAVALE PROJET UNIX

```
                                default :
                                printf( "Code retour inconnu !!!\n" );
                                }
                                break;
default:
    printf( "commande inconnue\n" );
}

while( choice[ 0 ] != 'q' );
}

/* Partie */

int LancementPartie()
{
    int x, y;
    int endPartie = 0;
    if ( AttenteInitPartie() != 0 )
        return( 1 );
    while( endPartie == 0 )
    {
        AfficheMap( map );
        printf( "Attente ...\n" );
        switch( AttendDebutPartie() )
        {
            case -1 :
                return( 1 );
                break;
            case 0:
                printf( "Saisie d'une position :\n" );
                printf( "X ? " );
                scanf( "%d", &x );
                x -= 1;
                printf( "Y ? " );
                scanf( "%d", &y );
                y -= 1;
                switch( EnvoiePositionJeu( x, y )
                {
                    case 0:
                        printf( "Dans l'eau.\n" );
                        break;
                    case 1:
                        printf( "Hors des limites\n" );
                        break;
                    case 2:
                        printf( "Touché !!!\n" );
                        break;
                    case 3:
                        printf( "Coulé !!!\n" );
                        break;
                    case 4:
                        printf( "Fin de partie !!!\n" );
                        break;
                    case 5:
                        printf( "Problème technique\n" );
                        break;
                    default:
                        printf( "Réponse inconnue\n" );
                        break;
                }
                break;
            case 1:
                // HIT
                break;
            case 2:
                // HIT
                printf( "Fin de partie\n" );
                endPartie = 1;
                break;
            default:
                printf( "Problème inconnu !!!\n" );
                break;
        }
    }
}
```

```
}

/* Menu pour recuperation des infos */

int menuInformation()
{
    char choice[ 80 ];
    do
    {
        printf( "\n--- Menu des Informations ---\n\n" );
        printf( "m - map\n" );
        printf( "q - Quitter\n" );
        printf( "\nChoix ? " );
        scanf( "%s", &choice );
        switch( (int) choice[ 0 ] )
        {
            case 'm':
                DemandeMap();
                break;
            case 'q':
                break;
            default:
                printf( "Commande inconnue\n" );
        }
    }

    while( choice[ 0 ] != 'q' );
}

/* Saisie d'une nouvelle partie */

int nouvellePartie()
{
    int nbJoueurs;
    int nbBateaux;
    int x, y, lg;
    int i;
    char dir[ 80 ];
    printf( "Nombre de joueurs max partie ? " );
    scanf( "%d", &nbJoueurs );
    printf( "Nombre de bateaux ? " );
    scanf( "%d", &nbBateaux );
    printf( "Taille de la Carte (map) en largeur (X) ? " );
    scanf( "%d", &x );
    printf( "Taille de la Carte (map) en hauteur (Y) ? " );
    scanf( "%d", &y );
    i= EnvoieCreationPartie( x, y, nbJoueurs, nbBateaux );
    if ( i == 0 )
    {
        for( i=0; i < nbBateaux; i++ )
        {
            printf( "Bateau N°%c :\n", i + 'A' );
            printf( "Origine en X ? " );
            scanf( "%d", &x );
            x -= 1;
            printf( "Origine en Y ? " );
            scanf( "%d", &y );
            y -= 1;
            printf( "Taille du Bateau ? " );
            scanf( "%d", &lg );
            printf( "Direction du bateau (N,E,S,O) ? " );
            scanf( "%s", dir );
            switch ( EnvoieBateau( x, y, lg, dir, i + 'A' ) )
            {
                case 0 :
                    printf( "OK\n" );
                    break;
                case 1 :
                    printf( "Le bateau est hors limite\n" );
                    i--;
                    break;
                case 2 :
                    printf( "Chevauchement entre les bateaux \n" );
                    i--;
                    break;
                case 3 :

```

## ANNEXES BATAILLE NAVALE PROJET UNIX

```
                printf( "Problème technique\n" );
                i--;
                break;
            default:
                printf( "Problème inconnu\n" );
                i--;
        }
    }
}
else
{
    switch( i )
    {
        case 1:
            printf( "La partie est déjà créée\n" );
            break;
        case 2:
            printf( "Nombre de joueurs trop important\n" );
            break;
        case 3:
            printf( "La carte (map) est trop grande\n" );
            break;
        case 4:
            printf( "Le nombre de bateaux est trop important\n" );
            break;
        case 5:
            printf( "Problème technique\n" );
            break;
        default:
            printf( "Problème inconnu\n" );
    }
}
return( i );
}
```

### Joueurs.c

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                    */
/*  FAB-NICO BATAILLE NAVALE                                */
/*-----*/

#include "joueurs.h"

/* Initialisation d'une liste a partir d'un fichier */
/* return : nbUser ou -1 */

int initieListe( char * filename, ListeUsers_t * liste )
{
    FILE * io;
    char buffer[ 80 ];
    char * delimitateur;

    strcpy( liste->fileName, filename );
    liste->nbUsed = 0;

    io = fopen( filename, "rt" );
    if ( io == NULL )
    {
        printf( "Erreur d'ouverture du fichier %s\n", filename );
        return( -1 );
    }

    while( fgets( buffer, 80, io ) != NULL )
    {
        delimitateur = strchr( buffer, DELIMITEUR );

        strncpy( liste->liste[ liste->nbUsed ].nom, buffer, delimitateur - buffer );
        liste->liste[ liste->nbUsed ].nom[ delimitateur - buffer ] = '\0';

        strcpy( liste->liste[ liste->nbUsed ].password, delimitateur+1 );
        liste->liste[ liste->nbUsed ].password[ strlen( liste->liste[ liste->nbUsed ]. password ) - 1 ] = '\0';
    }
}
```

```
        liste->nbUsed++;
    }
    return( liste->nbUsed );
}

/* Recherche d'un Joueur dans la liste */
/* return indice user ou -1 */

int TrouverJoueur( char * name, ListeUsers_t * liste )
{
    int i;
    for ( i=0; i < liste->nbUsed && strcmp( name, liste->liste[ i ].nom ); i++ );
    if ( i < liste->nbUsed )
        return( i );
    else
        return( -1 );
}

/* Affichage d'une liste de Joueurs */

void AfficheListe( ListeUsers_t * liste )
{
    int i;
    printf( "Nom du Fichier (FileName):%s nbMax: %d nbUsed= %d\n",
        liste->fileName,
        liste->nbMax,
        liste->nbUsed );
    for ( i=0; i < liste->nbUsed ; i++ )
    {
        printf( "Nom: %-20s\tPassword: %-20s\n",
            liste->liste[ i ].nom,
            liste->liste[ i ].password );
    }
}

/* Ecriture fichier d'une liste */
/* return : nbUsed ou -1 */

int EcritureListe( ListeUsers_t * liste )
{
    FILE * io;
    int i;

    io = fopen( liste->fileName, "wt" );
    if ( io == NULL )
    {
        printf( "Erreur d'ouverture du fichier %s\n", liste->fileName );
        return( -1 );
    }

    for( i=0; i < liste->nbUsed; i++ )
    {
        fprintf( io, "%s%c%s\n",
            liste->liste[ i ].nom,
            DELIMITEUR,
            liste->liste[ i ].password );
    }

    return( liste->nbUsed );
}

/* Ajout d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=deja enregistré */

int AjoutJoueur( char * name, char * password, ListeUsers_t * liste )
{
    if ( TrouverJoueur( name, liste ) != -1 )
        return( -2 );

    strcpy( liste->liste[ liste->nbUsed ].nom, name );
    strcpy( liste->liste[ liste->nbUsed ].password, password );
    liste->nbUsed++;

    if ( EcritureListe( liste ) == -1 )
```

```
        return( -1 );

    return( liste->nbUsed );
}

/* Suppression d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=inconnu */

int SupprimeJoueur( char * name, ListeUsers_t * liste )
{
    int i;

    i=TrouverJoueur( name, liste );
    if ( i == -1 )
        return( -2 );
    i++;
    for( ; i < liste->nbUsed; i++ )
    {
        strcpy( liste->liste[ i-1 ].nom, liste->liste[ i ].nom );
        strcpy( liste->liste[ i-1 ].password, liste->liste[ i ].password );
    }
    liste->nbUsed--;

    if ( EcrireListe( liste ) == -1 )
        return( -1 );

    return( liste->nbUsed );
}
```

### **Joueurs.h**

```
/*-----*/
/*   Projet de Fin d'annee de C Unix - ESGI 3 eme Annee   */
/*   Richeton Nicolas et Romelard Fabrice                */
/*   FAB-NICO BATAILLE NAVALE                           */
/*-----*/

#ifndef __USERS__
#define __USERS__

#include <stdio.h>
#include <string.h>

#define DELIMITEUR ';'

/* Structure d'un user */

typedef struct users
{
    char nom[ 20 ];
    char password[ 20 ];
} User_t;

/* Structure de la liste des users */

typedef struct listeusers
{
    char fileName[ 80 ];
    int nbMax;
    int nbUsed;
    User_t liste[ 1 ];
} ListeUsers_t;

/* Initialisation d'une liste a partir d'un fichier */
/* return : nbUser ou -1 */

int initieListe( char * filename, ListeUsers_t * liste );

/* Recherche d'un Joueur dans la liste */
/* return indice user ou -1 */

int TrouverJoueur( char * name, ListeUsers_t * liste );
```

```
/* Affichage d'une liste de Joueurs */

void AfficheListe( ListeUsers_t * liste );

/* Ecriture fichier d'une liste */
/* return : nbUsed ou -1 */

int EcritureListe( ListeUsers_t * liste );

/* Ajout d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=deja enregistré */

int AjoutJoueur( char * name, char * password, ListeUsers_t * liste );

/* Suppression d'un Joueur */
/* return : nbUsed ou -1=pb tech, -2=inconnu */

int SupprimeJoueur( char * name, ListeUsers_t * liste );

#endif
```

### Partie.c

```
/*-----*/
/*   Projet de Fin d'annee de C Unix - ESGI 3 eme Annee   */
/*   Richeton Nicolas et Romelard Fabrice                */
/*   FAB-NICO BATAILLE NAVALE                            */
/*-----*/

#include "partie.h"

/* Allocation d'une MAP */

Map_t * allocationMap( int lgX, int lgY )
{
    Map_t * map;

    map = (Map_t *) malloc( sizeof( int ) * 2 + lgX * lgY + 1 );
    if ( map == NULL )
    {
        return( NULL );
    }

    map->sizeX = lgX;
    map->sizeY = lgY;

    return( map );
}

/* Initialisation de la MAP */

void initialisationMap( Map_t * map )
{
    int x, y;

    for( x=0; x < map->sizeX; x++ )
    {
        for( y=0; y < map->sizeY; y++ )
        {
            map->data[ y*map->sizeX + x ] = MAP_VIDE;
        }
        map->data[ x*y + 1 ] = '\0';
    }
}

/* Récupération d'une case de la MAP */

char RecupMapPos( Map_t * m, Pos_t * p )
{
    return( m->data[ p->y*m->sizeX + p->x ] );
}

/* Affichage de la MAP */
```

```
void AfficheMap(Map_t * map)
{
    int x, y, division;

    printf( "\\X|" );
    for( x=0; x < map->sizeX; x++ )
    {
        division = (x+1) / 10 ;
        if ( division != 0 )
            printf( "%d|", division );
        else
            printf( "0|" );
    }
    printf( "\\nY|" );
    for( x=0; x < map->sizeX; x++ )
    {
        division = (x+1) % 10;
        printf( "%d|", division );
    }

    printf( "\\n----" );
    for( x=0; x < map->sizeX; x++ )
    {
        printf( "--" );
    }

    printf( "\\n" );
    for( y=0; y < map->sizeY; y++ )
    {
        printf( "%02d |", y+1 );
        for( x=0; x < map->sizeX; x++ )
        {
            printf( "%c|", map->data[ y*map->sizeX + x ] );
        }
        printf( "\\n" );
    }
    printf( "----" );
    for( x=0; x < map->sizeX; x++ )
    {
        printf( "--" );
    }
    printf( "\\n" );
}

/* Calcule une liste de position */
/* !!! Attention, retourne une liste allouées de positions */

Pos_t * CalculPosition( Pos_t * origine, char * direction, int lg )
{
    Pos_t * p = (Pos_t *) malloc( sizeof( Pos_t ) * lg );
    int incX, incY, nb;

    if ( strcmp( "N", direction ) == 0 )
    {
        incX = 0;
        incY = -1;
    }
    else
    if ( strcmp( "E", direction ) == 0 )
    {
        incX = 1;
        incY = 0;
    }
    else
    if ( strcmp( "S", direction ) == 0 )
    {
        incX = 0;
        incY = 1;
    }
    else
    if ( strcmp( "O", direction ) == 0 )
    {
        incX = -1;
    }
}
```

```
        incY = 0;
    }
    for( nb = 0; nb < lg; nb++ )
    {
        p[ nb ].x = origine->x + ( incX * nb );
        p[ nb ].y = origine->y + ( incY * nb );
    }

    return( p );
}

/* Affiche une position */

void AffichePosition( Pos_t * p )
{
    printf( "X=%d Y=%d\n", p->x + 1, p->y + 1);
}

/* Affiche une liste de position */

void AfficheAllPosition( Pos_t * p, int nb )
{
    int n;
    printf( "Position(s) :\n" );
    for( n=0; n < nb; n++ )
        AffichePosition( &( p[ n ] ) );
}

/* Verification qu'une position est dans le map */
/* Return */
/* 0 si OK */
/* 1 si out of map */
/* 2 si overlap */

int verifiePositionMap( Map_t * m, Pos_t * p )
{
    /* Out of ranges */

    if ( p->x < 0 || p->y < 0 || p->x >= m->sizeX || p->y >= m->sizeY )
    {
        return( 1 );
    }

    /* Overlap */

    if ( RecupMapPos( m, p ) != MAP_VIDE )
        return( 2 );

    return( 0 );
}

/* Verifie une liste de positions */

int verifieListePositionMap( Map_t * m, Pos_t * p, int nbPos )
{
    int n = 0, res = 0;

    for( n=0; n < nbPos && res == 0; n++ )
        res = verifiePositionMap( m, &( p[ n ] ) );

    return( res );
}

/* Modifie une position du map */

int ModifiePositionMap( Map_t * m, Pos_t * p, char val )
{
    int res;
    res = verifiePositionMap( m, p );
    if ( res == 0 || res == 2 )
    {
        m->data[ p->y*m->sizeX + p->x ] = val;
    }
    return( res );
}
```

```
/* Modifie une liste de positions du map */

int ModifieListePositionMap( Map_t * m, Pos_t * p, int nbPos, char val )
{
    int n = 0, res = 0;

    for( n=0; n < nbPos && (res == 0 || res == 2); n++ )
        res = ModifiePositionMap( m, &( p[ n ] ), val );
    return( res );
}

/* Verification et allocation d'un bateau */

Bateau_t * nouveauBateau( int * result, char idBateau, Map_t * map, Pos_t * origine, char * dir, int lg )
{
    Bateau_t * bateau;
    Pos_t * listpos;

    listpos = CalculPosition( origine, dir, lg );

    *result = verifieListePositionMap( map, listpos, lg );

    if ( *result == 0 )
    {
        ModifieListePositionMap( map, listpos, lg, idBateau );
    }
    else
        return( NULL );
    free( listpos );

    bateau = (Bateau_t *) malloc( sizeof( Bateau_t ) );
    if ( bateau == NULL )
    {
        *result = 3;
        return( NULL );
    }
    bateau->origine.x = origine->x;
    bateau->origine.y = origine->y;
    strcpy( bateau->dir, dir );
    bateau->lg = lg;
    bateau->id = idBateau;
    bateau->nbTouche = 0;

    return( bateau );
}

/* Affichage d'un bateau */

void AfficheBateau( Bateau_t * bateau )
{
    printf( "Bateau N° %c\n", bateau->id );
    printf( "\tOrigine : " );
    AffichePosition( &( bateau->origine ) );
    printf( "\tDirection : %s\n", bateau->dir );
    printf( "\tLongueur : %d\n", bateau->lg );
    printf( "\tNombre de touche : %d\n", bateau->nbTouche );
}

/* Affichage d'une liste de bateaux */

void AfficheListeBateaux( Liste_Bateaux_t * liste )
{
    int i;
    printf( "Nombre maximal de bateaux = %d\n", liste->nbMax );
    printf( "Nombre de bateaux = %d\n", liste->nbBateaux );
    for( i=0; i < liste->nbBateaux; i++ )
        AfficheBateau( &( liste->bateaux[ i ] ) );
}

/* Ajout d'un bateau à une liste (recopie) */

int AjoutBateau( Liste_Bateaux_t * liste, Bateau_t * bateau )
{
    if( liste->nbBateaux < liste->nbMax )
```

```
{
    memcpy( &(liste->bateaux[ liste->nbBateaux ]), bateau, sizeof( Bateau_t ) );
    liste->nbBateaux++;
    return( liste->nbBateaux );
}
else
    return( 0 );
}

/* Verif fin de partie */

int verificationFinPartie( Liste_Bateaux_t * liste )
{
    int i;
    for( i=0; i < liste->nbBateaux && liste->bateaux[ i ].lg == liste->bateaux[ i ].nbTouche; i++ );

    if ( i == liste->nbBateaux )
        return 1;

    return 0;
}

/* Joue une case */
/* 0 : dans l'eau */
/* 1 : out of range */
/* 2 : touche */
/* 3 : coule */
/* 4 : fin de partie */
/* 5 : pb technique */

int Joue( Map_t * map, Liste_Bateaux_t * liste, Pos_t * pos )
{
    int result;
    char val;
    Pos_t * listePos;

    result = verifiePositionMap( map, pos );

    switch( result )
    {
        case 0: /* Dans l'eau */
            printf( "Dans l'eau\n" );
            break;
        case 1: /* Out of range */
            printf( "Hors des limites\n" );
            break;
        case 2: /* Touch something */
            val = RecupMapPos( map, pos );
            if ( val != MAP_TOUCHE && val != MAP_COULE )
            {
                printf( "Touché.\n" );
                ModifiePositionMap( map, pos, MAP_TOUCHE );
                liste->bateaux[ val - 'A' ].nbTouche++;

                if ( liste->bateaux[ val - 'A' ].nbTouche == liste->bateaux[ val - 'A' ].lg )
                {
                    printf( "Coulé !!!\n" );
                    listePos = CalculPosition(&(liste->bateaux[ val - 'A' ].origine),
                                                liste->bateaux[ val - 'A' ].dir,
                                                liste->bateaux[ val - 'A' ].lg );
                    ModifieListePositionMap( map, listePos, liste->bateaux[ val - 'A' ].lg, MAP_COULE );
                }
            }
            free( listePos );
            result = 3;

            if ( verificationFinPartie( liste ) == 1 )
            {
                printf( "Fin de la partie\n" );
                result = 4;
            }
        }
    }
    else
        result = 0;
}
```

## ANNEXES BATAILLE NAVALE PROJET UNIX

```
                break;
            default:
                printf( "Code non géré !!!\n" );
                result = 5;
        }
        return( result );
    }
}
```

### **Parite.h**

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                    */
/*  FAB-NICO BATAILLE NAVALLE                              */
/*-----*/

#ifndef __JEU__
#define __JEU__

#include <stdio.h>

#define MAP_VIDE ''
#define MAP_TOUCHE 'T'
#define MAP_COULE 'C'

/* Structure de la carte (MAP) */

typedef struct Map
{
    int sizeX;           /* Taille X du tableau */
    int sizeY;           /* Taille Y du tableau */
    char data[ 1 ];      /* Valeur de chaque case du tableau */
} Map_t;

/* Structure d'un position de bateau */

typedef struct Pos
{
    int x;               /* Position X sur la MAP */
    int y;               /* Position Y sur la MAP */
} Pos_t;

/* Structure d'un bateau sur la MAP */

typedef struct Bateau
{
    char id;             /* Identifiant du bateau (caractere sur la MAP) */
    Pos_t origine;       /* Position d'origine du bateau sur la MAP */
    char dir[ 3 ];       /* Direction du bateau (N E S O) */
    int lg;              /* Taille du bateau */
    int nbTouche;        /* Nb d'elements du bateau touche */
} Bateau_t;

/* Liste de bateaux dans le jeu */

typedef struct Liste_Bateaux
{
    int nbMax;           /* Nombre max de bateaux défini par l'admin */
    int nbBateaux;       /* Nombre de bateaux dans la liste */
    Bateau_t bateaux[ 1 ]; /* Liste des bateaux défini par l'admin */
} Liste_Bateaux_t;

/* Allocation d'une MAP */

Map_t * allocationMap( int lgX, int lgY );

/* Initialisation de la MAP */

void initialisationMap( Map_t * map );

/* Récupération d'une case de la MAP */

char RecupMapPos( Map_t * m, Pos_t * p );
```

```
/* Affichage de la MAP */

void AfficheMap(Map_t * map);

/* Calcule une liste de position sur la MAP */

Pos_t * CalculPosition( Pos_t * origine, char * direction, int lg );

/* Affiche une position sur la MAP */

void AffichePosition( Pos_t * p );

/* Affiche une liste de position sur la MAP */

void AfficheAllPosition( Pos_t * p, int nb );

/* Verification qu'une position est dans la MAP pour les placements des bateaux */
/* Return */
/* 0 si OK */
/* 1 si out of map */
/* 2 si overlap */

int verifiePositionMap( Map_t * m, Pos_t * p );

/* Verifie une liste de positions des bateaux */

int verifieListePositionMap( Map_t * m, Pos_t * p, int nbPos );

/* Modifie une position de la MAP */

int ModifiePositionMap( Map_t * m, Pos_t * p, char val );

/* Modifie une liste de positions de la MAP */

int ModifieListePositionMap( Map_t * m, Pos_t * p, int nbPos, char val );

/* Verification et allocation d'un bateau (Création du bateau par l'admin */

Bateau_t * nouveauBateau( int * result, char idBateau, Map_t * map,
                          Pos_t * origine, char * dir, int lg );

/* Affichage d'un bateau sur la MAP*/

void AfficheBateau( Bateau_t * bateau );

/* Affichage d'une liste de bateaux sur la MAP*/

void AfficheListeBateaux( Liste_Bateaux_t * liste );

/* Ajout d'un bateau à une liste (recopie) */

int AjoutBateau( Liste_Bateaux_t * liste, Bateau_t * bateau );

/* Verif fin de partie */

int verificationFinPartie( Liste_Bateaux_t * liste );

/* Joue une case (un tir) */

int Joue( Map_t * map, Liste_Bateaux_t * liste, Pos_t * pos );

#endif
```

### **Protoserv.h**

```
/*-----*/
/*   Projet de Fin d'annee de C Unix - ESGI 3 eme Annee   */
/*   Richeton Nicolas et Romelard Fabrice                */
/*   FAB-NICO BATAILLE NAVALE                           */
/*-----*/

#ifndef __PROTOSERV__
#define __PROTOSERV__
```

```
enum ID_MSG
{
    MSG_IDENT=0,
    MSG_REP_IDENT=1,
    MSG_INSCRIPTION=2,
    MSG_REP_INSCRIPTION=3,
    MSG_INIT=5,
    MSG_INIT_JOUEUR=15,
    MSG_GO_PLAY=7,
    MSG_PLAY=8,
    MSG_MISSED=9,
    MSG_HIT=11,
    MSG_HIT_LIST=17,
    MSG_END=13,
    MSG_DECNX=14,
    MSG_ADD_USER=20,
    MSG_REP_ADD_USER=21,
    MSG_DEL_USER=22,
    MSG_REP_DEL_USER=23,
    MSG_CREATE=24,
    MSG_REP_CREATE=29,
    MSG_BATEAU=26,
    MSG_REP_BATEAU=25,
    MSG_DEBUT=30,
    MSG_GETMAP=28,
    MSG_MAP=31,
    MSG_DIFFUSE_MAP=33,
    MSG_DATAS=34
};

/* MSG_IDENT */

typedef struct msg_ident
{
    char nom[ 30 ];
    char password[ 30 ];
} msg_ident_t;

/* MSG_REP_IDENT, MSG_REP_INSCRIPTION */
/* MSG_REP_ADD_USER, MSG_REP_DEL_USER */
/* MSG_REP_CREATE, MSG_REP_BATEAU */

typedef struct msg_rep
{
    char status;
} msg_rep_t;

/* MSG_INIT */

typedef struct msg_init
{
    int nbJoueur;
} msg_init_t;

/* MSG_INIT_JOUEUR */

typedef struct msg_init_joueur
{
    char nom[ 30 ];
    int score;
} msg_init_joueur_t;

/* MSG_PLAY */

typedef struct msg_play
{
    int x;
    int y;
} msg_play_t;

/* MSG_HIT */

typedef struct msg_hit
```

```
{
    char nom[ 30 ];
    char valeur;
    int x;
    int y;
    char dir[ 3 ];
    int lg;
} msg_hit_t;

/* MSG_ADD_USER */

typedef struct msg_add_user
{
    char nom[ 30 ];
    char password[ 30 ];
} msg_add_user_t;

/* MSG_DEL_USER */

typedef struct msg_del_user
{
    char nom[ 30 ];
} msg_del_user_t;

/* MSG_CREATE */

typedef struct msg_create
{
    int x;
    int y;
    int nbJoueurs;
    int nbBateaux;
} msg_create_t;

/* MSG_BATEAU */

typedef struct msg_bateau
{
    int x;
    int y;
    int lg;
    char id;
    char dir[ 3 ];
} msg_bateau_t;

/* MSG_MAP */

typedef struct msg_map
{
    int x;
    int y;
    /* Vient ensuite le nombre de caracteres correspondant */
    /* a x*y */
} msg_map_t;

#endif
```

### Servjeu.c

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                  */
/*  FAB-NICO BATAILLE NAVALE                             */
/*-----*/

#include "socket.h"
#include <signal.h>

/* Drapeau pour signaler une interruption */
int nInterrupt = 0;

/* Handler de signaux */
```

```
void SignalsHandler( int sig )
{
    switch( sig )
    {
        case SIGTERM:
        case SIGINT:
            printf( "Demande d'interruption prise en compte\n" );
            nInterrupt = 1;
            break;

        default:
            nInterrupt = 2;
    }
}

void elimineZombie( int sig )
{
    printf( "Terminaison d'un processus fils\n" );
    wait( NULL );
}

int LanceProcessFils( int socket_service )
{
    int pid;

    switch( pid = fork() )
    {
        case -1 : /* Erreur */
            perror( "Création de processus" );
            break;
        case 0 : /* Processus fils */
            signal( SIGINT, SIG_IGN ); /* CTRL-C ignore par les fils */
            signal( SIGTERM, SignalsHandler ); /* Terminaison controle */
            printf( "Démarrage fils %d\n", getpid() );
            CreationFils( socket_service );
            printf( "Fermeture fils %d\n", getpid() );
            exit( 0 );
            break;
        default : /* Toujours le processus pere */
            printf( "Fils %d lance\n", pid );
    }
    return( pid );
}

int main( int argc, char * argv[] )
{
    struct sockaddr_in adresse;
    int lg_adr;
    int port;
    int socket_ecoute;
    int socket_service;

    if ( argc != 2 )
    {
        printf( "FAB-NICO BATAILLE NAVALE\n");
        printf( "Syntaxe : %s 'port' (ex: %s 6666)\n", argv[ 0 ], argv[ 0 ] );
        return( 1 );
    }

    port = atoi( argv[ 1 ] );

    /* Creation de la socket d'ecoute */

    if ( ( socket_ecoute = creation_socket( SOCK_STREAM, &port, &adresse ) ) == -1 )
    {
        fprintf( stderr, "FAB-NICO BATAILLE NAVALE\n");
        fprintf( stderr, "Création socket écoute impossible sur le port %d\n", port );
        return( -1 );
    }

    initialisationSHM();

    /* Declaration d'ouverture du service */

    if ( listen( socket_ecoute, SOMAXCONN ) == -1 )
```

```
{
    perror( "listen" );
    close( socket_ecoute );
    return( -1 );
}

/* Boucle d'attente de connexion */

signal( SIGTERM, SignalsHandler ); /* Terminaison controlee */
signal( SIGCHLD, elimineZombie ); /* Terminaison controlee */

nInterrupt = 0;

while ( nInterrupt == 0 )
{
    printf( "FAB-NICO BATAILLE NAVALE\n");
    printf( "Serveur en attente de connexion...\n" );
    socket_service = accept( socket_ecoute, &adresse, &lg_adr );

    /* Reception d'un signal (probablement SIGCHLD) */

    if ( socket_service == -1 && errno == EINTR )
    {
        continue;
    }

    /* Erreur grave */

    if ( socket_service == -1 )
    {
        perror( "accept" );
        return( -1 );
    }

    printf( "Nouvelle connexion sur le port %d\n", socket_service );
    LanceProcessFils( socket_service );
}
}
```

### **Socket.c**

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                    */
/*  FAB-NICO BATAILLE NAVALE                                */
/*-----*/

#include "socket.h"

/* Adresse globale socket d'ecoute */
static struct sockaddr_in adr_in;

int creation_socket( int type, int * ptr_port, struct sockaddr_in * ptr_adr )
{
    int desc;
    int lg = sizeof( struct sockaddr_in );

    /* Creation de la socket */

    if( ( desc = socket( AF_INET, type, 0 ) ) == -1 )
    {
        perror( "Création de la socket impossible" );
        return( -1 );
    }

    /* Creation de l'adresse d'attachement */

    adr_in.sin_family = AF_INET;
    adr_in.sin_addr.s_addr = htonl( INADDR_ANY );
    adr_in.sin_port = htons( *ptr_port );

    /* Demande d'attachement de la socket */
}
```

```
if ( bind( desc, &adr_in, lg ) == -1 )
{
    perror( "Attachement de la socket impossible" );
    close( desc );
    return( -1 );
}

/* Récupération de l'adresse effective d'attachement */

if ( ptr_adr != NULL )
{
    getsockname( desc, ptr_adr, &lg );
}

return( desc );
}
```

### **Socket.h**

```
/*-----*/
/*  Projet de Fin d'annee de C Unix - ESGI 3 eme Annee      */
/*  Richeton Nicolas et Romelard Fabrice                    */
/*  FAB-NICO BATAILLE NAVALE                                */
/*-----*/

#ifndef __socket__

#include <errno.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <unistd.h>

int creation_socket( int type, int * ptr_port, struct sockaddr_in * ptr_adr);

#endif
```