TP MULTITACHE REALISATION

B3330

Paul-Emmanuel SOTIR

Victoire CHAPELLE

# MAIN.CPP

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main - Tache mere de l'application

----------------------------------

debut : 18/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

///////////////////////////////////////////////////////////////// INCLUDE

//------------------------------------------------------- Include systeme

#include <string>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/sem.h>

#include <sys/ipc.h>

#include <sys/wait.h>

#include <sys/types.h>

//------------------------------------------------------ Include personnel

#include "process\_utils.h"

#include "gestionSortie.h"

#include "gestionEntree.h"

#include "clavier.h"

#include "Outils.h"

#include "Heure.h"

/////////////////////////////////////////////////////////////////// PRIVE

namespace

{

//-------------------------------------------------------------- Types

//! Structure regroupant les pid des processus fils et les outils de

//! communication entre processus.

struct

{

pid\_t heure\_pid = -1;

pid\_t clavier\_pid = -1;

pid\_t porte\_gb\_pid = -1;

pid\_t porte\_bp\_autres\_pid = -1;

pid\_t porte\_bp\_profs\_pid = -1;

pid\_t porte\_sortie\_pid = -1;

ipc\_id\_t parking\_state\_id = 0;

ipc\_id\_t waiters\_id = 0;

ipc\_id\_t parking\_places\_id = 0;

ipc\_id\_t entrance\_mqid = -1;

ipc\_id\_t exit\_mqid = -1;

ipc\_id\_t semaphores\_id = -1;

} ressources;

//-------------------------------------------------- Fonctions privees

//! Termine la tache mère et ses taches filles

void quit\_app()

{

// Fonction lambda utilisée pour terminer une tache

auto kill\_task = [](pid\_t task, const std::string& message, signal\_t sig = SIGUSR2)

{

if (task >= 0)

{

kill(task, sig);

Effacer(MESSAGE);

Afficher(MESSAGE, message.data());

waitpid(task, nullptr, 0);

}

};

// Delete application tasks

kill\_task(ressources.porte\_bp\_profs\_pid, "FIN porte bp profs");

kill\_task(ressources.porte\_bp\_autres\_pid, "FIN porte bp autres");

kill\_task(ressources.porte\_gb\_pid, "FIN porte gb");

kill\_task(ressources.porte\_sortie\_pid, "FIN porte sortie");

kill\_task(ressources.clavier\_pid, "FIN clavier");

kill\_task(ressources.heure\_pid, "FIN heure");

// Suppression des message queues

delete\_message\_queue(ressources.entrance\_mqid);

delete\_message\_queue(ressources.exit\_mqid);

// Suppression de la memoire partagée

delete\_shared\_memory(ressources.parking\_state\_id);

delete\_shared\_memory(ressources.waiters\_id);

delete\_shared\_memory(ressources.parking\_places\_id);

// Suppression des semaphores

semctl(ressources.semaphores\_id, 0, IPC\_RMID, 0);

// On retire le handler deu signal SIGCHLD

unsubscribe\_handler<SIGCHLD>();

TerminerApplication();

exit(EXIT\_SUCCESS);

}

//! Crée les taches filles et les ressources pour l'ipc

void init()

{

// Fonction lambda utilisée pour quitter en cas d'erreur

auto quit\_if\_failed = [](bool condition) {

if (!condition)

quit\_app();

};

InitialiserApplication(XTERM);

// On ajoute un handler pour le signal SIGCHLD indiquant qu'une tache fille s'est

// terminée

quit\_if\_failed(handle<SIGCHLD>([](signal\_t sig) {

quit\_app(); // On quitte proprement

}));

// Création de la 'message queue' utilisée pour communiquer entre le clavier et les

// barrières

quit\_if\_failed(create\_message\_queue(ressources.entrance\_mqid, 1));

// Création de la 'message queue' utilisée pour communiquer entre le clavier et la

// sortie

quit\_if\_failed(create\_message\_queue(ressources.exit\_mqid, 2));

// Création de la mémoire partagée (utilisée par les barrieres)

ressources.parking\_state\_id = create\_detached\_shared\_mem<parking>(4);

quit\_if\_failed(ressources.parking\_state\_id != -1);

ressources.waiters\_id = create\_detached\_shared\_mem<waiting\_cars>(5);

quit\_if\_failed(ressources.waiters\_id != -1);

ressources.parking\_places\_id = create\_detached\_shared\_mem<places>(6);

quit\_if\_failed(ressources.parking\_places\_id != -1);

// Creation des semaphores pour l'acces a la memoire partagée et l'attente au niveau

// des barrières en cas de parking plein

ressources.semaphores\_id = semget(ftok(".", 3), 4U, IPC\_CREAT | 0600);

quit\_if\_failed(ressources.semaphores\_id != -1);

quit\_if\_failed(semctl(ressources.semaphores\_id, 0, SETVAL, 1) != -1);

quit\_if\_failed(semctl(ressources.semaphores\_id, PROF\_BLAISE\_PASCAL, SETVAL, 0) != -1);

quit\_if\_failed(semctl(ressources.semaphores\_id, AUTRE\_BLAISE\_PASCAL, SETVAL, 0) != -1);

quit\_if\_failed(semctl(ressources.semaphores\_id, ENTREE\_GASTON\_BERGER, SETVAL, 0) !=-1);

// Creation de la tache 'heure'

ressources.heure\_pid = ActiverHeure();

quit\_if\_failed(ressources.heure\_pid != -1);

// Creation de la tache 'clavier'

ressources.clavier\_pid = activer\_clavier();

quit\_if\_failed(ressources.clavier\_pid != -1);

// Creation des taches 'gestionEntree' pour chaque barrieres a l'entree

ressources.porte\_gb\_pid = ActiverPorte(ENTREE\_GASTON\_BERGER);

quit\_if\_failed(ressources.porte\_gb\_pid != -1);

ressources.porte\_bp\_profs\_pid = ActiverPorte(PROF\_BLAISE\_PASCAL);

quit\_if\_failed(ressources.porte\_bp\_profs\_pid != -1);

ressources.porte\_bp\_autres\_pid = ActiverPorte(AUTRE\_BLAISE\_PASCAL);

quit\_if\_failed(ressources.porte\_bp\_autres\_pid != -1);

// Creation de la tache 'gestionSortie'

ressources.porte\_sortie\_pid = ActiverPorteSortie();

quit\_if\_failed(ressources.porte\_sortie\_pid != -1);

}

}

////////////////////////////////////////////////////////////////// PUBLIC

//----------------------------------------------------- Fonctions publique

//! Tache mère

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

{

// Creation des taches filles et des ressources pour l'ipc

init();

// Attente de la fin de la tache clavier

waitpid(ressources.clavier\_pid, nullptr, 0);

// Destruction de la tache mere

quit\_app();

}

# PROCESS\_UTILS.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Process utils - aide a la manipulation de processus

---------------------------------------------------

debut : 20/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef PROCESS\_UTILS\_H

#define PROCESS\_UTILS\_H

///////////////////////////////////////////////////////////////// INCLUDE

//-------------------------------------------------- Interfaces utilis�es

#include <array>

#include <signal.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/msg.h>

#include <algorithm>

#include <functional>

#include <sys/types.h>

//----------------------------------------------------------------- Types

using signal\_t = int;

using handler\_t = void(\*)(signal\_t);

using sa\_falg\_t = int;

using ipc\_id\_t = int;

template<typename T>

struct shared\_mem

{

ipc\_id\_t id;

T\* data;

};

////////////////////////////////////////////////////////////////// PUBLIC

//---------------------------------------------------- Fonctions publiques

//! Permet de creer un processus fils executant la fonction 'code\_fils'.

//! Le parametre optionnel 'code\_pere' permet de specifier du code specifique

//! au processus pere (avec pour parametre le pid du processus fils).

//! La fonction retourne le pid du processus fils.

pid\_t fork(const std::function<void(void)>& code\_fils

, const std::function<void(pid\_t)>& code\_pere = [](pid\_t) {});

//! Permet de gerer la reception de signaux. Les signaux qui seront geres

//! sont specifies sous la forme d'arguments template (variadic template).

//! La reception d'un des signaux specifies declanchera l'appel de la

//! donnee en parametre 'handler'.

//! Le parametre optionnel 'flags' permet de specifier les flags qui

//! seront donnes aux appels a 'sigaction'.

//! La fonction retourne true si tout les handlers pu etre inscrit à un signal.

//! Exemple pour gerer les signaux SIGUSR1 et SIGINT :

//! bool success = handle<SIGUSR1, SIGINT>([](signal\_t sig)

//! {

//! /\* code a executer lors de la reception d'un signal \*/

//! });

template<signal\_t... signals>

bool handle(handler\_t handler, sa\_falg\_t falgs = SA\_RESTART)

{

if (handler != nullptr)

{

struct sigaction action;

action.sa\_handler = handler;

sigemptyset(&action.sa\_mask);

action.sa\_flags = falgs;

// On execute sigaction pour recouvrir chaque signals en recuperant la valeur retournee

std::array<int, sizeof...(signals)> return\_codes = { sigaction(signals, &action

, nullptr)... };

// On verifie si des appels a sigaction on retournes '-1' (erreur)

if (std::find(std::begin(return\_codes), std::end(return\_codes), -1)

!= std::end(return\_codes))

return false;

return true;

}

return false;

}

//! Permet de remettre le comportement par defaut à la reception des signaux 'signals...'.

//! Les signaux concernés sont specifies sous la forme d'arguments template (variadic template).

//! La fonction retourne true si tout les signaux ont été desinscrits.

template<signal\_t... signals>

bool unsubscribe\_handler()

{

// Sigaction permettant de remettre le comportement par defaut

struct sigaction unsub\_action;

unsub\_action.sa\_handler = SIG\_DFL;

sigemptyset(&unsub\_action.sa\_mask);

unsub\_action.sa\_flags = 0;

// On execute sigaction pour chaque signal donné en paramètre template

std::array<int, sizeof...(signals)> return\_codes = { sigaction(signals, &unsub\_action

, nullptr)... };

// On verifie si des appels à sigaction on retournés '-1' (erreur)

if (std::find(std::begin(return\_codes), std::end(return\_codes), -1) != std::end(return\_codes)) return false;

return true;

}

//! Envoie un message sur la 'message queue' dont l'id est donne en parametre.

//! Le type 'Buffer' est le type du message envoye devant etre conigue et contenir

//! un attribut 'long mtype' identifiant le type du message.

template<typename Buffer>

bool send\_message(ipc\_id\_t msqid, const Buffer& message)

{

return msgsnd(msqid, &message, sizeof(message) - sizeof(long), 0) != -1;

}

//! Recoit un message issu de la 'message queue' dont l'id est donne en parametre.

//! Le message recut est ecrit dans 'qbuf'.

//! Les messages recus sont filtres en fonction de leur types (parametre 'type').

//! Si la lecture a echouee, la fonction renvoie 'false'.

//! Un appel a cette fonction est bloquant (appel systeme bloquant).

template<typename Buffer>

bool read\_message(ipc\_id\_t msqid, long type, Buffer& qbuf)

{

int rslt = 0;

do

{

rslt = msgrcv(msqid, &qbuf, sizeof(Buffer) - sizeof(long), type, 0);

} while (rslt == -1 && errno == EINTR);

return rslt >= 0;

}

//! Ouvre une 'message queue' dont l'id sera donne dans 'msquid'.

//! La message queue est identifiee par les parametres 'id' et 'path' permettant

//! de creer une cle (ftok).

//! Le parametre 'permission' permet de specifier les autorisation d'acces a la

//! 'message queue' (600 par defaut).

bool open\_message\_queue(ipc\_id\_t& msqid, int id, int permission = 0600, std::string path = std::string("."));

//! Crée une 'message queue' dont l'id sera donne dans 'msquid'.

//! La message queue est identifiee par les parametres 'id' et 'path' permettant

//! de creer une cle (ftok).

//! Le parametre 'permission' permet de specifier les autorisation d'acces a la

//! 'message queue' (600 par defaut).

//! Le parametre optionnel 'fail\_if\_exist' permet de choisir si la creation doit

//! echouer si la message\_queue existe deja ('false' par defaut).

bool create\_message\_queue(ipc\_id\_t& msqid, int id, int permission = 0600

, std::string path = std::string("."), bool fail\_if\_exist = false);

//! Supprime la 'message\_queue' dont l'id est donne en parametre.

//! Retourne 'true' si la suppression a bien eut lieu.

bool delete\_message\_queue(ipc\_id\_t msqid);

namespace

{

template<typename T>

shared\_mem<T> get\_or\_create\_shared\_memory(ipc\_id\_t id, int permisssion, std::string path

, bool fail\_if\_exist, bool create, bool attach = true)

{

// Cree une cle identifiant la memoire partagee

key\_t key = ftok(path.data(), id);

if (key == -1)

return{ -1, nullptr };

// Cree ou ouvre la memoire partagee

int flags = permisssion;

if (create)

flags |= IPC\_CREAT;

if (fail\_if\_exist)

flags |= IPC\_EXCL;

auto shmid = shmget(key, sizeof(T), flags);

if (shmid == -1)

return{ -1, nullptr };

if (attach)

{

// Recupere un pointeur vers la memoire partagee

T\* data = reinterpret\_cast<T\*>(shmat(shmid, nullptr, 0));

if (data == reinterpret\_cast<T\*>(-1))

return{ -1, nullptr };

// Execute le constructeur du type 'T' a l'adresse allouee

if (create)

new (data) T();

return{ shmid, data };

}

return{ shmid, nullptr };

}

}

//! Ouvre une memoire partagée pour un objet de type 'T'.

//! La memoire partagée est identifiee par les parametres 'id' et 'path' permettant

//! de creer une cle (ftok).

//! Le parametre 'permission' permet de specifier les autorisation d'acces a la

//! memoire partagée (600 par defaut).

//! Le type 'T' doit disposer d'un constructeur par defaut.

template<typename T>

shared\_mem<T> get\_shared\_memory(ipc\_id\_t id, int permisssion = 0600, std::string path = ".")

{

return get\_or\_create\_shared\_memory<T>(id, permisssion, path, false, false);

}

//! Crée une memoire partagée pour un objet de type 'T'.

//! La memoire partagée est identifiee par les parametres 'id' et 'path' permettant

//! de creer une cle (ftok).

//! Le parametre 'permission' permet de specifier les autorisation d'acces a la

//! memoire partagée (600 par defaut).

//! Le parametre optionnel 'fail\_if\_exist' permet de choisir si la creation doit

//! echouer si la memoire partagée existe deja ('false' par defaut).

//! Le type 'T' doit disposer d'un constructeur par defaut.

template<typename T>

shared\_mem<T> create\_shared\_memory(ipc\_id\_t id, int permisssion = 0600, std::string path = "."

, bool fail\_if\_exist = false)

{

return get\_or\_create\_shared\_memory<T>(id, permisssion, path, fail\_if\_exist, true);

}

//! Detache l'objet de type 'T' de la mémoire partagée

//! Retourne 'true' si l'operation a eut lieu sans erreurs.

template<typename T>

bool detach\_shared\_memory(T& shared\_data)

{

return (shmdt(&shared\_data) != -1);

}

//! Crée une memoire partagée pour un objet de type 'T' sans l'attacher au processus courant.

//! La memoire partagée est identifiee par les parametres 'id' et 'path' permettant

//! de creer une cle (ftok).

//! Le parametre 'permission' permet de specifier les autorisation d'acces a la

//! memoire partagée (600 par defaut).

//! Le parametre optionnel 'fail\_if\_exist' permet de choisir si la creation doit

//! echouer si la memoire partagée existe deja ('false' par defaut).

//! Le type 'T' doit disposer d'un constructeur par defaut.

//! Retourne l'id de la mémoire partagée ou -1 en cas d'erreur

template<typename T>

ipc\_id\_t create\_detached\_shared\_mem(ipc\_id\_t id, int permisssion = 0600, std::string path = "."

, bool fail\_if\_exist = false)

{

auto memory = get\_or\_create\_shared\_memory<T>(id, permisssion, path, fail\_if\_exist, true, false);

return memory.id;

}

//! Détruit la mémoire partagée identifiée par sont id donné en paramètre.

//! Retourne 'true' si la suppression a bien eut lieu.

bool delete\_shared\_memory(ipc\_id\_t id);

//! Ajoute la valeur 'num\_to\_add' à la 'sem\_num'-ème semaphore du tableau de

//! semaphores 'sems\_id' (ajoute 1 par defaut).

bool sem\_pv(ipc\_id\_t sems\_id, short unsigned int sem\_num, short num\_to\_add = 1);

//! Protège/execute la fonction 'func' donnée en paramètre avec la 'sem\_num'-ème

//! semaphore du tableau de semaphores 'sems\_id'.

bool lock(ipc\_id\_t sems\_id, short unsigned int sem\_num, const std::function<void(void)>& func);

#endif // PROCESS\_UTIL

# PROCESS\_UTILS.CPP

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Process utils - aide a la manipulation de processus

---------------------------------------------------

debut : 20/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

///////////////////////////////////////////////////////////////// INCLUDE

//-------------------------------------------------------- Include systeme

#include <string>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <mqueue.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/stat.h>

#include <functional>

#include <sys/types.h>

//------------------------------------------------------ Include personnel

#include "process\_utils.h"

#include "Outils.h"

/////////////////////////////////////////////////////////////////// PRIVE

//------------------------------------------------------ Fonctions privees

namespace

{

bool create\_or\_open\_message\_queue(ipc\_id\_t& msqid, int id, int perm, std::string path, int flags)

{

key\_t key = ftok(path.data(), id);

if (key == -1)

return false;

msqid = msgget(key, perm | flags);

return msqid >= 0;

}

}

////////////////////////////////////////////////////////////////// PUBLIC

//----------------------------------------------------- Fonctions publique

pid\_t fork(const std::function<void(void)>& code\_fils, const std::function<void(pid\_t)>& code\_pere)

{

pid\_t fils\_pid;

if ((fils\_pid = fork()) == 0)

code\_fils();

else

code\_pere(fils\_pid);

return fils\_pid;

}

bool open\_message\_queue(ipc\_id\_t& msqid, int id, int permission, std::string path)

{

return create\_or\_open\_message\_queue(msqid, id, permission, path, 0);

}

bool create\_message\_queue(ipc\_id\_t& msqid, int id, int permission, std::string path, bool fail\_if\_exist)

{

return create\_or\_open\_message\_queue(msqid, id, permission, path, fail\_if\_exist ? (IPC\_CREAT | IPC\_EXCL) : IPC\_CREAT);

}

bool delete\_message\_queue(ipc\_id\_t msqid)

{

return msgctl(msqid, IPC\_RMID, nullptr) != -1;

}

bool delete\_shared\_memory(ipc\_id\_t id)

{

return (shmctl(id, IPC\_RMID, nullptr) != -1);

}

bool sem\_pv(ipc\_id\_t sems\_id, short unsigned int sem\_num, short num\_to\_add)

{

sembuf sem\_buf{ sem\_num, num\_to\_add, 0 };

int rslt = 0;

do { rslt = semop(sems\_id, &sem\_buf, 1); } while (rslt == -1 && errno == EINTR);

return rslt >= 0;

}

bool lock(ipc\_id\_t sems\_id, short unsigned int sem\_num, const std::function<void(void)>& func)

{

// Decremente la semaphore

if (!sem\_pv(sems\_id, sem\_num, -1))

return false;

// Execute le code protégé par la semaphore

func();

// Incremente la semaphore

return sem\_pv(sems\_id, sem\_num, 1);

# CLAVIER.H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Clavier - tache gerant l'entree clavier

-----------------------------------------

debut : 18/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//---------- Interface du module <CLAVIER> (fichier clavier.h) -----------

#ifndef CLAVIER\_H

#define CLAVIER\_H

///////////////////////////////////////////////////////////////// INCLUDE

//--------------------------------------------------- Interfaces utilisees

#include <sys/types.h>

#include <stdlib.h>

#include "Outils.h"

//------------------------------------------------------------------ Types

//! Structure utilisee par la message queue entre le clavier et les barrieres

struct car\_incomming\_msg

{

long barriere\_type;

TypeUsager type\_usager;

};

//! Structure utilisee par la message queue entre le clavier et la sortie

struct car\_exit\_msg

{

long place\_num;

};

//---------------------------------------------------- Fonctions publiques

//! Démare la tache gerant l'interface console

//! Retourne le PID du processus fils ou -1 en cas d'echec.

pid\_t activer\_clavier();

//! Gere les commandes déclanchées par le menu

void Commande(char code, unsigned int valeur);

#endif // CLAVIER\_H

# CLAVIER.CPP

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Clavier - tache gerant l'entree clavier

-----------------------------------------

debut : 18/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//---------- Realisation du module <CLAVIER> (fichier clavier.cpp) -------

///////////////////////////////////////////////////////////////// INCLUDE

//-------------------------------------------------------- Include système

#include <string>

#include <errno.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <signal.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <functional>

#include <sys/types.h>

//------------------------------------------------------ Include personnel

#include "process\_utils.h"

#include "clavier.h"

#include "Outils.h"

#include "Menu.h"

/////////////////////////////////////////////////////////////////// PRIVE

namespace

{

//------------------------------------------------ Variables statiques

static ipc\_id\_t incomming\_car\_mq\_id;

static ipc\_id\_t car\_exit\_mq\_id;

//-------------------------------------------------- Fonctions privees

//! Termine la tache gèrant l'interface console

void quit\_clavier()

{

// On retire le handler du signal SIGUSR2

unsubscribe\_handler<SIGUSR2>();

// On notifie la tache mère de la fin pour qu'elle quitte les autres taches

kill(getppid(), SIGCHLD);

exit(EXIT\_SUCCESS);

}

}

////////////////////////////////////////////////////////////////// PUBLIC

//---------------------------------------------------- Fonctions publiques

pid\_t activer\_clavier()

{

// Creation de la tache clavier

return fork([]()

{

// On ajoute un handler pour le signal SIGUSR2 indiquant que la tache doit se terminer

if (!(handle<SIGUSR2>([](signal\_t sig) { quit\_clavier(); })))

quit\_clavier();

// Ouverture de la 'message queue' utilisée pour communiquer avec les taches des

// barrières

if (!open\_message\_queue(incomming\_car\_mq\_id, 1))

quit\_clavier();

// Ouverture de la 'message queue' utilisée pour communiquer avec la tache de sortie

if (!open\_message\_queue(car\_exit\_mq\_id, 2))

quit\_clavier();

// Execute la phase moteur (interface console)

while (true)

Menu();

});

}

void Commande(char code, unsigned int valeur)

{

switch (code)

{

case 'e':

case 'E':

quit\_clavier();

break;

case 'p':

case 'P':

// On notifie la tache gerant la barriere de l'arrivee de la voiture

car\_incomming\_msg msg={(valeur == 1 ? PROF\_BLAISE\_PASCAL:ENTREE\_GASTON\_BERGER), PROF};

if (!send\_message(incomming\_car\_mq\_id, msg))

quit\_clavier();

break;

case 'a':

case 'A':

// On notifie la tache gerant la barriere de l'arrivee de la voiture

car\_incomming\_msg msg={(valeur == 1 ? AUTRE\_BLAISE\_PASCAL:ENTREE\_GASTON\_BERGER), PROF};

if (!send\_message(incomming\_car\_mq\_id, msg))

quit\_clavier();

break;

case 's':

case 'S':

// On notifie la tache de sortie qu'une place doit être liberée (s’il y a une voiture)

if (!send\_message(car\_exit\_mq\_id, car\_exit\_msg{ valeur }))

quit\_clavier();

break;

}

}

# GESTION\_ENTREE.H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Gestion entree - tache gerant les barrieres d'entree

----------------------------------------------------

debut : 18/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//--- Interface du module <GESTION\_ENTREE> (fichier gestionEntree.h) -----

#ifndef GESTION\_ENTREE\_H

#define GESTION\_ENTREE\_H

///////////////////////////////////////////////////////////////// INCLUDE

//--------------------------------------------------- Interfaces utilisées

#include <array>

#include "Outils.h"

#include "process\_utils.h"

//------------------------------------------------------------------ Types

//! Type contenant les information nescessaires sur une voiture

//! Ce type est utilise sur de la memoire partagee.

struct car\_info

{

// Les constructeurs ne sont plus nescessaires en C++14 pour avoir des initilizer list + des

// valeurs par defaut

car\_info() = default;

car\_info(unsigned int num, TypeUsager type, time\_t t)

: car\_number(num), user\_type(type), heure\_arrivee(t) { };

volatile unsigned int car\_number = 0U;

volatile int user\_type = AUCUN;

volatile time\_t heure\_arrivee = 0;

};

//! Represente une requete d'un voiture voulant entrer dans le parking plein

//! Ce type est utilise sur de la memoire partagee.

struct request

{

// Les constructeurs ne sont plus nescessaires en C++14 pour avoir des initilizer list + des

// valeurs par defaut

request() = default;

request(TypeUsager t, time\_t h) : type(t), heure\_requete(h) { };

volatile TypeUsager type = AUCUN;

volatile time\_t heure\_requete = 0;

};

//! Struture contenant le prochain id donne a une voiture et le nombre de

//! voitures dans le parking.

//! Ce type est utilise sur de la memoire partagee.

struct parking

{

volatile unsigned int next\_car\_id = 1U;

volatile unsigned int car\_count = 0U;

bool is\_full() const { return car\_count >= NB\_PLACES; }

};

//! Ce type est utilise sur de la memoire partagee.

struct places

{ car\_info places[NB\_PLACES]; };

//! Ce type est utilisé sur de la memoire partagée.

struct waiting\_cars

{ request waiting\_fences[NB\_BARRIERES\_ENTREE]; };

//---------------------------------------------------- Fonctions publiques

//! Démare la tache gèrant la barrière identifiée par sont type (TypeBarriere)

//! Retourne le PID du processus fils ou -1 en cas d'echec.

pid\_t ActiverPorte(TypeBarriere t);

#endif // GESTION\_ENTREE\_H

# GESTION\_ENTREE.CPP

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Gestion entree - tache gerant les barrieres d'entree

----------------------------------------------------

debut : 18/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//-- Realisaion de la tache <GESTION\_ENTREE> (fichier gestionEntree.cpp) -

///////////////////////////////////////////////////////////////// INCLUDE

//-------------------------------------------------------- Include système

#include <unordered\_map>

#include <sys/wait.h>

#include <sys/sem.h>

#include <unistd.h>

#include <utility>

#include <string>

#include <ctime>

//------------------------------------------------------ Include personnel

#include "gestionEntree.h"

#include "process\_utils.h"

#include "clavier.h"

#include "Outils.h"

/////////////////////////////////////////////////////////////////// PRIVE

namespace

{

//------------------------------------------------ Variables statiques

//! On utilise des variables statiques plutot que automatiques pour qu'elles

//! soient accessibles depuis les handlers de signaux car ce sont des lambdas

//! qui ne peuvent rien capturer de leur contexte pour qu'elles puissent être

//! castés en pointeur de fonction C.

ipc\_id\_t car\_message\_queue\_id;

ipc\_id\_t sem\_id;

shared\_mem<parking> parking\_state;

shared\_mem<waiting\_cars> waiting;

shared\_mem<places> parking\_places;

std::unordered\_map<pid\_t, car\_info> car\_parking\_tasks;

//-------------------------------------------------- Fonctions privées

void quit\_entree()

{

// On retire les handlers des signaux SIGCHLD et SIGUSR2

unsubscribe\_handler<SIGCHLD, SIGUSR2>();

// On quitte tout les processus garant des voitures

for (const auto& parking\_task : car\_parking\_tasks) {

if (parking\_task.first > 0)

{

kill(parking\_task.first, SIGUSR2);

waitpid(parking\_task.first, nullptr, 0);

}

}

// Detache 'parking\_state' de la memoire partagée

detach\_shared\_memory(parking\_state.data);

detach\_shared\_memory(waiting.data);

detach\_shared\_memory(parking\_places.data);

// On notifie la tache mère de la fin pour qu'elle quitte les autres taches

kill(getppid(), SIGCHLD);

exit(EXIT\_SUCCESS);

}

}

////////////////////////////////////////////////////////////////// PUBLIC

//----------------------------------------------------- Fonctions publique

pid\_t ActiverPorte(TypeBarriere type)

{

// Creation du processus fils

return fork([type]()

{

// Fonction lambda aidant a quitter en cas d'erreur

auto quit\_if\_failed = [](bool condition) {

if (!condition)

quit\_entree();

};

// On ajoute un handler pour le signal SIGUSR2 indiquant que la tache doit se terminer

quit\_if\_failed(handle<SIGUSR2>([](signal\_t sig) { quit\_entree(); }));

// Ouverture de la message queue utilisée pour recevoir les voitures

quit\_if\_failed(open\_message\_queue(car\_message\_queue\_id, 1));

// Récuperation des semaphores

sem\_id = semget(ftok(".", 3), 4U, 0600);

quit\_if\_failed(sem\_id != -1);

// Ouvre les memoires partagées

parking\_state = get\_shared\_memory<parking>(4);

quit\_if\_failed(parking\_state.data != nullptr);

waiting = get\_shared\_memory<waiting\_cars>(5);

quit\_if\_failed(waiting.data != nullptr);

parking\_places = get\_shared\_memory<places>(6);

quit\_if\_failed(parking\_places.data != nullptr);

// On ajoute un handler pour le signal SIGCHLD indiquant qu'une voiture s'est garée

quit\_if\_failed(handle<SIGCHLD>([](signal\_t sig)

{

int status;

pid\_t chld = waitpid(-1, &status, WNOHANG);

if (chld > 0 && WIFEXITED(status))

{

// On affiche la place et on met à jour la memoire partagée

auto car = car\_parking\_tasks[chld];

auto place\_num = WEXITSTATUS(status);

lock(sem\_id, 0, [&car, place\_num]() {

parking\_places.data->places[place\_num - 1] = car;

});

car\_parking\_tasks.erase(chld);

AfficherPlace(WEXITSTATUS(status), (TypeUsager)car.user\_type

, car.car\_number, car.heure\_arrivee);

}

}));

// Phase moteur

while (true)

{

// On attend la reception d'un message de la tache clavier

car\_incomming\_msg message;

quit\_if\_failed(read\_message(car\_message\_queue\_id, type, message));

DessinerVoitureBarriere(type, message.type\_usager);

AfficherRequete(type, message.type\_usager, time(nullptr));

bool is\_full = false;

unsigned int next\_car\_id = 0;

do

{

quit\_if\_failed(lock(sem\_id, 0, [&message, type, &is\_full

, &next\_car\_id]()

{

// On vérifie que le parking n'est pas (de nouveau) plein

// avant de laisser passer la voiture

is\_full = parking\_state.data->is\_full();

if (!is\_full) {

next\_car\_id = parking\_state.data->next\_car\_id++;

parking\_state.data->car\_count++;

}

else

waiting.data->waiting\_fences[type - 1] =

{ message.type\_usager, time(nullptr) };

}));

if (is\_full)

// On attend qu'une place se libère

sem\_pv(sem\_id, type, -1);

} while (is\_full);

lock(sem\_id, 0, [type]() {

waiting.data->waiting\_fences[type - 1] = { AUCUN, 0 };

});

// La barrière s'ouvre et on gare la voiture

pid\_t garage\_pid = GarerVoiture(type);

quit\_if\_failed(garage\_pid);

car\_parking\_tasks.emplace(garage\_pid

, car\_info{ next\_car\_id, message.type\_usager, time(nullptr) });

sleep(1); // Temps pour avancer...

Effacer(static\_cast<TypeZone>(REQUETE\_R1 + type - 1));

}

});

}

# GESTION\_SORTIE.H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Gestion sortie - tache gerant les barrieres de sortie

-----------------------------------------------------

debut : 18/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//-- Interface de la tache <GESTION\_SORTIE> (fichier gestionSortie.h) ----

#ifndef GESTION\_SORTIE\_H

#define GESTION\_SORTIE\_H

///////////////////////////////////////////////////////////////// INCLUDE

//--------------------------------------------------- Interfaces utilisées

#include "process\_utils.h"

#include "gestionEntree.h"

//---------------------------------------------------- Fonctions publiques

//! Demare la tache gestionSortie pour permettre aux voitures de sortir

//! Retourne le PID du processus fils ou -1 en cas d'echec.

pid\_t ActiverPorteSortie();

#endif // GESTION\_SORTIE\_H

# GESTION\_SORTIE.CPP

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Gestion sortie - tache gerant les barrieres de sortie

-----------------------------------------------------

debut : 18/03/2016

binome : B3330

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//- Realisaion de la tache <GESTION\_SORTIE> (fichier gestionSortie.cpp) --

///////////////////////////////////////////////////////////////// INCLUDE

//-------------------------------------------------------- Include système

#include <unordered\_map>

#include <sys/wait.h>

#include <sys/sem.h>

#include <unistd.h>

#include <stdlib.h>

#include <utility>

#include <string>

#include <ctime>

//------------------------------------------------------ Include personnel

#include "gestionSortie.h"

#include "gestionEntree.h"

#include "process\_utils.h"

#include "clavier.h"

/////////////////////////////////////////////////////////////////// PRIVE

namespace

{

//------------------------------------------------ Variables statiques

//! On utilise des variables statiques plutot que automatiques pour qu'elles

//! soient accessibles depuis les handlers de signaux car ce sont des lambdas

//! qui ne peuvent rien capturer de leur contexte pour qu'elles puissent être

//! castés en pointeur de fonction C.

ipc\_id\_t car\_message\_queue\_id;

ipc\_id\_t sem\_id;

shared\_mem<parking> parking\_state;

shared\_mem<waiting\_cars> waiting;

shared\_mem<places> parking\_places;

std::unordered\_map<pid\_t, car\_info> car\_exiting\_tasks;

//-------------------------------------------------- Fonctions privées

//! Termine la tache Gestion sortie

void quit\_sortie()

{

// On retire les handlers des signaux SIGCHLD et SIGUSR2

unsubscribe\_handler<SIGCHLD, SIGUSR2>();

// Termine toutes les taches filles de sortie de voiture

for (const auto& task : car\_exiting\_tasks) {

if (task.first > 0)

{

kill(task.first, SIGUSR2);

waitpid(task.first, nullptr, 0);

}

}

// Detache les memoires partagées

detach\_shared\_memory(parking\_state.data);

detach\_shared\_memory(waiting.data);

detach\_shared\_memory(parking\_places.data);

// On notifie la tache mère de la fin pour qu'elle quitte les autres taches

kill(getppid(), SIGCHLD);

exit(EXIT\_SUCCESS);

}

}

////////////////////////////////////////////////////////////////// PUBLIC

//----------------------------------------------------- Fonctions publique

pid\_t ActiverPorteSortie()

{

// Creation du processus fils

return fork([]()

{

// Fonction lambda aidant a quitter en cas d'erreur

auto quit\_if\_failed = [](bool condition)

{

if (!condition)

quit\_sortie();

};

// On ajoute un handler pour le signal SIGUSR2 indiquant que la tache doit se terminer

quit\_if\_failed(handle<SIGUSR2>([](signal\_t sig) { quit\_sortie(); }));

// Ouverture de la message queue utilisée pour recevoir les demandes de sortie

quit\_if\_failed(open\_message\_queue(car\_message\_queue\_id, 2));

// Récuperation des semaphores

sem\_id = semget(ftok(".", 3), 4U, 0600);

quit\_if\_failed(sem\_id != -1);

// Ouvre les memoires partagées

parking\_state = get\_shared\_memory<parking>(4);

quit\_if\_failed(parking\_state.data != nullptr);

waiting = get\_shared\_memory<waiting\_cars>(5);

quit\_if\_failed(waiting.data != nullptr);

parking\_places = get\_shared\_memory<places>(6);

quit\_if\_failed(parking\_places.data != nullptr);

// On ajoute un handler pour le signal SIGCHLD indiquant qu'une voiture est sortie

quit\_if\_failed(handle<SIGCHLD>([](signal\_t sig)

{

// On affiche les information de sortie, on efface les informations de la

// place et on met à jour la memoire partagée

int status;

pid\_t chld = waitpid(-1, &status, WNOHANG);

if (chld > 0 && WIFEXITED(status))

{

auto car = car\_exiting\_tasks[chld];

auto place\_num = WEXITSTATUS(status);

request req\_prof, req\_gb, req\_autre;

lock(sem\_id, 0, [&car, place\_num, &req\_prof, &req\_gb, &req\_autre]() {

// Met a jour le nombre de voitures dans le parking

if (parking\_state.data->car\_count > 0)

parking\_state.data->car\_count--;

req\_prof = waiting.data->waiting\_fences[PROF\_BLAISE\_PASCAL-1];

req\_gb = waiting.data->waiting\_fences[ENTREE\_GASTON\_BERGER-1];

req\_autre=waiting.data->waiting\_fences[AUTRE\_BLAISE\_PASCAL-1];

});

// Laisse entrer une voiture attendant à une barrière en prenant en

// compte les priorités

if (req\_prof.type != AUCUN && !(req\_gb.type == PROF

&& req\_gb.heure\_requete < req\_prof.heure\_requete)) {

// Ont débloque la barrière PROF\_BLAISE\_PASCAL

sem\_pv(sem\_id, PROF\_BLAISE\_PASCAL, 1);

}

else if (req\_gb.type != AUCUN

&& (req\_gb.type == PROF

|| !(req\_autre.type != AUCUN && req\_gb.type == AUTRE

&& req\_autre.heure\_requete < req\_gb.heure\_requete))) {

// Ont débloque la barrière ENTREE\_GASTON\_BERGER

sem\_pv(sem\_id, ENTREE\_GASTON\_BERGER, 1);

}

else if (req\_autre.type != AUCUN) {

// Ont débloque la barrière AUTRE\_BLAISE\_PASCAL

sem\_pv(sem\_id, AUTRE\_BLAISE\_PASCAL, 1);

}

car\_exiting\_tasks.erase(chld);

Effacer(static\_cast<TypeZone>(ETAT\_P1 + place\_num - 1));

AfficherSortie((TypeUsager)car.user\_type, car.car\_number

, car.heure\_arrivee, time(nullptr));

}

}));

// Phase moteur

while (true)

{

// On attend la reception d'un message issu de la tache clavier

car\_exit\_msg message;

quit\_if\_failed(read\_message(car\_message\_queue\_id, 0, message));

// On recupère les informations a la place spécifiée dans la memoire partagée

car\_info car;

quit\_if\_failed(lock(sem\_id, 0, [&car, &message]() {

car = parking\_places.data->places[message.place\_num - 1];

parking\_places.data->places[message.place\_num - 1] = { 0, AUCUN, 0 };

}));

// On sort la voiture si une voiture occupe la place spécifiée

pid\_t child\_pid = SortirVoiture(message.place\_num);

car\_exiting\_tasks.emplace(child\_pid, car);

}

});

}

# MAKEFILE

############################################################################################

################################## GENERIC MAKEFILE ########################################

############################################################################################

# TODO: gerer les sous-dossiers / fichiers ayants le mêmes noms dans des dossiers différts

# TODO: gerer les extentions .hpp, .cxx, ...

# Debug mode (comment this line to build project in release mode)

DEBUG = true

# Compiler

CC = g++

# Command used to remove files

RM = rm -f

# Compiler and pre-processor options

CPPFLAGS = -Wall -std=c++11 -Ofast

# Debug flags

DEBUGFLAGS = -g

# Resulting program file name

EXE\_NAME = Parking

# The source file extentions

SRC\_EXT = cpp

# The header file types

# TODO permettre les .hpp

HDR\_EXT = h

# Source directory

SRCDIR = source

# Headers directory

INCDIR = include

# Main output directory

OUTPUT\_DIR = bin

# Release output directory

RELEASEDIR = release

# Debug output directory

DEBUGDIR = debug

# Dependency files directory

DEPDIR = dep

# Libraries paths

LIB\_DIRS = -L libs

# Library file names (e.g. '-lboost\_serialization-mt')

LIBS = -ltp -ltcl8.5 -lncurses

# List of include paths

INCLUDES = ./$(INCDIR)

ifdef DEBUG

BUILD\_PATH = ./$(OUTPUT\_DIR)/$(DEBUGDIR)

else

DEBUGFLAGS =

BUILD\_PATH = ./$(OUTPUT\_DIR)/$(RELEASEDIR)

endif

# Source directory path

SRC\_PATH = ./$(SRCDIR)

# Dependencies path

DEP\_PATH = ./$(BUILD\_PATH)/$(DEPDIR)

# List of source files

SOURCES = $(wildcard $(SRC\_PATH)/\*.$(SRC\_EXT))

# List of object files

OBJS = $(SOURCES:$(SRC\_PATH)/%.$(SRC\_EXT)=$(BUILD\_PATH)/%.o)

# List of dependency files

DEPS = $(SOURCES:$(SRC\_PATH)/%.$(SRC\_EXT)=$(DEP\_PATH)/%.d)

.PHONY: all clean rebuild

all: $(BUILD\_PATH)/$(EXE\_NAME)

clean:

#dos2unix clear\_ipc.sh

./clear\_ipc.sh

$(RM) $(BUILD\_PATH)/\*.o

$(RM) $(BUILD\_PATH)/$(EXE\_NAME)

$(RM) $(DEP\_PATH)/\*.d

rebuild: clean all

# Build object files

$(BUILD\_PATH)/%.o: $(SRC\_PATH)/%.$(SRC\_EXT)

@mkdir -p $(DEP\_PATH)

$(CC) $(CPPFLAGS) $(DEBUGFLAGS) -I $(INCLUDES) -MMD -MP -MF $(DEP\_PATH)/$\*.d -c $<

-o $@

# Build main target

$(BUILD\_PATH)/$(EXE\_NAME): $(OBJS)

$(CC) $(LIB\_DIRS) -o $(BUILD\_PATH)/$(EXE\_NAME) $(OBJS) $(LIBS)

# Copie l'executable dans le dossier principal

cp $(BUILD\_PATH)/$(EXE\_NAME) ./$(EXE\_NAME