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)
```

```
56                  {
57                          if (task >= 0)
58                          {
59                                  kill(task, sig);
60                                  Effacer(MESSAGE);
61                                  Afficher(MESSAGE, message.data());
62                                  waitpid(task, nullptr, 0);
63                          }
64                  };
65
66                  // Delete application tasks
67                  kill_task(ressources.porte_bp_profs_pid, "FIN porte bp profs");
68                  kill_task(ressources.porte_bp_autres_pid, "FIN porte bp autres");
69                  kill_task(ressources.porte_gb_pid, "FIN porte gb");
70                  kill_task(ressources.porte_sortie_pid, "FIN porte sortie");
71                  kill_task(ressources.clavier_pid, "FIN clavier");
72                  kill_task(ressources.heure_pid, "FIN heure");
73
74                  // Suppression des message queues
75                  delete_message_queue(ressources.entrance_mqid);
76                  delete_message_queue(ressources.exit_mqid);
77
78                  // Suppression de la memoire partagée
79                  delete_shared_memory(ressources.parking_state_id);
80                  delete_shared_memory(ressources.waiters_id);
81                  delete_shared_memory(ressources.parking_places_id);
82
83                  // Suppression des semaphores
84                  semctl(ressources.semaphores_id, 0, IPC_RMID, 0);
85
86                  // On retire le handler deu signal SIGCHLD
87                  unsubscribe_handler<SIGCHLD>();
88
89                  TerminerApplication();
90                  exit(EXIT_SUCCESS);
91          }
92
93          //! Crée les taches filles et les ressources pour l'ipc
94          void init()
95          {
96                  // Fonction lambda utilisée pour quitter en cas d'erreur
97                  auto quit_if_failed = [](bool condition) {
98                          if (!condition)
99                                  quit_app();
100                 };
101
102                 InitialiserApplication(XTERM);
103
104                 // On ajoute un handler pour le signal SIGCHLD indiquant qu'une tache fille s'est
105                 // terminée
106                 quit_if_failed(handle<SIGCHLD>([](signal_t sig) {
107                         quit_app(); // On quitte proprement
108                 }));
109                 // Création de la 'message queue' utilisée pour communiquer entre le clavier et les
110                 // barrières
111                 quit_if_failed(create_message_queue(ressources.entrance_mqid, 1));
112
113                 // Création de la 'message queue' utilisée pour communiquer entre le clavier et la
114                 // sortie
115                 quit_if_failed(create_message_queue(ressources.exit_mqid, 2));
116
117                 // Création de la mémoire partagée (utilisée par les barrieres)
118                 ressources.parking_state_id = create_detached_shared_mem<parking>(4);
119                 quit_if_failed(ressources.parking_state_id != -1);
120                 ressources.waiters_id = create_detached_shared_mem<waiting_cars>(5);
121                 quit_if_failed(ressources.waiters_id != -1);
122                 ressources.parking_places_id = create_detached_shared_mem<places>(6);
123                 quit_if_failed(ressources.parking_places_id != -1);
124
125                 // Creation des semaphores pour l'acces a la memoire partagée et l'attente au niveau
126                 // des barrières en cas de parking plein
127                 ressources.semaphores_id = semget(ftok(".", 3), 4U, IPC_CREAT | 0600);
128                 quit_if_failed(ressources.semaphores_id != -1);
129                 quit_if_failed(semctl(ressources.semaphores_id, 0, SETVAL, 1) != -1);
```

```
130                 quit_if_failed(semctl(ressources.semaphores_id, PROF_BLAISE_PASCAL, SETVAL, 0) != -1);
131                 quit_if_failed(semctl(ressources.semaphores_id, AUTRE_BLAISE_PASCAL, SETVAL, 0) != -1);
132                 quit_if_failed(semctl(ressources.semaphores_id, ENTREE_GASTON_BERGER, SETVAL, 0) !=-1);
133
134                 // Creation de la tache 'heure'
135                 ressources.heure_pid = ActiverHeure();
136                 quit_if_failed(ressources.heure_pid != -1);
137
138                 // Creation de la tache 'clavier'
139                 ressources.clavier_pid = activer_clavier();
140                 quit_if_failed(ressources.clavier_pid != -1);
141
142                 // Creation des taches 'gestionEntree' pour chaque barrieres a l'entree
143                 ressources.porte_gb_pid = ActiverPorte(ENTREE_GASTON_BERGER);
144                 quit_if_failed(ressources.porte_gb_pid != -1);
145                 ressources.porte_bp_profs_pid = ActiverPorte(PROF_BLAISE_PASCAL);
146                 quit_if_failed(ressources.porte_bp_profs_pid != -1);
147                 ressources.porte_bp_autres_pid = ActiverPorte(AUTRE_BLAISE_PASCAL);
148                 quit_if_failed(ressources.porte_bp_autres_pid != -1);
149
150                 // Creation de la tache 'gestionSortie'
151                 ressources.porte_sortie_pid = ActiverPorteSortie();
152                 quit_if_failed(ressources.porte_sortie_pid != -1);
153         }
154 }
155
156 ////////////////////////////////////////////////////////////////  PUBLIC
157 //------------------------------------------------------ Fonctions publique
158 //! Tache mère
159 int main(int argc, char* argv[])
160 {
161         // Creation des taches filles et des ressources pour l'ipc
162         init();
163
164         // Attente de la fin de la tache clavier
165         waitpid(ressources.clavier_pid, nullptr, 0);
166
167         // Destruction de la tache mere
168         quit_app();
169 }
```

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)...
};

```

```
72
73                  // On verifie si des appels a sigaction on retournes '-1' (erreur)
74                  if (std::find(std::begin(return_codes), std::end(return_codes), -1)
75                                                                          != std::end(return_codes))
76                          return false;
77                  return true;
78          }
79          return false;
80  }
81
82  //! Permet de remettre le comportement par defaut à la reception des signaux 'signals...'.
83  //! Les signaux concernés sont specifies sous la forme d'arguments template (variadic template).
84  //! La fonction retourne true si tout les signaux ont été desinscrits.
85  template<signal_t... signals>
86  bool unsubscribe_handler()
87  {
88          // Sigaction permettant de remettre le comportement par defaut
89          struct sigaction unsub_action;
90          unsub_action.sa_handler = SIG_DFL;
91          sigemptyset(&unsub_action.sa_mask);
92          unsub_action.sa_flags = 0;
93
94          // On execute sigaction pour chaque signal donné en paramètre template
95          std::array<int, sizeof...(signals)> return_codes = { sigaction(signals, &unsub_action
96                                                                          , nullptr)... };
97
98          // On verifie si des appels à sigaction on retournés '-1' (erreur)
99          if (std::find(std::begin(return_codes), std::end(return_codes), -1) != std::end(return_codes))
100                 return false;
101         return true;
102 }
103
104 //! Envoie un message sur la 'message queue' dont l'id est donne en parametre.
105 //! Le type 'Buffer' est le type du message envoye devant etre conigue et contenir
106 //! un attribut 'long mtype' identifiant le type du message.
107 template<typename Buffer>
108 bool send_message(ipc_id_t msqid, const Buffer& message)
109 {
110         return msgsnd(msqid, &message, sizeof(message) - sizeof(long), 0) != -1;
111 }
112
113 //! Recoit un message issu de la 'message queue' dont l'id est donne en parametre.
114 //! Le message recut est ecrit dans 'qbuf'.
115 //! Les messages recus sont filtres en fonction de leur types (parametre 'type').
116 //! Si la lecture a echouee, la fonction renvoie 'false'.
117 //! Un appel a cette fonction est bloquant (appel systeme bloquant).
118 template<typename Buffer>
119 bool read_message(ipc_id_t msqid, long type, Buffer& qbuf)
120 {
121         int rslt = 0;
122         do
123         {
124                 rslt = msgrcv(msqid, &qbuf, sizeof(Buffer) - sizeof(long), type, 0);
125         } while (rslt == -1 && errno == EINTR);
126         return rslt >= 0;
127 }
128
129 //! Ouvre une 'message queue' dont l'id sera donne dans 'msquid'.
130 //! La message queue est identifiee par les parametres 'id' et 'path' permettant
131 //! de creer une cle (ftok).
132 //! Le parametre 'permission' permet de specifier les autorisation d'acces a la
133 //! 'message queue' (600 par defaut).
134 bool open_message_queue(ipc_id_t& msqid, int id, int permission = 0600, std::string path =
135 std::string("."));
136
137 //! Crée une 'message queue' dont l'id sera donne dans 'msquid'.
138 //! La message queue est identifiee par les parametres 'id' et 'path' permettant
139 //! de creer une cle (ftok).
140 //! Le parametre 'permission' permet de specifier les autorisation d'acces a la
141 //! 'message queue' (600 par defaut).
142 //! Le parametre optionnel 'fail_if_exist' permet de choisir si la creation doit
143 //! echouer si la message_queue existe deja ('false' par defaut).
144 bool create_message_queue(ipc_id_t& msqid, int id, int permission = 0600
145                                 , std::string path = std::string("."), bool fail_if_exist = false);
```

```
146
147	//! Supprime la 'message_queue' dont l'id est donne en parametre.
148	//! Retourne 'true' si la suppression a bien eut lieu.
149	bool delete_message_queue(ipc_id_t msqid);
150
151	namespace
152	{
153		template<typename T>
154		shared_mem<T> get_or_create_shared_memory(ipc_id_t id, int permisssion, std::string path
155		                                          , bool fail_if_exist, bool create, bool attach = true)
156		{
157			// Cree une cle identifiant la memoire partagee
158			key_t key = ftok(path.data(), id);
159			if (key == -1)
160				return{ -1, nullptr };
161
162			// Cree ou ouvre la memoire partagee
163			int flags = permisssion;
164			if (create)
165				flags |= IPC_CREAT;
166			if (fail_if_exist)
167				flags |= IPC_EXCL;
168			auto shmid = shmget(key, sizeof(T), flags);
169			if (shmid == -1)
170				return{ -1, nullptr };
171
172			if (attach)
173			{
174				// Recupere un pointeur vers la memoire partagee
175				T* data = reinterpret_cast<T*>(shmat(shmid, nullptr, 0));
176				if (data == reinterpret_cast<T*>(-1))
177					return{ -1, nullptr };
178
179				// Execute le constructeur du type 'T' a l'adresse allouee
180				if (create)
181					new (data) T();
182
183				return{ shmid, data };
184			}
185			return{ shmid, nullptr };
186		}
187	}
188
189	//! Ouvre une memoire partagée pour un objet de type 'T'.
190	//! La memoire partagée est identifiee par les parametres 'id' et 'path' permettant
191	//! de creer une cle (ftok).
192	//! Le parametre 'permission' permet de specifier les autorisation d'acces a la
193	//! memoire partagée (600 par defaut).
194	//! Le type 'T' doit disposer d'un constructeur par defaut.
195	template<typename T>
196	shared_mem<T> get_shared_memory(ipc_id_t id, int permisssion = 0600, std::string path = ".")
197	{
198		return get_or_create_shared_memory<T>(id, permisssion, path, false, false);
199	}
200
201	//! Crée une memoire partagée pour un objet de type 'T'.
202	//! La memoire partagée est identifiee par les parametres 'id' et 'path' permettant
203	//! de creer une cle (ftok).
204	//! Le parametre 'permission' permet de specifier les autorisation d'acces a la
205	//! memoire partagée (600 par defaut).
206	//! Le parametre optionnel 'fail_if_exist' permet de choisir si la creation doit
207	//! echouer si la memoire partagée existe deja ('false' par defaut).
208	//! Le type 'T' doit disposer d'un constructeur par defaut.
209	template<typename T>
210	shared_mem<T> create_shared_memory(ipc_id_t id, int permisssion = 0600, std::string path = "."
211	                                   , bool fail_if_exist = false)
212	{
213		return get_or_create_shared_memory<T>(id, permisssion, path, fail_if_exist, true);
214	}
215
216	//! Detache l'objet de type 'T' de la mémoire partagée
217	//! Retourne 'true' si l'operation a eut lieu sans erreurs.
218	template<typename T>
219	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;
```

```
72  }
73
74  bool delete_shared_memory(ipc_id_t id)
75  {
76          return (shmctl(id, IPC_RMID, nullptr) != -1);
77  }
78
79  bool sem_pv(ipc_id_t sems_id, short unsigned int sem_num, short num_to_add)
80  {
81          sembuf sem_buf{ sem_num, num_to_add, 0 };
82          int rslt = 0;
83
84          do { rslt = semop(sems_id, &sem_buf, 1); } while (rslt == -1 && errno == EINTR);
85
86          return rslt >= 0;
87  }
88
89  bool lock(ipc_id_t sems_id, short unsigned int sem_num, const std::function<void(void)>& func)
90  {
91          // Decremente la semaphore
92          if (!sem_pv(sems_id, sem_num, -1))
93                  return false;
94
95          // Execute le code protégé par la semaphore
96          func();
97
98          // Incremente la semaphore
99          return sem_pv(sems_id, sem_num, 1);
```

CLAVIER.H

```
1  /*************************************************************************
2               Clavier  -  tache gerant l'entree clavier
3               -------------------------------------
4       debut         : 18/03/2016
5       binome        : B3330
6  *************************************************************************/
7
8  //---------- Interface du module <CLAVIER> (fichier clavier.h) -----------
9  #ifndef CLAVIER_H
10 #define CLAVIER_H
11
12 //////////////////////////////////////////////////////////////  INCLUDE
13 //-------------------------------------------------- Interfaces utilisees
14 #include <sys/types.h>
15 #include <stdlib.h>
16 #include "Outils.h"
17
18 //------------------------------------------------------------------ Types
19 //! Structure utilisee par la message queue entre le clavier et les barrieres
20 struct car_incomming_msg
21 {
22       long barriere_type;
23       TypeUsager type_usager;
24 };
25
26 //! Structure utilisee par la message queue entre le clavier et la sortie
27 struct car_exit_msg
28 {
29       long place_num;
30 };
31
32 //------------------------------------------------------ Fonctions publiques
33 //! Démare la tache gerant l'interface console
34 //! Retourne le PID du processus fils ou -1 en cas d'echec.
35 pid_t activer_clavier();
36
37 //! Gere les commandes déclanchées par le menu
38 void Commande(char code, unsigned int valeur);
39
40 #endif // CLAVIER_H
```

CLAVIER.CPP

```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)
```

```
72                          Menu();
73          });
74  }
75
76  void Commande(char code, unsigned int valeur)
77  {
78          switch (code)
79          {
80          case 'e':
81          case 'E':
82                  quit_clavier();
83                  break;
84          case 'p':
85          case 'P':
86                  // On notifie la tache gerant la barriere de l'arrivee de la voiture
87                  car_incomming_msg msg={(valeur == 1 ? PROF_BLAISE_PASCAL:ENTREE_GASTON_BERGER), PROF};
88                  if (!send_message(incomming_car_mq_id, msg))
89                          quit_clavier();
90                  break;
91          case 'a':
92          case 'A':
93                  // On notifie la tache gerant la barriere de l'arrivee de la voiture
94                  car_incomming_msg msg={(valeur == 1 ? AUTRE_BLAISE_PASCAL:ENTREE_GASTON_BERGER), PROF};
95                  if (!send_message(incomming_car_mq_id, msg))
96                          quit_clavier();
97                  break;
98          case 's':
99          case 'S':
100                 // On notifie la tache de sortie qu'une place doit être liberée (s'il y a une voiture)
101                 if (!send_message(car_exit_mq_id, car_exit_msg{ valeur }))
102                         quit_clavier();
103                 break;
104         }
105 }
```

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

```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)
{
```

```
72          // Creation du processus fils
73          return fork([type]()
74          {
75                  // Fonction lambda aidant a quitter en cas d'erreur
76                  auto quit_if_failed = [](bool condition) {
77                          if (!condition)
78                                  quit_entree();
79                  };
80
81                  // On ajoute un handler pour le signal SIGUSR2 indiquant que la tache doit se terminer
82                  quit_if_failed(handle<SIGUSR2>([](signal_t sig) { quit_entree(); }));
83
84                  // Ouverture de la message queue utilisée pour recevoir les voitures
85                  quit_if_failed(open_message_queue(car_message_queue_id, 1));
86
87                  // Récuperation des semaphores
88                  sem_id = semget(ftok(".", 3), 4U, 0600);
89                  quit_if_failed(sem_id != -1);
90
91                  // Ouvre les memoires partagées
92                  parking_state = get_shared_memory<parking>(4);
93                  quit_if_failed(parking_state.data != nullptr);
94                  waiting = get_shared_memory<waiting_cars>(5);
95                  quit_if_failed(waiting.data != nullptr);
96                  parking_places = get_shared_memory<places>(6);
97                  quit_if_failed(parking_places.data != nullptr);
98
99                  // On ajoute un handler pour le signal SIGCHLD indiquant qu'une voiture s'est garée
100                 quit_if_failed(handle<SIGCHLD>([](signal_t sig)
101                 {
102                         int status;
103                         pid_t chld = waitpid(-1, &status, WNOHANG);
104                         if (chld > 0 && WIFEXITED(status))
105                         {
106                                 // On affiche la place et on met à jour la memoire partagée
107                                 auto car = car_parking_tasks[chld];
108                                 auto place_num = WEXITSTATUS(status);
109
110                                 lock(sem_id, 0, [&car, place_num]() {
111                                         parking_places.data->places[place_num - 1] = car;
112                                 });
113
114                                 car_parking_tasks.erase(chld);
115                                 AfficherPlace(WEXITSTATUS(status), (TypeUsager)car.user_type
116                                                                         , car.car_number, car.heure_arrivee);
117                         }
118                 }));
119
120                 // Phase moteur
121                 while (true)
122                 {
123                         // On attend la reception d'un message de la tache clavier
124                         car_incomming_msg message;
125                         quit_if_failed(read_message(car_message_queue_id, type, message));
126
127                         DessinerVoitureBarriere(type, message.type_usager);
128                         AfficherRequete(type, message.type_usager, time(nullptr));
129
130                         bool is_full = false;
131                         unsigned int next_car_id = 0;
132                         do
133                         {
134                                 quit_if_failed(lock(sem_id, 0, [&message, type, &is_full
135                                                                                 , &next_car_id]()
136                                 {
137                                         // On vérifie que le parking n'est pas (de nouveau) plein
138                                         // avant de laisser passer la voiture
139                                         is_full = parking_state.data->is_full();
140                                         if (!is_full) {
141                                                 next_car_id = parking_state.data->next_car_id++;
142                                                 parking_state.data->car_count++;
143                                         }
144                                         else
145                                                 waiting.data->waiting_fences[type - 1] =
```

```
146                                                     { message.type_usager, time(nullptr) };
147                             }));
148
149                             if (is_full)
150                                     // On attend qu'une place se libère
151                                     sem_pv(sem_id, type, -1);
152                     } while (is_full);
153
154                     lock(sem_id, 0, [type]() {
155                             waiting.data->waiting_fences[type - 1] = { AUCUN, 0 };
156                     });
157
158                     // La barrière s'ouvre et on gare la voiture
159                     pid_t garage_pid = GarerVoiture(type);
160                     quit_if_failed(garage_pid);
161                     car_parking_tasks.emplace(garage_pid
162                             , car_info{ next_car_id, message.type_usager, time(nullptr) });
163
164                     sleep(1); // Temps pour avancer...
165                     Effacer(static_cast<TypeZone>(REQUETE_R1 + type - 1));
166             }
167     });
168 }
```

GESTION_SORTIE.H

```
1  /*************************************************************************
2               Gestion sortie - tache gerant les barrieres de sortie
3               -------------------------------------------------
4        debut                : 18/03/2016
5        binome               : B3330
6  *************************************************************************/
7
8  //-- Interface de la tache <GESTION_SORTIE> (fichier gestionSortie.h) ----
9  #ifndef GESTION_SORTIE_H
10 #define GESTION_SORTIE_H
11
12 //////////////////////////////////////////////////////////////  INCLUDE
13 //--------------------------------------------------- Interfaces utilisées
14 #include "process_utils.h"
15 #include "gestionEntree.h"
16
17 //------------------------------------------------------ Fonctions publiques
18
19 //! Demare la tache gestionSortie pour permettre aux voitures de sortir
20 //! Retourne le PID du processus fils ou -1 en cas d'echec.
21 pid_t ActiverPorteSortie();
22
23 #endif // GESTION_SORTIE_H
```

GESTION_SORTIE.CPP

```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
```

```
72
73  pid_t ActiverPorteSortie()
74  {
75          // Creation du processus fils
76          return fork([]()
77          {
78                  // Fonction lambda aidant a quitter en cas d'erreur
79                  auto quit_if_failed = [](bool condition)
80                  {
81                          if (!condition)
82                                  quit_sortie();
83                  };
84
85                  // On ajoute un handler pour le signal SIGUSR2 indiquant que la tache doit se terminer
86                  quit_if_failed(handle<SIGUSR2>([](signal_t sig) { quit_sortie(); }));
87
88                  // Ouverture de la message queue utilisée pour recevoir les demandes de sortie
89                  quit_if_failed(open_message_queue(car_message_queue_id, 2));
90
91                  // Récuperation des semaphores
92                  sem_id = semget(ftok(".", 3), 4U, 0600);
93                  quit_if_failed(sem_id != -1);
94
95                  // Ouvre les memoires partagées
96                  parking_state = get_shared_memory<parking>(4);
97                  quit_if_failed(parking_state.data != nullptr);
98                  waiting = get_shared_memory<waiting_cars>(5);
99                  quit_if_failed(waiting.data != nullptr);
100                 parking_places = get_shared_memory<places>(6);
101                 quit_if_failed(parking_places.data != nullptr);
102
103                 // On ajoute un handler pour le signal SIGCHLD indiquant qu'une voiture est sortie
104                 quit_if_failed(handle<SIGCHLD>([](signal_t sig)
105                 {
106                         // On affiche les information de sortie, on efface les informations de la
107                         // place et on met à jour la memoire partagée
108                         int status;
109                         pid_t chld = waitpid(-1, &status, WNOHANG);
110                         if (chld > 0 && WIFEXITED(status))
111                         {
112                                 auto car = car_exiting_tasks[chld];
113                                 auto place_num = WEXITSTATUS(status);
114
115                                 request req_prof, req_gb, req_autre;
116                                 lock(sem_id, 0, [&car, place_num, &req_prof, &req_gb, &req_autre]() {
117                                         // Met a jour le nombre de voitures dans le parking
118                                         if (parking_state.data->car_count > 0)
119                                                 parking_state.data->car_count--;
120
121                                         req_prof = waiting.data->waiting_fences[PROF_BLAISE_PASCAL-1];
122                                         req_gb = waiting.data->waiting_fences[ENTREE_GASTON_BERGER-1];
123                                         req_autre=waiting.data->waiting_fences[AUTRE_BLAISE_PASCAL-1];
124                                 });
125
126                                 // Laisse entrer une voiture attendant à une barrière en prenant en
127                                 // compte les priorités
128                                 if (req_prof.type != AUCUN && !(req_gb.type == PROF
129                                                 && req_gb.heure_requete < req_prof.heure_requete)) {
130                                         // Ont débloque la barrière PROF_BLAISE_PASCAL
131                                         sem_pv(sem_id, PROF_BLAISE_PASCAL, 1);
132                                 }
133                                 else if (req_gb.type != AUCUN
134                                                 && (req_gb.type == PROF
135                                                 || !(req_autre.type != AUCUN && req_gb.type == AUTRE
136                                                 && req_autre.heure_requete < req_gb.heure_requete))) {
137                                         // Ont débloque la barrière ENTREE_GASTON_BERGER
138                                         sem_pv(sem_id, ENTREE_GASTON_BERGER, 1);
139                                 }
140                                 else if (req_autre.type != AUCUN) {
141                                         // Ont débloque la barrière AUTRE_BLAISE_PASCAL
142                                         sem_pv(sem_id, AUTRE_BLAISE_PASCAL, 1);
143                                 }
144
145
```

```
146                                 car_exiting_tasks.erase(chld);
147                                 Effacer(static_cast<TypeZone>(ETAT_P1 + place_num - 1));
148                                 AfficherSortie((TypeUsager)car.user_type, car.car_number
149                                                                         , car.heure_arrivee, time(nullptr));
150                         }
151                 }));
152
153                 // Phase moteur
154                 while (true)
155                 {
156                         // On attend la reception d'un message issu de la tache clavier
157                         car_exit_msg message;
158                         quit_if_failed(read_message(car_message_queue_id, 0, message));
159
160                         // On recupère les informations a la place spécifiée dans la memoire partagée
161                         car_info car;
162                         quit_if_failed(lock(sem_id, 0, [&car, &message]() {
163                                 car = parking_places.data->places[message.place_num - 1];
164                                 parking_places.data->places[message.place_num - 1] = { 0, AUCUN, 0 };
165                         }));
166
167                         // On sort la voiture si une voiture occupe la place spécifiée
168                         pid_t child_pid = SortirVoiture(message.place_num);
169                         car_exiting_tasks.emplace(child_pid, car);
170                 }
171         });
172 }
```

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
```