2017

Réseaux et Technologie Internet

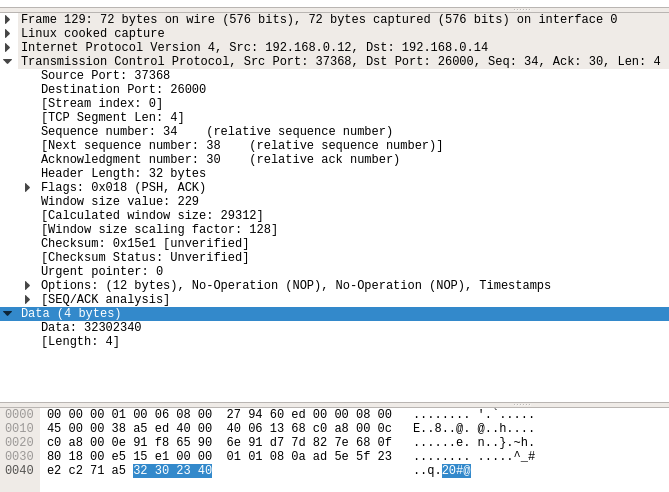
Laurent Reynders

# Tableau de commandes

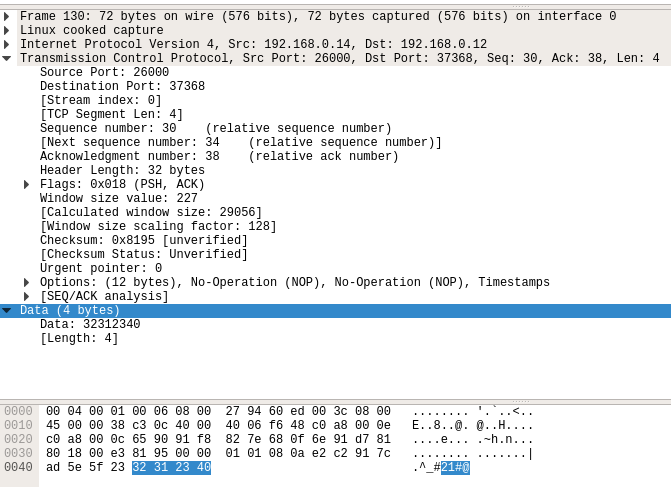
|  |  |  |
| --- | --- | --- |
| Commande | Sémantique | Réponse éventuelle |
| LOGIN\_OFFICER | Un travailleur de l’aéroport veut se connecter.  *Paramètres : login + mot de passe* | * Oui (validation à l’aide du fichier Login.csv) (LOGIN\_OFFICER\_OK) * Non + la raison (LOGIN\_OFFICER\_KO) |
| LOGOUT\_OFFICER | Le travailleur veut se déconnecter sans pour autant éteindre le client.  *Paramètres : -* | * Oui (LOGOUT\_OFFICER\_OK) * Non + la raison (LOGOUT\_OFFICER\_KO) |
| CHECK\_TICKET | Le travailleur veut vérifier si le numéro de ticket du client est bien valide.  *Paramètres : structure contenant le numéro de ticket et le nombre de compagnons* | * Oui (validation à l’aide du fichier Ticket.csv) (CHECK\_TICKET\_OK) * Non + la raison (CHECK\_TICKET\_KO) |
| CHECK\_LUGGAGE | Le travailleur encode les bagages avant de les mettre sur le tapis.  *Paramètres : vecteur de structures, chacune contenant le numéro de ticket, le poids, et si c’est une valise ou non* | * Oui (calcul du tarif pour surpoids et sauvegarde temporaire en attente du payement) (CHECK\_LUGGAGE\_OK) Retour d’une structure avec le prix supplémentaire, le poids total des bagages, l’excédent de poids, et le numéro de ticket. * Non + la raison (CHECK\_LUGGAGE\_KO) |
| PAYMENT\_DONE | Le travailleur indique si le client a payé pour le supplément ou pas.  *Paramètres : un booléen pour l’indication de payement.* | * Oui (PAYMENT\_DONE\_OK) * Non = la raison (PAYMENT\_DONE\_KO) |
|  |  |  |
|  |  |  |

# Trames vues sur WireShark

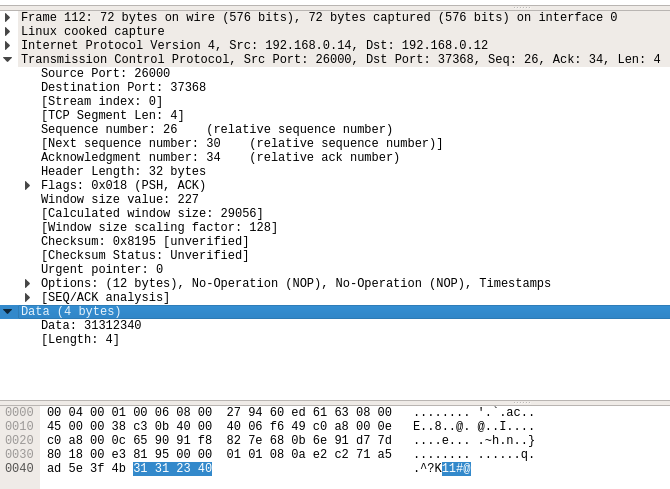
## LOGOUT\_OFFICER



LOGOUT\_OFFICER\_OK



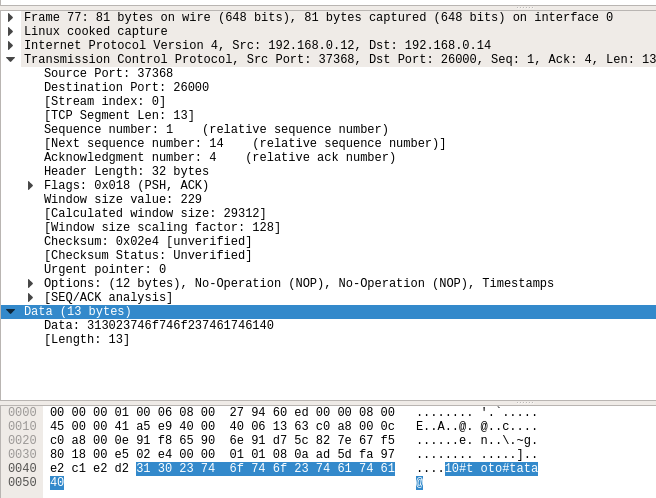
## LOGIN\_OFFICER\_OK



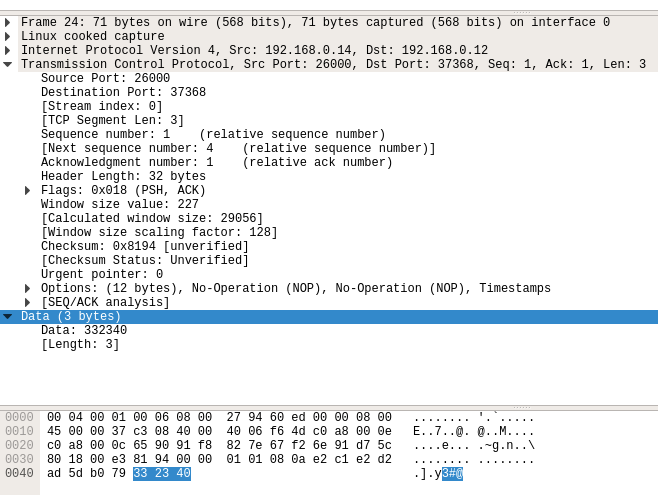
## LOGIN\_OFFICER\_KO

## ../../../../../../Desktop/Screen%20Shot%202017-10-02%20at%201

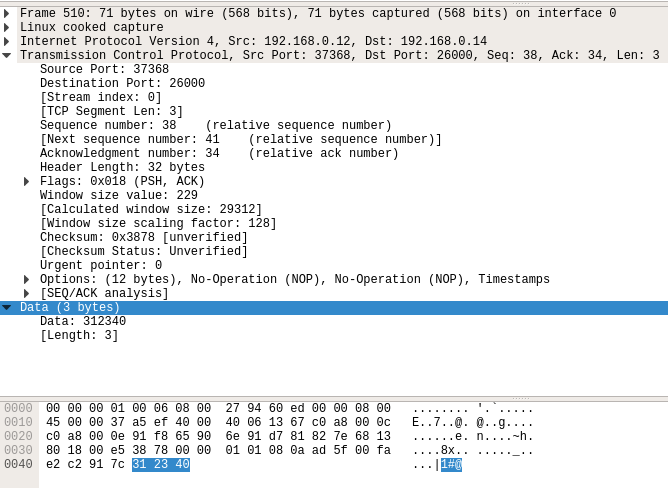
## LOGIN\_OFFICER



## OK



## EOC



## PAYMENT\_DONE\_OK

## ../../../../../../Desktop/Screen%20Shot%202017-10-02%20at%201

## PAYMENT\_DONE

## ../../../../../../Desktop/Screen%20Shot%202017-10-02%20at%201

## CHECK\_LUGGAGE\_OK

## ../../../../../../Desktop/Screen%20Shot%202017-10-02%20at%201

## CHECK\_LUGGAGE

## ../../../../../../Desktop/Screen%20Shot%202017-10-02%20at%201

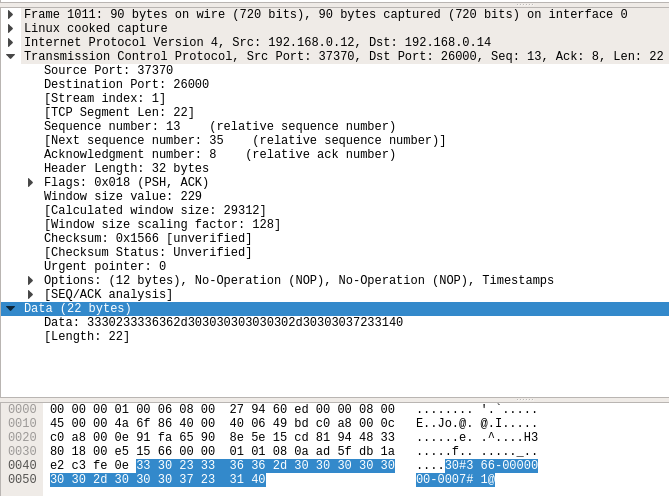
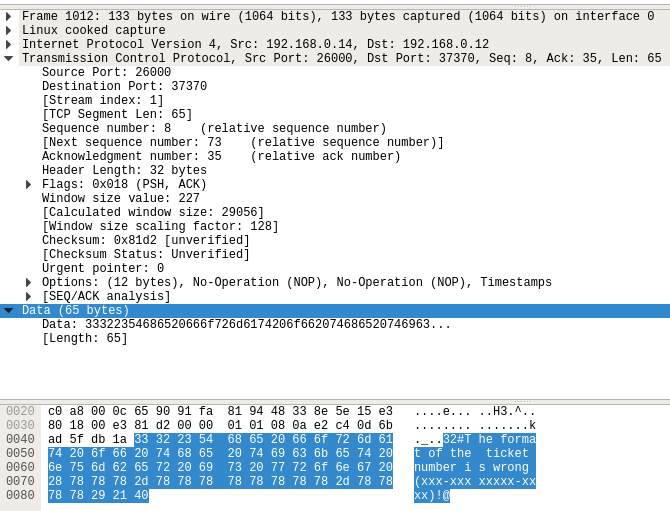
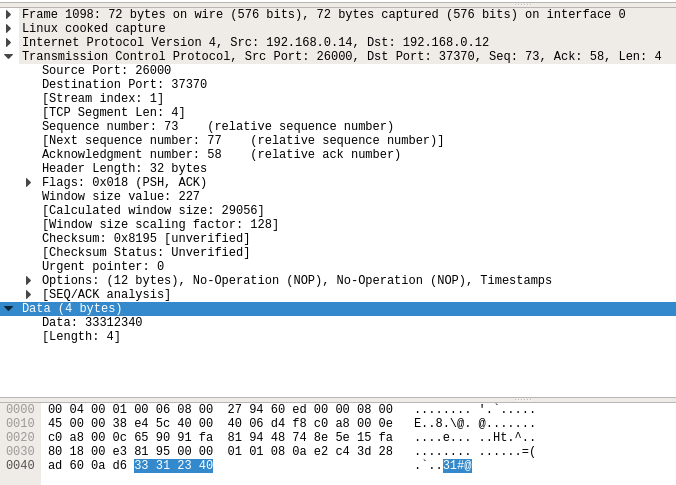


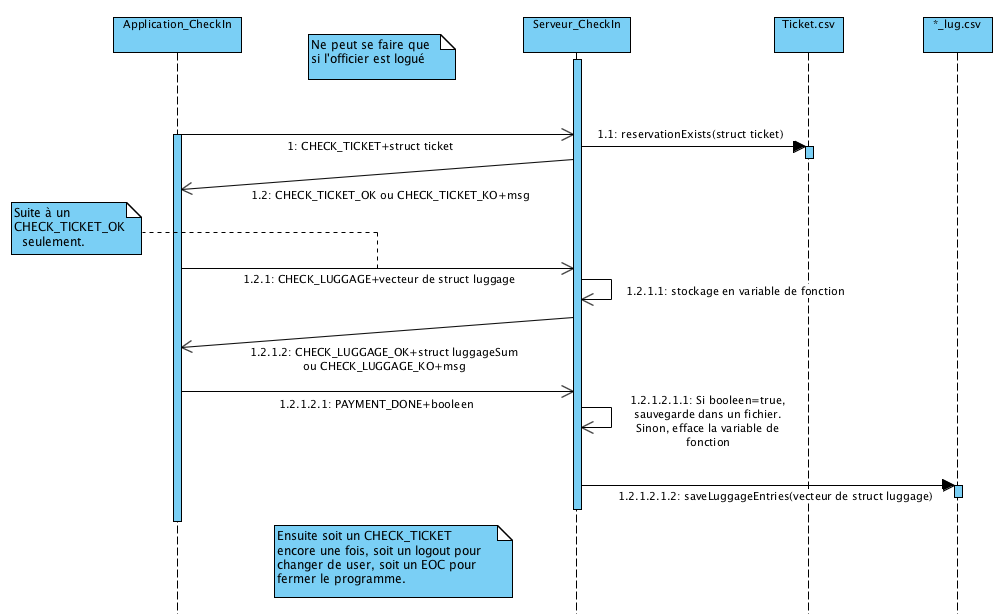
Diagramme du protocole CIMP

## LOGIN\_OFFICER../../../../../../Desktop/Screen%20Shot%202017-10-02%20at%201

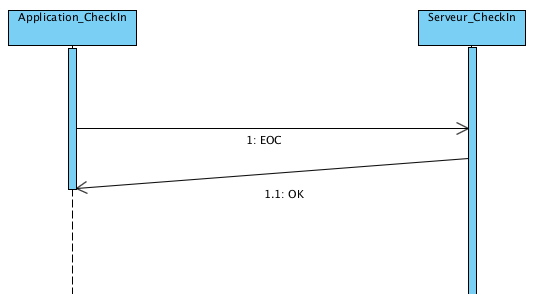
## LOGOUT\_OFFICER

## ../../../../../../Desktop/Screen%20Shot%202017-10-02%20at%201

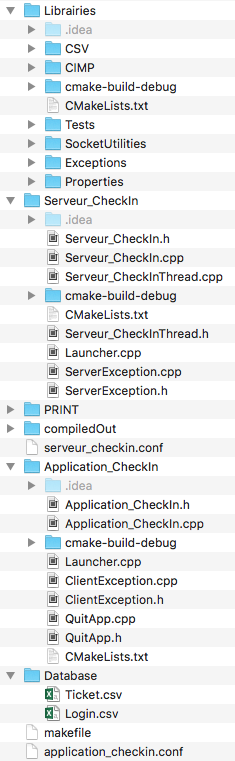
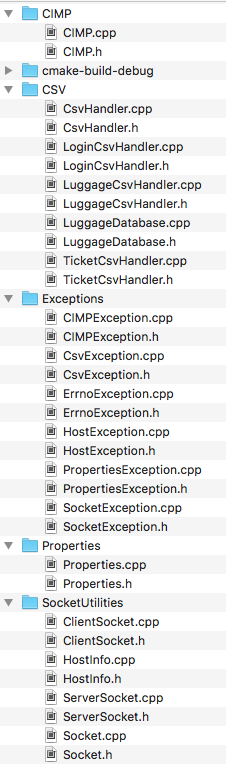
## CHECK\_TICKET



## EOC



## Arborescence du projet

## Code du serveur

### Launcher

#include **<iostream>**#include **<csignal>**#include **"Serveur\_CheckIn.h"  
  
void** fonctionSig(**int**);  
  
Serveur\_CheckIn \*serveur\_checkIn = **nullptr**;  
  
**using namespace** std;  
  
**int** main(**int** argc, **char**\* argv[]) {  
 **if** (argc < 2) {  
 cerr << **"Please provide the properties file as argument when launching me!"** << endl;  
 exit(1);  
 }  
 affServeur(1, **"Launcher starting..."**);  
  
 **struct** sigaction sa;  
 sa.sa\_handler = fonctionSig;  
 sa.sa\_flags = 0;  
 sigemptyset(&sa.sa\_mask);  
 sigaddset(&sa.sa\_mask, SIGUSR1);  
 sigaction(SIGINT, &sa, **nullptr**);  
  
 **try** {  
 serveur\_checkIn = **new** Serveur\_CheckIn(argv[1]);  
 serveur\_checkIn->init();  
 serveur\_checkIn->run();  
 }**catch**(**const** exception &e){  
 affServeur(1, e.what());  
 }  
  
 **delete** serveur\_checkIn;  
  
 affServeur(1, **"Server is closing.\nGood bye!\n\n"**);  
 **return** 0;  
}  
  
**void** fonctionSig(**int** signal){  
 affServeur(1, **"Signal received : calling ~Serveur\_CheckIn()..."**);  
  
 **delete** serveur\_checkIn;  
  
 affServeur(1, **"Server is closing.\nGood bye!\n\n"**);  
 exit(0);  
}

### Serveur\_CheckIn.h

Serveur

#ifndef **SERVEURCHECKIN\_SERVEUR\_CHECKIN\_H**#define **SERVEURCHECKIN\_SERVEUR\_CHECKIN\_H**#include **<vector>**#include **"Serveur\_CheckInThread.h"**#include **"ServerException.h"**#include **"../Librairies/Exceptions/ErrnoException.h"**#include **"../Librairies/SocketUtilities/HostInfo.h"**#include **"../Librairies/SocketUtilities/ServerSocket.h"**#include **"../Librairies/CIMP/CIMP.h"**#include **"../Librairies/CSV/LoginCsvHandler.h"**#include **"../Librairies/Properties/Properties.h"***//#define affServeur(lvl, msg) printf("\e[33;1m|%.\*s>\e[0;1m [%s] \e[0m%s\n", (lvl)\*2, "========================", \_\_PRETTY\_FUNCTION\_\_, msg);fflush(stdout)*#define **affServeur**(**lvl**, **msg**) printf(**"\e[33;1m|%.\*s>\e[0;1m [%s] \e[0m%s\n"**, (**lvl**)\*2, **"========================"**, **\_\_FUNCTION\_\_**, **msg**);fflush(**stdout**)  
  
**using namespace** std;  
  
**class** Serveur\_CheckIn {  
**private**:  
 string tramSeparator;  
 string tramEnding;  
 string host;  
  
 **unsigned int** port;  
 **unsigned int** maxNbClient;  
  
 HostInfo \*hostInfo;  
 ServerSocket \*socketEcoute;  
 Socket \*socketService;  
 vector<pthread\_t> tabThread;  
  
**public**:  
 **bool** newSocketHasBeenAdded;  
 string dbPath;  
 string loginFilePath;  
 string ticketFilePath;  
 string csvSeparator;  
 CIMP \*cimp;  
 vector<Socket \*> tabConnectedSocket;  
 **mutable** pthread\_mutex\_t mutexNbConnectedClients;  
 **mutable** pthread\_cond\_t condNbConnectedClients;  
  
 **explicit** Serveur\_CheckIn(**const char** \*propertiesFilePath);  
 **virtual** ~Serveur\_CheckIn();  
  
 **void** readConfigFile(**const char** \*propertiesFilePath) **throw**(ServerException);  
 **void** init() **throw**(ErrnoException);  
 **void** run() **throw**(ServerException);  
};  
  
  
#endif *//SERVEURCHECKIN\_SERVEUR\_CHECKIN\_H*

### Serveur\_CheckIn.cpp

#include **"Serveur\_CheckIn.h"**#pragma clang diagnostic push  
#pragma clang diagnostic ignored **"-Wmissing-noreturn"**Serveur\_CheckIn::Serveur\_CheckIn(**const char** \*propertiesFilePath) {  
 **this**->readConfigFile(propertiesFilePath);  
  
 **affServeur**(2, **"Acquiring host information..."**);  
 **this**->hostInfo = **new** HostInfo(**this**->host.c\_str());  
 cout << \*(**this**->hostInfo) << endl;  
  
 **affServeur**(2, **"Getting SocketEcoute ready..."**);  
 **this**->socketEcoute = **new** ServerSocket(**this**->port, \*(**this**->hostInfo), **this**->tramEnding, **this**->maxNbClient);  
 cout << \*(**this**->socketEcoute) << endl;  
}  
  
Serveur\_CheckIn::~Serveur\_CheckIn() {  
 printf(**"\n"**);  
 **affServeur**(2, **"Shutting down server..."**);  
  
 */\*\*\* Envoie le signal de fermeture aux threads (pas immédiat) \*\*\*/* **affServeur**(2, **"Sending kill signal to all threads..."**);  
 **for**(pthread\_t thread : **this**->tabThread){  
 pthread\_kill(thread, **SIGUSR1**);  
 pthread\_detach(thread);  
 }  
  
 */\*\*\* Attend que les threads soient tous arrêté \*\*\*/  
// affServeur(2, "Waiting for all threads to die...");* **affServeur**(2, **"Destroying CIMP..."**);  
 **delete this**->cimp;  
  
 **affServeur**(2, **"Destroying socketService..."**);  
 **delete this**->socketService;  
  
 **affServeur**(2, **"Destroying mutex and cond..."**);  
 pthread\_cond\_destroy(&(**this**->condNbConnectedClients));  
 pthread\_mutex\_destroy(&(**this**->mutexNbConnectedClients));  
  
 **affServeur**(2, **"Destroying socketEcoute..."**);  
 **delete this**->socketEcoute;  
  
 **affServeur**(2, **"Destroying hostInfo..."**);  
 **delete this**->hostInfo;  
}  
  
**void** Serveur\_CheckIn::readConfigFile(**const char** \*propertiesFilePath) **throw**(ServerException) {  
 string temp;  
 **affServeur**(1, **"Opening configuration file..."**);  
 Properties properties(propertiesFilePath, **"="**);  
  
 **if**((**this**->host=properties.getValue(**"HOST"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : HOST"**);  
  
 **if**((temp=properties.getValue(**"PORT\_CHCK"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : PORT\_CHCK"**);  
 **this**->port = **static\_cast**<**unsigned int**>(atoi(temp.c\_str()));  
 **if**(**this**->port == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Error, the value for PORT\_CHCK is either invalid or 0."**);  
  
 **if**((**this**->tramEnding=properties.getValue(**"TRAM\_END"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : TRAM\_END"**);  
  
 **if**((**this**->tramSeparator=properties.getValue(**"TRAM\_SEP"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : TRAM\_SEP"**);  
  
 **if**((temp=properties.getValue(**"MAX\_NB\_CLIENTS"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : MAX\_NB\_CLIENTS"**);  
 **this**->maxNbClient = **static\_cast**<**unsigned int**>(atoi(temp.c\_str()));  
 **if**(maxNbClient == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Error, the value for MAX\_NB\_CLIENTS is either invalid or 0."**);  
  
 **if**((**this**->csvSeparator=properties.getValue(**"CSV\_SEP"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : CSV\_SEP"**);  
  
 **if**((**this**->dbPath=properties.getValue(**"DB\_PATH"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : DB\_PATH"**);  
  
 **if**((**this**->loginFilePath=properties.getValue(**"LOGIN\_DB\_FILE"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : LOGIN\_DB\_FILE"**);  
 **this**->loginFilePath = **this**->dbPath + **this**->loginFilePath;  
  
 **if**((**this**->ticketFilePath=properties.getValue(**"TICKET\_DB\_FILE"**)).length() == 0)  
 **throw** ServerException(**"Serveur\_CheckIn::readConfigFile() >> Properties.getValue() : TICKET\_DB\_FILE"**);  
 **this**->ticketFilePath = **this**->dbPath + **this**->ticketFilePath;  
}  
  
**void** Serveur\_CheckIn::init() **throw**(ErrnoException) {  
 **affServeur**(2, **"Initializing server..."**);  
 stringstream ss;  
  
 **this**->socketService = **nullptr**;  
 **this**->socketEcoute->bind();  
  
 pthread\_mutex\_init(&(**this**->mutexNbConnectedClients), **nullptr**);  
 pthread\_cond\_init(&(**this**->condNbConnectedClients), **nullptr**);  
  
 **affServeur**(2, **"Initializing CIMP protocol..."**);  
 cimp = **new** CIMP(**this**->tramSeparator);  
  
  
 **affServeur**(2, **"Creating pool of threads..."**);  
 pthread\_mutex\_lock(&(**this**->mutexNbConnectedClients));  
 **this**->tabThread.reserve(**this**->maxNbClient);  
 cout << **"MAXNB="** << **this**->maxNbClient;  
 **for** (**unsigned int** i=0; i < **this**->maxNbClient; i++) {  
 **this**->tabThread.push\_back(pthread\_t());  
 **if**(pthread\_create(&**this**->tabThread[i], **nullptr**, fctThread, (**void** \*) **this**) !=0) {  
 ss.clear();  
 ss << **"void Serveur\_CheckIn::init() >> Failed to create thread["** << i + 1 << **"]."**;  
 **throw** ErrnoException(**errno**, ss.str());  
 }  
  
 ss.clear();  
 ss.str(string());  
 ss << **"Thread n°"** << i+1 << **" created."**;  
 **affServeur**(2, ss.str().c\_str());  
 }  
 pthread\_mutex\_unlock(&(**this**->mutexNbConnectedClients));  
 pthread\_cond\_signal(&(**this**->condNbConnectedClients));  
  
 **affServeur**(2, **"Initializing done!"**);  
}  
  
**void** Serveur\_CheckIn::run() **throw**(ServerException) {  
 **affServeur**(2, **"Starting server..."**);  
 stringstream ss;  
 **unsigned long** i;  
  
 **do**{  
 **affServeur**(2, **"Starting listening for incoming connection..."**);  
 **this**->socketEcoute->listen();  
  
 **affServeur**(2, **"Waiting for a client..."**);  
 **this**->socketService = **this**->socketEcoute->accept();  
 **affServeur**(2, **"A client just connected!"**);  
  
 **affServeur**(2, **"Searching for a free socket..."**);  
 pthread\_mutex\_lock(&(**this**->mutexNbConnectedClients));  
 i = **this**->tabConnectedSocket.size() + **this**->maxNbClient;  
  
 **if**(i > **SOMAXCONN**){  
 **affServeur**(2, **"No available socket has been found."**);  
  
 **try**{  
 **this**->socketService->sendMessage(cimp->encodeDOC(**"Sorry, no available socket has been found."**));  
 }**catch**(**const** ErrnoException &ex){  
 stringstream ss1;  
 ss1 << **"Serveur\_CheckIn::run() >> Failed to send DOC (no free socket available)."**;  
 ss1 << **"["** << ex.what() << **"]"**;  
 **throw** ServerException(ss1.str());  
 }  
 close(**this**->socketService->getSocketHandle());  
 }  
 **else**{  
 **affServeur**(2, **"A socket is available for the client!"**);  
 **try**{  
 **this**->socketService->sendMessage(cimp->encodeOK());  
 }**catch**(**const** ErrnoException &ex){  
 ss.clear();  
 ss << **"Serveur\_CheckIn::run() >> Failed to send OK after finding a free socket."**;  
 ss << **"["** << ex.what() << **"]"**;  
 **throw** ServerException(ss.str());  
 }  
  
 **affServeur**(2, **"Letting a thread continue the conversation with the client..."**);  
  
 **this**->tabConnectedSocket.push\_back(**new** Socket(\*(**this**->socketService)));  
 **this**->newSocketHasBeenAdded = **true**;  
 pthread\_mutex\_unlock(&(**this**->mutexNbConnectedClients));  
 pthread\_cond\_signal(&(**this**->condNbConnectedClients));  
 }  
 }**while**(**true**);  
}  
  
#pragma clang diagnostic pop

### Serveur\_CheckInThread.h

#ifndef **SERVEURCHECKIN\_SERVEUR\_CHECKINTHREAD\_H**#define **SERVEURCHECKIN\_SERVEUR\_CHECKINTHREAD\_H**#include **<string>**#include **<sstream>**#include **<pthread.h>**#include **<unistd.h>**#include **<csignal>**#include **<iomanip>**#include **<ctime>**#include **<regex>**#include **"Serveur\_CheckIn.h"**#include **"../Librairies/CSV/TicketCsvHandler.h"**#include **"../Librairies/SocketUtilities/Socket.h"**#include **"../Librairies/CIMP/CIMP.h"***//#define affThread(lvl, msg) printf("\e[34;1m|%.\*s>\e[0;1m [th\_%s] [%s] \e[0m%s\n", (lvl)\*8, "========================", getThreadId(), \_\_PRETTY\_FUNCTION\_\_, msg);fflush(stdout)*#define **affThread**(**lvl**, **msg**) printf(**"\e[34;1m|%.\*s>\e[0;1m [th\_%s] [%s] \e[0m%s\n"**, (**lvl**)\*8, **"========================"**, getThreadId(), **\_\_FUNCTION\_\_**, **msg**);fflush(**stdout**)  
  
**using namespace** std;  
  
**void** \* fctThread(**void**\* param);  
**const char**\* getThreadId();  
**bool** ticketNumberIsValid(**const** string ticketNumber);  
string getTodayString();  
**void** fctSIGUSR1(**int** sign);  
**void** fctEndOfThread(**void** \* param);  
  
  
#endif *//SERVEURCHECKIN\_SERVEUR\_CHECKINTHREAD\_H*

### Serveur\_CheckInThread.cpp

#include **"Serveur\_CheckInThread.h"**#include **"../Librairies/CIMP/CIMP.h"**#include **"../Librairies/CSV/LuggageCsvHandler.h"**#pragma clang diagnostic push  
#pragma ide diagnostic ignored **"OCDFAInspection"**#pragma clang diagnostic ignored **"-Wmissing-noreturn"  
void**\* fctThread(**void** \*param) {  
 **bool** clientIsLoggedIn, finishTransaction;  
 **int** previousState;  
 stringstream ss;  
 string flightNumber = **"362"**;  
 string receivedMessage;  
 Socket \*serverSocket;  
 vector<luggageStruct> luggageVector;  
 **auto** \*serveur\_checkIn = (Serveur\_CheckIn \*) param;  
 **auto** CIMP = serveur\_checkIn->cimp;  
 **auto** loginCsvHandler = **new** LoginCsvHandler(serveur\_checkIn->loginFilePath, serveur\_checkIn->csvSeparator);  
 **auto** ticketCsvHandler = **new** TicketCsvHandler(serveur\_checkIn->ticketFilePath, serveur\_checkIn->csvSeparator);  
  
  
 sigset\_t emptyMask, interruptMask;  
 **struct** sigaction sa;  
 sa.**sa\_handler** = fctSIGUSR1;  
 sa.sa\_flags = 0;  
 **sigemptyset**(&sa.sa\_mask);  
 **sigaddset**(&sa.sa\_mask, **SIGINT**);  
 sigaction(**SIGUSR1**, &sa, **NULL**);  
  
 **sigemptyset**(&emptyMask);  
 **sigemptyset**(&interruptMask);  
 **sigaddset**(&interruptMask, **SIGINT**);  
  
 **pthread\_cleanup\_push**(fctEndOfThread, serverSocket);  
  
 **affThread**(1, **"Thread started."**);  
 **while**(**true**){  
 **affThread**(1, **"Waiting for a client..."**);  
 pthread\_mutex\_lock(&(serveur\_checkIn->mutexNbConnectedClients));  
 **while**( !serveur\_checkIn->newSocketHasBeenAdded )  
 pthread\_cond\_wait(&(serveur\_checkIn->condNbConnectedClients), &(serveur\_checkIn->mutexNbConnectedClients) );  
 serverSocket = serveur\_checkIn->tabConnectedSocket.front();  
 serveur\_checkIn->tabConnectedSocket.erase(serveur\_checkIn->tabConnectedSocket.begin());  
 serveur\_checkIn->newSocketHasBeenAdded = **false**;  
 pthread\_mutex\_unlock(&(serveur\_checkIn->mutexNbConnectedClients));  
  
 finishTransaction = **false**;  
 clientIsLoggedIn = **false**;  
 previousState = 0;  
  
 **affThread**(1, **"Starting transaction with client..."**);  
 **do**{  
 pthread\_sigmask(**SIG\_SETMASK**, &emptyMask, **nullptr**);  
 pthread\_sigmask(**SIG\_SETMASK**, &interruptMask, **nullptr**);  
  
 **affThread**(1, **"Waiting for client's request..."**);  
 receivedMessage.clear();  
 **try** {  
 serverSocket->receiveMessage(receivedMessage);  
  
 */\*\*\* On parse le message dans le type adapté \*\*\*/* **void** \*parsedData;  
 **int** protocolType = CIMP->parse(receivedMessage, &parsedData);  
 **switch** (protocolType){  
 **case EOC** :  
 **affThread**(1, **"Received an EOC request..."**);  
  
 serverSocket->sendMessage(CIMP->encodeOK());  
 finishTransaction = **true**;  
 clientIsLoggedIn = **false**;  
 previousState = 0;  
 luggageVector.clear();  
 **break**;  
 **case LOGIN\_OFFICER** :  
 **affThread**(1, **"Received a LOGIN\_OFFICER request..."**);  
 **if**(clientIsLoggedIn){  
 **affThread**(1, **"The client was already connected!"**);  
  
 **delete** ((loginStruct \*) parsedData);  
 serverSocket->sendMessage(CIMP->encodeLOGIN\_OFFICER\_KO(**"You are already connected!"**));  
 }  
 **else**{  
 **try** {  
 **affThread**(1, **"Connecting to login database to verify credentials..."**);  
 **if**( loginCsvHandler->isValid( ((loginStruct \*)parsedData)->login, ((loginStruct \*)parsedData)->password ) ){  
 **delete** ((loginStruct \*) parsedData);  
  
 **affThread**(1, **"The client connected successfuly!"**);  
 clientIsLoggedIn = **true**;  
 previousState = **LOGIN\_OFFICER**;  
  
 serverSocket->sendMessage(CIMP->encodeLOGIN\_OFFICER\_OK());  
 }  
 **else**{  
 **delete** ((loginStruct \*) parsedData);  
 **affThread**(1, **"The client entered wrong credentials!"**);  
 serverSocket->sendMessage(CIMP->encodeLOGIN\_OFFICER\_KO(**"Wrong credentials!"**));  
 }  
 } **catch**(**const** CsvException &ex) {  
 **affThread**(1, **"Could not connect to login database!"**);  
 **affThread**(1, ex.what());  
 clientIsLoggedIn = **false**;  
 finishTransaction = **true**;  
 previousState = 0;  
 kill(getpid(), **SIGINT**);  
 }  
 }  
 **break**;  
 **case LOGOUT\_OFFICER** :  
 **affThread**(1, **"Received a LOGOUT\_OFFICER request..."**);  
 serverSocket->sendMessage(CIMP->encodeLOGOUT\_OFFICER\_OK());  
  
 previousState = 0;  
 clientIsLoggedIn = **false**;  
 **break**;  
 **case CHECK\_TICKET** :{  
 **affThread**(1, **"Received a CHECK\_TICKET request..."**);  
 **if**(!clientIsLoggedIn){  
 **affThread**(1, **"The client wasn't even logged in!"**);  
 serverSocket->sendMessage(CIMP->encodeCHECK\_TICKET\_KO(**"Please log in before using our features!"**));  
 }  
 **else** {  
 string toReturn;  
  
 **if** (ticketNumberIsValid(((ticketStruct \*) parsedData)->ticketNumber) &&  
 ticketCsvHandler->reservationExists(\*((ticketStruct \*) parsedData))) {  
 toReturn = CIMP->encodeCHECK\_TICKET\_OK();  
 } **else** {  
 toReturn = CIMP->encodeCHECK\_TICKET\_KO( **"The format of the ticket number is wrong (xxx-xxxxxxxx-xxxx)!"**);  
 }  
  
 serverSocket->sendMessage(toReturn);  
 previousState = **CHECK\_TICKET**;  
 }  
 }  
 **break**;  
 **case CHECK\_LUGGAGE** :  
 **affThread**(1, **"Received a CHECK\_LUGGAGE request..."**);  
 **if**(!clientIsLoggedIn){  
 **affThread**(1, **"The client wasn't even logged in!"**);  
 serverSocket->sendMessage(CIMP->encodeCHECK\_LUGGAGE\_KO(**"Please log in before using our features!"**));  
 } **else**{  
 **if**(previousState != **CHECK\_TICKET**){  
 **affThread**(1, **"The client isn't following the script!"**);  
 serverSocket->sendMessage(CIMP->encodeCHECK\_LUGGAGE\_KO(**"Please encode the ticket first!"**));  
 }  
 **else**{  
 luggageSumStruct luggageSum;  
 **float** totalWeight, excessWeight;  
 string tickNb;  
  
 totalWeight = excessWeight = 0;  
 tickNb = ((vector<luggageStruct> \*)parsedData)->back().ticketNumber;  
 ((vector<luggageStruct> \*)parsedData)->pop\_back();  
 **for**(luggageStruct luggage : \*((vector<luggageStruct> \*)parsedData)){  
 luggageVector.reserve(1);  
 luggageVector.push\_back(luggage);  
 **float** weight = **static\_cast**<**float**>(atof(luggage.weight.c\_str()));  
 totalWeight = totalWeight + weight;  
 **if**(weight>20.0)  
 excessWeight += weight-20;  
 }  
  
 luggageSum.ticketNumber = tickNb;  
 luggageSum.totalWeight = to\_string(totalWeight);  
 luggageSum.excessWeight = to\_string(excessWeight);  
 luggageSum.priceToPay = to\_string(excessWeight\*2.95);  
  
 serverSocket->sendMessage(CIMP->encodeCHECK\_LUGGAGE\_OK(luggageSum));  
 previousState = **CHECK\_LUGGAGE**;  
 }  
 }  
 **break**;  
 **case PAYMENT\_DONE** :  
 **affThread**(1, **"Received a PAYMENT\_DONE request..."**);  
 **if**(!clientIsLoggedIn){  
 **affThread**(1, **"The client wasn't even logged in!"**);  
 serverSocket->sendMessage(CIMP->encodePAYMENT\_DONE\_KO(**"Please log in before using our features!"**));  
 } **else**{  
 **if**(previousState != **CHECK\_LUGGAGE**){  
 **affThread**(1, **"The client isn't following the script!"**);  
 serverSocket->sendMessage(CIMP->encodePAYMENT\_DONE\_KO(**"Please encode the luggages first or empty!"**));  
 }  
 **else**{  
 cout << **"serveur : "** << boolalpha << \*((**bool** \*)parsedData) << endl;  
 **if**(\*((**bool** \*)parsedData)) {  
 *//362\_22082017\_lug.csv* stringstream ss2;  
 ss2 << serveur\_checkIn->dbPath << flightNumber << **"\_"** << getTodayString()  
 << **"\_lug.csv"**;  
 **auto** \*luggageCsvHandler = **new** LuggageCsvHandler(ss2.str(),  
 serveur\_checkIn->csvSeparator);  
  
 **if** (luggageCsvHandler->saveLuggageEntries(luggageVector))  
 serverSocket->sendMessage(CIMP->encodePAYMENT\_DONE\_OK());  
 **else** serverSocket->sendMessage(CIMP->encodePAYMENT\_DONE\_KO( **"Error trying to save it in database!"**));  
 previousState = **CHECK\_LUGGAGE**;  
 }  
 **else**{  
 previousState = 0;  
 luggageVector.clear();  
 serverSocket->sendMessage(CIMP->encodePAYMENT\_DONE\_KO( **"Reverted to beginning of transaction."**));  
 }  
 }  
 }  
 **break**;  
 **default**:  
 **affThread**(1, **"Received an unknown request..."**);  
  
 **break**;  
 }  
 } **catch** (**const** ErrnoException &ex) {  
 **affThread**(1, ex.what());  
 clientIsLoggedIn = **false**;  
 finishTransaction = **true**;  
 previousState = 0;  
 luggageVector.clear();  
 serverSocket->sendMessage(CIMP->encodeEOC());  
 } **catch** (**const** CIMPException &ex) {  
 **affThread**(1, ex.what());  
 clientIsLoggedIn = **false**;  
 finishTransaction = **true**;  
 previousState = 0;  
 luggageVector.clear();  
 serverSocket->sendMessage(CIMP->encodeEOC());  
 } **catch**(**const** CsvException &ex) {  
 **affThread**(1, **"Could not connect to database!"**);  
 **affThread**(1, ex.what());  
 clientIsLoggedIn = **false**;  
 finishTransaction = **true**;  
 previousState = 0;  
 luggageVector.clear();  
 serverSocket->sendMessage(CIMP->encodeEOC());  
 kill(getpid(), **SIGINT**);  
 }  
 }**while**(!finishTransaction);  
  
 **affThread**(1, **"End of transaction with the client."**);  
  
 **delete** serverSocket;  
 }  
  
 **pthread\_cleanup\_pop**(1);  
 **return** param;  
}  
  
**const char**\* getThreadId() {  
 stringstream ss;  
 **char** \*threadId;  
  
 ss << getpid() << **"."** << pthread\_self();  
 threadId = (**char** \*) malloc(ss.str().length()+1);  
 strcat(threadId, ss.str().c\_str());  
  
 **return** threadId;  
}  
  
**bool** ticketNumberIsValid(**const** string ticketNumber){  
*// 362-22082017-0070* regex pieces\_regex(**"([0-9]{3})\\-([0-9]{8})\\-([0-9]{4})"**);  
 smatch pieces\_match;  
  
 **return** regex\_match(ticketNumber, pieces\_match, pieces\_regex) && ticketNumber.substr(4, 8) == getTodayString();  
}  
  
string getTodayString(){  
 **auto** t = time(**nullptr**);  
 **auto** tm = \*localtime(&t);  
 ostringstream oss;  
  
  
 oss << put\_time(&tm, **"%d%m%Y"**);  
 **return** oss.str();  
}  
  
**void** fctSIGUSR1(**int**) {  
 **affThread**(1, **"Main Thread is asking me to shutdown..."**);  
 pthread\_exit(0);  
}  
  
**void** fctEndOfThread(**void** \*param) {  
 **delete**((Socket \*)param);  
 **affThread**(1, **"Good bye!"**);  
}  
  
#pragma clang diagnostic pop

### ServeurException.h

#ifndef **SERVEURCHECKIN\_SERVEREXCEPTION\_H**#define **SERVEURCHECKIN\_SERVEREXCEPTION\_H**#include **<string>**#include **<cstring>  
  
using namespace** std;  
  
**class** ServerException : **public** exception {  
**protected**:  
 string message;  
**public**:  
 ServerException(string msg);  
 ServerException(**const char** \*msg);  
 ServerException(**const** ServerException &orig);  
 ~ServerException() **throw**();  
 **virtual const char**\* what() **const throw**();  
};  
  
  
#endif *//SERVEURCHECKIN\_SERVEREXCEPTION\_H*

### ServeurException.cpp

#include **"ServerException.h"**ServerException::ServerException(**const** string msg) : exception(), message(msg) {}  
  
ServerException::ServerException(**const char** \*msg) : exception(), message(msg) {}  
  
ServerException::ServerException(**const** ServerException &orig) : exception(orig), message(orig.message) {}  
  
ServerException::~ServerException() **throw**() {}  
  
**const char** \*ServerException::what() **const throw**() {  
 **char** \*returnVal;  
  
 returnVal = **new char**[**this**->message.length() + 1];  
 strcat(returnVal, **this**->message.c\_str());  
  
 **return** returnVal;  
}

### serveur\_checkin.conf

PORT\_CHCK=26000  
TRAM\_SEP=#  
TRAM\_END=@  
CSV\_SEP=;  
#DB\_PATH=../../Database/  
DB\_PATH=Database/  
LOGIN\_DB\_FILE=Login.csv  
TICKET\_DB\_FILE=Ticket.csv  
MAX\_NB\_CLIENTS=5  
HOST=192.168.56.2

Makefile

.SILENT:  
.PHONY: clean, clobber  
  
COMP=compiledOut  
LIB\_EXC=Librairies/Exceptions  
LIB\_CIMP=Librairies/CIMP  
LIB\_CSV=Librairies/CSV  
LIB\_PROP=Librairies/Properties  
LIB\_SOCK=Librairies/SocketUtilities  
LIB\_TEST=Librairies/Tests  
SERV=Serveur\_CheckIn  
APPL=Application\_CheckIn  
  
CC = g++ -m64 -std=c++11 -Wall -pthread  
  
OBJS=$(COMP)/CsvException.o $(COMP)/ErrnoException.o $(COMP)/PropertiesException.o $(COMP)/SocketException.o $(COMP)/HostException.o $(COMP)/CIMPException.o $(COMP)/CIMP.o $(COMP)/LuggageDatabase.o $(COMP)/LuggageCsvHandler.o $(COMP)/TicketCsvHandler.o $(COMP)/LoginCsvHandler.o $(COMP)/CsvHandler.o $(COMP)/Properties.o $(COMP)/HostInfo.o $(COMP)/Socket.o $(COMP)/ClientSocket.o $(COMP)/ServerSocket.o $(COMP)/ServerException.o $(COMP)/Serveur\_CheckInThread.o $(COMP)/Serveur\_CheckIn.o $(COMP)/QuitApp.o $(COMP)/ClientException.o $(COMP)/Application\_CheckIn.o  
PROGRAMS=CsvExceptionTest CsvHandlerTest ErrnoExceptionTest HostInfoTest LoginCsvHandlerTest PropertiesTest test launcherServeur\_CheckIn launcherApplication\_CheckIn  
ALL: $(PROGRAMS)  
  
  
launcherApplication\_CheckIn: $(APPL)/launcher.cpp $(OBJS)  
 echo Creating launcherApplication\_CheckIn executable ...  
 $(CC) -o launcherApplication\_CheckIn $(APPL)/launcher.cpp $(OBJS)  
  
$(COMP)/Application\_CheckIn.o: $(APPL)/Application\_CheckIn.h $(APPL)/Application\_CheckIn.cpp $(OBJS)  
 echo Creating Application\_CheckIn.o ...  
 $(CC) -c $(APPL)/Application\_CheckIn.cpp  
 mv Application\_CheckIn.o $(COMP)  
  
$(COMP)/ClientException.o: $(APPL)/ClientException.h $(APPL)/ClientException.cpp  
 echo Creating ClientException.o ...  
 $(CC) -c $(APPL)/ClientException.cpp  
 mv ClientException.o $(COMP)  
  
$(COMP)/QuitApp.o: $(APPL)/QuitApp.h $(APPL)/QuitApp.cpp  
 echo Creating QuitApp.o ...  
 $(CC) -c $(APPL)/QuitApp.cpp  
 mv QuitApp.o $(COMP)  
  
launcherServeur\_CheckIn: $(SERV)/launcher.cpp $(OBJS)  
 echo Creating launcherServeur\_CheckIn executable ...  
 $(CC) -o launcherServeur\_CheckIn $(SERV)/launcher.cpp $(OBJS)  
  
$(COMP)/Serveur\_CheckInThread.o: $(SERV)/Serveur\_CheckInThread.h $(SERV)/Serveur\_CheckInThread.cpp $(COMP)/Socket.o $(COMP)/Serveur\_CheckIn.o $(COMP)/LoginCsvHandler.o  
 echo Creating Serveur\_CheckInThread.o ...  
 $(CC) -c $(SERV)/Serveur\_CheckInThread.cpp  
 mv Serveur\_CheckInThread.o $(COMP)  
  
$(COMP)/Serveur\_CheckIn.o: $(SERV)/Serveur\_CheckIn.h $(SERV)/Serveur\_CheckIn.cpp $(COMP)/Serveur\_CheckInThread.o $(COMP)/ServerException.o $(COMP)/HostInfo.o $(COMP)/ServerSocket.o $(COMP)/CIMP.o $(COMP)/LoginCsvHandler.o $(COMP)/Properties.o  
 echo Creating Serveur\_CheckIn.o ...  
 $(CC) -c $(SERV)/Serveur\_CheckIn.cpp  
 mv Serveur\_CheckIn.o $(COMP)  
  
$(COMP)/ServerException.o: $(SERV)/ServerException.h $(SERV)/ServerException.cpp  
 echo Creating ServerException.o ...  
 $(CC) -c $(SERV)/ServerException.cpp  
 mv ServerException.o $(COMP)  
  
  
  
test: $(LIB\_TEST)/test.cpp  
 echo Creating test executable ...  
 $(CC) -o test $(LIB\_TEST)/test.cpp  
  
PropertiesTest: $(LIB\_TEST)/PropertiesTest.cpp $(COMP)/Properties.o $(COMP)/PropertiesException.o  
 echo Creating PropertiesTest executable ...  
 $(CC) -o PropertiesTest $(LIB\_TEST)/PropertiesTest.cpp $(COMP)/Properties.o $(COMP)/PropertiesException.o  
  
LoginCsvHandlerTest: $(LIB\_TEST)/LoginCsvHandlerTest.cpp $(COMP)/LoginCsvHandler.o $(COMP)/CsvHandler.o $(COMP)/CsvException.o  
 echo Creating LoginCsvHandlerTest executable ...  
 $(CC) -o LoginCsvHandlerTest $(LIB\_TEST)/LoginCsvHandlerTest.cpp $(COMP)/LoginCsvHandler.o $(COMP)/CsvHandler.o $(COMP)/CsvException.o  
  
HostInfoTest: $(LIB\_TEST)/HostInfoTest.cpp $(COMP)/HostInfo.o $(COMP)/HostException.o  
 echo Creating HostInfoTest executable ...  
 $(CC) -o HostInfoTest $(LIB\_TEST)/HostInfoTest.cpp $(COMP)/HostInfo.o $(COMP)/HostException.o  
  
ErrnoExceptionTest: $(LIB\_TEST)/ErrnoExceptionTest.cpp $(COMP)/ErrnoException.o  
 echo Creating ErrnoExceptionTest executable ...  
 $(CC) -o ErrnoExceptionTest $(LIB\_TEST)/ErrnoExceptionTest.cpp $(COMP)/ErrnoException.o  
  
CsvHandlerTest: $(LIB\_TEST)/CsvHandlerTest.cpp $(COMP)/CsvHandler.o $(COMP)/CsvException.o  
 echo Creating CsvHandlerTest executable ...  
 $(CC) -o CsvHandlerTest $(LIB\_TEST)/CsvHandlerTest.cpp $(COMP)/CsvHandler.o $(COMP)/CsvException.o  
  
CsvExceptionTest: $(LIB\_TEST)/CsvExceptionTest.cpp $(COMP)/CsvException.o  
 echo Creating CsvExceptionTest executable ...  
 $(CC) -o CsvExceptionTest $(LIB\_TEST)/CsvExceptionTest.cpp $(COMP)/CsvException.o  
  
$(COMP)/ServerSocket.o: $(LIB\_SOCK)/ServerSocket.h $(LIB\_SOCK)/ServerSocket.cpp $(COMP)/Socket.o  
 echo Creating ServerSocket.o ...  
 $(CC) -c $(LIB\_SOCK)/ServerSocket.cpp  
 mv ServerSocket.o $(COMP)  
  
$(COMP)/ClientSocket.o: $(LIB\_SOCK)/ClientSocket.h $(LIB\_SOCK)/ClientSocket.cpp $(COMP)/Socket.o  
 echo Creating ClientSocket.o ...  
 $(CC) -c $(LIB\_SOCK)/ClientSocket.cpp  
 mv ClientSocket.o $(COMP)  
  
$(COMP)/Socket.o: $(LIB\_SOCK)/Socket.h $(LIB\_SOCK)/Socket.cpp $(COMP)/ErrnoException.o $(COMP)/SocketException.o $(COMP)/HostInfo.o  
 echo Creating Socket.o ...  
 $(CC) -c $(LIB\_SOCK)/Socket.cpp  
 mv Socket.o $(COMP)  
  
$(COMP)/HostInfo.o: $(LIB\_SOCK)/HostInfo.h $(LIB\_SOCK)/HostInfo.cpp $(COMP)/HostException.o  
 echo Creating HostInfo.o ...  
 $(CC) -c $(LIB\_SOCK)/HostInfo.cpp  
 mv HostInfo.o $(COMP)  
  
$(COMP)/Properties.o: $(LIB\_PROP)/Properties.h $(LIB\_PROP)/Properties.cpp $(COMP)/PropertiesException.o  
 echo Creating Properties.o ...  
 $(CC) -c $(LIB\_PROP)/Properties.cpp  
 mv Properties.o $(COMP)  
  
$(COMP)/LuggageCsvHandler.o: $(LIB\_CSV)/LuggageCsvHandler.h $(LIB\_CSV)/LuggageCsvHandler.cpp $(COMP)/CsvHandler.o $(COMP)/LuggageDatabase.o  
 echo Creating LuggageCsvHandler.o ...  
 $(CC) -c $(LIB\_CSV)/LuggageCsvHandler.cpp  
 mv LuggageCsvHandler.o $(COMP)  
  
$(COMP)/LuggageDatabase.o: $(LIB\_CSV)/LuggageDatabase.h $(LIB\_CSV)/LuggageDatabase.cpp  
 echo Creating LuggageDatabase.o ...  
 $(CC) -c $(LIB\_CSV)/LuggageDatabase.cpp  
 mv LuggageDatabase.o $(COMP)  
  
$(COMP)/TicketCsvHandler.o: $(LIB\_CSV)/TicketCsvHandler.h $(LIB\_CSV)/TicketCsvHandler.cpp $(COMP)/CsvHandler.o $(COMP)/CIMP.o  
 echo Creating TicketCsvHandler.o ...  
 $(CC) -c $(LIB\_CSV)/TicketCsvHandler.cpp  
 mv TicketCsvHandler.o $(COMP)  
  
$(COMP)/LoginCsvHandler.o: $(LIB\_CSV)/LoginCsvHandler.h $(LIB\_CSV)/LoginCsvHandler.cpp $(COMP)/CsvHandler.o  
 echo Creating LoginCsvHandler.o ...  
 $(CC) -c $(LIB\_CSV)/LoginCsvHandler.cpp  
 mv LoginCsvHandler.o $(COMP)  
  
$(COMP)/CsvHandler.o: $(LIB\_CSV)/CsvHandler.h $(LIB\_CSV)/CsvHandler.cpp $(COMP)/CsvException.o  
 echo Creating CsvHandler.o ...  
 $(CC) -c $(LIB\_CSV)/CsvHandler.cpp  
 mv CsvHandler.o $(COMP)  
  
$(COMP)/CIMP.o: $(LIB\_CIMP)/CIMP.h $(LIB\_CIMP)/CIMP.cpp $(COMP)/CIMPException.o  
 echo Creating CIMP.o ...  
 $(CC) -c $(LIB\_CIMP)/CIMP.cpp  
 mv CIMP.o $(COMP)  
  
$(COMP)/CIMPException.o: $(LIB\_EXC)/CIMPException.cpp $(LIB\_EXC)/CIMPException.h  
 echo Creating CIMPException.o ...  
 $(CC) -c $(LIB\_EXC)/CIMPException.cpp  
 mv CIMPException.o $(COMP)  
  
$(COMP)/HostException.o: $(LIB\_EXC)/HostException.cpp $(LIB\_EXC)/HostException.h  
 echo Creating HostException.o ...  
 $(CC) -c $(LIB\_EXC)/HostException.cpp  
 mv HostException.o $(COMP)  
  
$(COMP)/SocketException.o: $(LIB\_EXC)/SocketException.cpp $(LIB\_EXC)/SocketException.h  
 echo Creating SocketException.o ...  
 $(CC) -c $(LIB\_EXC)/SocketException.cpp  
 mv SocketException.o $(COMP)  
  
$(COMP)/PropertiesException.o: $(LIB\_EXC)/PropertiesException.cpp $(LIB\_EXC)/PropertiesException.h  
 echo Creating PropertiesException.o ...  
 $(CC) -c $(LIB\_EXC)/PropertiesException.cpp  
 mv PropertiesException.o $(COMP)  
  
$(COMP)/ErrnoException.o: $(LIB\_EXC)/ErrnoException.cpp $(LIB\_EXC)/ErrnoException.h  
 echo Creating ErrnoException.o ...  
 $(CC) -c $(LIB\_EXC)/ErrnoException.cpp  
 mv ErrnoException.o $(COMP)  
  
$(COMP)/CsvException.o: $(LIB\_EXC)/CsvException.cpp $(LIB\_EXC)/CsvException.h  
 echo Creating CsvException.o ...  
 $(CC) -c $(LIB\_EXC)/CsvException.cpp  
 mv CsvException.o $(COMP)  
  
  
clean:  
 @rm -f $(COMP)/\*.o core  
  
clobber: clean  
 @rm -f tags $(PROGRAMS)