Laboratoire de Réseaux et technologie Internet (TCP-UDP/IP & HTTP en C/C++/Java) :

3ème Informatique de gestion
3ème Informatique et systèmes
opt. Informatique industrielle et Réseaux-télécommunications
2018-2019

Claude Vilvens, Christophe Charlet, Alfonso Romio, Sébastien Calmant
(Network programming team)




1. Préambule

L'Unité d'Enseignement "**Programmation réseaux, web et mobiles**" (10 ECTS - 135h) comporte des Activités d'apprentissage :

- AA: Réseaux et technologies Internet (dans toutes les options);
- AA: Programmation.Net (dans toutes les options);
- AA: Technologie de l'e-commerce et mobiles (informatique de gestion seulement);
- AA: Compléments de programmation réseaux (informatique réseaux-télécoms seulement).

Les travaux de programmation réseaux présentés ici constituent la description technique des travaux de laboratoire de l'AA "**Réseaux et technologie Internet**". Il s'agit ici de maîtriser les divers paradigmes de programmation réseau TCP/IP et HTTP dans les environnements UNIX et Windows, en utilisant les langages C/C++ et Java. Ces travaux ont été conçus de manière à pouvoir collaborer avec les travaux de laboratoire des AA "**Compléments programmation réseaux**" (3ème informatique réseaux-télécoms) et "**Technologies du e-commerce**" (3ème informatique de gestion); certains points correspondants sont donc déjà vaguement évoqués ici.

Les développements C/C++ UNIX sont sensés se faire avec les outils de développement disponibles sur la machine Sunray; on pourra aussi utiliser une machine virtuelle Sun Solaris ou Linux. Cependant, Code::Blocks sur les PCs peut vous permettre de développer du code en dehors de ces machines. Les développements Java sont à réaliser avec **NetBeans 8.***. Le serveur Web avec moteur à servlets/JSP utilisé est **Tomcat 7*/8*** sous Windows (PC) et/ou Unix (machines U2 ou INXS). La gestion des fichiers élémentaires se fera au moyen de fichiers à enregistrements classiques, de fichiers textes ou de fichiers CSV (dans le contexte C/C++) ou properties (dans le contexte Java). La gestion des bases de données intervenant dans cet énoncé se fera avec le SGBD **MySQL**, interfacé avec la ligne de commande ou, de manière plus attrayante, avec des outils comme **MySQL Workbench** ou **Toad for MySQL**.

Les travaux peuvent être réalisés

- soit par pour **une équipe de deux étudiants** qui devront donc se coordonner intelligemment et se faire confiance, sachant qu'il faut être capable d'expliquer l'ensemble du travail (pas seulement les éléments dont on est l'auteur);
- soit par un **étudiant travaillant seul** (les avantages et inconvénients d'un travail par deux s'échangent : on a moins de problèmes quand il faut s'arranger avec soi-même et on ne perd pas de temps en coordination).

2. Règles d'évaluation

Comme on sait, la note finale pour l'UE considérée se calcule par une moyenne des notes des AA constitutives, sachant que le seul cas de réussite automatique d'une UE est une note de 10/20 minimum dans chacune des AAs.

Pour ce qui concerne l'évaluation de l'AA "Réseaux et technologie Internet", voici les règles de cotation utilisées par les enseignants de l'équipe responsable de cette AA.

1) L'évaluation établissant la note de l'AA "Réseaux et technologie Internet" est réalisée de la manière suivante :

- théorie : un examen écrit en janvier 2018 (sur base d'une liste de points de théorie à développer fournis au fur et à mesure de l'évolution du cours théorique) et coté sur 20;
- laboratoire : 7 évaluations pondérées selon leur importance (5 non remédiables car visant à acquérir des notions de base, 2 remédiables en 2ème session), chacune notée sur un total établi en fonction de l'importance des notions à assimiler et de la charge de travail correspondante; la moyenne de ces notes fournit une note de laboratoire sur 20;
- note finale : **moyenne de la note de théorie (poids de 50%) et de la note de laboratoire (poids de 50%)**.

Dans ces conditions, *il est clair qu'une note de théorie beaucoup trop basse (du type 5/20 ou moins encore) ne peut conduire qu'à l'échec de l'AA considérée*.

Cette procédure est d'application tant en 1ère qu'en 2ème session.

2) *Dans le cas où les travaux sont présentés par une équipe de deux étudiants*, chacun d'entre eux doit être capable d'expliquer et de justifier l'intégralité du travail sans de longues recherches dans le code de l'application proposée (pas seulement les parties du travail sur lesquelles il aurait plus particulièrement travaillé).

3) *Dans tous les cas*, tout étudiant doit être capable d'expliquer de manière générale (donc, sans entrer dans les détails) les notions et concepts théoriques qu'il manipule dans ses travaux (par exemple: socket, machine à états de TCP, signature électronique, certificat, etc).

4) En 2ème session, un **report de note** est possible pour **des notes supérieures ou égales à 10/20** en ce qui concerne :

- la note de théorie;
- les notes de laboratoire des évaluations 6 et 7.

Les évaluations de théorie, laboratoire 6 et laboratoire 7 ayant des **notes inférieures à 10/20** sont donc **à représenter dans leur intégralité** (le refus de représenter une évaluation complète de laboratoire entraîne automatiquement la cote de 0).

Les notes de laboratoire des évaluations 1,2,3,4 et 5 ne sont pas remédiables : comme elles ont pour but de faire acquérir des techniques élémentaires, il n'a plus de sens de tester à nouveau ces acquis en 2ème session et elles resteront donc à la valeur acquise lors de l'évaluation de 1ère session.

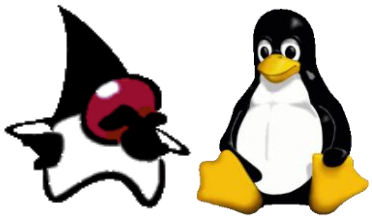
3. Agenda des évaluations

Pour chaque évaluation, le délai est à *respecter* impérativement.

Evaluation	**Evaluation continue : semaine d'évaluation**	**Pondération dans la cote de laboratoire**	**Remédiable en 2ème session**
Evaluation 1 : client-serveur multithread TCP en C/C++	1/10/2018-5/10/2018	20	non
Evaluation 2 : JDBC	8/10/2018-12/10/2018	10	non
Evaluation 3 : client-serveur multithread TCP en Java	22/10/2018-26/10/2018	20	non
Evaluation 4 : programmation Web Java classique	19/11/2018-23/11/2018	20	non
Evaluation 5 : client-serveur UDP en C/C++ et Java	3/12/2018-7/12/2018	10	non
Evaluation 6 : client-serveur sécurisé en Java et complément caddie virtuel en Java	Examen de laboratoire de janvier 2019	80	oui
Evaluation 7 : communications réseaux C/C++-Java	Examen de laboratoire de janvier 2019	20	oui

Remarque importante : Pour rappel, lors de chaque évaluation, chaque étudiant est sensé connaître les bases théoriques qui lui ont permis de réaliser les développements proposés. Dans le cas contraire, on sera amené à considérer qu'il a développé sans comprendre ce qu'il faisait …

4. Avertissements préalables



1) Dans ce qui suivra, un certain nombre de points ont été simplifiés par rapport à la réalité. **Certains points ont également été volontairement laissés dans le vague ou l'obscur** : il vous appartient d'élaborer vous-mêmes les éléments qui ne sont pas explicitement décris dans l'énoncé. Cependant, pour vous éviter de vous fourvoyer dans une impasse ou perdre votre temps à des options de peu d'intérêt, il vous est vivement recommandé de prendre conseil au près de l'un de vos professeurs concernés par le projet.

2) Le contenu des évaluations a été déterminé en fonction de l'avancement du cours théorique ET des besoins des autres cours de 3ème bachelier, avec lesquels nous nous sommes synchronisés dans la mesure du possible. Sans ces contraintes, le schéma de développement eût été différent.

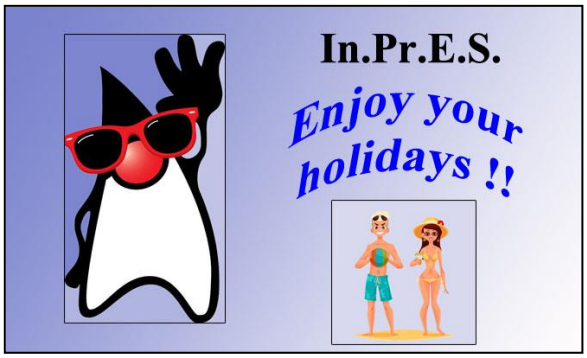
3) Autre chose : une condition sine-qua-non de réussite est le traitement systématique et raisonné des **erreurs** courantes (codes de retour, errno, exceptions).

Une plage de 10 ports TCP-UDP vous est attribuée sur les machines Unix et Linux. Il vous est demandé de la respecter pour des raisons évidentes …

5. Le contexte général

5.1 Présentation générale

Nous sortons de la période des vacances d'été, un été particulièrement torride ... Afin de prolonger cette délicieuse tranche de vie, nous vous proposons de mettre en place une plate-forme de gestion d'hôtels et de villages de vacances appartenant à la célèbre chaîne de loisirs Inpres-EnjoyYourHolidays (**IEYH** - prononcer "yeheee!").



Cette infrastructure se compose de :

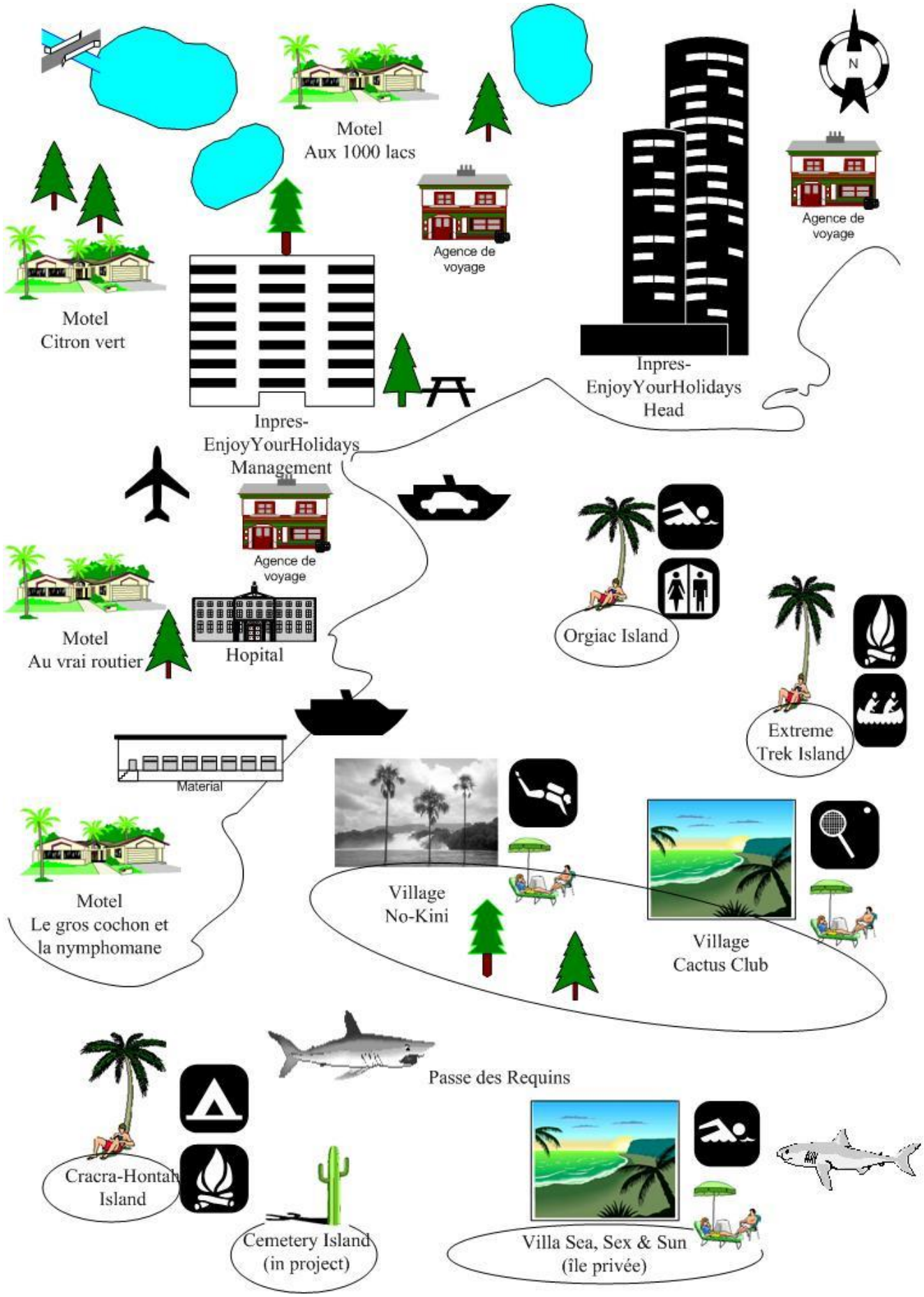
♦ plusieurs motels (comme " Motel Au vrai routier ", "Motel Citron vert", etc), situés sur le continent (le plus souvent à proximité des autoroutes et des zonings industriels) et qui se limitent à la location de chambres pour routiers, voyageurs de passage, etc; seul le petit déjeuner peut y être obtenu;

♦ deux villages de vacances ("Village No-Kini" et "Village Cactus Club") installés sur une île paradisiaque et qui proposent, outre des mangeailles monstrueuses, de nombreuses activités pour passer un séjour agréable : plages, pédalos, court de tennis, cinéma (pas de réservation nécessaire) ou cours et formation diverses (ski nautique, vie dans la nature, etc) mais encore aussi participation à des "excursions" à "cracra-hontah" (durée : 5 jours), "extreme trek" (durée : 7 jours) ou "orgiac island" (durée : 3 jours – faut pas exagérer); ces excursions sont organisées le 1er et le 15 de chaque mois;

♦ un centre de gestion des réservations de séjours, un centre de gestion des activités et un centre de maintenance de matériel nécessaire, tous situés sur le continent et en relation réseau avec les motels (seulement pour les réservations de séjours) et les villages;

♦ le siège central de la société IEYH et le siège du management, avec ses bureaux d'études (notamment en gestion-marketing).

Schématiquement, l'environnement est donc celui-ci :


Motel
Aux 1000 lacs
Agence de
voyage
Agence de
voyage
Motel
Citron vert
Inpres-
EnjoyYourHolidays
Head
Inpres-
EnjoyYourHolidays
Management
Agence de
voyage
Motel
Au vrai routier
Hopital
Orgiac Island
Extreme
Trek Island
Material
Motel
Le gros cochon et
la nymphomane
Village
No-Kini
Village
Cactus Club
Passe des Requins
Cracra-Hontal
Island
Cemetery Island
(in project)
Villa Sea, Sex & Sun
(île privée)

La gestion correcte d'une telle organisation touristique réclame évidemment une infrastructure informatique importante. Nous allons tout au long de ce projet en mettre en place les acteurs selon des schémas client-serveur.

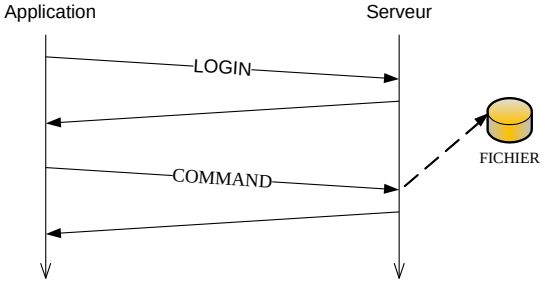
Les travaux de l'évaluation 1 : client-serveur multithread TCP en C/C++

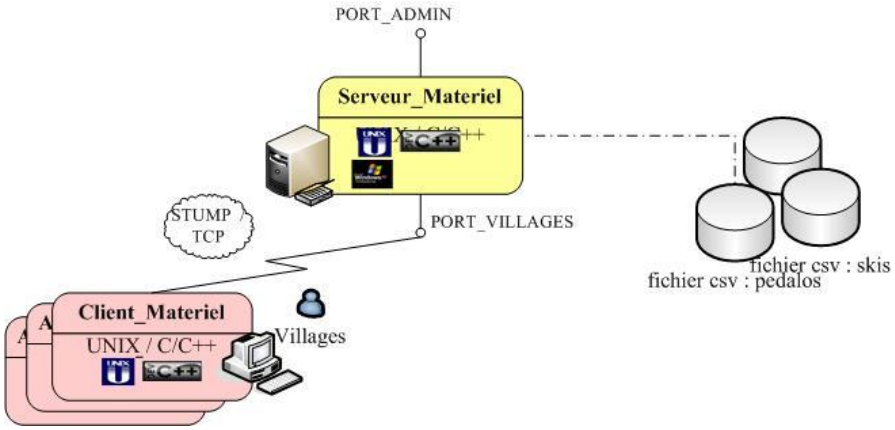
Compétences développées :

- ♦ Maîtriser l'implémentation du modèle client-serveur avec sockets TCP en C/C++;
- ♦ Concevoir une librairie de fonctions/classes utiles dans un but d'utilisation simplifiée et réutilisable dans le développement;
- ♦ Développer des fonctions/classes génériques (dissimulant suffisamment leur fonctionnement interne pour que leur utilisation ne souffre pas d'un changement d'implémentation).

Dossier attendu :

1. tableau des commandes et diagramme du protocole STUMP - du type ci-contre;
2. code C/C+ du serveur Serveur_Materiel
3. vues des trames échangées par un sniffer lors du début des connexions (états TCP).





1. Serveur_Materiel

1.1 Le service matériel pour les villages

Pour rappel, il s'agit d'un **serveur multithread C-C++/ Unix** (en *modèle pool de threads*) qui est essentiellement dévolu à la gestion (livraison, maintenance, déclassement, etc) du matériel nécessaire aux activités spécifiques des villages de vacances (pédalos, hors-bord, orgies, …). Il ne s'agit donc pas ici des réservations de matériel pour l'une ou l'autre activité (voir plus loin).

Le matériel disponible est mémorisé de manière simple dans des fichiers .csv (consultables avec un éditeur comme JEdit ou un tableur comme Excel). Chaque type de matériel sera caractérisé par un identifiant, une catégorie et son état (OK ou KO[panne, à réparer] ou DES [=destroyed, supprimé - mais on conserve sa trace], Des librairies d'accès aux bases de données relationnelles existent bien sûr en C/C++, mais elles ne sont pas du tout portables et/ou posent des problèmes de déploiement d'un système à un autre: nous les éviterons donc.

Le serveur est chargé de satisfaire les requêtes provenant d'une application **Client_Materiel** (C/C++) utilisée par les responsables des villages de vacances; le serveur attend ce type de requête sur le PORT_VILLAGE. Il utilise le ***protocole applicatif*** (basé TCP) **STUMP** (**STU**ff **M**anagement **P**rotocol) dont les commandes sont

protocole **STUMP**		
commande	**sémantique**	**réponse éventuelle**
LOGIN	un gestionnaire de matériel veut se connecter *paramètres* : nom et mot de passe	oui ou non + cause
HMAT	Handling Material : demande d'une action (livraison, réparation ou déclassement) portant sur du matériel existant (pédalo, jet-ski, barque, …) *paramètres* : action, matériel, date souhaitée	oui + identifiant donné à l'action ou non + pourquoi (par exemple : livraison impossible car produit commandé pas encore arrivé)
LISTCMD	LIST CoMmanDs : demande la liste des actions commandées	oui + liste ou non: "aucune action demandée"
CHMAT	Cancel Handling Material: suppression d'une action commandée antérieurement *paramètres* : identifiant de l'action, nom	oui ou non + date dépassée
ASKMAT	Asking for Material : commande de matériel d'un type existant déjà ou pas (1 seul item avec accessoires éventuels) *paramètres* : type existant ou pas, libellé, marque, prix approximatif, liste accessoires	oui + identifiant donné à l'action à utiliser pour solliciter la livraison + délai ou non + pourquoi

Il est clair qu'il faudra donc accéder à divers fichiers csv. Les noms de ces fichiers seront contenus dans un container associatif permettant de retrouver ces noms au moyen d'une clé simple (K_PEDALOS, K_BARQUES, K_FOUETS, …) permettant de changer facilement de fichier. Voir point 1.2 ci-dessous pour l'accès à ces fichiers.

1.2 Quelques conseils méthodologiques pour le développement de STUMP

1) Il faut tout d'abord choisir la manière d'implémenter les requêtes et les réponses des protocoles STUMP et plusieurs possibilités sont envisageables pour écrire les trames;
- uniquement par chaîne de caractères contenant des séparateurs pour isoler les différents paramètres;
- sous forme de structures, avec des champs suffisamment précis pour permettre au serveur d'identifier la requête et de la satisfaire si possible;
- un mélange des deux : une chaîne pour déterminer la requête, une structure pour les données de la requête;
- fragmenter en plusieurs trames chaînées dans le cas d'une liste à transmettre.

On veillera à utiliser des constantes (#DEFINE et fichiers *.h) et pas des nombres ou des caractères explicites.

2) On peut ensuite construire un squelette de serveur multithread et y intégrer les appels de base des primitives des sockets. Mais il faudra très vite (sous peine de réécritures qui feraient perdre du temps) remplacer ces appels par les appels correspondants d'une librairie de fonctions **SocketsUtilities** facilitant la manipulation des instructions réseaux. Selon ses goûts, il s'agira

- soit d'une bibliothèque **C** de fonctions réseaux TCP/IP;
- soit, mais c'est peut-être un peu moins évident, d'une bibliothèque **C++** implémentant une *hiérarchie de classes* **C++** utiles pour la programmation réseau TCP/IP : par exemple, Socket, SocketClient et SocketServeur;); on évitera cependant la construction de flux réseaux d'entrée et de sortie (genre NetworkStreamBase, ONetworkStream et INetworkStream) car cela devient très(trop) ambitieux pour le temps dont on dispose - des fonctions classiques send(), receive(), etc suffiront.

Dans les deux cas, un mécanisme d'erreur robuste sera mis au point.

3) Quelques remarques s'imposent :
3.1) Une remarque pleine de bon sens ;-) : une fonction (ou une méthode) de bibliothèque ne présente d'intérêt que si elle est plus facile à utiliser que la (les) fonction(s) qu'elle remplace …

3.2) Une deuxième remarque pleine de bon sens ;-) : une fonction (ou une méthode) de bibliothèque ne présente d'intérêt que si elle est indépendante du cas particulier du projet "Inpres-EnjoyYourHolidays" … Ainsi :

bien ☺	**pas bien ☹**
xxx **send**(void *,int size) /* couche basse : réutilisable dans une autre application */ xxx **AskForMaterial**(...,...) /* couche haute : propre à cette application */	xxx **send**(Material *,int size) // et pas de xxx AskForMaterial (...,...) /* une seule couche : la fonction send ne peut être réutilisée dans une autre application */

3.3) Une troisième remarque pleine de bon sens ;-) : multiplier les couches exagérément est certainement le meilleur moyen de compliquer le développement et de ralentir l'exécution !

3.4) Enfin, en tenant compte de l'administration du serveur, il serait avisé de faire intervenir dans le code du serveur **la notion d'état** de celui-ci (*certaines commandes n'ont de sens que si elles sont précédées d'une autre*).

4) Il est impérieux de surveiller les écoutes, les connexions et les communications réseaux
- au moyen des commandes **ping** et surtout **netstat** (savoir expliquer les états TCP);
- en utilisant un **sniffer** comme Wireshark ou autre encore analysant le trafic réseau (attention au localhost qui ne permet pas de sniffer simplement). Cette pratique sera demandée lors des évaluations.

5) Il serait aussi intéressant de prévoir un fichier de configuration lu par le serveur à son démarrage. A l'image des fichiers properties de Java, il s'agit d'un simple fichier texte dont chaque ligne comporte une propriété du serveur et la valeur de celle-ci :

serveur_checkin.conf

```
Port_Service=70000
Port_Admin=70009
sep-trame=$
fin-trame=#
sep-csv=;
pwd-master=tusaisquetuesbeautoi
pwd-admin=sebclachralf.
…
```

1.3 L'accès aux données du matériel

On utilise donc ici des fichiers de données de type csv. Il est bien clair que ce serveur Serveur_Materiel devra agir ultérieurement sur la base de données BD_HOLIDAYS (exposée au point suivant) et il le fera en passant par une Serveur_Data (voir plus loin).

Des librairies d'accès aux bases de données relationnelles existent bien sûr en C/C++, mais elles ne sont pas du tout portables et/ou posent des problèmes de déploiement d'un système à un autre). Nous les éviterons donc.

Comme la base BD_Holidays et le Serveur_Data ne sont pas encore disponibles, il convient de prévoir une conception logicielle qui isole les demandes à ce serveur dans une librairie de fonctions dont l'implémentation sera modifiée ultérieurement (avec le minimum de réécriture de code). On pense donc ici à des fonctions du type suivant (*ce ne sont que deux exemples - libre à vous d'en concevoir d'autres du même style*) :

librairie **AccessMaterial**		
fonction (ou méthode)	**sémantique**	**valeur retournée**
yyy getMaterial (char * number)	récupérer une élément de matériel existant	structure donnant les renseignements sur ce matériel
int addMaterial (char * number, char * category)	enregistrement d'un nouvel élément de matériel	enregistrement effectué ou pas

Les travaux de l'évaluation 2 : JDBC

Compétences développées :

- Mettre en œuvre les techniques de threads et JDBC en Java;

Dossier attendu :

1. schéma relationnel de BD_HOLIDAYS;
2. diagramme de classes UML des classes de database. facility.

2. Les accès aux bases de données

2.1 La base de données BD_ HOLIDAYS

Cette base MySQL BD_HOLIDAYS doit contenir les informations utiles concernant les voyageurs et les réservations les concernant (chambres ou activités). Ses tables, si on admet un certain nombre d'approximations, de simplifications et d'omissions sans importance pour le projet tel que défini ici (donc, avec le nombre minimum de champs pour pouvoir effectuer une démo valable du projet développé), seront en première analyse celles d'une base de données on ne peut plus classique, que l'on peut définir dans un premier temps sommairement comme contenant les tables :

- **voyageurs** : on peut distinguer le Voyageur référent (au nom de qui une réservation est faite) et le Voyageur accompagnant (qui donc fait référence à un Voyageur référent); on trouvera en tous cas un numéro de client, nom, prénom, le groupe [adresse du domicile, code postal, commune], nationalité, date de naissance, adresse e-mail;
- **chambres** : numéro (le 1er chiffre identifie un village ou un motel), équipement (comme douche/baignoire), nombre d'occupants, prix HTVA;
- **activités** : identifiant, type (choisi dans une liste), nombre maximum de participants, nombre de participants inscrits, durée, prix HTVA;
- **reservations** : identifiant (de la forme 20180915-RES01), voyageur-titulaire, référence à la chambre ou l'activité, data de début, data de fin, prix net, payé (O/N), …

On a bien sûr toute liberté pour ajouter des tables ou des champs supplémentaires aux tables existantes, voire des vues ou des contraintes, mais de manière limitée et strictement justifiée par le projet à développer ici.

2.2 Un outil d'accès aux bases de données

L'accès à la base de données ne devrait pas se faire avec les primitives JDBC utilisées telles quelles, mais plutôt au moyen d'objets métiers encapsulant le travail (typiquement de type Java Beans, mais de toute manière sans utilisation d'un mécanisme d'events).

On demande donc de construire un ensemble de telles classes (package **database.facility**) permettant l'accès (c'est-à-dire à tout le moins la connexion et les sélections de base) le plus simple possible. On souhaite pouvoir accéder, au minimum, à des bases relationnelles de type MySQL et Oracle

Le programme de test APP_TEST_LIB_JDBC de la petite librairie ainsi construite proposera un interface graphique de base permettant de se connecter à la base MySQL BD_HOLIDAYS pour y réaliser

♦ tout d'abord des requêtes élémentaires de type "select * from … where …", "select count(*) from" et "update … set … where" avec affichage des résultats (requêtes adaptées à la table visée – pas de tentative de généricité à ce stade);
♦ ensuite quelques requêtes qui seront utiles dans la suite (elles seront intégrées dans les applicatifs à développer ultérieurement) comme la liste des voyageurs inscrits à une activité donnée à une date donnée, liste des voyageurs sur une certaine période regroupés par nationalité, âge moyen des participants à une activité donnée sur une certaine période.

Rien n'interdit de lancer simultanément plusieurs utilisations de cette application qui est donc "**threadée**": attention donc aux **accès concurrents** !

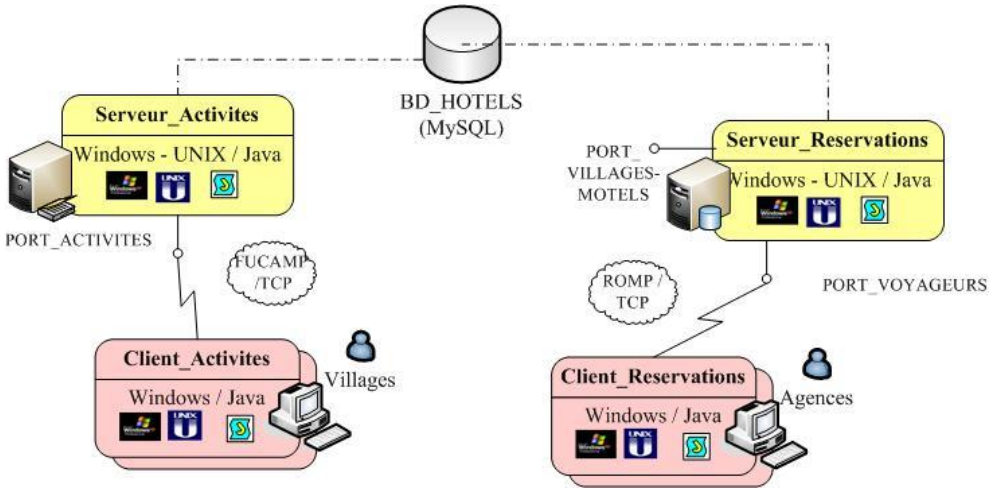
Les travaux de l'évaluation 3 : client-serveur multithread TCP en Java

Compétences développées :

- Maîtriser l'implémentation du modèle client-serveur sur base des sockets TCP en Java;
- Mettre en œuvre les techniques de threads et JDBC en Java;

Dossier attendu :

1. définition et implémentation des commandes du protocole FUCAMP;
2. code Java du serveur Serveur_Activites;
3. trames échangées et vues par un sniffer.



3. Serveur_Activites

Il s'agit d'un serveur multithread Java/Windows-Unix (en modèle pool de threads) qui est essentiellement dévolu à la gestion des activités spécifiques des villages de vacances (principalement l'inscription aux activités qui nécessitent une réservation). Comme ce genre d'informations concerne directement les voyageurs (et leur facture !), toutes les données nécessaires sont stockées dans la base de données BD_HOTELS.

Il est chargé de satisfaire les requêtes provenant d'une application **Client_Activites** (Java) utilisée par les responsables seuls habilités au travail de gestion des activités; le serveur attend ce type de requête sur le PORT_ACTIVITES. Il utilise le ***protocole applicatif*** (basé TCP) **FUCAMP** (**FU**nny and orgia**C** **A**ctivities **M**anagement **P**rotocol).

Il vous appartient de définir les commandes du protocole FUCAMP (et donc de leur donner un nom) en fonction du cahier des charges et de choisir la manière de les implémenter (objets, chaînes de caractères, etc).

Cahier des charges sous forme d'exemple (sans les éléments du GUI laissé à votre convenance):

1) >> Login ? *André Sellig*
>> Mot de passe ? *Ouftinenni*

Bonjour André !

2) Une boîte de liste propose alors un menu-liste offrant les différentes activités (d'un jour ou plus longue)

3) Un double-clic sur un item de la liste fait apparaître dans une boîte de dialogue les éléments suivants :
1. Inscription à une activité d'un jour :
paramètres : choix de l'activité (ski nautique, cours de survie, préparation du cou de girafe farci, etc), date souhaitée, nom du client

2. Inscription à une activité d'une certaine durée ("cracra-hontah" (5 jours), "extreme trek" (7 jours) ou "orgiac island" (3 jours))
paramètres : type d'activité, session du 1er ou du 15 du mois, nom

3. Liste des activités programmées participants à une activité donnée

4. Désistement d'un participant pour une activité donnée
paramètres: identification de l'activité, nom du client, date

Bien sûr, les différentes tables de la base BD_HOLIDAYS reflèteront en temps réel toutes les opérations réalisées (transactions immédiates, non différées).

4. Serveur_Reservations : réservations par agence

Nous allons ici nous préoccuper de l'implémentation du modèle client-serveur pour le serveur **Serveur_Reservations** (clients: **Client_Reservations**).

Il s'agit d'un serveur multithread Java/Windows-Unix (en modèle pool de threads) très analogue au précédent, mais qui est cette fois dévolu à la gestion des chambres des motels et des villages.

Il est tout d'abord chargé de satisfaire les requêtes provenant d'une application (Java) **Client_Reservations** utilisée par les agences de voyages qui effectuent des réservations pour les motels et les villages (ou les annulent); le serveur attend ce type de requête sur le PORT_VOYAGEURS. Comme il faut éviter que n'importe qui connaisse les noms des clients qui ont réservés (spécialement pour les villages aux activités spéciales) ou, pire, les sabote par des inscriptions fictives (par exemple), il faut d'ores et déjà envisager que ces communications réseaux soient protégées au moyen des outils de cryptographie (clés publique et privée, message digest, signature digitale). Cependant, **nous commencerons par implémenter ici provisoirement une version non sécurisée**.

Le serveur utilise le ***protocole applicatif*** (basé TCP) **ROMP** (**RO**om **M**anagement **P**rotocol), dont les commandes utilisant des techniques cryptologiques, sont

protocole **ROMP**

commande	sémantique	réponse éventuelle
LOGIN	un gestionnaire de chambres veut se connecter *paramètres* : nom et mot de passe	oui ou non
BROOM	Booking Room : réservation d'une chambre *paramètres* : catégorie (Motel ou Village), type de chambre (Simple, Double, Familiale), date d'arrivée, nombre de nuitées, nom du client de référence	oui + n° de chambre proposée + prix ou non + pourquoi

PROOM	Pay Room : accord et paiement de la réservation *paramètres* : n° de chambre, nom du client de référence et numéro de carte de crédit	
CROOM	Cancel Room : suppression d'une réservation de chambre *paramètres* : n° de chambre, nom du client	oui ou non + date dépassée
LROOMS	List of Rooms : liste des chambres d'hôtel réservées ce jour *paramètres* : -	numéros de chambres + noms des clients correspondant

Inutile sans doute de le préciser : comme ce genre d'informations concerne directement les voyageurs (et leur facture !), toutes les données nécessaires sont stockées dans la base de données BD_HOLIDAYS.