IUT de Montpellier **décembre 2019**
Département Informatique **Durée : 2h30**
2ème année

SYSTEME-RESEAUX

AVERTISSEMENT : Les sujets de système et réseaux sont prévus dans une même plage horaire. Vous utiliserez le temps à votre convenance. **Vous devrez, cependant , répondre sur deux copies différentes**

Remarque : toute ressemblance avec des choses existantes n'est que fortuite

Le UTI, organisme de formation réputé, souhaite mettre en place un concours destiné aux étudiants en informatique.

Ce concours a pour objectif de faire le point sur les connaissances acquises au cours de leur formation et s'articule autour de 3 grandes épreuves. Une épreuve sur les bases de données , une sur le système et une dernière sur la sécurité des systèmes.

Partie réseaux

A rendre sur une copie séparée de la partie système

Pour ce concours, l'établissement met à disposition deux grandes salles qui pourront accueillir plus d'une centaine de personnes. La plus grande salle pourra accueillir 15 groupes de 4 étudiants répartis en îlots et dans la deuxième salle, plus petite, seront installées deux serveurs et un poste de travail pour l'administration des équipements (voir schéma annexe 1).

Le problème est que l'on veut mettre en oeuvre un réseau, pour faire communiquer tous ces équipements et qu'il n'existe pas à ce jour.

1 - Sachant que chaque îlot doit pouvoir accueillir 4 machines qui pourront se connecter via un câble ou du Wifi. Que ces machines doivent pouvoir accéder aux 2 serveurs et que le poste administration doit pouvoir aussi accéder aux serveurs. Tous les postes étudiants doivent pouvoir accéder à internet. Proposer le câblage du site .

Remarque : Vous préciserez sur le schéma, les équipements de connexion nécessaires (hub, switch, routeur, ..), les types de câbles , …

2 – L'établissement met à disposition l'adresse IP suivante : 162.38.222. Proposez un plan d'adressage du réseau. Vous justifierez vos choix.

3 . Expliquer en 5 lignes maximum ce que fait l'application de l'annexe 2

4. Ecrire, non pas l'application qui va communiquer avec celle de l'annexe 2, mais une application qui va réussir à bloquer son fonctionnement , en quelque sorte la « planter ». Bien sûr vous expliquerez, comment vous y prendre. En fait vous êtes déjà dans le concours …

Partie système

A rendre sur une copie séparée de la partie réseaux

Pour déterminer le classement final, une application installée sur un des postes de chaque ilot (une application qui ne sera pas étudiée ici) va envoyer différentes évaluations (note correspondant à un niveau d'avancement) . En fait pour chaque épreuve, il y aura 4 notes, par groupe d'étudiants. Une note correspond à un niveau atteint, ou -1 si le niveau n'a pas été atteint. Nous allons programmer les différents processus côté serveur uniquement.

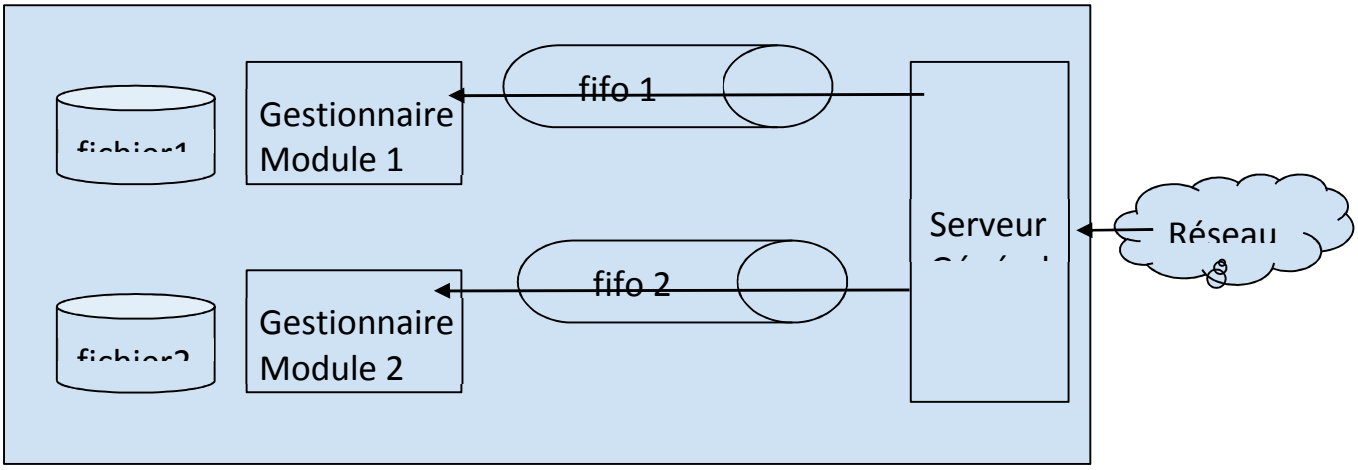


Figure 1: Architecture du service de gestion des notes.

Côté serveur, nous proposons l'architecture suivante (voir Figure 1):

- Ce serveur général va recevoir les messages réseaux en provenance des machines des différentes îlots;
- nous allons avoir un processus gestionnaire par épreuve. Chaque gestionnaire de module prendra en charge une seule épreuve. Ceci comprend l'archivage des notes des groupes et la gestion des notes absentes.

Chaque épreuve dispose d'un code unique d'identification. Ce code sera utilisé pour construire le nom de la fifo du gestionnaire correspondant et le fichier de sauvegarde avec les conventions suivantes :

- le nom de la fifo sera: '/gestionnotes/<code epreuve>'
- le nom du fichier sera: '<code epreuve>.csv'

Par exemple, si l'épreuve système à pour code : SYS1001 ; alors le path de la fifo sera : */gestionnotes/SYS1001* et le fichier de sauvegarde des notes aura pour nom: *SYS1001.csv*

Gestion des communications avec le serveur:

Le serveur va recevoir un message qui comporte exactement 4 évaluations. Pour chaque évaluation, nous allons avoir le code de l'épreuve, le code de l'étape, l'identifiant du groupe et la valeur de la note exprimée en pourcentage entre 0 et 1. En cas d'absence de note (niveau non atteint) , la valeur de la note sera de -1.

Voici les structures C qui représentent ces messages:

```
struct evaluation_t {
    char code_epreuve[8];
    char code_etape[8];
    int  id_groupe;
    float note;
};
```

```
struct message_t {
    struct evaluation_t data[4];
};
```

Les communications entre le serveur général et les gestionnaires de modules se feront par des tubes nommés.

Exercice 1: Récupération des notes

Nous proposons, le code source suivant pour le serveur général:

```
int main(int argc, char** argv)
{   etablir_connexion_reseau();
   while( en_service() )
   {  message_t * message = reception_nouveau_message();
      traitement(message); // a faire
   }
   fermeture_connexion_reseau();
}
```

Complétez le code de la fonction de traitement de message *traitement(message)* en supposant que toutes les autres fonctions (*etablir_connexion_reseau(), en_service() , reception_nouveau_message(), fermeture_connexion_reseau()*) sont données.

Aide: Voici un prototype de la fonction à compléter:

```
void traitement( struct message_t* message)
{   for(int i = 0;i < 4;i++)
 {   struct evaluation_t* eval_ptr = & message->data[i];
     char path_fifo[80];
     sprintf(path_fifo, "/gestionnotes/%s",eval_ptr->code_epreuve);
     //todo ouverture de la fifo (1 pt + 1 pt si bon mode)
     //todo ecriture de la note (1 pt + 1 pt si bons paramètres)
     //todo fermeture de la fifo (1 pt)
 }
}
```

Exercice 2: Sauvegarde des notes

Notre objectif maintenant est d'écrire le code du gestionnaire d'une épreuve pour sauvegarder les évaluations reçues dans un fichier. Compléter le code suivant pour sauvegarder les évaluations reçues dans un fichier texte. Pour chaque évaluation reçue une nouvelle ligne sera écrite au format suivant: <code etape> ; <num groupe> ; <note>

Aide: Voici un prototype de la fonction à compléter (la fonction *en_service()* est donnée):

```
int main(int argc, char** argv)
{
  if(argc != 2) { perror("usage : gestionnaire <code matiere>"); }
  char path_fifo[80];
  char path_file[80];
  sprintf(path_fifo, "/gestionnotes/%s",argv[1]);
  //creation de la fifo
  mkfifo(path_fifo, 0666);
  //todo ouverture de la fifo en lecture (1 pt)
  while( en_service() )
{
   //todo attente d une note de la fifo (1 pt)
   //todo ouverture du fichier de sauvegarde des notes (1 pt + 1 pt si bon mode ouverture)
   //todo ecriture d une nouvelle ligne au format: <code etape> ; <num groupe> ; <note> (1 pt + 1 pt))
   //fermeture du fichier          (1pt)
```

}
//todo fermeture de la fifo (1pt)
}

Exercice 3: Gestion des notes absentes

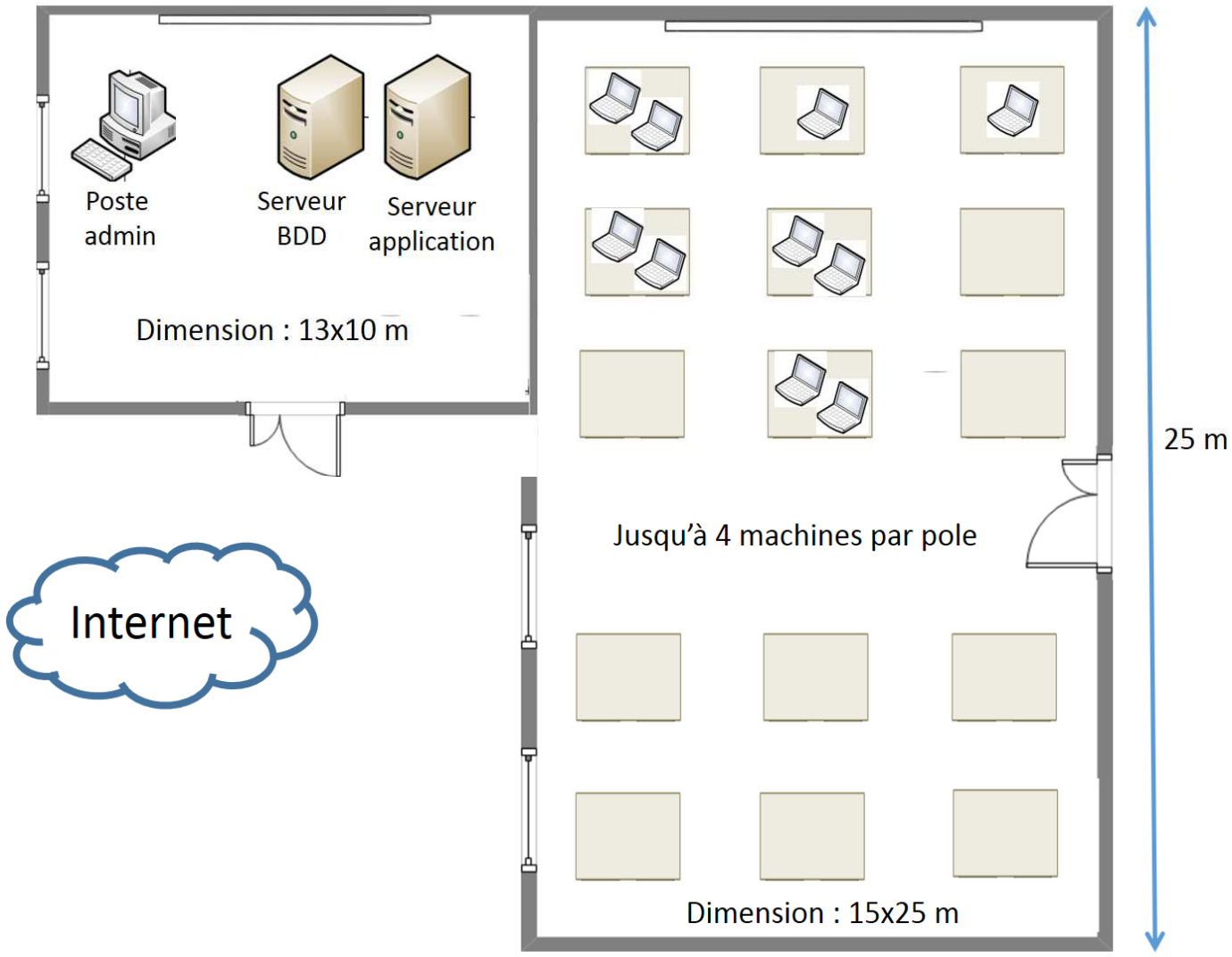
Nous souhaitons à présent gérer les évaluations manquantes. Comme souligné auparavant si un niveau n'est pas atteint alors sa note sera de -1.

Nous disposons d'un programme externe existant nommé : *alerteabsence*. Ce programme va lire sur son entrée standard une chaine de caractères au format suivant: "<num_groupe>@<code epreuve>" et va s'occuper de lancer toutes les procédures nécessaires pour gérer l'absence d'une évaluation.

Modifiez le code du gestionnaire d'épreuve de l'exercice 2 pour pouvoir gérer les absences d'évaluations des différents groupes.

Attention: Le programme *alerteabsence* existe déjà, on ne vous demande pas de le programmer mais simplement de l'utiliser avec des tubes processus.

ANNEXE 1 : Plan des locaux



ANNEXE 2 : Code serveur épreuve

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/types.h>

#define PORT 12345

main()
{  int sock, socket2, lg, nb_read;
int fd_aller[2],fd_retour[2];
  char mess[80];
  struct sockaddr_in local;
  struct sockaddr_in distant;

  bzero(&local, sizeof(local));
  local.sin_family = AF_INET;
  local.sin_port = htons(PORT);
  local.sin_addr.s_addr = INADDR_ANY;
  bzero(&(local.sin_zero),8);

  lg = sizeof(struct sockaddr_in);
  if((sock=socket(AF_INET, SOCK_STREAM,0)) == -1)
   {perror("socket"); exit(1);}
  if(bind(sock, (struct sockaddr *)&local, sizeof(struct sockaddr)) == -1)
   {perror("bind");exit(1);}
  if(listen(sock, 5) == -1){perror("listen");exit(1);}
  while(1)
   {      socket2=accept(sock, (struct sockaddr *)&distant, &lg));
             if (fork()==0)
     {    strcpy(mess,"");
         read(socket2,mess,80);
         pipe(fd_aller);
                              pipe(fd_retour);
         if (fork()!= 0)
         {            write(fd_aller[1], mess, strlen(mess));
                                    nb_read=read(fd_retour[0], mess, 80);
                                    write(socket2, mess, nb_read);
                                    close(socket2);
                              } else {
                                    dup2(fd_aller[0], 0);
                                    dup2(fd_retour[1], 1);
                                    execlp("bc", "bc", NULL);
                              }
                  }
          }
}
```

ANNEXE 3 :

int open(const char *_pathname_, int _flags_);

int close(int _fd_);

ssize_t read(int _fd_, void *_buf_, size_t _count_);

ssize_t write(int _fd_, const void *_buf_, size_t _count_);

FILE *fopen(const char *_path_, const char *_mode_);

size_t fread(void *_ptr_, size_t _size_, size_t _nmemb_, FILE *_stream_);

size_t fwrite (const void *_ptr_, size_t _size_, size_t _nmemb_, FILE *_stream_);

int sprintf(char *_str_, const char *_format_, ...);

size_t strlen(const char *_s_);

char *strcpy(char *_dest_, const char *_src_);