Examen du 26 Mars 2015 (1h30)
RSA : Réseau et Système avancés
Telecom Nancy - Deuxième année par apprentissage

Documents interdits, à l'exception d'une feuille A4 manuscrite à rendre avec votre copie.

Toutes les réponses doivent être justifiées et commentées.

Exercice 1 : questions de cours (8 pts)

1. Que signifie l'acronyme IGMP ? Expliquer entre quels équipements ce protocole intervient et à quoi il sert.
2. Qu'est-ce qu'un raw socket ? Donner un exemple de protocole qui nécessite l'utilisation de ce type de socket.
3. En 2008, suite à l'annonce d'une nouvelle version de BitTorrent utilisant le protocole UDP pour l'échange de fichiers au lieu du protocole TCP, de nombreux experts ont décrit ce changement comme irrespectueux des autres utilisateurs d'Internet et prédit une dégradation du trafic. Expliquer quels pouvaient être leurs arguments.
4. Expliquer les différences entre les modes de diffusion suivants : unicast, broadcast et multicast. Donner un exemple de service pour chacun d'eux.
5. Certaines versions avancées de TCP (par exemple TCP Vegas) essayent d'anticiper la congestion avant qu'elle ne se produise et que des paquets ne soient perdus. Expliquer la méthode utilisée.
6. Qu'est ce qu'une entrée/sortie bloquante ? Dans le cadre de l'utilisation de sockets C, donner un exemple d'une telle fonction et deux moyens de contourner ce problème.
7. Pourquoi dit-on que les connexions TCP courtes exploitent mal la bande passante ? Quel mécanisme a été introduit pour limiter ce problème ?
8. Qu'est-ce que le domaine AF_UNIX ? Quelle est la forme d'une adresse pour ce domaine ?

Exercice 2 : contrôle de congestion (6 pts)

1. Décrire deux améliorations introduites par TCP Reno par rapport à TCP Tahoe.
2. Étant donné les courbes suivantes, indiquer les moments où des mécanismes de contrôle de congestion de TCP sont mis en œuvre (Slow Start, Congestion Avoidance, Fast Recovery, Fast Retransmit, etc.). Justifier.
3. Identifier quelle implantation de TCP est illustrée par chaque courbe. Justifier.

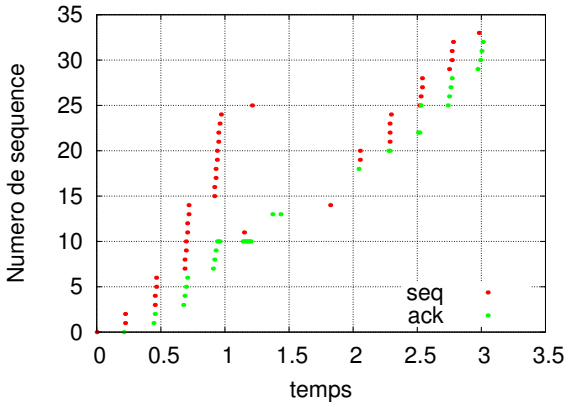

FIGURE 1 – Évolution des acquittements TCP 1

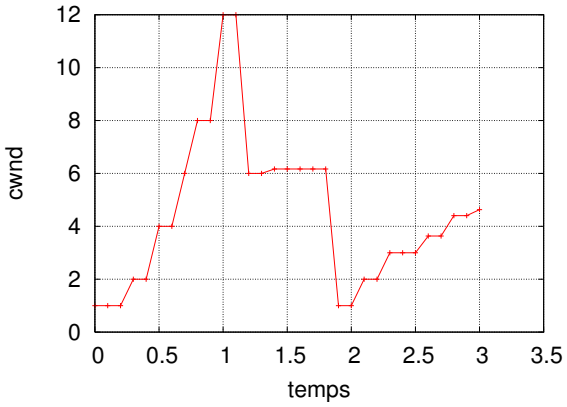

FIGURE 2 – Fenêtre de congestion TCP 1

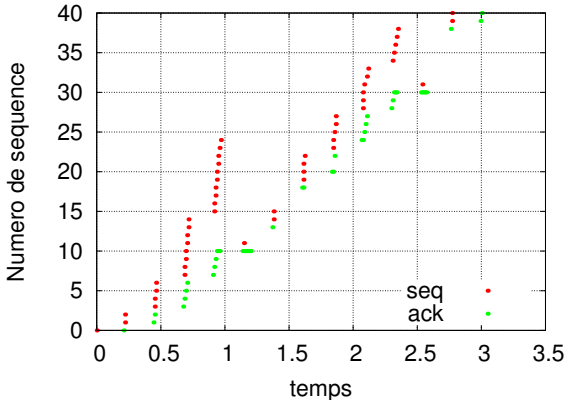

FIGURE 3 – Évolution des acquittements TCP 2

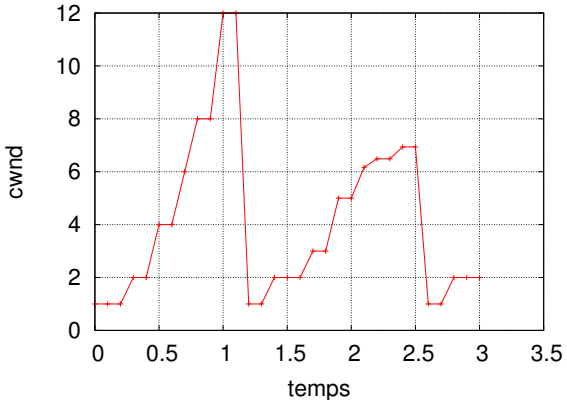

FIGURE 4 – Fenêtre de congestion TCP 2

Exercice 3 : programmation socket (6 pts)

Étant donné le code source ci-dessous d'un serveur TCP :

1. Remplacer les six commentaires mentionnés dans le code par de véritables commentaires indiquant ce que réalise la ligne le suivant immédiatement (3pts).
2. Compléter les six noms de fonctions ou de variables remplacés par "..X.." dans le code source (3pts).

```
1  #include <sys/types.h>
2  #include <sys/socket.h>
3  #include <netinet/in.h>
4  #include <arpa/inet.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include <stdlib.h>
8  #include <stdio.h>
9
10 #define BUFSIZE 1500
11 int str_echo (sockfd)
12       int sockfd;
13 {
14   int nrcv, nsnd;
15   char msg[BUFSIZE];
16
17   memset((void *)  msg, 0, sizeof(msg) );
18   if ( (nrcv= read ( ..1.., msg, sizeof(msg)-1) ) < 0 )  {
19     printf ("servselect : : read error on socket");
20     return (-1);
21   }
22
23   msg[nrcv]='\0';
24   printf ("servselect :message recu=%s nombre octets recus = %d \n",msg,..2..);
25
26   if ( (nsnd = write (sockfd, msg, nrcv) ) <0 ) {
27     printf ("servselect : write error on socket");
28     return (-1);
29   }
30   printf ("nsnd = %d \n", nsnd);
31   return (nsnd);
32 }
33
34 usage(){
35   printf("usage : servmulti numero_port_serveur\n");
36 }
37
38 main (argc, argv)
39 int argc;
40 char *argv[];
41
42 {
43   int sockfd, n, newsockfd, childpid, servlen,fin;
44   socklen_t clilen;
45   struct sockaddr_in  serv_addr, cli_addr;
46   int tab_clients[FD_SETSIZE];
47   fd_set rset, allset;
48   int maxfdp1, nbfd, mindex, i,sockcli;
```

```
49
50  printf (" argc = %d\n", argc);
51    if (argc != 2){
52      usage();
53      exit(1);
54    }
55
56    if ((sockfd = socket(AF_INET, ..3.., 0)) <0) {
57      perror("servmulti : Probleme socket\n");
58      exit (2);
59    }
60
61    memset( (char*) &serv_addr,0, sizeof(serv_addr) );
62    serv_addr.sin_family = AF_INET;
63    /*Commentaire 1*/
64    serv_addr.sin_addr.s_addr = htonl (INADDR_ANY);
65    serv_addr.sin_port = htons(atoi(..4..));
66
67    if (bind(..5..,(struct sockaddr *)&serv_addr, sizeof(serv_addr) ) <0) {
68      perror ("servmulti : erreur bind\n");
69      exit (1);
70    }
71
72    /*Commentaire 2*/
73    if (listen(sockfd, SOMAXCONN) <0) {
74      perror ("servmulti : erreur listen\n");
75      exit (1);
76    }
77
78    maxfdp1 = sockfd+1;
79    for (i=0 ; i < FD_SETSIZE; i++) {
80      tab_clients[i] = -1 ;
81    }
82    mindex = -1;
83    FD_ZERO (&allset);
84    FD_ZERO (&rset);
85    FD_SET(sockfd, &rset);
86
87    for (;;) {
88      allset = rset;// ou FD_COPY(&rset, &allset)
89
90
91      if ( (nbfd = select (maxfdp1, &allset, NULL, NULL, NULL)) < 0) {
92        perror ("probleme select \n");
93        exit(1);
94      }
95
96      /*Commentaire 3*/
97      if (FD_ISSET(sockfd, &allset)) {
98        clilen = sizeof(cli_addr);
99        newsockfd = accept(sockfd, (struct sockaddr *) ..6..,&clilen);
100       if (newsockfd < 0) {
101         perror("servmulti : erreur accept\n");
102         exit (1);
103       }
104
105       i=0;
106       while ( (i < FD_SETSIZE) && (tab_clients[i] >= 0)) i++;
107       if (i == FD_SETSIZE) {
108         printf(" trop de clients \n");
109         exit(1);
110       }
111       /*Commentaire 4*/
112       tab_clients[i]=newsockfd;
113       /*Commentaire 5*/
114       FD_SET (newsockfd, &rset);
115       if (i > mindex) mindex = i;
116       if (newsockfd >= maxfdp1)
117         maxfdp1= newsockfd+1;
118       nbfd--;
119     }
120
121     i=0;
122     /*Commentaire 6a*/
123     while ( (nbfd > 0)  && (i <= mindex)) {
124       /*Commentaire 6b*/
125       if ( (( sockcli = tab_clients[i]) >= 0)
126            && (FD_ISSET (sockcli, &allset))) {
```

```
127             if (str_echo (sockcli) == 0) {
128               close(sockcli);
129               tab_clients[i] = -1;
130               FD_CLR(sockcli, &rset);
131             }
132             nbfd--;
133           }
134           i++;
135         }
136
137     }
138   }
```