UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS
UFR FACULTÉ DES SCIENCES

LICENCE D'INFORMATIQUE 2004-2005

L36I – programmation système et réseaux

CORRIGÉ

Corrigé de l'examen de juin

PREMIERE PARTIE

Question 1 (Informatique et EEA) *(3 points ≃ 18 mn)*

- 1. `socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)`
- 2. `AF_INET`
- 3. `INADDR_ANY`
- 4. `if(bind(mess_sock, (SOCKADDR) &seraddr, sizeof(seraddr))<0) ERRNO_EXIT;`
- 5. `if(listen(mess_sock, NBMAXCLIENTS)<0) ERRNO_EXIT;`
- 6. `if((clisock= accept(mess_sock, (SOCKADDR) &cliaddr, &cliaddr_len))<0) ERRNO_EXIT;`
- 7. `close(mess_sock);`
- 8. `close(clisock);`
- 9. `if (signal(SIGUSR1, traitantUsr1)==SIG_ERR) ERRNO_EXIT;`

Question 2 (Informatique et EEA) *(6 points ≃ 36 mn)*

```
void clientProcessing (int clisock, struct sockaddr_in* cliaddr){
  char buf[MESSMAX];
  int  buflen;
  int  OK;
  do {
    OK=0;
    if((buflen= recv(clisock, buf, 1, AF_UNSPEC))<0) ERRNO_EXIT;
    buf[buflen]='\0';
    if (buf[0] == 'U') {
      OK=(kill(getppid(), SIGUSR1)==0);
    } else if (buf[0] == 'D') {
      // lecture longueur
      int textlen;
      if((buflen= recv(clisock, buf, 10, AF_UNSPEC))<0) ERRNO_EXIT;
      buf[buflen]='\0';
      sscanf(buf, "%10d", &textlen);
      // lecture du texte
      int ntoread= textlen;
      int nread=0;
      while(ntoread && (buflen=read(clisock, buf, ntoread))>0){
        buf[buflen]='\0';
        printf("%s", buf);
        ntoread-=buflen;
        nread+= buflen;
      }
      OK=(nread==ntoread);
    } else if (buf[0] == 'E') {
      return;
    }
    if(OK)
      if(send(clisock, "S", 1, AF_UNSPEC)<0) ERRNO_EXIT;
      else
        if(send(clisock, "F", 1, AF_UNSPEC)<0) ERRNO_EXIT;
  } while (OK);
}
```

Question 3 (Informatique et EEA) *(3 points ≃ 18 mn)*

Question 3.1 *(1 point ≃ 6 mn)*

Le traitant du signal SIGUSR1 doit envoyer (primitive kill) un signal de terminaison (SIGTERM) à tous les processus SPS. Cela suppose que lors des connexions des clients, le processus père note dans un tableau les pids des processus fils de type SPS (ou qu'il forme un groupe de processus avec les processus SPS, mais cela n'a pas été étudié cette année). Il faut alors mettre à jour ce tableau par le père par des primitives wait afin de connaitre les processus SPS qui se sont terminés. Il faut aussi que chaque processus fils dispose d'un traitant du signal TERM (armé par signal au début de la procédure clientProcessing) et que celui-ci ferme la socket qui le lie au client.

Question 3.2 *(2 points ≃ 12 mn)*

L'écriture des textes envoyés par des clients simultanés peut être entrelacée. Pour éviter ce problème, on doit considérer la voie standard de sortie comme une ressource critique et l'acquérir par un sémaphore d'exclusion mutuelle. Ce sémaphore doit être créé et initialisé à 1 avant que le serveur crée un processus SPS :

```
static int mutex;
if((mutex= semget(IPC_PRIVATE, 1, S_IRWXU))<0) ERRNO_EXIT;
V(mutex,1); // initialise le sémaphore mutex à 1
```

La séquence de code dans la procédure clientProcessing doit être enrichie de la manière suivante :

```
void clientProcessing (int clisock, struct sockaddr_in* cliaddr){
...
} else if (buf[0] == 'D') {
...
P(mutex,1)
/* lecture et affichage du texte */
V(mutex,1)
} else if (buf[0] == 'E') {
...
```

Les opérations P et V peuvent être programmées de la façon suivante :

```
struct sembuf action;
void P (int sem, int i) {
action.sem_num=0; action.sem_op= -i; action.sem_flg= 0;
if(semop(sem, &action, 1)<0) ERRNO_EXIT;
}
void V (int sem, int i) {
action.sem_num=0; action.sem_op= i; action.sem_flg= 0;
if(semop(sem, &action, 1)<0) ERRNO_EXIT;
}
```

Question 4 (Informatique seulement) *(6 points ≃ 36 mn)*

```
#include <stevens.h>

void findExec ( char *pathName, FILE_HANDLER *func ) {
  struct stat     statbuf;          /* tampon des infos du i-noeud          */
  struct dirent   *entry_ptr;       /* curseur de parcours dans répertoire  */
  DIR             *directory_ptr;   /* descripteur abstrait de répertoire   */
  char            *pathName_ptr;   /* pointeur courant dans pathName       */

  /* Place dans statbuf les infos du fichier pathName */
  if (lstat(pathName, &statbuf) < 0) ERRNO_EXIT;

  /* Applique func au fichier ou répertoire pathName */
  func(pathName, &statbuf );

  /* Si ce n'est pas un répertoire, on retourne */
  if ( S_ISDIR(statbuf.st_mode) == 0) return;

  /* Ajouter "/" à pathName */
  pathName_ptr = pathName + strlen(pathName);
  *pathName_ptr++ = '/';
  *pathName_ptr = '\0';

  /* lecture du répertoire */
  if ( (directory_ptr = opendir(pathName)) == NULL) ERRNO_EXIT;

  /* parcours séquentiel des fichiers du répertoire */
  while ( (entry_ptr = readdir(directory_ptr)) != NULL) {

    /* ignorer "." er ".." */
    if (strcmp(entry_ptr->d_name, ".") == 0  || strcmp(entry_ptr->d_name, "..") == 0)
      continue;

    /* ajout du nom après le "/" */
    strcpy(pathName_ptr, entry_ptr->d_name);

    /* appel récursif de func pour full_path */
    findExec (pathName, func);
  }

  /* restaure 'pathname' dans l'état qu'il avait à l'entrée de cette routine */
  pathName_ptr[-1] = '\0';

  if (closedir(directory_ptr) < 0) ERRNO_EXIT;
}
```

DEUXIEME PARTIE

Question 5 (Informatique et EEA) *(4 points ≃ 24 mn)*

Déclarations communes

```
int nb_vide = N; /* compteur de cases vides */
int nb_plein = 0; /* compteur de cases pleines */
condition plein, vide; /* variables de condition */
```

```
void ajouter_obj(objet)
debut
if (nb_vide == 0)
WAIT(vide);
nb_vide-; nb_plein++;
add(buffer,objet);
SIGNAL(plein);
fin
```

```
objet retirer_obj(void)
debut
if (nb_plein == 0)
WAIT(plein);
nb_plein-; nb_vide++;
objet = remove(buffer);
SIGNAL(vide);
return objet;
fin
```

Question 6 (Informatique et EEA) *(4 points ≃ 24 mn)*

Question 6.1 *(0,5 points ≃ 3 mn)*

La représentation réseau suit le format MSB (*Most Significant Byte First*). Donc la valeur des 4 octets pour cette adresse est : $O_1 = 138$, $O_2 = 96$, $O_3 = 232$ et $O_4 = 4$.

Question 6.2 *(0,5 points ≃ 3 mn)*

Les octets sont stokés au format réseau, donc : 138,96,232 et 4.

Question 6.3 *(1 point ≃ 6 mn)*

On ajoute 1 à l'octet de poids faible mais dans la représentation interne du processeur, qui est $O_3O_1O_4O_2$. Dans l'ordre décroissant des adresses mémoire, c'est donc le 3e octet (O_4) qui sera augmenté de 1 (octet de poids faible). On a vu à la question précédente que cet octet contient 232, donc sa valeur devient 233. Par conséquent la fonction `inet_ntoa` retourne la chaîne "138.96.233.4", ce qui aboutit à l'affichage suivant :
"En ajoutant 1 j'obtiens : 138.96.233.4".

Question 6.4 *(2 points ≃ 12 mn)*

ntohl convertit du format réseau au format host. L'octet de poids fort de l'adresse au format réseau se trouve dans l'octet du tableau ayant l'indice 0, et l'octet de poids faible dans dans celui d'indice 3. Dans le résultat au format host, les 4 octets doivent être placés dans le tableau de la façon suivante $O_3O_1O_4O_2$, où O_3 se trouve à l'indice 0, et 0_2 à l'indice 3.

```
char *ntohl(char *netval)
{
    char hostval[4];
    hostval[0] = netval[2];
    hostval[1] = netval[0];
    hostval[2] = netval[3];
    hostval[3] = netval[1];
    return hostval;
}
```

Question 7 (Informatique seulement) *(4 points ≃ 24 mn)*

Question 7.1 *(0,5 points ≃ 3 mn)*

0, 0, 1, 1, 0, 3, 1, 2, 2, 4, 4, 3

Question 7.2 *(1 point ≃ 6 mn)*

Le processus dispose de 2 cadres (car 200 mots). Notons les C_1 et C_2.
Alors, l'occupation des cadres au cours du temps est la suivante (le signe + indique la ou les pages qui seront référencées le plus tard dans le futur) :

OPT												
refs :	0	0	1	1	0	3	1	2	2	4	4	3
C_1	+0	+0	+0	0	+0	+3	3	+3	3	3	3	+3
C_2	-	-	1	+1	1	1	+1	2	+2	+4	+4	+4
def ?	D		D	-	-	D	-	D	-	D	-	-

Il y a eu 5 défauts de page sur 12 accès au total, donc le taux de défauts de pages est $T_{OPT} = \frac{5}{12}$.

Question 7.3 *(1 point ≃ 6 mn)*

Le processus dispose de 2 cadres (car 200 mots). Notons les C_1 et C_2.
Alors, l'occupation des cadres au cours du temps est la suivante (le signe + indique la page qui a été référencée le plus loin dans le passé) :

LRU												
refs :	0	0	1	1	0	3	1	2	2	4	4	3
C_1	+0	+0	+0	+0	0	+0	1	+1	+1	4	4	+4
C_2	-	-	1	1	+1	3	+3	2	2	+2	+2	3
def ?	D		D	-	-	D	D	D	-	D	-	D

Il y a eu 7 défauts de page sur 12 accès au total, donc le taux de défauts de pages est $T_{OPT} = \frac{7}{12}$.

Question 7.4 *(0,5 points ≃ 4 mn)*

OPT n'est pas réaliste (et donc réalisable) car il suppose que l'on sait à l'avance quelle seront les futures références du programme.
La raison pour laquelle on l'étudie quand même est que OPT est l'algorithme théoriquement optimal (d'où son nom). On l'utilise donc à titre de comparaison pour savoir à quel point un algorithme est bon (ou mauvais).

Question 7.5 *(1 point ≃ 5 mn)*

L'algorithme qui choisit de vider le cadre qui contient la page qui sera utilisée le plus tôt dans le futur est un bon candidat. C'est bien le pire algorithme, car il garantit qu'à chaque défaut de page, on fait le plus mauvais choix possible en déchargeant systématiquement les pages qui seront utiles à nouveau dans un avenir proche.