Master1 – POSIX	- Durée : 3 heures
Décembre 2006	- Toute documentation autorisée
Examen	- Barème indicatif
	Luciana Arantes, Olivier Marin, Bertil Folliot

1. THREADS

Considérez le programme ci-dessous dont la thread *main* crée NB_THREADS joignables en attendant après, leur fin, avant de se terminer elle-même.

```
1: #define NB_THREADS 3

2: void *func_thread (void *arg) {
3:  printf ("arg :%d\n", *((int*)arg));
4:  return NULL;
5: }


6: int main (int argc, char ** argv) {
7:  pthread_t tid[NB_THREADS]; int i=0; int *pt_ind;

8:  for (i=0; i<NB_THREADS; i++) {
9:     pt_ind = (int *) malloc (sizeof (i));
10:    *pt_ind =i;

11:   if (pthread_create (&(tid[i]), NULL, func_thread,
         (void *)pt_ind ) != 0)
          exit (1);
12:  }

13:  for (i=0; i < NB_THREADS; i++)
14:     if (pthread_join (tid[i], NULL) !=0)
          exit (1);


15:   printf ("Toutes les threads créées se sont terminées ");
16:   return EXIT_SUCCESS;
}
```

1.1

Est-ce que le programme marcherait toujours de la même façon si au lieu des threads joignables il crée des threads détachées ? Justifiez votre réponse.

Considérez maintenant que nous voulons modifier le programme afin de créer des threads **détachées** mais de façon que la thread *main* continue toujours à se terminer après les autres NB_THREADS threads. Pour cela nous devrons aussi y ajouter un mécanisme de synchronisation/contrôle entre la thread main et les autres threads.

1.2

Modifiez le programme en conséquence en utilisant comme mécanisme de synchronisation/contrôle, des variables de conditions.

1.3

Même question en utilisant comme mécanisme de synchronisation/contrôle, des signaux.

1.4

Même question en utilisant comme mécanisme de synchronisation/contrôle, des sémaphores POSIX. Si vous voulez, ce n'est pas nécessaire de donner le code du programme. Vous pouvez seulement décrire comment les threads utiliseront les sémaphores, où les placer dans chaque thread, ainsi que préciser leur déclaration et initialisation.

1.5

En considérant vos 3 solutions précédentes, est-ce que la thread *main* se terminerait toujours après les autres si une des threads créées décidait d'annuler l'exécution d'une deuxième ? Justifiez votre réponse.

2. IPC POSIX

On voudrait obtenir un programme composé de trois processus : l'un lit ce que l'utilisateur rentre sur le terminal, un autre encode la chaîne de caractères lue, puis le troisième enregistre le résultat dans un fichier log. Programmé naïvement, voici le code produit pour une telle application :

```
#define BUFSZ 1024
char *buf;

int main(int argc, char **argv)
{

    int or = 1, pid, i;

    pid = fork();
    if (pid == 0) {
        while (1) {
            i = 0;
            while (buf[i] != '\0') {
                buf[i] = toupper(buf[i]);
                i++;
            }
            printf("son1: %s\n", buf);
        }
    }
    if (fork()) {
        while (or > 0) {
            printf("father: ");
            fflush(NULL);
            for (i = 0; i < BUFSZ ; i++)
                buf[i] = '\0';
            or = read(STDIN_FILENO, buf, BUFSZ);
            if (or > 0) {
                printf("%s\n", buf);
            }
        }
    } else {
        int fd = open("log", O_WRONLY | O_TRUNC | O_CREAT, 0600);
        while (1) {
            printf("son2: logging \"%s\"\n", buf);
            write(fd, buf, (int)strlen(buf));
        }
    }
}
```

2.1

Expliquez en deux points pourquoi ce programme ne peut pas fonctionner.

2.2

Donnez les modifications à apporter au code pour que le programme fonctionne comme spécifié.

2.3

En prenant soin d'expliquer vos choix d'implémentation, reprenez le code de ce programme pour obtenir le même résultat en n'utilisant **qu'une seule et unique** file de messages IPC System V.