Programmation système II

Évaluation - Timothée Ravier

1e année cycle ingénieur STI, juin 2014

1h20 – Documents non autorisés

1 Threads et synchronisation (7 points)

- **Question 1.1 (2 points) :** Quelles sont les différences entre un processus et un thread ?
- **Question 1.2 (1 point) :** Expliquez le concept de section critique.
- **Question 1.3 (1 point) :** Quelle est la différence entre un mutex et un semaphore ?
- **Question 1.4 (1 points) :** Expliquez ce que fait ce programme et pourquoi le résultat n'est pas celui attendu :

Code du programme :

```
1  int NB_THREADS = 2;
2  int parts_de_gateau = 4;
3
4  void *prendre_part(void * num)
5  {
6      int numero = *((int *)num);
7
8      while (parts_de_gateau > 0) {
9          printf("Thread %d : il y a %d parts de gateau.\n", numero, parts_de_gateau);
10         printf("Thread %d : mange une part de gateau.\n", numero);
11         --parts_de_gateau;
12         printf("Thread %d : il reste %d parts de gateau.\n", numero, parts_de_gateau);
13         usleep(1);
14     }
15
16     pthread_exit(NULL);
17 }
18
19 int main(int argc, const char *argv[])
20 {
21     pthread_t thread[NB_THREADS];
22     int numero_thread[NB_THREADS];
23     int i, ret;
24
25     for (i = 0; i < NB_THREADS; ++i) {
26         numero_thread[i] = i;
27         pthread_create(&thread[i], NULL, prendre_part, &numero_thread[i]);
28     }
29
30     for (i = 0; i < NB_THREADS; ++i) {
31             pthread_join(thread [i], NULL);
32     }
33
```

```
34      printf("Main : il reste %d parts de gateau.\n", parts_de_gateau);
35
36      return EXIT_SUCCESS;
37  }
```

Sortie du programme :

```
Thread 1 : il y a 4 parts de gateau.
Thread 1 : mange une part de gateau.
Thread 1 : il reste 3 parts de gateau.
Thread 0 : il y a 4 parts de gateau.
Thread 0 : mange une part de gateau.
Thread 0 : il reste 2 parts de gateau.
Thread 1 : il y a 2 parts de gateau.
Thread 1 : mange une part de gateau.
Thread 1 : il reste 1 parts de gateau.
Thread 0 : il y a 1 parts de gateau.
Thread 0 : mange une part de gateau.
Thread 0 : il reste 0 parts de gateau.
Thread 1 : il y a 1 parts de gateau.
Thread 1 : mange une part de gateau.
Thread 1 : il reste -1 parts de gateau.
Main : il reste -1 parts de gateau.
```

- **Question 1.1 (2 points) :** Proposez une correction du programme. Ne recopiez que le minimum nécessaire sur votre copie.

2 IPC (3 points)

- **Question 2.1 (1 points) :** Citez quatre mécanismes de communication inter processus (SystemV ou POSIX).
- **Question 2.2 (2 points) :** Expliquez le comportement de ces deux programmes :

Programme 1 :

```
1   int main()
2   {
3       ssize_t taille_lue = 0, taille_ecrite = 0, retour_write = 0;
4       const size_t buffer_size = 2048;
5       char buffer[buffer_size];
6       char prio_buffer[256];
7       mqd_t file_message;
8       unsigned int priorite;
9       int ret = 0;
10
11      file_message = mq_open("/filemessage", O_RDONLY);
12
13      while (1) {
14          taille_lue = mq_receive(file_message, buffer, buffer_size, &priorite);
15          if (taille_lue == 0) {
16              break;
17          } else {
18              snprintf(prio_buffer, 256, "Priorite %d: ", priorite);
19              taille_ecrite = 0;
```

```
20                 while (taille_ecrite < ret) {
21                     retour_write = write(STDOUT_FILENO, prio_buffer + taille_ecrite, ret -taille_ecrite);
22                     taille_ecrite += retour_write;
23                 }
24                 taille_ecrite = 0;
25                 while (taille_ecrite < taille_lue) {
26                     retour_write = write(STDOUT_FILENO, buffer + taille_ecrite, taille_lue -taille_ecrite);
27                     taille_ecrite += retour_write;
28                 }
29             }
30         }
31
32         return EXIT_SUCCESS;
33 }
```

Programme 2 :

```
1  int main()
2  {
3      ssize_t taille_lue = 0;
4      const size_t buffer_size = 2048;
5      char buffer[buffer_size];
6      mqd_t file_message;
7      struct mq_attr attr;
8      int ret = 0;
9
10     memset(&attr, 0, sizeof(struct mq_attr));
11     attr.mq_maxmsg = 10;
12     attr.mq_msgsize = 2048;
13
14     file_message = mq_open("/filemessage", O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR, &attr);
15     if (file_message == (mqd_t) -1) {
16             errExit("mq_open");
17     }
18
19     while (1) {
20         taille_lue = read(STDIN_FILENO, buffer, buffer_size);
21         if (taille_lue == 0) {
22             mq_send(file_message, buffer, 0, 0);
23             break;
24         } else {
25             mq_send(file_message, buffer, (unsigned int)taille_lue, rand() % 10);
26         }
27     }
28
29     mq_unlink("/filemessage");
30
31     return EXIT_SUCCESS;
32 }
```

3 Signaux (5 points)

- **Question 3.1 (1 point) :** Qu'est-ce qu'un signal ?
- **Question 3.2 (2 points) :** Donner quatre exemples de signaux et expliquer leur sens.

- **Question 3.3 (2 points) :** Voici un le code source d'un programme et les commandes qui ont été tapées dans un autre terminal pendant son exécution. Quelle est la sortie de ce programme ? Expliquez.

Code du programme :

```
1  int temperature = 20;
2
3  static void cas_special(int sig)
4  {
5        --temperature;
6  }
7
8  static void urgence(int sig)
9  {
10       ++temperature;
11 }
12
13 int main(int argc, const char *argv[])
14 {
15     struct sigaction sa1, sa2, sa3;
16
17     sigfillset(&sa1.sa_mask);
18     sa1.sa_flags = SA_RESTART;
19     sa1.sa_handler = cas_special;
20     sigaction(SIGUSR1, &sa1, NULL);
21
22     sigfillset(&sa2.sa_mask);
23     sa2.sa_flags = SA_RESTART;
24     sa2.sa_handler = urgence;
25     sigaction(SIGUSR2, &sa2, NULL);
26
27     sigfillset(&sa3.sa_mask);
28     sa3.sa_flags = SA_RESTART;
29     sa3.sa_handler = SIG_IGN;
30     sigaction(SIGINT, &sa, NULL);
31
32     printf("Mon PID est : %ld\n", (long) getpid());
33
34     while (1) {
35         printf("Temprature courante : %d\n", temperature);
36         sleep(1);
37     }
38
39     return EXIT_SUCCESS;
40 }
```

Le début de la sortie du programme :

```
Mon PID est : 7153
Temprature courante : 20
Temprature courante : 20
...
```

Commandes exécutées :

```
$ kill -SIGUSR1 7153
$ kill -SIGUSR2 7153
```

```
$ kill -SIGINT 7153
$ kill -SIGUSR2 7153
$ kill -SIGUSR1 7153
$ kill -SIGINT 7153
$ kill -SIGUSR2 7153
$ kill -SIGINT 7153
$ kill -SIGTERM 7153
$ kill -SIGUSR1 7153
$ kill -SIGUSR2 7153
```

4 Socket UNIX (3 points)

- **Question 4.1 (1 point) :** Quelle est la différence entre une socket réseau et une socket UNIX ?
- **Question 4.2 (2 point) :** Donner le pseudo-code d'un client et d'un serveur communiquant à l'aide d'un socket UNIX. Ce qui importe ici ce sont le nom des fonctions appelées et leur ordre.

Exemple de pseudo-code :

```
1  main()
2  {
3      char buffer[256];
4      int fichier;
5
6      fichier = open("fichier", READ);
7      while(read(fichier, buffer, 256)) {
8          printf(buffer);
9      }
10
11     return 0;
12 }
```

5 Gestion des entrées sorties (2 points)

- **Question 5.1 (1 point) :** Que signifie l'attribut `O_NONBLOCK` ?
- **Question 5.2 (1 point) :** A quoi servent les fonctions `select`, `poll` et `epoll` ?