


# Compte Rendu STR - TP0

## Exemple 1 : 2 Taches sous POSIX

### Code



```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <pthread.h>
5
6  void *fonc(void *arg)
7  {
8      int i;
9      for (i = 0; i < 7; i++)
10     {
11         printf("Tache %d : %d\n", (long)arg, i);
12         usleep(1000000);
13     }
14 }
15 int main(void)
16 {
17     pthread_t tache1, tache2;
18     pthread_create(&tache1, NULL, fonc, (void *)1);
19     pthread_create(&tache2, NULL, fonc, (void *)2);
20     pthread_join(tache1, NULL);
21     pthread_join(tache2, NULL);
22     return 0;
23 }
```

### Execution

```
wa101@wa101-latitude5490:~/STR-TP0
~/STR-TP0 > ./example1
Tache 1 : 0
Tache 2 : 0
Tache 1 : 1
Tache 2 : 1
Tache 1 : 2
Tache 2 : 2
Tache 2 : 3
Tache 1 : 3
Tache 1 : 4
Tache 2 : 4
Tache 2 : 5
Tache 1 : 5
Tache 2 : 6
Tache 1 : 6
~/STR-TP0 > 7s
```

## Exemple 2 : Priorite et ordonnancement de Taches

Code

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <pthread.h>
5  void *fonc(void *arg)
6  {
7      int i;
8      for (i = 0; i < 7; i++)
9      {
10         printf("Tache %d : %d /**/", (long)arg, i);
11         usleep(1000000);
12     }
13 }
14 int main(void)
15 {
16     pthread_t tache1, tache2;
17     pthread_attr_t attr;
18     struct sched_param param;
19     pthread_attr_init(&attr);
20     param.sched_priority = 12;
21     pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
22
23     pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
24     pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
25     param.sched_priority = 10;
26     pthread_attr_setschedparam(&attr, &param);
27     pthread_create(&tache1, &attr, fonc, (void *)1);
28     param.sched_priority = 7;
29     pthread_attr_setschedparam(&attr, &param);
30     pthread_create(&tache2, &attr, fonc, (void *)2);
31
32     pthread_attr_destroy(&attr);
33     pthread_join(tache1, NULL);
34     pthread_join(tache2, NULL);
35     return 0;
36 }

```

## Execution

```
wa101@wa101-latitude5490:~/STR-TP0
~/STR-TP0 > sudo ./exemple2
Tache 1 : 0 /**/Tache 2 : 0 /**/Tache 2 : 1 /**/Tache 1 : 1 /**/Tache 2 : 2 /**/
Tache 1 : 2 /**/Tache 2 : 3 /**/Tache 1 : 3 /**/Tache 2 : 4 /**/Tache 1 : 4 /**/
Tache 2 : 5 /**/Tache 1 : 5 /**/Tache 2 : 6 /**/Tache 1 : 6 /**/
~/STR-TP0 > 7s
```

### Exemple 3 : Exclusion mutuelle

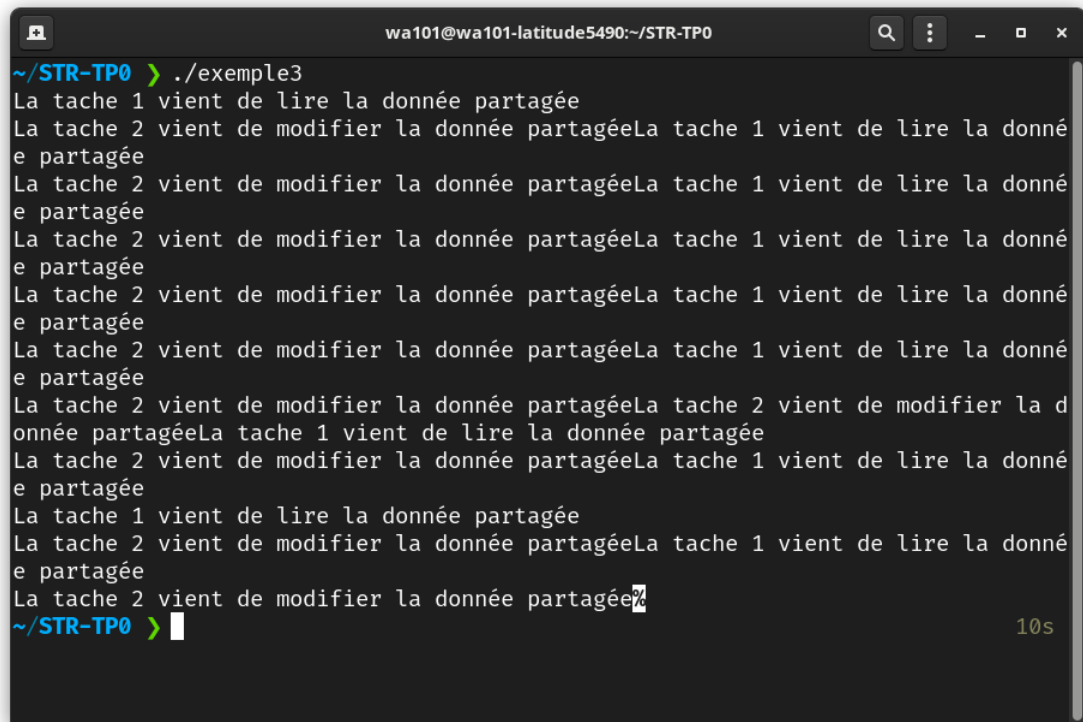
Code

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <pthread.h>
5  typedef struct
6  {
7      float taille;
8      float poids;
9  } type_donneePartagee;
10 pthread_mutex_t verrou;
11
12 type_donneePartagee donneePartagee;
13
14 void *tache1(void *arg)
15 {
16
17     type_donneePartagee ma_donneePartagee;
18
19     int i = 0;
20
21     while (i < 10)
22     {
23         pthread_mutex_lock(&verrou);
24         ma_donneePartagee = donneePartagee;
25         pthread_mutex_unlock(&verrou);
26         printf("La tache %s vient de lire la donnée partagée\n", (char *)arg);
27         usleep(1000000);
28         i++;
29     }
30 }
31
32 void *tache2(void *arg)
33 {
34     int i = 0;
35     while (i < 10)
36     {
37         pthread_mutex_lock(&verrou);
38         donneePartagee.taille = 100 + rand() % 101;
39         donneePartagee.poids = 10 + rand() % 101;
40         pthread_mutex_unlock(&verrou);
41         printf("La tache %s vient de modifier la donnée partagée", (char *)arg);
42         usleep(1000000);
43         i++;
44     }
45 }
46
47 int main(void)
48 {
49     pthread_t th1, th2;
50     pthread_mutex_init(&verrou, NULL);
51     donneePartagee.taille = 100 + rand() % 101;
52     donneePartagee.poids = 10 + rand() % 101;
53     pthread_create(&th1, NULL, tache1, "1");
54     pthread_create(&th2, NULL, tache2, "2");
55     pthread_join(th1, NULL);
56     pthread_join(th2, NULL);
57     return 0;
58 }

```

## Execution



```
wa101@wa101-latitude5490:~/STR-TP0
~/STR-TP0 > ./exemple3
La tache 1 vient de lire la donnée partagée
La tache 2 vient de modifier la donnée partagéeLa tache 1 vient de lire la donn
e partagée
La tache 2 vient de modifier la donnée partagéeLa tache 1 vient de lire la donn
e partagée
La tache 2 vient de modifier la donnée partagéeLa tache 1 vient de lire la donn
e partagée
La tache 2 vient de modifier la donnée partagéeLa tache 1 vient de lire la donn
e partagée
La tache 2 vient de modifier la donnée partagéeLa tache 2 vient de modifier la d
onnée partagéeLa tache 1 vient de lire la donnée partagée
La tache 2 vient de modifier la donnée partagéeLa tache 1 vient de lire la donn
e partagée
La tache 1 vient de lire la donnée partagée
La tache 2 vient de modifier la donnée partagéeLa tache 1 vient de lire la donn
e partagée
La tache 2 vient de modifier la donnée partagée%
~/STR-TP0 > 10s
```

## Exemple 4 : Exclusion mutuelle & variable condition

### Code

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <pthread.h>
5  typedef struct
6  {
7      float taille;
8      float poids;
9  } type_donneePartagee;
10 pthread_mutex_t verrou;
11 pthread_cond_t cond =
12     PTHREAD_COND_INITIALIZER;
13 type_donneePartagee donneePartagee;
14 void *tache1(void *arg)
15 {
16     type_donneePartagee ma_donneePartagee;
17     int i = 0;
18     while (i < 10)
19     {
20         pthread_mutex_lock(&verrou);
21         pthread_cond_wait(&cond, &verrou);
22         ma_donneePartagee = donneePartagee;
23         pthread_mutex_unlock(&verrou);
24         printf("La tache %s vient de lire la donnee partagee\n", (char *)arg);
25         usleep(1000000);
26         i++;
27     }
28 }
29 void *tache2(void *arg)
30 {
31     int i = 0;
32     while (i < 10)
33     {
34         pthread_mutex_lock(&verrou);
35         donneePartagee.taille = 100 + rand() % 101;
36         donneePartagee.poids = 10 + rand() % 101;
37         if (donneePartagee.taille >= 120 && donneePartagee.poids >= 60)
38         {
39             pthread_cond_signal(&cond);
40         }
41         pthread_mutex_unlock(&verrou);
42         printf("La tache %s vient de modifier la donnee partagee\n", (char *)arg);
43         usleep(1000000);
44         i++;
45     }
46 }
47 int main(void)
48 {
49     pthread_t th1, th2;
50     pthread_mutex_init(&verrou, NULL);
51     donneePartagee.taille = 100 + rand() % 101;
52     donneePartagee.poids = 10 + rand() % 101;
53     pthread_create(&th1, NULL, tache1, "1");
54     pthread_create(&th2, NULL, tache2, "2");
55     pthread_join(th1, NULL);
56     pthread_join(th2, NULL);
57     return 0;
58 }

```

## Execution

```
./exemple4
~ > cd STR-TP0
~/STR-TP0 master !1 ?5 > ls
exemple1  exemple2  exemple3  exemple4  exemple5  exemple6
exemple1.c  exemple2.c  exemple3.c  exemple4.c  exemple5.c  exemple6.c
~/STR-TP0 master !1 ?5 > ./exemple4
La tache 2 vient de modifier la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 1 vient de lire la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 1 vient de lire la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 1 vient de lire la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 2 vient de modifier la donnee partagee
La tache 1 vient de lire la donnee partagee
```

## Exemple 5 : Taches periodique sous POSIX

### Code



```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <time.h>
5  void *tachePeridique(void *periode)
6  {
7      pthread_cond_t cond;
8      pthread_mutex_t verrou;
9      struct timespec time;
10     pthread_cond_init(&cond, NULL);
11     pthread_mutex_init(&verrou, NULL);
12     int i = 0;
13     clock_gettime(CLOCK_REALTIME, &time);
14     while (i < 10)
15     {
16         pthread_mutex_lock(&verrou);
17         time.tv_sec = time.tv_sec + (long)periode;
18         printf("La tache %s s'execute periodiquement à l'instant %d secondes\n", "t1", (int)time.tv_sec);
19         //suite du code
20         pthread_cond_timedwait(&cond, &verrou, &time);
21         pthread_mutex_unlock(&verrou);
22         i++;
23     }
24 }
25
26 int main(void)
27 {
28     pthread_t tache1;
29     pthread_create(&tache1, NULL, tachePeridique, (void *)5);
30     pthread_join(tache1, NULL);
31     return 0;
32 }

```

## Execution

```

wa101@wa101-latitude5490:~/STR-TP0
~ > cd STR-TP0
~/STR-TP0 master !1 ?4 > ./exemple5
La tache t1 s'execute periodiquement à l'instant 1635348516 secondes
La tache t1 s'execute periodiquement à l'instant 1635348521 secondes
La tache t1 s'execute periodiquement à l'instant 1635348526 secondes
^C
~/STR-TP0 master !1 ?4 >

```

## **Exemple 6 : Synchronisation entre Taches sous POSIX**

**Code**

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4  #include <semaphore.h>
5  sem_t evt;
6  //déclaration du sémaphore représentant
7  void *tache1(void *arg)
8  {
9      int i = 0;
10     while (i < 10)
11     {
12         printf("La tâche %s s'exécute\n", (char *)arg);
13         //suite du code
14         sem_post(&evt);
15         //la tâche 1 émet l'événement à la fin de son
16
17         i++;
18     }
19 }
20 void *tache2(void *arg)
21 {
22     int i = 0;
23     while (i < 10)
24     {
25         sem_wait(&evt);
26         //la tâche 2 est bloquée en attente de l'émission de
27         printf("La tâche %s s'exécute enfin\n", (char *)arg);
28         //suite du code
29         i++;
30     }
31 }
32 int main()
33 {
34     pthread_t th1, th2;
35     sem_init(&evt, 0, 0);
36     // le sémaphore est local au processus issu de
37     // la fonction main() et a un compteur initialisé à 0
38     pthread_create(&th1, NULL, tache1, "1");
39     pthread_create(&th2, NULL, tache2, "2");
40     pthread_join(th1, NULL);
41     pthread_join(th2, NULL);
42     return 0;
43 }

```

## Execution

