```
/***** A compiler avec

   gcc -o Exemple2_await_generate Exemple2_await_generate.c \
       traceinstantsf.c -I $CHEMIN/ft_v1.1/include -L $CHEMIN/ft_v1.1/lib \
       -lfthread -lpthread
*****/


#include "fthread.h"
#include "stdio.h"
#include "unistd.h"
#include "traceinstantsf.h"
#include "stdlib.h"
#include "pthread.h"


ft_thread_t    ft_trace, ft_generator;
ft_thread_t    ft_awaiter[3];
ft_scheduler_t sched;
ft_event_t     evt;


void awaiter (void *arg)
{
  long i, n, res;

  for (i = 0; i< 5; i++) {

    /*****
          ft_thread_await(evt) permet au thread courant de se mettre en
          attente dans la liste d'evenement evt.
          Si dans l'instant courant l'evenement est genere, tous les threads
          de la liste d'evenement evt sont liberes et continuent leur
          execution dans l'instant courant.

          Attention, sur le meme sched, un seul thread s'execute, les autres
          se mettent en attente de leur tour par cooperation.

          Attention, on fait un await d'un evenement appartenant au meme
          sched que l'appel de await. Sinon, ca renvoie une erreur sans
          etre bloquant.
    *****/

    fprintf(stdout, "awater%d en attente d'un evenement.\n", (long)arg);
    res = ft_thread_await(evt);

    if (res == OK) {

      /*****
            Plusieurs evenements differents peuvent etre generes dans un meme
            instant.
            Mais un meme evenement peut aussi etre genere plusieurs fois dans
            un meme instant.
            Plusieurs meme evt peuvent etre generes dans le meme instant.
            Le 0 de ft_thread_get_value(evt, 0, (void *)&n) indique que l'on
            veut la valeur du premier meme evt genere.
            Si on choisit 1 a la place, cela indique que c'est la valeur du
            2eme meme evt genere, etc, ...

            Attention : si l'evt n'existe pas ou ce nombre est plus grand que
            le nombre (- 1) du meme evt, ca renvoie l'erreur NEXT avec
            n <== NULL et ft_thread_get_value() termine au prochain instant.
      *****/
      ft_thread_get_value(evt, 0, (void *)&n);
      fprintf(stdout,
              "awaiter%d a recu l'evenement numero %d.\n",
```



```
              (long)arg, n);
    }
  }
}


void generator (void *arg)
{
  long i;

  for (i=0;; ++i) {
    fprintf(stdout, "generator genere l'evenement numero %d.\n", i);

    /*****
          Attention : on genere un evt qui appartient au meme sched sinon
          ca renvoie une erreur. Pire ==> ca bloque.
    *****/
    ft_thread_generate_value(evt, (void *)i);
    ft_thread_cooperate();
  }
}


void join_awaiters (void *arg)
{
  long i;
  pthread_t pthread_sched;
  /*****
        Les joins ne servent pas vraiment ici car les awaiters et generator
        finissent très souvent avant tous leurs affichages sur stdout.

        Attention : le ft_thread qui appelle le join, doit etre dans le meme
                    ft_scheduler que les ft_threads passes en argument du join.

  *****/

        fprintf(stdout, "Debut de join_awaiters.\n");

        for (i = 0; i < 3; ++i) {
          ft_thread_join(ft_awaiter[i]);
        }

        fprintf(stdout, "Fin de tous les awaiter.\n");

        fprintf(stdout, "Stoper la trace.\n");
        ft_scheduler_stop(ft_trace);

        fprintf(stdout, "Stoper le generateur.\n", (long)arg);
        ft_scheduler_stop(ft_generator);

        fprintf(stdout, "********** exit(0) **********\n");
        exit(0);
}



int main(int argc, char *argv[])
{
  long i;

  sched = ft_scheduler_create ();

  evt = ft_event_create(sched);

  ft_trace      = ft_thread_create(sched, traceinstants, NULL, (void *)50);
```

```
  ft_generator  = ft_thread_create(sched, generator, NULL, NULL);

  for (i = 0; i < 3; ++i) {
    ft_awaiter[i] = ft_thread_create(sched, awaiter, NULL, (void *)(i + 1));
  }

  ft_thread_create(sched, join_awaiters, NULL, NULL);

  ft_scheduler_start(sched);

  fprintf(stdout, "\n\nC'est fini pour le programme principal.\n");


  /*****
        ft_exit() force le pthread qui fait tourner le main() à terminer et
        laisse les autres threads continuer.
        Il ne retourne jamais ==> il est bloquant.

        Ca laisse tout le temps aux autres threads d'afficher par le stdout.

        A la place de ft_exit() et pour cette exemple, on peut mettre par
        exemple sleep(50) qui permet aussi le temps aux autres threads
        d'afficher.
  *****/
  ft_exit();    /* Bloquant donc. */

  return 0;     /* Never reached.*/
                /* Juste pour calmer le compilo pur avoir ecrit int main() */
}




/*
$ Exemple2_await_generate

C'est fini pour le programme principal.
>>>>>>>>>>> instant 0 :
generator genere l'evenement numero 0.
awater1 en attente d'un evenement.
awaiter1 a recu l'evenement numero 0.
awater1 en attente d'un evenement.
awaiter1 a recu l'evenement numero 0.
awater1 en attente d'un evenement.
awaiter1 a recu l'evenement numero 0.
awater1 en attente d'un evenement.
awaiter1 a recu l'evenement numero 0.
awater1 en attente d'un evenement.
awaiter1 a recu l'evenement numero 0.
awater2 en attente d'un evenement.
awaiter2 a recu l'evenement numero 0.
awater2 en attente d'un evenement.
awaiter2 a recu l'evenement numero 0.
awater2 en attente d'un evenement.
awaiter2 a recu l'evenement numero 0.
awater2 en attente d'un evenement.
awaiter2 a recu l'evenement numero 0.
awater2 en attente d'un evenement.
awaiter2 a recu l'evenement numero 0.
awater3 en attente d'un evenement.
awaiter3 a recu l'evenement numero 0.
awater3 en attente d'un evenement.
awaiter3 a recu l'evenement numero 0.
awater3 en attente d'un evenement.
awaiter3 a recu l'evenement numero 0.
```



```
awater3 en attente d'un evenement.
awaiter3 a recu l'evenement numero 0.
awater3 en attente d'un evenement.
awaiter3 a recu l'evenement numero 0.
Debut de join_awaiters.
Fin de tous les awaiter.
Stoper la trace.
Stoper le generateur.
********** exit(0) **********
*/
```