```
#define T_MAX 100
static pthread_mutex_t searchLock = PTHREAD_MUTEX_INITIALIZER;

enum state {NOTUSED=0, READY=1, SLEEPING=2};
struct thread {
   pthread_t  tid;
   enum state state;
} thread[T_MAX];

struct thread *insert(void) {
   pthread_mutex_lock(&searchLock);

   for (i = 0; i < T_MAX; i++) {
      if (thread[i].state != NOTUSED) continue;

      pthread[i].state = READY;
      pthread[i].tid = pthread_self();
      pthread_mutex_unlock(&searchLock);
      return (&thread[i]);
   }
   pthread_mutex_unlock(&searchLock);
   return (NULL);
}

struct *search(pthread_t tid) {
   pthread_mutex_lock(&searchLock);

   for (i = 0; i < T_MAX; i++) {
      if (thread[i].state != NOTUSED && thread[i].tid == tid) {
         pthread_mutex_unlock(&searchLock);
         return (&thread[i]);
      }
   }
   pthread_unlock(&searchLock);
   return (NULL);
}
```

```
enum state {NOTUSED=0, PARTICIPANT=1, WAITING=2};

static struct thread {
   pthread_t  tid;
   enum state state;
   int group;
} thread[T_MAX];

static struct thread *insert(int group) {// MODIFIED FUNCTION
   pthread_mutex_lock(&searchLock);

   for (i = 0; i < T_MAX; i++) {
      if (thread[i].state != NOTUSED) continue;
      thread[i].tid = thread_self();
      thread[i].state = PARTICIPANT;
      thread[i].group = group;
      pthread_unlock(&searchLock);
      return(&thread[i]);
   }
   pthread_unlock(&searchLock);
   return (NULL);
}


static struct *search(pthread_t tid) {// MODIFIED FUNCTION
   pthread_mutex_lock(&searchLock);

   for (i = 0; i < T_MAX; i++) {
      if (thread[i].state != NOTUSED && thread[i].tid == tid)
         return (&thread[i]);
   }
   pthread_mutex_unlock(&searchLock);
   return (NULL);
}

int barrier_participate(int group) {
   if (search(pthread_self()) {
      pthread_unlock(&searchLock);
```

```
        return (-1);
    }
    if (insert(group) == NULL)
        return (-1);
    return(0);
}

static int all_waiting(group) {
    for (int i = 0; i < T_MAX; i++) {
        if (thread[i].state == PARTICIPATE  &&
                thread[i].group == group)
                return (0);
    }
    return (1);
}

static int all_continue(group) {
    for (int i = 0; i < T_MAX, i++) {
        if (thread[i].state == WAITING  &&
                thread[i].group == group)
                    thread[i].state = PARTICIPATE;
    }
    pthread_cond_broadcast(&searchCond);
    return (0);
}

int barrier_wait() {
    struct thread *p = search(pthread_self());
    if (p == NULL) return (-1);
    if (p->state != PARTICIPATE) {
        pthread_unlock(&searchLock);
        return(-1);
    }
    p->state = WAITING;

    if (!all_waiting(p->group)) {
        while (p->state == WAITING)
            pthread_cond_wait(&searchCond, &searchLock);
```

```
        pthread_mutex_unlock(&searchLock);
        return (0);
    }

    all_continue(p->group);
    pthread_mutex_unlock(&searchLock);
    return (0);
}

int barrier_leave() {
    struct thread *p = search(pthread_self());
    if (p == NULL) return (-1);
    if (p->state != PARTICIPATE) {
        pthread_mutex_unlock(&searchLock);
        return(-1);
    }
    p->state = NOTUSED;
    if (all_waiting(p->group))
        all_continue();
    pthread_mutex_unlock(&searchLock);
}
```