UNIVERSIDAD DEL BÍO-BÍO

EJERCICIO DE MENSAJES

Sistemas Operativos
Profesor: Luis Gajardo

ICI

La federación de tenis de mesa de Chile organiza un torneo en donde todos los jugadores se enfrentan con todos. En el torneo participan 8 jugadores, numerados de 0 a 7 que se enfrentan en 7 rondas de 4 partidos. Para disminuir la duración de los torneos se requiere que los partidos se jueguen en paralelo, sujeto a que se dispone de 3 mesas identificadas como A, B, C.

Para hacer enfrentarse los jugadores *x* e *y*, en la mesa *m*. Usted dispone del procedimiento *enfrentar(x, y, m)*. Este procedimiento toma mucho tiempo y solo retorna cuando el partido finalizó. Algunos partidos pueden ser muy cortos (3 sets) mientras otros pueden ser muy extensos (5 sets muy disputados).

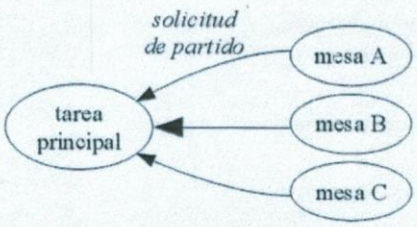
Se ha programado el siguiente procedimiento para realizar el torneo pedido:

```
struct { int x, y; } partidos[4*7]= /* 7 fechas, 4 partidos por ronda */
  { {0,4}, {1,5}, {2,6}, {3,7}, /* 0 enfrenta a 4, 1 a 5, etc. por la 1era. ronda, */
    {0,1}, {2,4}, {3,5}, {7,6}, /* 0 enfrenta a 1, 2 a 4, etc. por la 2da. ronda, */
    {0,2}, {3,1}, {7,4}, {6,5}, ... }; /* 3era. ronda ... y así hasta la 7ma. */
int prox; /* índice del próximo partido que se debe jugar */
```

```
void torneo() {
  prox= 0;
  nTask m1= nEmitTask(mesa, "A");
  nTask m2= nEmitTask(mesa, "B");
  nTask m3= nEmitTask(mesa, "C");
  nWaitTask(m1);
  nWaitTask(m2);
  nWaitTask(m3);
}
```

```
int mesa(char *m) {
  while (prox<4*7) {
    int x= partidos[prox].x;
    int y= partidos[prox].y;
    prox++;
    enfrentar(x, y, m);
  }
  return 0;
}
```

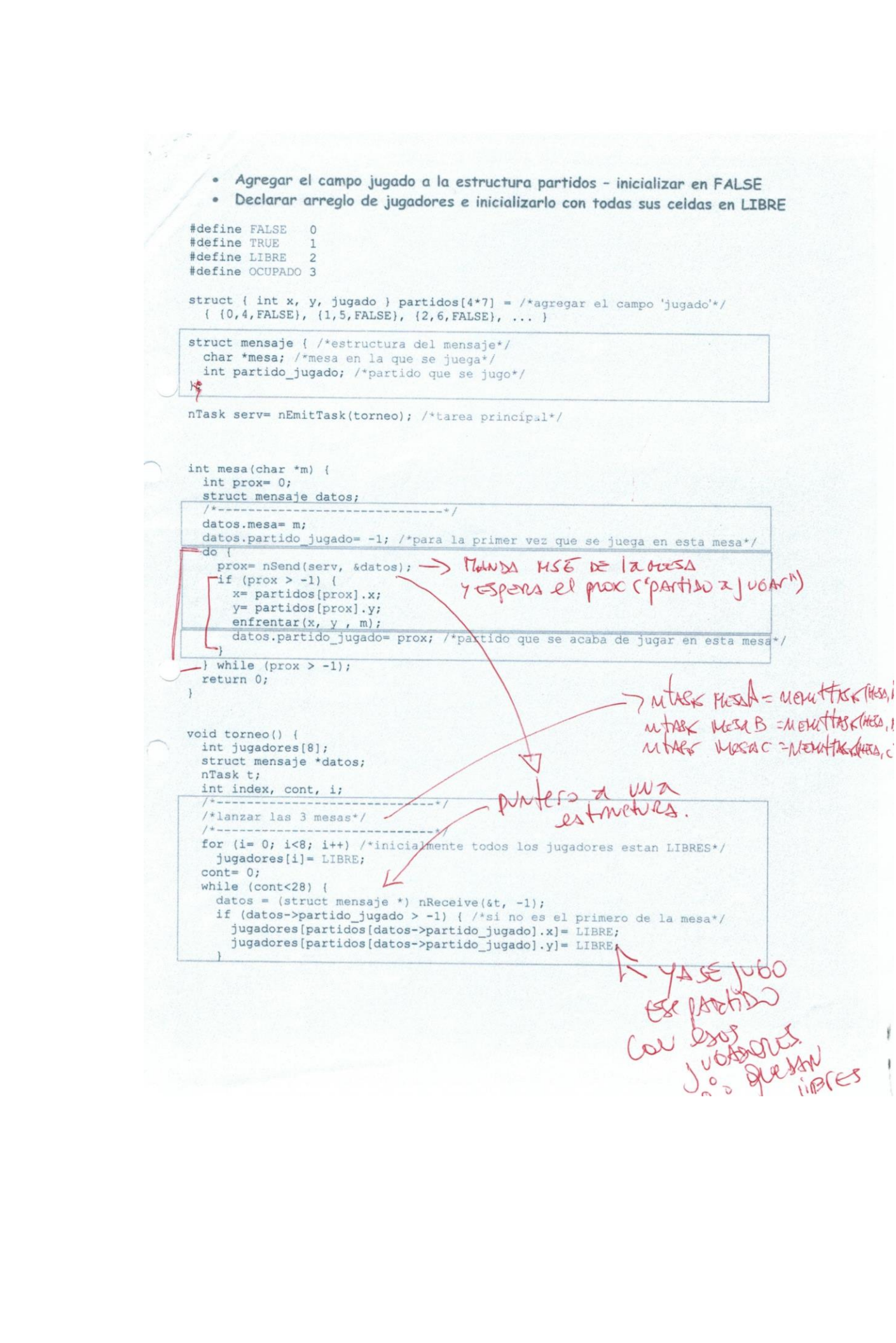
Resuelva el problema utilizando como única herramienta de sincronización los mensajes de nSystem. Utilice la organización de tareas que se indica en la figura de más abajo. Cada mesa es una tarea que solicita partidos a la tarea principal. Para ello la mesa envía un mensaje a la tarea principal, la que responde con el partido que se debe jugar. Entonces, la mesa realiza el enfrentamiento. Al finalizar, la mesa envía un nuevo mensaje a la tarea principal indicando qué partido se jugó. La tarea principal responderá este mensaje con el nuevo partido que se debe jugar a continuación (-1 si no quedan partidos para jugar), etc.



Restricción: si una mesa está disponible para jugar un nuevo partido, pero el próximo partido en el arreglo no se puede jugar porque uno de los jugadores todavía está jugando en otra mesa, Usted debe buscar algún otro partido que sí se pueda jugar y entregárselo a la mesa. Como en la pregunta anterior, *torneo* solo retorna cuando todos los partidos finalizaron.

- Agregar el campo jugado a la estructura partidos - inicializar en FALSE
- Declarar arreglo de jugadores e inicializarlo con todas sus celdas en LIBRE

```
#define FALSE   0
#define TRUE    1
#define LIBRE   2
#define OCUPADO 3

struct { int x, y, jugado } partidos[4*7] = /*agregar el campo 'jugado'*/
  { {0,4,FALSE}, {1,5,FALSE}, {2,6,FALSE}, ... }

struct mensaje { /*estructura del mensaje*/
  char *mesa; /*mesa en la que se juega*/
  int partido_jugado; /*partido que se jugo*/
};

nTask serv= nEmitTask(torneo); /*tarea principal*/


int mesa(char *m) {
  int prox= 0;
  struct mensaje datos;
  /*------------------------------*/
  datos.mesa= m;
  datos.partido_jugado= -1; /*para la primer vez que se juega en esta mesa*/
  do {
    prox= nSend(serv, &datos);
    if (prox > -1) {
      x= partidos[prox].x;
      y= partidos[prox].y;
      enfrentar(x, y , m);
      datos.partido_jugado= prox; /*partido que se acaba de jugar en esta mesa*/
    }
  } while (prox > -1);
  return 0;
}


void torneo() {
  int jugadores[8];
  struct mensaje *datos;
  nTask t;
  int index, cont, i;
  /*------------------------------*/
  /*lanzar las 3 mesas*/
  /*------------------------------*/
  for (i= 0; i<8; i++) /*inicialmente todos los jugadores estan LIBRES*/
    jugadores[i]= LIBRE;
  cont= 0;
  while (cont<28) {
    datos = (struct mensaje *) nReceive(&t, -1);
    if (datos->partido_jugado > -1) { /*si no es el primero de la mesa*/
      jugadores[partidos[datos->partido_jugado].x]= LIBRE;
      jugadores[partidos[datos->partido_jugado].y]= LIBRE;
    }
```



```
    index= -1; /*buscar un partido no jugado*/
    for (i=0; i<28; i++) {
      if (partidos[i].jugado == FALSE) {
        if ((jugadores[partidos[i].x] == LIBRE) &&
            (jugadores[partidos[i].y] == LIBRE)) {
          index= i;
          break;
        }
      }
    }
    if (index == -1 && i==28) {
      nReplay(t, -1); /*no hay mas partidos por jugar*/
    }
    else {
      partidos[index].jugado= TRUE;
      jugadores[partidos[index].x]= OCUPADO;
      jugadores[partidos[index].y]= OCUPADO;
      cont++;
      nReplay(t, index); /*index es el proximo partido a jugar*/
    }
  }
}
```

VERIFICO

→ quiebra el For.

While

Método