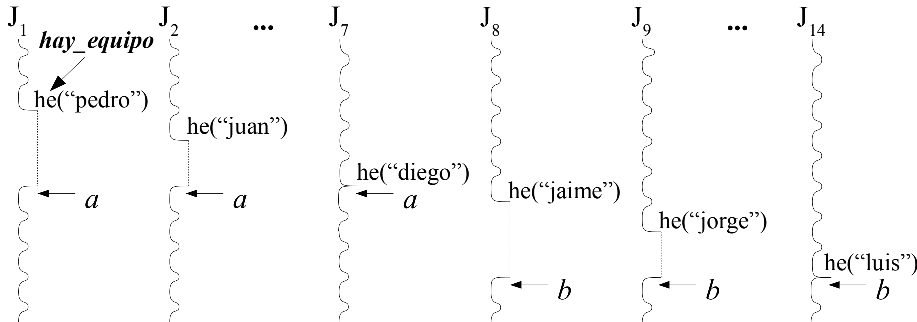
CC4302 Sistemas Operativos – Control 1 – Semestre Otoño 2018 – Prof.: Luis Mateu

Pregunta 1

Parte a.- Se necesita formar equipos de 7 jugadores de futbolito. Los jugadores son representados por threads que invocan la función *hay_equipo* indicando como argumento su nombre. Esta función espera hasta que 7 jugadores hayan invocado *hay_equipo* retornando un arreglo de 7 strings con los nombres del equipo completo. Ejemplo de uso:



En donde *a* es un arreglo con los nombres "pedro", "juan", …, "diego", y *b* es un arreglo con los nombres "jaime", "jorge", …, "luis".

La siguiente implementación de *hay_equipo* funciona casi siempre, pero es incorrecta:

```
char **equipo; // = malloc(7*sizeof(char*)); equipo en formación
int k= 0;      // cuantos jugadores han llegado
nSem m;        // = nMakeSem(1); para garantizar la exclusión mutua
nSem tickets;  // = nMakeSem(0); para hacer esperar hasta 6 jugadores
```

```
char **hay_equipo(char *nom) {
  nWaitSem(m); // inicio secc. crítica
  equipo[k++]= nom;
  char **miequipo= equipo;
  if (k!=7) { // aún no hay equipo
    nSignalSem(m); // fin secc. crít.
    nWaitSem(tickets); // espera
  }
  else { // hay equipo: despertar a los
         // 6 jugadores en espera
    for (int i= 0; i<6; i++)
      nSignalSem(tickets);
    // preparar nuevo equipo
    equipo=
      malloc(7*sizeof(char*));
    k= 0;
    nSignalSem(m); // fin secc. crít.
  }
  return miequipo;
}
```

Haga un diagrama de threads que muestre una situación en donde un jugador recibe como resultado un equipo que todavía no se forma. Considere que en nSystem los tickets de un semáforo se otorgan por orden de llegada.

Parte b.- Programe una solución de *hay_equipo* que sea correcta y eficiente usando los semáforos de nSystem. *Hint*: un jugador debe crear su propio semáforo cuando necesita esperar, colocando su dirección en un arreglo de 6 semáforos. Al despertarse, el jugador destruye ese semáforo.

Pregunta 2

La función *masParecidas* de más abajo, encuentra las 2 tareas de programación de software de sistemas que más se parecen. Esta función recibe como parámetros un arreglo *tareas* con las *n* tareas de los alumnos y 2 punteros *pi* y *pj* con la direcciones de las variables en donde se deben almacenar los índices de las 2 tareas más parecidas en el arreglo.

```
void masParecidas(Tarea *tareas, int n,
                  int *pi, int *pj) {
  int min= MAXINT;
  for (int i= 1; i<n; i++)
    for (int j= 0; j<i; j++) {
      int similitud= compTareas(tareas[i], tareas[j]);
      if (similitud<min) {
        min= similitud;
        *pi= i; *pj= j;
} } } }
```

En este código la función *compTareas* es dada y entrega un coeficiente de similitud. El 0 significa que son iguales. Para comparar 2 tareas esta función toma tiempo muy variable: desde unos pocos milisegundos a minutos. Las $O(n^2)$ comparaciones toman mucho tiempo de CPU.

Paralelice las función *masParecidas* considerando una máquina octa-core con una implementación de nSystem que ejecuta los threads en paralelo. Ud. no puede ejecutar más de 8 comparaciones de tareas en paralelo porque se requeriría mucha memoria. Un core no puede permanecer ocioso mientras queden comparaciones por realizar.

Ayuda: Al inicio de la función cree 8 threads adicionales que se usarán para realizar las comparaciones y que llamaremos los *threads comparadores*. El thread principal (el que ejecuta *masParecidas*) se usa para alimentar a los threads comparadores con pares (i, j) de tareas que se deben comparar. Cada thread comparador espera hasta obtener del thread principal un par (i, j) de tareas, las compara, revisa si se trata del par más similar del momento y luego espera a recibir un nuevo par. Por otra parte el thread principal se dedica a generar un par (i, j), espera a que se desocupe algún thread comparador, se lo asigna y luego genera un nuevo par, espera un thread comparador, se lo asigna, …, etc.