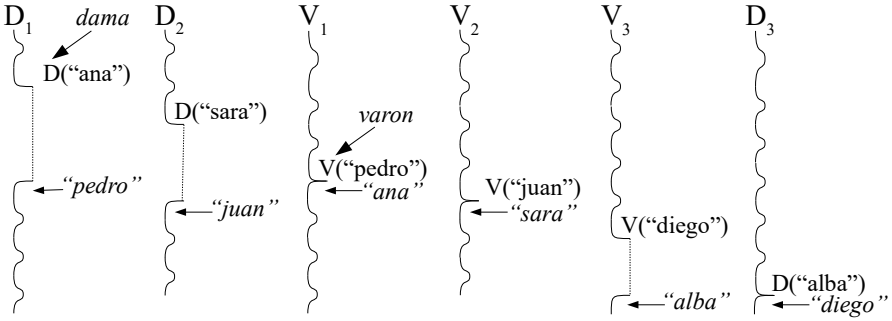
CC4302 Sistemas Operativos – Control 1 – Semestre Otoño 2026 – Profs.: Mateu, Torrealba

Pregunta 1

```
char *dama(char *nom);
char *varon(char *nom);
```

Múltiples threads representan damas y varones que buscan una pareja para bailar en una discoteca invocando las funciones *dama* y *varon*, cuyos encabezados se muestran en el cuadro de arriba. La función *dama* recibe como parámetro el nombre de la dama. Si hay varones esperando pareja, *dama* entrega de inmediato el nombre del varón que llegó primero. Es decir que la asignación de parejas se hace por orden de llegada. Si no hay varones en espera, *dama* espera por una invocación de la función *varon*. Análogamente *varon* entrega al instante el nombre de la primera dama que todavía busca su pareja, o espera una invocación de *dama*. El siguiente es un diagrama de threads que muestra un ejemplo de ejecución:



Programe las funciones *dama* y *varon* usando obligatoriamente un mutex y una sola condición. No puede usar colas para hacer esperar los threads y por lo tanto debe usar 2 distribuidores de número, uno para las damas y otro para los varones de modo que se les asigne pareja por orden de llegada.

Pregunta 2

I. (5 puntos) La función *contarSec* del cuadro de la derecha recibe en *doc* un arreglo de *ndoc* documentos y en *pal* un arreglo de *npal* palabras. El arreglo *frecs* de *npal* enteros es de salida: en *frecs[j]* se entrega la cantidad de ocurrencias de *pal[j]* en el arreglo *doc*. La función *contarDoc* es dada. **Programe la función *contarPar***, que recibe los mismos parámetros que *contarSec* más el parámetro adicional *int p*. Debe entregar el mismo resultado pero realizando el conteo **en paralelo** con *p* threads, asignando a cada thread la misma cantidad de documentos (por simplicidad suponga que *n* es múltiplo de *p*). Debe invocar *contarSec* desde *contarPar*.

```
void contarSec(Doc *doc[],
    int ndoc, char *pal[],
    int frecs[], int npal ) {
  // inicializa frecs en cero
  memset(frecs, 0,
         npal*sizeof(int));
  for(int i=0; i<ndoc; i++) {
    int frec1[npal];
    memset(frec1, 0, // frec1 en cero
           npal*sizeof(int) );
    contarDoc(doc[i], pal,
              frec1, npal );
    for (int j=0; j<npal; j++)
      frecs[j] += frec1[j];
} }
void contarPar(…, int p);
```

II. (1 punto) Explique cuál sería el problema de distribuir uniformemente los documentos a los threads y explique en palabras cómo lo arreglaría.

Pregunta 3

A) Un estadio posee un único baño que debe ser compartido por hinchas rojos y azules. El baño es amplio y admite un número ilimitado de personas. El problema consiste en evitar que los hinchas rojos se encuentren con los hinchas azules dentro del baño. Los hinchas rojos solicitan entrar al baño invocando *entrar(ROJO)* y notifican su salida con *salir(ROJO)*, mientras que los hinchas azules invocan *entrar(AZUL)* y *salir(AZUL)*. La siguiente implementación ***incorrecta*** usa un mutex, múltiples condiciones y el patrón *request* para resolver el problema.

```
pthread_mutex_t m=
          PTHREAD_MUTEX_INITIALIZER;
int adentro[2]= {0, 0};
Queue *q[2]; // = makeQueue()*2;
enum { ROJO=0,AZUL=1};
```

```
typedef struct {
  int ready;
  pthread_cond_t w;
} Request;
```

```
void entrar(int color) {
  pthread_mutex_lock(&m);
  int oponente= !color;
  if (adentro[oponente]>0 ||
      !empty(q[oponente])) {
    Request req= { 0,
         PTHREAD_COND_INITIALIZER };
    put(q[color], &req);
    while (!req.ready)
      pthread_cond_wait(
                &req.w, &m );
  }
  adentro[color]++;
  pthread_mutex_unlock(&m);
}
```

```
void salir(int color) {
  pthread_mutex_lock(&m);
  int oponente= !color;
  adentro[color]--;
  if (adentro[color]==0) {
    while (!empty(
              q[oponente] )) {
      Request *preq= get(
                q[oponente] );
      preq->ready= 1;
      pthread_cond_signal(
                    &preq->w );
  } }
  pthread_mutex_unlock(&m);
}
```

Haga un diagrama de threads que muestre un jugador azul y un jugador rojo entrando al baño al mismo tiempo incorrectamente.

B) Haga cambios menores en esta implementación que corrijan el defecto, manteniendo el patrón request. Puede responder esta parte sin responder *A*.