CC4302 Sistemas Operativos – Control 1 – Semestre Primavera 2023 – Prof.: Luis Mateu

Pregunta 1

Parte 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, 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 ||
      !emptyQueue(
             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 (!emptyQueue(
             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.

Parte b.- Corrija la implementación haciendo cambios menores.

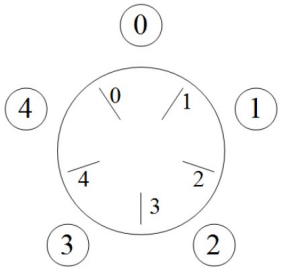
Pregunta 2

I.- (2 puntos) La función *palindro* de arriba a la derecha entrega 1 si un arreglo *s* de *n* enteros es palíndrome, es decir que al invertirlo se lee el mismo arreglo. En caso contrario entrega 0. **Paralelice la función palindro** de manera que se ejecute la primera mitad del ciclo en un nuevo thread y la segunda mitad en el thread original. *¡Revise que su solución posee paralelismo!*

```
int palindro(int *a, int n) {
  int i= 0;
  while (i<n/2) {
    if(a[i]!=a[n-1-i])
      return 0;
    i++;
  }
  return 1;
}
```

II.- (4 puntos) A la derecha se muestra una solución parcial de la cena de filósofos, en donde 5 filósofos numerados de 0 a 4 cenan juntos en una mesa redonda con 5 palitos, numerados de 0 a 4. Cada filósofo alterna entre comer y pensar. Para poder comer el filósofo *id* necesita los palitos *id* y *(id+1)%5*. No necesita ningún palito para pensar. Los filósofos deben comer en paralelo cuando sea posible pero no deben comer con el mismo palito simultáneamente, y tampoco pueden sufrir hambruna.

```
void pedir(int id);
void devolver(int id);
void init( );
void filosofo(int id) {
  for (;;) {
    pedir(id);
    comer(id,(id+1)%5);
    devolver(id);
    pensar();
  }
}
```



Programe las funciones *pedir* y *devolver*. Programe además *init* para inicializar las variables globales si lo necesita. Para evitar la hambruna se exige que el filósofo *id* no puede comenzar a comer si su vecino de la izquierda, el filósofo (*id*+1)%5, lleva esperando más tiempo que él o si su vecino de la derecha, el filósofo (*id+4*)%5, lleva esperando más tiempo que él. Por ejemplo si el filósofo 2 come y el filósofo 3 espera, si llega el filósofo 4 también debe esperar porque su vecino de la derecha solicitó comer antes que él. Observe que no es atención por orden de llegada porque si el filósofo 2 come y el filósofo 3 espera, si llega el filósofo 0, sí come de inmediato, porque sus vecinos 4 y 1 no están esperando ni comiendo.

Restricciones: Para resolver este problema de sincronización Ud. *debe utilizar un solo mutex y una sola condición*. Use un arreglo de 5 enteros que indica si el palito *id* está ocupado o no y otro arreglo de 5 enteros que indica en qué momento el filósofo *id* solicitó comer, o -1 si no está esperando. Para representar los tiempos use un contador global que se incrementa en 1 cada vez que un filósofo solicita comer.