CC4302 Sistemas Operativos – Control 1 – Semestre Otoño 2024
Profs.: Mateu, Torrealba, Arenas

Pregunta 1

Los threads A y B necesitan *intercambiar* valores una sola vez durante la ejecución. Para lograrlo invocan la función *intercambiar* implementada en el cuadro de la derecha. Por ejemplo si A invoca primero *intercambiar(1)*, deberá esperar. Cuando B invoca *intercambiar(2)*, la invocación en A retorna de inmediato el valor 2 y la invocación en B retorna 1. Conteste:

```
int val= 0;
int cont= 0;
pthread_mutex_t m=PTHREAD_MUTEX_INITIALIZER;
int intercambiar(int mi_val) {
  lock(&m);       // (O)
  int ret= val;
  val= mi_val;
  cont++;
  if (cont<2) {
    while (cont!=2) {
      unlock(&m); // (P)
      lock(&m);   // (Q)
      ret= val;
  } }
  unlock(&m);     // (R)
  return ret;
}
```

a.- ¿Hay algún datarace en la solución? Si lo hay haga un diagrama de threads que muestre una ejecución errónea. Si no hay dataraces, explique cómo los evita esta solución.

b.- Considere que se elimina de esta solución el mutex *m* y todas las invocaciones a *lock* y *unlock*. Haga un diagrama de threads que muestre una ejecución errónea.

c.- Considere que en la solución original solo se **eliminan** las llamadas a *unlock* en P y a *lock* en Q, pero se mantiene el mutex *m* y las llamadas en O y R. ¿Sería correcta la solución? Explique.

d.- Reprograme esta solución de manera que se espere ***eficientemente*** la segunda invocación de *intercambiar* y explique por qué es más eficiente que la solución planteada.

Pregunta 2

La función *masRapida* de la derecha recibe un arreglo de *n* funciones que buscan un elemento *x* en un arreglo *a*. Entrega la función en el arreglo *f* que toma menos tiempo en buscar *x* en *a*. Considerando que el computador tiene *n* cores disponibles, paralelice esta función de manera que no se demore más que la función más lenta en el arreglo.

Restricción: el computador no posee un reloj ni cronómetro y por lo tanto no puede llamar a *getTime*.

Ayuda: La función más rápida es la que termina primero.

```
typedef int (*SearchFun)(int *a, int x);
SearchFun masRapida(SearchFun f[], int n,
                    int *a, int x ) {
  int mejor_t= MAXINT;
  SearchFun mejor= NULL;
  for (int i= 0; i<n; i++) {
    int ini= getTime();
    (*f[i])(a,x);
    int t= getTime()-ini;
    if (t<mejor_t) {
      mejor_t= t;
      mejor= f[i];
  } }
  return mejor;
}
```

Pregunta 3

Muchos perros y gatos necesitan pasar por un puente para cruzar un río. Como perros y gatos se odian, no puede haber perros y gatos simultáneamente en el puente. Pero para mayor eficiencia, se permite que los perros sí crucen simultáneamente el puente y que los gatos también lo crucen simultáneamente. Los perros solicitan cruzar invocando *cruzaPerro()* y notifican la salida del puente con *salePerro()*. Análogamente los gatos invocan *cruzaGato()* y *saleGato()*.

Programe estas 4 funciones de manera que las solicitudes se otorguen por orden de llegada.

Restricción: Resuelva el problema usando un mutex y **una sola condición**. No puede usar otras herramientas de sincronización. Su solución debe hacer que los perros que llegan consecutivamente al puente lo crucen en paralelo y que análogamente los gatos consecutivos crucen el puente en paralelo.