CC4302 Sistemas Operativos – Control – Semestre Otoño 2023 – Profs.: Luis Mateu/Javier Bustos

Pregunta 1

Parte a.- La función *buscarValor* del cuadro de la derecha entrega la dirección del nodo que contiene el valor *val* en el árbol binario *a*. Como los valores no están ordenados, si no está en el subárbol izquierdo, también hay que buscar en el subárbol derecho. **Programe la función *buscarValorDual***, con el mismo encabezado que buscarValor y que entrega el mismo resultado pero realizando la búsqueda en la rama izquierda **en paralelo** con la búsqueda en la rama derecha **considerando una máquina dual core**. Puede invocar *buscarValor* desde *buscarValorDual*. No puede crear más de un thread adicional.

```
Nodo *buscarValor(
          Nodo *a, int val) {
  if (a==NULL)
    return NULL;
  if (a->val==val)
    return a;
  Nodo *resIzq=
    buscarValor(a->izq, val);
  if (resIzq!=NULL)
    return resIzq;
  return
    buscarValor(a->der, val);
}
```

Parte b.- Un puente puede soportar PESOMAX como peso máximo. Los vehículos deben solicitar permiso para ingresar al puente invocando la función *entrar(p)* y notificar su salida llamando a la función *salir(p)*, en donde *p* es el peso del vehículo. Este peso nunca superará a PESOMAX y al salir siempre se indicará el mismo peso que anunció ese vehículo al entrar. La siguiente implementación funciona el 99.999% de los casos.

```
pthread_mutex_t m= PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c= PTHREAD_COND_INITIALIZER;
double pesoActual= 0;
void entrar(double peso) {
  pthread_mutex_lock(&m);
  while (pesoActual+peso > PESOMAX)
    pthread_cond_wait(&c, &m);
  pesoActual += peso;
  pthread_mutex_unlock(&m);
}
void salir(double peso) {
  pesoActual -= peso;
  pthread_cond_broadcast(&c);
}
```

Haga un diagrama de threads que muestre un ejemplo de ejecución en donde se exceda el peso máximo soportado. *Ayuda*: un thread A invoca entrar(1); luego A invoca salir(1) en paralelo con un thread B que llama a entrar(PESOMAX-1), y; finalmente un thread C invoca entrar(2). Muestre cómo B y C podrían entrar juntos al puente alcanzando PESOMAX+1.

Parte c.- **Corrija** la solución para que no se exceda nunca el peso máximo soportado **y además identifique** el defecto de la solución ya corregida.

Pregunta 2

I. Implemente una solución eficiente para el mismo problema de la parte b de la pregunta 1, de modo que se garantice entradas por orden de llegada usando un mutex y múltiples condiciones. Defina la función *iniPuente( )* para inicializar las variables globales que necesite. Use el patrón request.

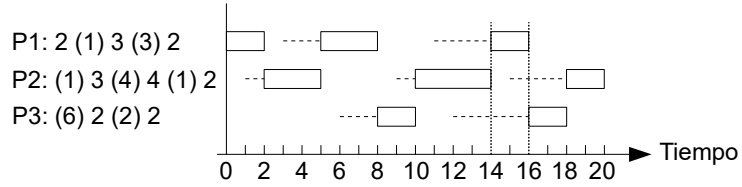
II. Implemente una solución nativa para nThreads del mismo problema de la pregunta 2.I. Las funciones que debe programar son:

```
void nEntrar(double peso);      // Solicitud de ingreso al puente
void nSalir(double peso);       // Notificación de salida
void nth_iniPuente( );          // Para inicializar variables globales
```

Use el campo *ptr* en el descriptor de thread para almacenar la dirección del parámetro *peso* de la función *entrar*.

Restricción: No puede usar herramientas de sincronización preexistentes en nThreads como mutex, condiciones o semáforos. Debe usar operaciones como *START_CRITICAL*, *END_CRITICAL*, *schedule*, *setReady*, *nth_putBack*, *nth_peekFront* (obtiene el primer thread de una NthQueue sin extraerlo), etc. Puede usar Queue en lugar de NthQueue.

III. El siguiente diagrama muestra el scheduling de 3 procesos.



Junto a cada proceso se indica la duración de las ráfagas de CPU y entre paréntesis la duración de los estados de espera. El rectángulo indica que el proceso está en estado RUN, la línea punteada que está READY y la línea en blanco que está en estado de espera. Observe que el scheduler de procesos solo toma decisiones en los tiempos 14 y 16.

Responda: **a.-** Explique por qué el scheduling de procesos no es consistente con *shortest job first non preemptive* considerando como predictor la duración de la última ráfaga ejecutada; **b.-** Modifique el diagrama para mostrar cómo se ejecutarían los mismos procesos considerando la estrategia *shortest job first* señalada en a; y **c.-** Explique cuál estrategia de scheduling sí es consistente con el diagrama de más arriba.