CC4302 Sistemas Operativos – Control – Semestre Otoño 2021 – Prof.: Luis Mateu

Nota: Después de resolver todo el control, si le queda tiempo para probar sus soluciones, descargue de U-cursos *ctl.zip* y descomprímalo. Los programas de prueba están en los directorios P1, P2 y P3. Lea los *Makefiles* para las instrucciones de compilación y ejecución. No necesita definir la variable NSYSTEM. Vienen incluidos pSystem y nSystem. Aprobar los tests de prueba no garantiza nota 7 porque los tests no son exhaustivos.

Pregunta 1

Considere los mismos *left/right lock* de la tarea 3. La siguiente es una implementación de un solo *left/right lock* global. Es *incorrecta* pero funciona el 99,99 % de los casos.

```
enum { LEFT= 0, RIGHT= 1 };
int busy[2]= { FALSE, FALSE }; // ambas mitades libres
nSem m; // = nMakeSem(1); parte con 1 ticket
```

```
int halfLock() {
  if (!busy[LEFT] &&
      !busy[RIGHT])
    nWaitSem(m);
  int L= -1;
  while (L==-1) {
    if (!busy[LEFT])
      L= LEFT;
    else if (!busy[RIGHT])
      L= RIGHT;
  } // ¡busy waiting!
  busy[L]= TRUE;
  return L;
}
```

```
void halfUnlock(int L) {
  busy[L]= FALSE;
  if (!busy[LEFT] &&
      !busy[RIGHT])
    nSignalSem(m);
}
void fullLock() {
  nWaitSem(m);
}
void fullUnlock() {
  nSignalSem(m);
}
```

A) (2 puntos) Haga un diagrama de threads que muestre que estando el mutex *m* completamente libre, puede ocurrir que 2 threads invoquen concurrentemente *halfLock* pero solo el primero obtenga una mitad. El segundo espera hasta que el primero invoque *halfUnlock*.

B) (4 puntos) Reeimplemente correctamente y sin *busy-waiting* las funciones *halfLock* y *halfUnlock*, ***sin modificar las funciones fullLock y fullUnlock***. Para ello preserve la idea de la solución de más arriba, manteniendo el semáforo *m*, pero agregando 2 nuevos semáforos de nSystem. El primer semáforo para lograr la exclusión mutua dentro de *halfLock* y dentro de *halfUnlock*. El segundo semáforo mantiene hasta 2 tickets, uno por cada mitad libre del mutex. No se preocupe por la hambruna.

Pregunta 2

Reprograme el buffer del productor/consumidor de manera que se garantice que al depositar un item, se otorgue al consumidor que lleva más tiempo esperando. Por simplicidad no interesa el orden en que se atienden los productores. Los encabezados de las funciones que debe implementar son:

```
typedef struct buffer *Buffer;
Buffer makeBuffer(int size);
void put(Buffer buf, void *ptr);
void *get(Buffer buf);
```

Restricciones: Debe evitar cambios de contexto explícitos innecesarios. Para ello debe usar un monitor y múltiples condiciones de nSystem, una condición para cada thread en espera. Use una *fifoqueue* para encolar las solicitudes *get* en espera.

Ayuda: Represente las solicitudes *get* mediante una estructura que incluya la condición para esperar, una variable booleana que señale cuando está lista la solicitud y un puntero para dejar ahí el ítem depositado. Cuando se deposite un item en el buffer y hay un consumidor en espera, no lo coloque en el buffer, colóquelo directamente en la solicitud y despierte a ese thread.

Pregunta 3

I. (4 puntos) Reprograme el mismo buffer de la pregunta 2, pero como herramienta de sincronización nativa de nSystem, es decir recurriendo a las funciones de bajo nivel de nSystem (*START_CRITICAL*, *ResumeNextReadyTask*, *PutTask*, etc.). Ud. no puede usar otros mecanismos de sincronización ya disponibles en nSystem como semáforos, monitores, mensajes, etc. Debe atender ***productores y consumidores por orden de llegada***. Puede usar las colas de threads de nSystem (tipo *Queue*, el mismo de la *ready_queue*). Los encabezados de las funciones que debe implementar son:

```
typedef struct buffer *nBuffer;
nBuffer nMakeBuffer(int size);
void nPut(nBuffer buf, void *ptr);
void *nGet(nBuffer buf);
```

II. (2 puntos) El diagrama de abajo muestra el scheduling de 3 procesos. En tiempo 0 los 3 procesos están READY (línea punteada). La estrategia de scheduling es en base a prioridades fijas y distintas. Junto a cada proceso se indica la duración de las ráfagas de CPU y entre paréntesis la duración de los estados de espera. Responda: **a.-** Ordene los procesos de mejor a peor prioridad (0,5 puntos) **b.-** Explique si se trata de scheduling *preemptive* o *non-preemptive* y por qué (0,5 puntos) **c.-** Complete el diagrama (1 punto).