CC4302 Sistemas Operativos – Control – Semestre Otoño 2022 – Prof.: Luis Mateu

Pregunta 1

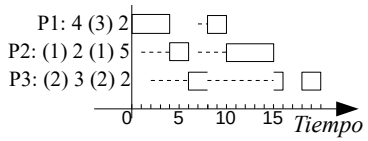
Parte a.- Considere que un thread *T1* invoca *intercambiar(v1)* y otro thread *T2* invoca *intercambiar(v2)*. Ambas invocaciones deben retornar sus parámetros intercambiados, es decir que se debe retornar *v2* en *T1* y *v1* en *T2*. Por lo tanto el primer thread que haga la invocación de *intercambiar* debe esperar la invocación de *intercambiar* en el segundo. No habrá una tercera invocación de *intercambiar*. Dibuje un diagrama de threads que demuestre que la solución de abajo es incorrecta. ¡Cuidado! Que sea ineficiente, no significa que sea incorrecta. Lo que se evalúa en esta pregunta es la capacidad de explicar el error con el diagrama. Una explicación en palabras no es suficiente.

```
volatile int g_val=0, g_flag=0;
semaphore_t g_m; // sema_init(&g_m, NULL, 1);
```

```
int intercambiar(int v) {
  sema_wait(&g_m);
  if (g_flag==0) {
    g_flag= 1;
    sema_post(&g_m);
    g_val= v;
    while (g_flag==1)
      ;
    return g_val;
  }
  else {
    int ret= g_val;
    g_flag= 0;
    sema_post(&g_m);
    g_val= v;
    return ret;
  }
}
```

Parte b.- Escriba una solución correcta y eficiente de la función *intercambiar* de la *parte a* agregando un segundo semáforo. No puede recurrir a otras herramientas de sincronización como mutex/condiciones.

Parte c.- El diagrama muestra las decisiones de scheduling para 3 procesos. Para cada proceso se indica la duración de la ráfaga y la duración de su estado de espera entre paréntesis. En el diagrama 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. Explique si la estrategia de scheduling es *preemptive* o *non preemptive*. Rehaga el diagrama completo considerando ahora la estrategia *shortest job first non preemptive*. Considere que el predictor de duración para la próxima ráfaga es la duración de la última ráfaga.



Pregunta 2

Considere un productor y múltiples consumidores. La función *get* de al lado programada con *pthreads* es usada por los consumidores para extraer un ítem del buffer. El problema de esta solución es que si hay múltiples consumidores en espera, cuando el productor deposita un ítem, cualquiera de las llamadas pendientes de *get* podría extraer ese ítem.

```
void *get(Buffer *b) {
  lock(&b->m);
  while (b->cnt==0)
    wait(&b->cond, &b->m);
  void *item=
          b->array[b->out];
  b->out= (b->out+1) %
           b->size;
  b->cnt--;
  broadcast(&b->cond);
  unlock(&b->m);
  return item;
}
```

Reprograme la función *get* para que los consumidores sean atendidos por orden de llegada. Señale los campos que necesita agregar a la estructura *Buffer* y qué valor inicial deben tener. Puede usar cualquiera de las 2 metodologías enseñadas en clases, pero si usa el patrón *request* necesitará reprogramar *put* también (recuerde que es un solo productor).

Pregunta 3

Se necesita agregar *buffers* en forma nativa a nThreads. Estos se usarán para sincronizar múltiples productores y consumidores. Defina la estructura *nBuffer* y programe las siguientes funciones:

- *nBuffer *nMakeBuffer( )*: Crea y retorna un buffer de tamaño ilimitado.
- *void nPut(nBuffer *b, void *ptr)*: Deposita el ítem *ptr* en el buffer *b*. Esta función nunca espera porque el buffer es de tamaño ilimitado. Si hay tareas en espera en un *nGet* del mismo buffer *b*, el ítem se otorga de inmediato a aquel que lleva más tiempo esperando (atención de consumidores por orden de llegada).
- *void *nGet(nBuffer *b)*: Extrae y retorna un ítem del buffer *b*. Si el buffer está vacío, espera hasta que se deposite un ítem con *nPut*.

Restricciones: Ud. debe usar las operaciones de bajo nivel de nThreads (*START_CRITICAL*, *suspend*, *setReady*, *schedule*, *nth_putBack*, *nth_getFront*, *nth_emptyQueue*, etc.). No puede usar otros mecanismos de sincronización ya disponibles en nThreads, como semáforos, mutex, mensajes, etc. Para almacenar los threads que esperan en *nGet* use una *NthQueue* y para almacenar los ítemes depositados con *nPut* use una *Queue* (no use un arreglo circular).