CC4302 Sistemas Operativos – Control 2 – Semestre Otoño 2025
Profs.: Mateu, Torrealba, Arenas

Pregunta 1

A) (3 puntos) El DCC acaba de adquirir N notebooks para prestar a los alumnos. Estos equipos se identifican por los enteros 0, 1, …, N-1. Para solicitar un notebook los alumnos invocan la función *solNtbk*, que retorna el identificador del notebook asignado. No se puede asignar el mismo notebook a otro alumno hasta que sea devuelto. Si no hay notebooks disponibles, esta función espera hasta que se devuelva uno. Para devolver un notebook, el alumno invoca la función *devNtbk(i)* suministrando en *i* el identificador del notebook que había sido asignado previamente por *solNtbk*. **Programe** las funciones *solNtbk*, *devNtbk* e *init* (para la inicialización). Sus encabezados se muestran en el cuadro de la derecha.

```
int solNtbk();
void devNtbk(int i);
void init();
```

Restricciones: (i) Para la sincronización Ud. debe usar un solo mutex y múltiples condiciones de pthreads. (ii) Las solicitudes deben ser atendidas por orden de llegada. (iii) Para evitar cambios de contexto inútiles debe usar el patrón request. (iv) Use una cola fifo (tipo Queue) para guardar las solicitudes pendientes.

B) (1,5 puntos) ¿Cómo se garantiza la exclusión mutua al acceder a la cola de procesos *ready* en un núcleo de sistema operativo para un computador single-core? ¿Y si el computador es multi-core?

C) (1,5 puntos) Dé un ejemplo en el que solo utilizando *shortest job first* (SJF) se genere hambruna.

Pregunta 2

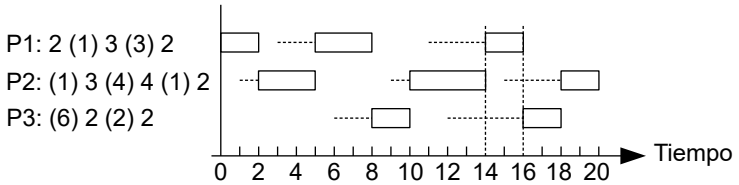
I. (4 puntos) Programe las funciones del cuadro de la derecha con la misma funcionalidad de las funciones de la pregunta 1 parte A, como herramientas de sincronización nativas de nThreads, es decir usando operaciones como *START_CRITICAL*, *setReady*, *suspend*, *schedule*, etc. Use el estado WAIT_NTBK para los threads en espera. También debe atender las solicitudes por orden de llegada. Como requisito adicional, para garantizar un envejecimiento equilibrado de los notebooks, en caso de haber múltiples equipos disponibles, Ud. debe asignar el notebook que fue devuelto hace más tiempo.

```
int nSolNtbk();
void nDevNtbk(int i);
void nth_init();
```

Ayuda: (a) Cree un arreglo de N enteros en donde el *i*-ésimo elemento contiene siempre el valor *i*. (b) Si el notebook *i* está disponible, incremente en 1 un contador global y agregue la dirección del *i*-ésimo elemento del arreglo a una cola de prioridades, usando el valor del contador global como prioridad. (c) Al solicitar un notebook, extraiga una dirección de la cola de prioridad. El contenido de esa dirección es el identificador del notebook que lleva más tiempo sin usar. El cuadro muestra las funciones que necesita para manipular la cola de prioridades. (d) Puede usar el campo *ptr* en el descriptor de thread para almacenar la dirección de alguna variable estratégica.

```
PriQueue *makePriQueue(void);
void *priGet(PriQueue *pq);
void priPut(PriQueue *pq,
        void *elem, double pri);
int emptyPriQueue(PriQueue *pq);
```

II. (2 puntos) 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) Explique cuál estrategia de scheduling sí es consistente con el diagrama de más arriba; y (c) Modifique el diagrama a partir del tiempo 14 para mostrar cómo se ejecutarían los mismos procesos considerando la estrategia *shortest job first non preemptive*.