CC4302 Sistemas Operativos – Control 2 – Semestre Primavera 2024 – Profs.: Mateu, Torrealba, Arenas

Pregunta 1

Un puente puede soportar PESOMAX como peso máximo. Los vehículos deben solicitar permiso para cruzar el puente invocando la función *nEntrar(p)*, en donde *p* es el peso del vehículo. Cuando esta función retorna, el vehículo puede entrar al puente. El vehículo notificará su salida llamando a la función *nSalir(p)*. El peso *p* nunca superará a PESOMAX. Los permisos para entrar al puente se otorgan por orden de llegada, lo que significa que se autoriza a un vehículo a entrar al puente si y solo si (i) su peso más la de los vehículos que ya están cruzando el puente no excede PESOMAX, y (ii) no hay un vehículo en espera que haya solicitado entrar al puente antes que él. Cuando un vehículo sale se permite que entren al puente tantos vehículos como sea posible sin violar (i) y (ii).

Programe las funciones del cuadro de la derecha como herramienta de sincronización nativa de nThreads, es decir usando operaciones como *START_CRITICAL*, *setReady*, *suspend*, *schedule*, etc. Puede definir nuevos estados y agregar nuevos campos al descriptor de los nano threads (como por ejemplo el peso). Para ello basta que lo explique en palabras. Debe evitar cambios de contexto inútiles. No puede implementar esta API en términos de otras herramientas pre-existentes en nThreads como semáforos, mutex o condiciones.

```
void iniPuente(void);
void nEntrar(double p);
void nSalir(double p);
```

Pregunta 2

Parte a.- (5,5 puntos) Programe las funciones del cuadro de la derecha con la misma funcionalidad de las funciones de la pregunta 1, pero resolviendo el problema con un mutex y múltiples condiciones evitando cambios de contexto inútiles, y por lo tanto necesitará el patrón request. Además **los permisos para ingresar al puente se deben otorgar por prioridad**. La prioridad se calcula para cada vehículo cuando solicita entrar al puente como su peso más una *penalidad* y no cambia mientras espera. La condición (ii) de la pregunta 1 cambiar por: se autoriza a un vehículo con prioridad Q a entrar al puente si no hay otro vehículo en espera con prioridad R mejor que la de Q, es decir R<Q. La *penalidad* es una variable global que parte en cero. Cuando un vehículo sale del puente, si hay vehículos en espera, la *penalidad* se incrementa en el peso del vehículo que sale. Si no hay vehículos en espera, la penalidad vuelve a cero.

```
void iniPuente(void);
void entrar(double p);
void salir(double p);
```

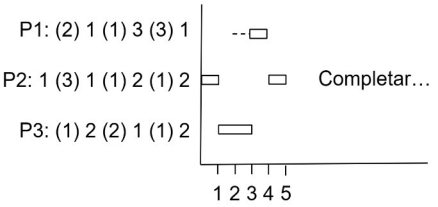
Ayuda: Use una cola de prioridad *q* de tipo *PriQueue* como la de la tarea 4. Las funciones para operar la cola son *priPeek(q)*, *priBest(q)*, *priGet(q)*, *priPut(q, e, pri)* y *emptyPriQueue(q)*. La diferencia entre *priPeek* y *priGet* es que *priPeek* no extrae el elemento. Use *priBest* para obtener la mejor prioridad en *q* en un instante dado. Cuando un vehículo deba esperar, agregue una solicitud (*request*) a la cola de prioridad. Un vehículo con prioridad *pri* no puede cruzar si *priBest(q)<pri*. Cuando salga un vehículo, use repetidamente *priPeek* y *priGet* para autorizar el cruce de los vehículos con mejor prioridad de modo que no se exceda el peso soportado por el puente. **Si le complica lo de la penalidad, ignórela**. Al final vea si puede incorporarla. La siguiente tabla muestra la evolución de un puente que soporta peso máximo 3. Inicialmente la penalidad es 0 y ya llegaron en este orden los vehículos (entre paréntesis se indica su peso): A(1.1), B(2.9), C(0.9), D(0.8) y E(0.7).

Evento (y peso)	*Pena-lidad*	*En espera (y prioridad)*	*Acción*	*Cruzando (y peso)*	*Peso utilizado*
estado inicial	0	E(0.7) B(2.9)		A(1.1) C(0.9) D(0.8)	2.8
Sale A	1.1	B(2.9)	Entra E	C(0.9) D(0.8) E(0.7)	2.4
Llega F(1.0)	1.1	F(2.1) B(2.9)	F espera	no cambia	2.4
Sale C	2.0	B(2.9)	Entra F	D(0.8) E(0.7) F(1.0)	2.5
Llega G(1.2)	2.0	B(2.9) G(3.2)	G espera	no cambia	2.5
Salen D, E, F	4.5	G(3.2)	**Entra B(*)**	B(2.9)	2.9
Sale B	7.4		Entra G	G(1.2)	1.2
Sale G	**0**			puente vacío	0

Note que en (*) debido a la penalidad la prioridad de G es peor que la de B.

Parte b.- (0,5 puntos) ¿Para qué sirve la penalidad?

Pregunta 3



I) El diagrama de la derecha 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. Complete el diagrama utilizando la estrategia *Shortest Job First non preemptive*. Considere que el predictor de duración de la próxima ráfaga es la duración de la última ráfaga.

II) ¿Tiene sentido usar spin-locks en una máquina monocore? Explique. Responda entonces cómo implementaría la exclusión mutua en un núcleo moderno de Unix para una máquina monocore.

III) ¿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?