CC4302 Sistemas Operativos – Examen – Semestre Primavera 2017 – Prof.: Luis Mateu

Pregunta 1

(i) Solución: No, no tiene sentido porque si un thread A intenta cerrar un spin-lock pero ese spin-lock lo posee otro thread B, el thread A hará busy-waiting inútilmente ocupando el 100% del tiempo del único core disponible. El thread A impedirá que el thread B se ejecute hasta que se acabe la tajada de tiempo del thread A. Recién ahí el thread B podrá liberar el spin-lock.

En vez de cerrar y abrir un spin-lock se deben inhibir las interrupciones antes de entrar a la sección crítica y activarlas nuevamente al salir. Así se asegura que un thread ejecutará toda la sección crítica de principio a fin, excluyendo la posibilidad de que una interrupción gatille un cambio de contexto que le dé la oportunidad a otro thread de ejecutarse en el único core disponible.

(ii) Solución: Depende de cuánto tiempo de CPU requiera la sección crítica. Si requiere muchos microsegundos conviene usar un semáforo porque si ya hay otro proceso en la sección crítica, se hará un cambio de contexto para retomar otro proceso, y así aprovechar mejor la CPU. En cambio, si la sección crítica es de corta duración, es mejor usar un spin-lock que hará busy-waiting para esperar que el otro core salga de la sección crítica. Si se usara un semáforo, el cambio de contexto tomaría más tiempo de CPU que el tiempo que toma la ejecución de la sección crítica.

(iii) Solución: Se perdería la protección entre procesos. Si una aplicación se "cae" por mal manejo de punteros, podría modificar la memoria de otra aplicación haciendo que esta también se "caiga". Por lo tanto, cada vez que una sola aplicación se "cae" habría que reiniciar el sistema operativo completo y todas las aplicaciones. Con paginamiento basta reiniciar la aplicación que se "cayó".

(iv) Solución:

	Disco	***SSD***
page faults	~ 100 por segundo	~ 40000 por segundo
velocidad	100 MB por segundo	400 MB por segundo
tiempo de vida	5 años	1000 a 10000 escrituras por celda

Pregunta 2

a.-

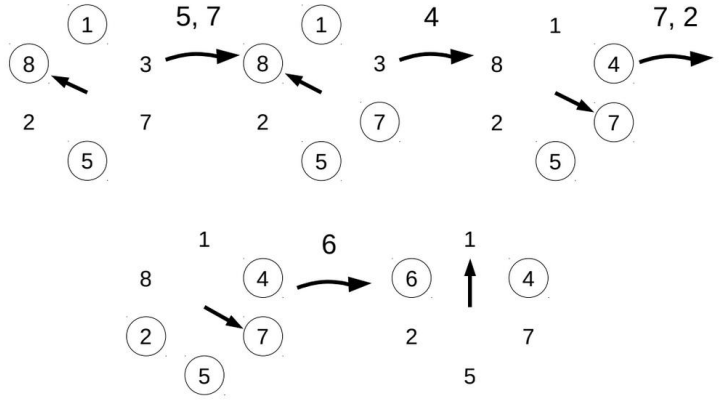
```
// Campos en el descriptor de tarea
typedef struct {
  ...
  Oxygen oxy;
  Hydrogen hydro;
} *nTask;

// Variables globales
Queue hydroQ; // = MakeQueue();
Queue oxyQ; // = MakeQueue();

void nOxygen(Oxygen *oxy, Hydrogen **pH1, Hydrogen **pH2) {
  // Garantizar exclusión mutua con S_C y E_C en ambas funciones
  START_CRITICAL();
  if (LengthQueue(hydroQ)<2) {
    // Suspender tarea si no están los 2 hidrógenos
    current_task->status= WAIT_HYDRO; // no es relevante cambiar estado
    PutTask(oxyQ, current_task);
    Resume();
  }
  nTask t1= GetTask(hydroQ); // Extraer 2 tareas hidrógeno
  nTask t2= GetTask(hydroQ);
  t1->oxy= t2->oxy= oxy;     // Depositar oxígeno en tareas hidrógeno
  t1->status= t2->status= READY;
  *pH1= t1->hydro; // Entregar hidrógenos en *pH1 y *pH2
  *pH2= t2->hydro;
  // Retomar las 2 tareas hidrógeno (no es relevante el orden en que se retoman)
  PushTask(ready_queue, current_task);
  PushTask(ready_queue, t1);
  PushTask(ready_queue, t2);
  Resume();
  END_CRITICAL();
}
```

```
nOxygen *nHydrogen(Hydrogen *h) {
  START_CRITICAL();
  // Registrar hidrógeno para ser extraído por la tarea oxígeno
  current_task->hydro= h;
  // Colocar tarea en hydroQ para suspenderla en (*)
  current_task->status= WAIT_OXY; // no es relevante cambiar estado
  PutTask(hydroQ, current_task);
  // Retomar una tarea oxígeno si hay 2 hidrógenos en hydroQ
  if (LengthQueue(hydroQ)>=2 && !EmptyQueue(oxyQ)) {
    nTask t= GetTask(oxyQ);
    t->status= READY;
    PushTask(ready_queue, t);
  }
  Resume(); // (*) Suspende tarea hidrógeno
  Oxygen *oxy= current_task->oxy; // Rescatar oxígeno y retornarlo
  END_CRITICAL();
  return oxy;
}
```

II. Solución:



b.- Solución:

(i) 300, 600, 200, 900, 800

(ii) 600, 800, 900, 300, 200

(iii) 300, 200, 600, 800, 900

Pregunta 3

I. Solución:

```
Oxygen combineHydro(Hydrogen *hydro) {
  spinLock(&_empty);       // wait buffer vacio
  _hydro= hydro;           // Colocar hidrógeno en buffer
  Oxy *oxy;
  _pOxy= &oxy;             // Dejar dirección de variable en donde debe quedar
                           // el oxígeno y retornar ese oxígeno (*)
  int slot= _k;
  _k= (_k+1)%2;
  spinUnlock(&_full);      // signal buffer lleno
  spinLock(&_wait[slot]); // Esperar hasta completar molécula de agua:
  // En realidad no se puede usar &wait[k] debido a un datarace, pero se acepta.
  return oxy;              // Junto con (*)
}
```