# CC4302 Sistemas Operativos – Examen – Semestre Primavera 2024 – Profs.: Mateu, Torrealba, Arenas

## Pregunta 1

Para reproducir un video se tiene: (i) un thread productor que lee cuadros del video por internet y los deposita en un buffer con la función *depositar*, (ii) un segundo thread consumidor que obtiene un cuadro del buffer con la función *obtener* y lo muestra en la pantalla. A la derecha se muestra una implementación de *depositar* y *obtener*. Para una reproducción más fluida cuando hay intermitencias en internet modifique esta implementación de manera que cuando la cola este vacía, se espere hasta llenarla completamente con N cuadros, porque es preferible hacer pocas pausas, aunque largas, que hacer muchas pausas de unos pocos cuadros. El video termina cuando se recibe un cuadro NULL, de manera que si el consumidor está en espera, en *depositar* solo debe despertar al consumidor cuando el cuadro es NULL o cuando la cola tiene N cuadros. Use una nueva variable global para indicar que el consumidor está en espera.

```
#define N 4096
pthread_mutex_t m=
      PTHREAD_MUTEX_INITIALIZER
;
pthread_cond_t cond=
     PTHREAD_COND_INITIALIZER;
Queue *q; // = makeQueue();
Cuadro *obtener() {
  lock(&m);
  while (emptyQueue(q))
    wait(&cond, &m);
  Cuadro *c= get(q);
  broadcast(&cond);
  unlock(&m);
  return c;
}
void depositar(Cuadro *c)
{
  lock(m);
   while(queueLength(q)==N
)
    wait(&cond, &m);
  put(q, c);
  broadcast(&cond);
  unlock(&m);
}
```

***Requerimiento adicional***: Debe evitar cambios de contexto inútiles, usando una condición para hacer esperar al productor y otra para hacer esperar al consumidor.

## Pregunta 2

Reprograme completamente su implementación de la pregunta 1, con los mismos requerimientos, pero con funciones *nObtener* y *nDepositar*, que sean herramientas de sincronización nativa de nThreads, es decir usando operaciones como *START_CRITICAL*, *setReady*, *suspend*, *schedule*, etc. No puede implementar esta API en términos de otras herramientas de sincronización pre-existentes en nThreads como semáforos, mutex o condiciones. Use los estados WAIT_OBTENER y WAIT_DEPOSITAR.

```
Cuadro *nObtener();
void      nDepositar(Cuadro
*c);
```

*Ayuda*: No necesita ninguna cola adicional, salvo *q*. Use 2 variables globales para los identificadores de thread del productor y consumidor cuando están en espera. Por ejemplo si una variable es nula quiere decir que ese thread está trabajando, si es no nula, ese es el identificador del thread en espera. Recuerde que es un solo consumidor y un solo productor.

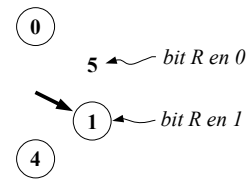
## Pregunta 3

A) Considere que las ráfagas de los procesos asociados a entrada/salida son pequeñas. Explique por qué la estrategia FCFS (*First Come First Served*) podría desfavorecer estos procesos.

B) Considere que las ráfagas de los procesos asociados a tareas de cálculo intensivo (por ejemplo, procesamiento de imágenes o simulaciones científicas) son grandes. Explique por qué la estrategia SJF (*Shortest Job First*) podría desfavorecer estos procesos.

C) Se tiene un archivo de 98 KB en una partición Unix con bloques de 4 KB (98/4=24.5). Haga un diagrama mostrando inodo, bloques de datos y de indirección.

D) Considere un sistema Unix que implementa la estrategia del reloj. El sistema posee 6 páginas reales disponibles y corre un solo proceso. La figura indica el estado inicial de la memoria, mostrando las páginas residentes en memoria, la posición del cursor y el valor del bit R.




Dibuje los estados por los que pasa la memoria para la siguiente traza de accesos a páginas virtuales: 4 8 7 1 6 5

E) 5 procesos se encuentra en estado de espera porque hicieron peticiones para leer sectores del disco en el siguiente orden: 100, 900, 200, 400, 300. El último sector leído fue el 700 y el penúltimo el 800. Indique en qué orden se harán las lecturas de estos 5 procesos cuando la estrategia de scheduling de disco es: (i) first come first served, (ii) shortest seek first, (iii) método del ascensor (o look).