CC4302 Sistemas Operativos – Control 1 – Semestre Otoño 2025 – Profs.: Mateu, Torrealba, Arenas

Pregunta 1 (30%)

```
void impr(Doc *docs, int n) {
  for (int i=0;i<n;i++)
    devImpr(docs[i],0);
}
```

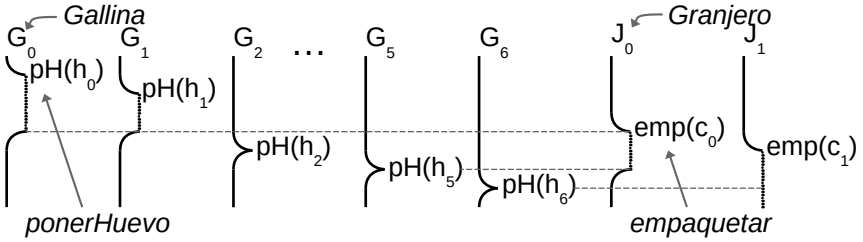
La función *impr* de la derecha imprime *n* documentos de distintos tamaños (por ejemplo de 1 a 100 páginas) y por lo tanto el tiempo de impresión es muy variable. La función *devImpr(doc, imp)* imprime el documento *doc* en la impresora *imp*, que puede ser 0 o 1, y no retorna hasta que se haya impreso completamente *doc*. A pesar de que hay 2 impresoras, la función *impr* usa solo la 0. Reprográmela de modo que use ambas impresoras (0 y 1) con la siguiente restricción: una impresora no puede permanecer ociosa si existe algún documento cuya impresión no se ha iniciado aún. Por lo tanto no puede usar el patrón subintervalo, porque la primera mitad del arreglo puede contener documentos de una sola página, mientras que la segunda mitad documentos de 100 páginas. El thread que le toque imprimir la primera mitad quedará ocioso la mayor parte del tiempo.

Use obligatoriamente esta metodología: Cree un solo thread adicional. El thread original imprime en la impresora 0 y el nuevo en la impresora 1. Declare variables globales para mantener por ejemplo el arreglo de documentos, la cantidad de documentos y la posición *P* del primer documento en el arreglo cuya impresión todavía no se inicia. Cuando un thread se desocupa, este elige imprimir el documento *P* e incrementa *P*. Se termina si $P \geq n$. Debe evitar dataraces al usar *P*.

Pregunta 2 (40%)

Múltiples gallinas llaman a la función *ponerHuevo(h)* cacareando que acaban de poner el huevo *h* y se quedan en espera (empollando *h*) hasta que un granjero retire el huevo. Múltiples granjeros llaman a la función *empaquetar(c)* para armar la caja *c* de huevos, en donde *c* es un arreglo de 6 huevos. Esta función retorna cuando se complete la caja con los 6 huevos puestos en el arreglo. Los huevos se deben depositar en las cajas por orden de llegada de las gallinas. Las cajas se deben llenar por orden de llegada de los granjeros. Cuando un granjero coloca el primer huevo en una caja, los siguientes 5 huevos deben ir en esa misma caja. El cuadro muestra los encabezados de las funciones solicitadas. El siguiente diagrama muestra un ejemplo de uso:

```
void ponerHuevo(Huevo huevo);
void empaquetar(Huevo *caja);
```



Observe que las gallinas G_0 y G_1 esperan hasta que llegue J_0, pero las gallinas G_2 a G_5 continúan de inmediato porque todavía hay espacio en la caja c_0 de J_0. Los huevos de las gallinas G_0 a G_5 quedan en el arreglo c_0. J_0 debe esperar hasta que se pone h_5. La gallina G_6 continúa de inmediato porque ya llegó un nuevo granjero J_1 y hay espacio vacío en su caja c_1. J_1 debe esperar a que se pongan otros 5 huevos.

Restricción: Debe resolver este problema de sincronización usando un solo mutex y **una sola condición**. Por lo tanto necesitará un distribuidor de números para las gallinas y otro para los granjeros. Por simplicidad deposite los huevo en una cola (tipo Queue) a medida que las gallinas los vayan poniendo.

Pregunta 3 (30%)

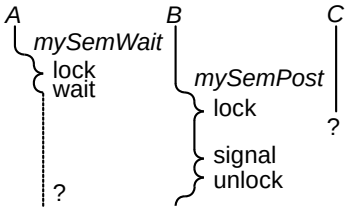
Parte a.- El cuadro de más abajo implementa ***incorrectamente*** un semáforo a partir de mutex y condiciones.

```
typedef struct {
  pthread_mutex_t m;
  int cnt;
  Queue *q;
} MySem;

void mySemWait(MySem *s) {
  lock(&s->m);
  if (s->cnt==0) {
    Req req={0,PTHREAD_COND_INITIALIZER};
    put(s->q, &req);
    while (!req.rdy)
      cond_wait(
            &req.w, &s->m);
  }
  s->cnt--;
  unlock(&s->m);
}
```

```
typedef struct {
  int rdy;
  pthread_cond_t w;
} Req;
MySem *mySemMake(int ini){
  …
}
void mySemPost(MySem *s) {
  lock(&s->m);
  s->cnt++;
  if (!emptyQueue(s->q)) {
    Req *preq= get(s->q);
    preq->rdy= 1;
    cond_signal(&preq->w);
  }
  pthread_mutex_unlock(&s->m);
}
```

En el diagrama de threads de la derecha el thread A invoca *mySemWait* cuando el semáforo está vacío y por lo tanto espera. Más tarde B invoca *mySemPost*. Complete los threads A y C en el diagrama de modo que muestre que 2 *mySemWait* obtienen la misma ficha que depositó B y por lo tanto ambos retornan incorrectamente.



Parte b.- Haga cambios menores en esta implementación que corrijan el problema. Puede responder esta parte sin responder *a*. No necesita programar *mySemMake*.