Pregunta 1

Se programó la función *match* para buscar en paralelo un patrón dentro de una imagen. Esta solución es incorrecta pero todos sus errores están relacionados con el uso de *fork*.

```
int region_match(Image *i, Pattern *p, int xi, int yi,
                 int xf, int yf, int *px, int *py);
int match(Image *img, Pattern *pat, int *x, int *y) {
  int i;
  int xstep= imgWidth(img)/8, h= imgHeight(img);
  for (i= 0; i<8; i++) {
    if (fork()==0) {
      if ( region_match(img, pat,
              i*xstep, 0, (i+1)*xstep-1, h-1, &x, &y) ) {
        return 1;
  } } }
  return 0;
}
```

La función *region_match* entrega verdadero si se encuentra el origen del patrón *p* en el rectángulo delimitado por (*xi*, *yi*) y (*xf*, *yf*) de la imagen *i*, entregando la coordenada de su origen en **px* y **py*. Esta función toma mucho tiempo de CPU.

Parte a.- (1 punto) Indique cuales son los 3 errores de programación de la solución de arriba que están relacionados con el uso de *fork*.

Parte b.- (3 puntos) Escriba la solución correcta y eficiente de la función *match*.

Parte c.- (2 puntos) La siguiente función compara un conjunto de *n* imágenes entre sí entregando verdadero si 2 imágenes son semejantes con un error máximo de *epsilon*. Los índices de las imágenes similares se entregan en **pi* y **pj*. Para ello se llama a la función *cmpImg* que toma mucho tiempo de CPU.

```
int cmpImgVec(Img *imgs[], int n, double epsilon,
              int *pi, int *pj,) {
  int i, j;
  for (i= 0; i<n; i++) {
    for (j= 0; j<i; j++) {
      if (cmpImg(imgs[i], imgs[j])<epsilon) {
        *pi= i; *pj= j;
        return 1
  } } }
  return 0;
}
```

Modifique la función *cmpImgVec* para que cada 10 segundos despliegue en la salida estándar una línea indicando cuantas llamadas a *cmpImg* se han realizado hasta el momento. Use señales. Ejemplo: 233 llamadas.

Pregunta 2

Se requiere programar el servidor y el cliente de un sistema para hacer colectas por Internet. El servidor se llama *colecta*, corre siempre en *localhost* y escucha a los clientes en el puerto 3000. Recibe como argumento el monto de la meta que se debe alcanzar. Ejemplo de invocación:

```
$ colecta 11
```

El cliente se llama *aportar* y recibe como argumento el monto del aporte. El cliente debe esperar hasta que se haya alcanzado la meta. En tal caso entrega el monto efectivo aportado. Por ejemplo si faltan 3 euros para alcanzar la meta y una persona aporta 5, entonces aportar despliega 3. Si una persona aporta cuando la meta ya fue alcanzada, entonces se despliega 0 de inmediato. El siguiente es un ejemplo de uso. La meta a recaudar es 11.

Persona 1	***Persona 2***	***Persona 3***	***Persona 4***
$ **aportar 5** *(espera)*			
	$ **aportar 2** *(espera)*		
5 $ *(termina)*	2 $ *(termina)*	$ **aportar 7** 4 $	
			$ **aportar 3** 0 $

Note que el *prompt* $ indica cuando un comando termina o si debe esperar. El tiempo avanza hacia abajo. En **negritas** aparece lo que escribió un usuario.

Requerimientos: Programe el cliente y el servidor de este sistema de colectas. El cliente debe reproducir exactamente la salida que se muestra en el ejemplo, exceptuando el texto que dice *(espera)* o *(termina)*. En el servidor: use threads para atender los clientes; no programe la función *main*; programe la función de servicio (*serv*); suponga que la meta a recaudar se encuentra en una variable global; deberá usar otras variables globales; señale cómo se inicializan en el *main*.