Primer Certamen
Tiempo 120 min. Al terminar, entregar sus respuestas vía AULA.

1.- (35 pts) Una página web con archivo principal index.html contiene referencias a varias imágenes ubicadas en otro directorio. Se desea listar todos los nombres de los archivos de imágenes que no son referenciados desde index.html. Es decir la página web se visualiza de manera completa incluso borrando esas imágenes no usadas.
Cree el script shell (unUsedImage.sh) que recibe como argumentos en la línea de comando el archivo html base y el directorio donde se encuentras las imágenes. Como salida el script muestra todas las imágenes, bajo el directorio señalado, que no son referenciadas en el archivo html dado. Suponga que el archivo html y el script se encuentran en el directorio actual y el directorio de imágenes es relativo a la ubicación del directorio actual. Una invocación posible sería:
$ unUsedImage.sh index.html img
Con index.html como archivo html base e img como directorio para las imágenes.

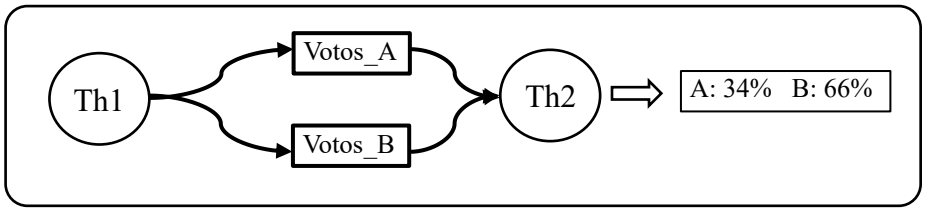
Para esta pregunta, consideraremos que una imagen "imagen.jpg" es referenciada en el archivo html "index.html" si el siguiente comando es exitoso:
$grep 'imagen.jpg' index.html

Aquí usted puede encontrar uno de tales archivos html con sus imágenes.

```
#! /bin/bash
dir=$2
htmlFile=$1
for i in $dir/*
do
if   test -f $i && ( file $i | grep image >> /dev/null ) && ! grep $i $htmlFile > /dev/null
#if   ! grep $i $htmlFile > /dev/null   # fue considera OK por redacción de enunciado.
then
       echo $i
fi
done
```

El código de la pauta lo puede ver aquí.

2.- Cree el programa "democracia.c" para simular el conteo de votos en una elección. Democracia.c recibe dos argumentos enteros E1 y E2 como en:
$ democracia 100 500 /* E1 = 100, E2 = 500. */
Se tiene una hebra Th1 que genera y procesa los votos de la elección de un cargo para el cual hay 2 candidatos A y B. Hay un contador de preferencias por cada candidato. Por simplicidad no hay nulos ni blancos.



Una segunda hebra lee los votos de cada candidato y muestra por pantalla el porcentaje de preferencias de cada candidato. Cada hebra desarrolla un lazo del tipo:

```
Th1() {
  for (i=0; i<50; i++) {
   r = valor entre 0 y 1 /* puede usar drand48() */
   if (r < 0.4)  Votos_A++;
   else    Votos_B++;
   esperar E1 [ms];     /* se sugiere usar usleep */
  }
  exit(0);
```

```
Th2() {
 while(1) {
   a = 100*Votos_A/(Votos_A+Votos_B);
   b = 100*Votos_B/(Votos_A+Votos_B);
   imprima A: a %    B: b % ;
   esperar E2 [ms];
 }
}
```

}	

a) 35 (pts) Desarrolle el programa democracia.c siguiendo los lineamientos datos y garantizando el acceso exclusivo a los datos compartidos.

b) 30 (pts) Desarrollo el programa democracia2.c en el cual Th2 actualiza el resultado mostrado por pantalla tan pronto se ha procesado otro voto. En este caso se omite el segundo parámetro E2 y se debe evitar estados de espera ocupado (Busy Wait).

Sugerencia: Considere un mutex único para controlar el acceso a ambos contadores.

El código de la pauta lo puede ver aquí.

a)

```
/*
Cree el programa "democracia.c" para simular el conteo de votos
en una elección. Democracia.c recibe dos argumentos enteros
E1 y E2 como en:
$ democracia 100 500 * E1 = 100, E2 = 500. *
Se tiene una hebra Th1 que genera y procesa los votos de la
elección de un cargo para el cual hay 2 candidatos A y B.
Hay un contador de preferencias por cada candidato.
Por simplicidad no hay nulos ni blancos.
Desarrolle el programa democracia.c siguiendo los lineamientos
datos y garantizando el acceso exclusivo a los datos compartidos.

$ gcc -o democracia -pthread democracia.c
*/

#include <stdio.h>
#include <stdlib.h>  /* drand48() */
#include <unistd.h>  /* usleep */
#include <pthread.h>

int votos_A=0;
int votos_B=0;

pthread_mutex_t mylock = PTHREAD_MUTEX_INITIALIZER;

void * showResult(void * arg) {
  int wait = *((int *)arg);
  float a, b;
  while (1) {
    pthread_mutex_lock(&mylock);
    a = (float)votos_A/(votos_A+votos_B);
    pthread_mutex_unlock(&mylock);
    b = 1-a;
    printf("A: %f %    b: %f %\n", 100*a, 100*b);
    usleep(wait);
  }
}

int main(int argc, char * argv[]) {
  int err, e1, e2, i;
  double r;
  pthread_t tid;

  e1 = 1000*atoi(argv[1]);
  e2 = 1000*atoi(argv[2]);

  err = pthread create(&tid, NULL, showResult, &e2);
  if (err != 0){
    printf("can't create thread \n");
    exit(0);
  }
  for (i = 0; i < 50; i++){
    r = drand48();
    pthread_mutex_lock(&mylock);
    if ( r < 0.4)
      votos_A++;
```

```
    else
      votos_B++;
    pthread_mutex_unlock(&mylock);
    usleep(e1);
  }
}
```

b)

```
/*
Cree el programa "democracia.c" para simular el conteo de votos
en una elección. Democracia.c recibe dos argumentos enteros
E1 y E2 como en:
$ democracia 100 500 * E1 = 100, E2 = 500. *
Se tiene una hebra Th1 que genera y procesa los votos de la
elección de un cargo para el cual hay 2 candidatos A y B.
Hay un contador de preferencias por cada candidato.
Por simplicidad no hay nulos ni blancos.
Desarrollo el programa democracia2.c en el cual Th2 actualiza
el resultado mostrado por pantalla tan pronto se ha procesado
otro voto. En este caso se omite el segundo parámetro E2 y
se debe evitar estados de espera ocupado (Busy Wait).

$ gcc -o democracia2 -pthread democracia2.c
*/

#include <stdio.h>
#include <stdlib.h>  /* drand48() */
#include <unistd.h>  /* usleep */
#include <pthread.h>

int votos_A=0;
int votos_B=0;

pthread_mutex_t mylock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t votesChange = PTHREAD_COND_INITIALIZER;

void * showResult(void * arg) {
  float a, b;
  while (1) {
    pthread_mutex_lock(&mylock);
    pthread_cond_wait(&votesChange, &mylock);
    a = (float)votos_A/(votos_A+votos_B);
    pthread_mutex_unlock(&mylock);
    b = 1-a;
    printf("A: %f %    b: %f %\n", 100*a, 100*b);
  }
}

int main(int argc, char * argv[]) {
  int err, e1, i;
  double r;
  pthread_t tid;

  e1 = 1000*atoi(argv[1]);

  err = pthread_create(&tid, NULL, showResult, NULL);
  if (err != 0){
    printf("can't create thread \n");
    exit(0);
  }
  for (i = 0; i < 50; i++){
    r = drand48();
    pthread_mutex_lock(&mylock);
    if ( r < 0.4)
      votos_A++;
    else
      votos_B++;
    pthread_mutex_unlock(&mylock);
    pthread_cond_signal(&votesChange);
    usleep(e1);
  }
}
```