Sistemas Operativos 1/2021
Laboratorio 3

Profesores:
Cristóbal Acosta (cristobal.acosta@usach.cl)
Fernando Rannou (fernando.rannou@usach.cl)
Ayudantes:
Cristóbal Fernández (cristobal.fernandez@usach.cl)
Manuel I. Manríquez (manuel.manriquez.b@usach.cl)
Benjamín Muñoz (benjamin.munoz.t@usach.cl)

I. Objetivos Generales

El objetivo del presente laboratorio es aplicar los conocimientos adquiridos en cátedra sobre concurrencia. Se espera que los estudiantes generen una solución teniendo en cuenta las consideraciones propias del diseño multihebreado y que realicen llamadas al SO Linux de manera exitosa.

II. Objetivos Específicos

1. Conocer y usar las funciones y tipos de datos de la librería *pthread* como método de creación y administración de hebras.
2. Implementar un *pipeline* de procesamiento de imagen en un programa multihebreado.
3. Implementar medidas de sincronización de hebras.
4. Implementar soluciones de exclusión mutua por software.
5. Practicar técnicas de buena documentación de programas.
6. Practicar uso de `Makefile` para compilación de programas.
7. Practicar distintas técnicas de procesamiento de imágenes.

III. Conceptos

III.A. Concurrencia y Sincronización

Cuando dos o más tareas (procesos o hebras) comparten algún recurso en forma concurrente o paralela, debe existir un mecanismo que sincronice sus actividades.

De no existir una sincronización, es posible sufrir una corrupción en los recursos compartidos u obtener soluciones incorrectas.

III.B. Sección Crítica

Porción de código que se ejecuta de forma concurrente y podría generar conflicto en la consistencia de datos debido al uso de variables globales.

III.C. Mutex

Provee exclusión mutua, permitiendo que sólo una hebra a la vez ejecute la sección crítica.

III.D. Hebras

Los hilos POSIX, usualmente denominados *pthreads*, son un modelo de ejecución que existe independientemente de un lenguaje, además es un modelo de ejecución en paralelo. Éstos permiten que un programa controle múltiples flujos de trabajo que se superponen en el tiempo.

Para poder utilizar hebras, es necesario incluir la librería **pthread.h**. Por otro lado, dentro de la función main, se debe instanciar la variable de referencia a las hebras, para esto se utiliza el tipo de dato **pthread_t** acompañado del nombre de variable.

Luego, el código que ejecutarán las hebras se debe construir en una función de la forma:

```
void * function (void * params)
```

La cual recibe parámetros del tipo `void`, por lo cual es necesario castear el o los parámetros de entrada para así poder utilizarlos sin problemas.

Algunas funciones para manejar las hebras son:

- **pthread_create:** función que crea una hebra. Recibe como parámetros de entrada:
  - La variable de referencia a la hebra que desea crear.
  - Los atributos de éste, los cuales no es obligación de modificar, por lo que en caso de no querer hacerlo, simplemente se deja NULL.
  - El nombre de la función que la hebra ejecutará (la cual debe cumplir con la descripción antes mencionada)
  - Por último, los parámetros de entrada (de la función que se ejecutará) previamente casteados.

  Un ejemplo sería:

  ```
  while(i < numeroHebras)
  {
      pthread_create(&hilo[i], NULL, escalaGris, (void *) &structHebra[i].id);
      i++;
  }
  ```

- **pthread_join:** función donde la hebra que la ejecuta, espera por las hebras que se ingresan por parámetro de entrada.

  Un ejemplo sería:

  ```
  while(i < numeroHebras)
  {
      pthread_join(hilo[i],NULL);
      i++;
  }
  ```

- **pthread_mutex_init:** función que inicializa un mutex, pasando por parámetros la referencia al mutex, y los atributos con que se inicializa.

  Para inicializar la estructura se utiliza:

```
while(i < numeroHebras)
{
    pthread_mutex_init(&MutexAcumulador, NULL);
    i++;
}
```

Donde **MutexAcumulador**, es una variable (de tipo `pthread_mutex_t`) que representa un mutex, el cual permitirá implementar exclusión mutua a una sección crítica que será ejecutada por varias hebras.

- **pthread_mutex_lock:** entrega una solución a la sección crítica. Ésta recibe como parámetros la variable que se desea bloquear para el resto de hebras. Un ejemplo de uso sería:

  ```
  pthread_mutex_lock(&MutexAcumulador);
  ```

  Cabe destacar que la primera hebra en ejecutar **pthread_mutex_lock** podrá ingresar a la sección crítica, el resto de hebras quedarán bloqueadas a la espera de que se libere el mutex.

- **pthread_mutex_unlock:** permite liberar una sección crítica. Ésta recibe como parámetro de entrada, la variable que se desea desbloquear. Su implementación es la siguiente:

  ```
  pthread_mutex_unlock(&MutexAcumulador);
  ```

- **pthread_barrier_init**: permite asignar recursos a una barrera e inicializar sus atributos. Su implementación es:

  ```
  pthread_barrier_init(&rendezvous, NULL, NTHREADS);
  ```

  Donde rendezvous es una variable de tipo pthread_barrier_t, NULL indica que se utilizan los atributos de barrera predeterminados (se pide usar estos atributos, no debiese ser necesario modificarlos) y NTHREADS es la cantidad de hebras que deben esperar en la barrera.

- **pthread_barrier_wait**: permite sincronizar los hilos en una barrera especificada. El hilo de llamada bloquea hasta que el número requerido de hilos ha llamado a `pthread_barrier_wait()` especificando la barrera. Un ejemplo de su uso:

  ```
  funcion1();
  pthread_barrier_wait(&rendezvous);
  funcion2();
  ```

- **pthread_barrier_destroy**: cuando ya no se necesita una barrera, se debe destruir. Su implementación es:

  ```
  pthread_barrier_destroy(&rendezvous);
  ```

IV. Enunciado

El trabajo consiste en un pipeline de procesamiento de imágenes, donde se deben aplicar las siguientes etapas: Zoom-in, suavizado y transformación de delineado. Es decir, la imagen de entrada debe pasar por cada una de las etapas y entregar una imagen resultante después de aplicar todos los filtros. Las etapas son:

IV.A. Zoom-in

Consiste en una técnica de replicación de pixeles para toda imagen de MxN, donde cada pixel es replicado según un factor de escala r que será pasado como parámetro de entrada, lo cual da origen a una imagen resultante de dimensiones *(M*r)x(N*r)*.

Por ejemplo, se tiene una imagen A de 2x2 y un factor de replicación de 2, por lo que la imagen resultante tendrá dimensiones resultantes de 4x4 y quedará similar al siguiente ejemplo:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad Zoom - in(A) = \begin{bmatrix} a & a & b & b \\ a & a & b & b \\ c & c & d & d \\ c & c & d & d \end{bmatrix}$$

Para una imagen de 2x2 con factor de replicación 3 se tiene:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad Zoom - in(A) = \begin{bmatrix} a & a & a & b & b & b \\ a & a & a & b & b & b \\ a & a & a & b & b & b \\ c & c & c & d & d & d \\ c & c & c & d & d & d \\ c & c & c & d & d & d \end{bmatrix}$$

IV.B. Filtro Suavizado

Técnica que sirve para suavizar los bordes de un imagen, reducir el ruido o disminuir los cambios de intensidad en la imagen. En este caso se hará uso del filtro media, en el cual se obtiene el pixel de salida obteniendo la media aritmética de su vecindario.

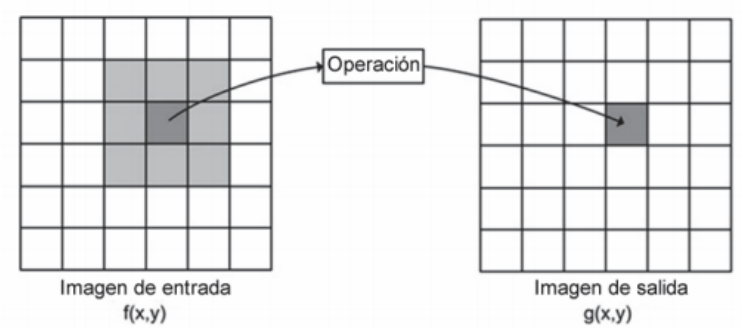

Figure 1. Ejemplo de obtención de pixel de salida

Figure 2. Ejemplo de imagen pasada por filtro de suavizado

IV.C. Transformación de delineado

La transformación de delineado realiza la operación contraria al suavizado, esta sirve para destacar y hacer más visibles las variaciones y bordes de la imagen. En este caso se hará uso de la máscara Laplaciana, la cual proviende del operador Laplaciano, el cual se define como:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Donde I es la intensidad del píxel actual y los valores x,y corresponden a la posición del píxel actual. Este operador hace uso de la segunda derivada espacial de la imagen para destacar cambios bruscos de intensidad. Para hace uso de esta transformación se debe convolucionar la máscara Laplaciana con la imagen entregada, esta máscara es:

$$MáscaraLaplaciana = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Para obtener el píxel de salida se debe tomar su vecindario, al igual que en la Figura 1. Para un caso general, suponemos una imagen representada por la matriz A, donde el píxel final *e'* tiene la forma:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

$$e' = a*-1 + b*-1 + c*-1 + d*-1 + e*8 + g*-1 + h*-1 + i*-1$$

Para el caso particular de los bordes, no se debe tomar en cuenta la máscara que sale de la imagen.

Figure 3. Ejemplo de imagen al aplicar máscara Laplaciana

V. Hebras

Para este laboratorio debe haber una hebra `productora` que leerá la imagen de manera secuencial produciendo filas y llenando un `buffer` de largo `b` (parámetro ingresado por consola). Después, se deben crear una cantidad `h` de hebras `consumidoras` (también ingresado por consola) que vaciarán el buffer cuando este se encuentre lleno, distribuyéndose filas adyacentes hasta tener la cantidad `m` de filas que deben recoger, lo cual se calcula de la siguiente forma:

$$m = \frac{cantidad\_filas\_imagen}{cantidad\ de\ hebras} \tag{1}$$

Por ejemplo, si se tiene una imagen de 256 filas y 4 hebras, cada hebra tendrá m = 256/4 = 64 filas. Si el número de filas por hebra no es un número entero, habrá una hebra que consumirá menos filas, y la hebra productora deberá avisar que no quedan más filas de la imagen por leer.

Para este caso, cada hebra manejará una región de la imagen y aplicará cada etapa del pipeline a su sección correspondiente, entregando una imagen resultante para ser recogida por las hebras al momento de aplicar el siguiente filtro. Debe existir una sincronización entre cada etapa para lo que se deben aplicar barreras, y con esto evitar errores de sincronización. Es decir, antes de cada etapa del pipeline debe haber

una barrera para esperar a todas las hebras consumidoras y continuar con el filtro correspondiente. Además de esto, también se debe tener en cuenta para el caso del filtro de suavizado y de delineado las condiciones de borde al momento de calcular el pixel, ya que necesitará valores que no están en la región correspondiente a una hebra.

```
void *hebra (void* parametros){
   //recoger porción de la imagen
   //aplicar zoom-in
   pthread_barrier_wait();
   //aplicar efecto de suavizado
   pthread_barrier_wait();
   ...
}
```

Una vez las hebras hayan completado todas las etapas del Pipeline, cada una debe escribir su porción resultante en una estructura común, para lo cual también debe proveerse exclusión mutua. Cuando todas las hebras terminen de escribir la imagen final, la hebra padre debe hacer uso de `join` para esperar que finalicen y después escribir el archivo final.

VI. Parámetros de Entrada

Para este trabajo se debe entregar al menos un archivo llamado `lab3.c` y un *Makefile* que entregue archivos ejecutables para lab3 y cada etapa del pipeline. Cada imagen estará en formato raw (`float`). La ejecución del programa tendrá los siguientes parámetros que deben ser procesados por `getopt()`:

- `-I filename` : especifica el nombre de la imagen de entrada
- `-O filename` : especifica el nombre de la imagen final resultante del pipeline.
- `-M número` : especifica el número de filas de la imagen
- `-N número` : especifica el número de columnas de la imagen
- `-h número` : especifica el número de hebras.
- `-r factor` : factor de replicación para Zoom-in
- `-b número` : especifica el tamaño del buffer de la hebra productora.
- `-f`: bandera que indica si se entregan resultados por consola. En caso de que se ingrese este flag deberá mostrar las dimensiones de la imagen antes y después de aplicar zoom-in.

Por ejemplo:

```
./lab3 -I imagen1.raw -O imagen1F.raw -M 512 -N 512 -r 2 -h 4 -b 64 -f
```

VII. Entregables

El laboratorio es individual o en parejas y se descontará 1 punto por día de atraso. Cabe destacar que no entregar un laboratorio implica la reprobación de la asignatura. Finalmente, debe subir en un archivo comprimido **formato zip** a Uvirtual los siguientes entregables:

- `Makefile`: Archivo para `make` que compila el programa. El makefile debe compilar cada vez que haya un cambio en el código. Si no hay cambios, el comando `make` no debe crear un nuevo objeto.

Figure 4. Ejemplo de funcionamiento de un `Makefile`

- `archivos .c y .h` Se debe tener como mínimo un archivo .c principal llamado lab3.c con el main del programa. Además, deben existir dos o mas archivos con el proyecto (*.c, *.h) . Se debe tener mínimo un archivo .h que tenga cabeceras de funciones, estructuras o datos globales. Se deben comentar todas las funciones de la forma:

```
//Entradas: explicar qué se recibe
//Funcionamiento: explicar qué hace
//Salidas: explicar qué se retorna
```

- Se considerará para evaluación las buenas prácticas de programación, como modularidad y nombres de variables representativos.
- Trabajos con códigos que hayan sido copiados de un trabajo de otro grupo serán calificados con la nota mínima.

El archivo comprimido debe llamarse: RUTESTUDIANTE1_RUTESTUDIANTE2.zip
Ejemplo: 19689333k_186593220.zip
NOTA: Si los laboratorios son en parejas, éstas puede ser de distintas secciones, pero se debe avisar previamente vía correo a los ayudantes (Cristóbal Fernández y Benjamín Muñoz).

VIII. Visualización de las imágenes

Tanto las imágenes de entrada como de salida estarán en formato binario (`float`). Por lo tanto, no pueden ser visualizadas directamente por los software de imágenes. Luego, usaremos Matlab (disponible gratuitamente a todos los estudiantes de la Universidad) para visualizar imágenes. El siguiente script `verimagenraw()` realiza el trabajo.

```
function a = verimangeraw('filename', M, N)
      f = fopen(filename, 'r');
      a = fread(f, 'float');
      a = reshape(s, N, M); % si, es N, M, no M, N
      a = a'; % al trasponer queda de MxN
      figure; imagesc(a); axis('square');axis('off');
      colormap(gray)
```

Aunque el script abre una ventana y muestra la imagen, también retorna el arreglo (la imagen) como una matriz, la cual usted podría seguir usando. Por ejemplo, para visualizar con una colormap distinto:

```
a = verimagenraw('imagen_1.raw', 512, 512);
figure; imagesc(a); axis('square');axis('off');
colormap(hot)
```

IX. Fecha de entrega

Domingo 25 de Julio 2021, hasta las 23:55 hrs.