Sistemas Operativos 1/2018
Laboratorio 4

Profesores:
Cristóbal Acosta (cristobal.acosta@usach.cl)
Miguel Cárcamo (miguel.carcamo@usach.cl)
Fernando Rannou (fernando.rannou@usach.cl)
Ayudantes:
Esteban Alarcón (esteban.alarcon.v@usach.cl)
Marcela Rivera (marcela.rivera.c@usach.cl)

I. Objetivos Generales

Con el uso de μC++, debe implementarse el problema de Productor-consumidor en el contexto de astroinformática, específicamente el programa Nearly Black, visto en los laboratorios anteriores.

II. Objetivos Específicos

1. Implementar programa Nearly black en μC++.
2. Programar el problema de productor consumidor usando monitores. Esto se explicará de forma detallada en las ayudantías, por lo que es de suma importancia su asistencia. Si requieren avanzar por su cuenta, lean la documentación en `https://plg.uwaterloo.ca/usystem/uC++.html`.

III. Problema Productor consumidor

El enunciado del problema productor/consumidor es que hay uno o más procesos generando algún tipo de datos y poniéndolos en un buffer. Hay un único consumidor que está extrayendo datos de dicho buffer de uno en uno. El sistema está obligado a impedir la superposición de las operaciones sobre los datos. Por tanto, sólo un agente (productor o consumidor) puede acceder al buffer en un momento dado.

IV. Monitores

Los monitores fueron construidos como una forma de subsanar las desventajas de la implementación de semáforos, los cuales no es sencillo ver el efecto global de sus operaciones (wait y signal) sobre los semáforos y sección de código que afectan. El monitor provee una funcionalidad equivalente a la de los semáforos pero es más fácil de controlar.

Un monitor soporta la sincronización mediante el uso de variables de condición que están contenidas dentro del monitor y son accesibles sólo desde el monitor; se manipulan mediante dos funciones:

- cwait(c): Suspende la ejecución del proceso llamante en la condición c. El monitor queda disponible para ser usado por otro proceso.
- csignal(c): Retoma la ejecución de algún proceso bloqueado por un cwait en la misma condición. Si hay varios procesos, elige uno de ellos; si no hay ninguno, no hace nada.

Aunque un proceso puede entrar en el monitor invocando cualquiera de sus procedimientos, puede entenderse que el monitor tiene un único punto de entrada que es el protegido, de ahí que sólo un proceso

pueda estar en el monitor a la vez. Otros procesos que intenten entrar en el monitor se unirán a una cola de procesos bloqueados esperando por la disponibilidad del monitor.

V. μC++

μC++ es una extensión de c++ para programas concurrentes, la cual agrega funcionalidades de corutinas, tareas y monitores. Para la instalación se debe clonar el repositorio desde `https://github.com/pabuhr/uCPP` y seguir las instrucciones de instalación.

Para verificar que se haya instalado correctamente, escribir en consola `u++`.

VI. Enunciado

Para este laboratorio, las etapas son ejecutadas por un objeto que realiza las respectivas funcionalidades (por ejemplo, Lector, Binarizador, etc) y se conectan a través de objetos Buffers, los cuales funcionan como monitores, donde va guardando en una cola los datos que reciben, para luego ir pasandolos a los objetos siguientes.

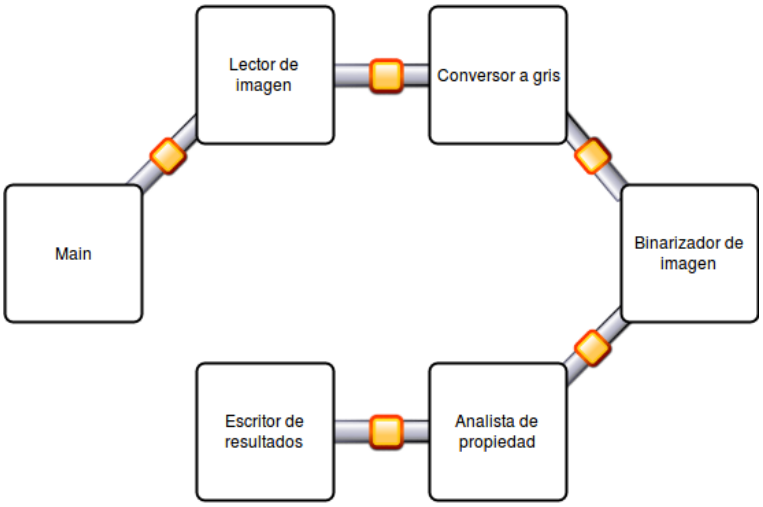

Figure 1. Diagrama de comunicación entre objetos mediante buffers

VII. Definición de etapas

La aplicación consiste en un *pipeline* de procesamiento de imágenes astronómicas. Cada imagen pasará por tres etapas de procesamiento tal que al final del *pipeline* se clasifique la imagen como satisfaciendo o no alguna condición a definir. El programa procesará varias imágenes, una a la vez.

Las etapas del *pipeline* son:

1. Lectura de imagen RGB
2. Conversión a imagen en escala de grises
3. Binarización de imagen
4. Análisis de propiedad
5. Escritura de resultados

VII.A. Lectura de imágenes

Las imágenes estarán almacendas en binario con formato bmp. Habrá n imágenes y sus nombres tendrán el mismo prefijo seguido de un número correlativo. Es decir, los nombres serán imagen_1, imagen_2, ... ,

imagen_n.

Para este laboratorio se leerá una nueva imagen cuando la anterior haya finalizado el *pipeline* completo. Es decir, en el *pipeline* sólo habrá una imagen a la vez.

VII.B. Conversión de RGB a escala de grises

En una imagen RGB, cada pixel de la imagen está representado por tres números, que representan el componente de color rojo (Red), el de color verde (Green) y el de color azul (Blue). Para convertir una imagen a escala de grises, se utiliza la siguiente ecuación de luminiscencia:

$$Y = R * 0.3 + G * 0.59 + B * 0.11 \tag{1}$$

donde R, G, y B son los componentes rojos, verdes y azul, respectivamente.

VII.C. Binarización de una imagen

Para binarizar la imagen basta con definir un umbral, el cual define cuáles píxeles deben ser transformados a blanco y cuáles a negro, de la siguiente manera. Para cada pixel de la imagen, hacer

```
si el pixel > UMBRAL
    pixel = 1
sino
    pixel = 0
```

Los valores 0 y 1 no son representados por un bit sino son valores enteros (`int`) Al aplicar el algoritmo anterior, obtendremos la imagen binarizada.

VII.D. Clasificación

Se debe crear una función que concluya si la imagen es *nearly black* (casi negra). Esta característica es típica de muchas imágenes astronómicas donde una pequeña fuente puntual de luz está rodeada de vacio u oscuridad. Entonces, si el porcentaje de píxeles negros es mayor o igual a un cierto umbral la imagen es clasificada como *nearly black*.

El programa debe imprimir por pantalla la clasificación final de cada imagen y escribir en disco la imagen binarizada resultante.

VIII. Otros conceptos

Como ayuda, la manera de leer una imagen es la siguiente:

- Abrir el archivo con el uso de open(). Recuerde que este archivo debe contener la matriz (imagen).
- Leer la imagen con read().

Con los pasos previamente descritos, se puede leer la imagen y manipularla de tal forma que se pueda crear la matriz. Recuerde que la imagen tiene una estructura para poder obtener el valor de los pixeles, por lo que se pide investigar al respecto.

Para el caso de escribir la imagen resultante:

- Se debe escribir con write().
- Finalmente se debe cerrar el archivo con close().

Por último, el programa debe recibir la cantidad de imágenes a leer, el valor del umbral de binarización, el valor del umbral de negrura (representa un porcentaje) y una bandera que indica sí se desea mostrar o no la conclusión final hecha por la cuarta etapa. Por lo tanto, el formato getopt es:

- -c: cantidad de imágenes
- -u: UMBRAL para binarizar la imagen.

- -n: UMBRAL para clasificación
- -b: bandera que indica si se deben mostrar los resultados por pantalla, es decir, la conclusión obtenida al leer la imagen binarizada.

Por ejemplo, la salida por pantalla al analizar 3 imágenes sería lo siguiente:

```
$ ./pipeline -c 3 -u 50 -n 60 -b
|     image    |     nearly black    |
|--------------|---------------------|
|   imagen_1   |         yes         |
|   imagen_2   |         no          |
|   imagen_3   |         no          |
```

Donde sólo la imagen_1 fue clasificada como *nearly black*.

IX. Entregables

Debe entregarse un archivo comprimido que contenga al menos los siguientes archivos:

1. *Makefile*: archivo make para que compile los programas.
2. uno o mas archivos con el proyecto (*.cpp, *.hpp).
3. Clara documentación, es decir, entregar códigos debidamente comentados, en donde a lo menos se incluya le descripción de cada función, sus parámetros de entrada y salida.

El archivo comprimido debe llamarse: RUTESTUDIANTE1_RUTESTUDIANTE2.zip
Ejemplo: 19689333k_186593220.zip
NOTA: los laboratorios son en parejas, las cuales deben ser de la misma sección. De lo contrario no se revisarán laboratorios.
Se descuenta un punto por cada día de retraso.

X. Fecha de entrega

Viernes 31 de Agosto.