

Sistemas Operativos 1/2021

Laboratorio 1

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

Este laboratorio tiene como objetivo aplicar técnicas de programación imperativa mediante lenguaje C, como la recepción de parámetros mediante `getopt` y compilación mediante `Makefile` sobre sistema operativo Linux. Además, usaremos los llamados al sistema `open()`, `read()`, y `write()`. Finalmente, la aplicación de procesamiento de imágenes de este lab se usará en labs posteriores usando otros servicios del sistema operativo. Por lo tanto, con el desarrollo de este lab construiremos las bases para el desarrollo de algunos labs que siguen.

II. Objetivos Específicos

1. Conocer y usar las funcionalidades de `getopt()` como método de recepción de parámetros de entradas.
2. Validar los parámetros recibidos con `getopt()`
3. Construir funciones de lectura y escritura de archivos de imágenes binarios usando `open()`, `read()`, y `write()`.
4. Practicar técnicas de documentación de programas.
5. Conocer y practicar uso de `Makefile` para compilación de programas.
6. Realizar distintas técnicas de procesamiento de imágenes.

III. Conceptos

III.A. Getopt

La función `getopt()` pertenece a la biblioteca `<unistd.h>` y sirve para analizar argumentos ingresados por línea de comandos, asignando *options* de la forma:"-x", en donde "x" puede ser cualquier letra. A continuación, se detallan los parámetros y el uso de la función:

```
int getopt(int argc, char * constargv[],const char *optstring)
```

- **argc**: Argumento de main de tipo `int`, indica la cantidad de argumentos que se introdujeron por línea de comandos.
- **argv[]**: Arreglo de main de tipo `char * const`, almacena los argumentos que se introdujeron por línea de comandos.

- **optstring:** String en el que se indican los "option characters", es decir, las letras que se deben acompañar por signos "-". Si un option requiere un argumento, se debe indicar en optstring seguido de ":", en el caso contrario, se considerará que el option no requiere argumentos y por lo tanto el ingreso de argumentos es opcional.
- **option character:** Si en `argv[]` se incluyó el option "-a" y optstring contiene "a", entonces `getopt` retorna el char "a" y además se setea a la variable `optarg` para que apunte al argumento que acompaña al option character encontrado.

`Getopt` retorna distintos valores, que dependen del análisis de los argumentos ingresados por línea de comandos (contenidos en `argv[]`) y busca los option characters reconocidos en optstring. Por ejemplo, retorna `-1` cuando se terminan de leer los option characters; y retorna `-?` cuando se lee un option no reconocido en optstring. También este es un valor de retorno cuando un option requiere un argumento, pero en línea de comandos se ingresó sin argumento.

III.B. Input/Output y procuración de memoria

El sistema operativo provee a los procesos de varios servicios para acceder a diversos recursos del sistema. Por ejemplo, la procuración de memoria es un servicio que el SO presta a los procesos. Los procesos en si no procuran memoria, sino le solicitan al SO tal servicio.

En este lab usaremos dos familias de llamados al sistema: los servicios de lectura y escritura en un archivo, y los servicios de procuración de memoria dinámica.

I/O :

Los llamados al sistema que usaremos en este lab para leer y escribir imágenes en archivo son

1. `int open(const char *pathname, int flags)`
2. `ssize_t read(int fd, void *buf, size_t count)`
3. `ssize_t write(int fd, const void *buf, size_t count)`
4. `int close(int fd)`

`open()` se usa para abrir (y crear) un archivo llamado `pathname`. La variable `flags` indica el modo en que se abrirá el archivo.

`open()` retorna un entero (`int`) que es el descriptor del archivo abierto. En caso que hubo algún error, `open()` retorna `-1`. Note que todos los llamados retornan un número. Es responsabilidad del programador recibir esos números y tomar las acciones apropiadas en caso de error. Por ejemplo,

```
int f1;

if ((f1 = open("/home/estudiante1/lab1/image1.raw", O_RDONLY)) == -1) {
    printf("Error al abrir archivo\n");
    exit(-1);
}
```

El descriptor de archivos (variable entera) representa al archivo en el sistema operativo, y es el argumento que se debe usar en `read()` y `write()`. Luego, asumiendo que `A` es un puntero y que tiene memoria procurada, suficiente para almacenar `N` bytes, usamos

```
int nbytes;
if ((nbytes = read(f1, A, N)) != N) {
    printf("Error al leer archivo\n");
    exit(-1);
}
```

Nuevamente, chequeamos el retorno del llamado al sistema para verificar si hubo o no algún error.

Note que para `read()`, no es importante el tipo de datos que está leyendo. No se especifica si se leen enteros, caracteres, flotantes, etc. Simplemente se dice cuantos bytes se debe leer del archivo. Estos bytes son almacenados en la memoria apuntada por `A`, el cual necesariamente tiene un tipo de datos. Luego, los datos que están en el archivo deben estar justamente en el formato binario o de representación interna del tipo de dato del puntero que apunta a la memoria donde serán leídos. Por ejemplo, si se desea leer `N` enteros en el arreglo `A[]`, éste debe ser declarado como `int`, y se debe especificar el número total de bytes que esos `N` enteros representan:

```
int A[N];
int nbytes;
if ((nbytes = read(f1, A, N*sizeof(int))) != N) {
    printf("Error al leer archivo\n");
    exit(-1);
}
```

Pero si queremos leer `N` caracteres (`char`),

```
char A[N];
int nbytes;
if ((nbytes = read(f1, A, N*sizeof(char))) != N) {
    printf("Error al leer archivo\n");
    exit(-1);
}
```

En el primer caso, de enteros, si se abre el archivo de entrada, no se verían los números como se ven cuando éstos están en formato ASCII, pues están almacenados en formato de representación binaria de `int`. Esta misma lógica se aplica a `write()`.

Al finalizar el uso de un archivo, siempre ciérralo con `close()`.

MANEJO DE MEMORIA DINÁMICA :

El programa de manejo de imágenes debe estar preparado para trabajar con imágenes de distinto tamaño. Por lo tanto, para procurar y manejar el espacio de memoria necesitado se usa los llamado al sistema:

1. `void *malloc(size_t size)`
2. `void free(void *ptr)`

`malloc()` procura `size` bytes y retorna un puntero (tipo `void`) que apunta a dicha memoria. Si hubo algún error, retorna `NULL`. Entonces, al igual que los llamados de I/O, `malloc()` no se preocupa del tipo de dato que será almacenado en dicha memoria. Estos se pueden usar apropiadamente especificando conversiones de tipo al puntero. Por ejemplo, si se desea procurar memoria para `N` numeros flotantes de precisión simple, usamos

```
float *p;
if ((p = (float *) malloc(N*sizeof(float))) == NULL) {
    printf("No se pudo procurar memoria\n");
    exit(-1);
}
```

Siempre libere la memoria con `free()`, cuando ya no la necesite.

IV. Enunciado

El trabajo consiste en un pipeline de procesamiento de imágenes de 2 etapas: zoom-in y suavizado. Es decir, la imagen de entrada debe pasar por cada una de las etapas entregando un archivo resultante para cada etapa. Las etapas a realizar son:

IV.A. Zoom in

Consiste en una técnica de replicación de pixeles para toda imagen de $N \times N$, 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 $(N*r) \times (N*r)$. Para esta etapa se debe entregar una imagen llamada *imagen_i_zoom.raw*

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

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

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

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

IV.B. Filtro Suavizado

Técnica que sirve para suavizar los bordes de una 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. Se debe entregar como salida una imagen con el filtro aplicado de nombre *imagen_i_suavizado.raw*

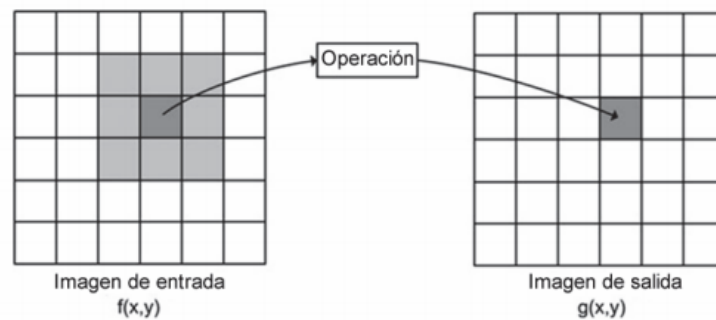


Figure 1. Ejemplo de obtención de pixel de salida



Figure 2. Ejemplo de imagen pasada por filtro de suavizado

V. Parámetros de Entrada

Para este trabajo se debe entregar al menos un archivo llamado `lab1.c` y un *Makefile* que entregue un archivo ejecutable llamado `lab1`. 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
- `-Z filename` : especifica el nombre de la imagen resultante del zoom-in
- `-S filename` : especifica el nombre de la imagen resultante del suavizado (a la resultante del zoom-in)
- `-M numero` : especifica el número de filas de la imagen
- `-N numero` : especifica el número de columnas de la imagen
- `-r factor` : factor de replicación para Zoom-in
- `-b`: bandera que indica si se entregan resultados por consola. En caso de que se ingrese este flag deberá mostrar: dimensiones de la imagen antes y después de aplicar zoom-in

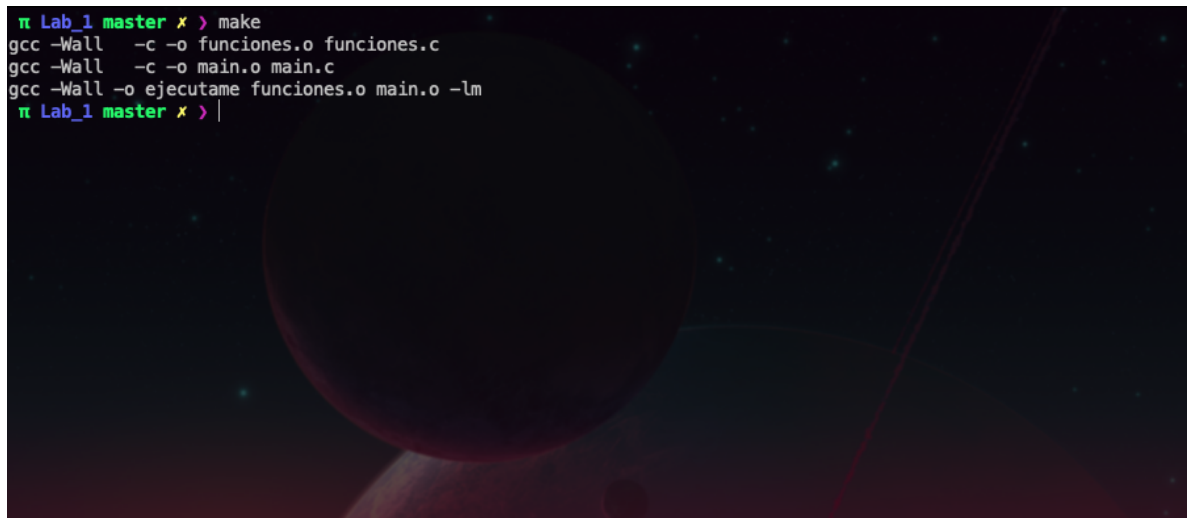
Por ejemplo:

```
$ ./lab1 -I imagen1.raw -Z imagen1Z.raw -S imagen1S.raw -M 512 -N 512 -r 2
```

VI. 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 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.



```
π Lab_1 master x > make
gcc -Wall -c -o funciones.o funciones.c
gcc -Wall -c -o main.o main.c
gcc -Wall -o ejecutame funciones.o main.o -lm
π Lab_1 master x > |
```

Figure 3. Ejemplo de funcionamiento de un Makefile

- **archivos .c y .h** Se debe tener como mínimo un archivo .c principal llamado `lab1.c` con el main del programa. 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.
- 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 deben ser de la misma sección. De lo contrario no se revisarán laboratorios.

VII. 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 = verimanageraw('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)
```

VIII. Fecha de entrega

Domingo 02 de Mayo 2021, hasta las 23:55 hrs.