

Sistemas Operativos 2/2021

Laboratorio 1

Profesores:

Cristóbal Acosta (cristobal.acosta@usach.cl)
Fernando Rannou (fernando.rannou@usach.cl)

Ayudantes:

Manuel I. Manríquez (manuel.manriquez.b@usach.cl)
Benjamín Muñoz (benjamin.munoz.t@usach.cl)
Camila Norambuena (camila.norambuena.v@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.

Finalmente, en este trabajo se aplicará un sistema de simulación de partículas que impactan a un material. El contexto de este trabajo servirá como base para futuros laboratorios, por lo que es importante la resolución de este en el curso de la asignatura.

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.
4. Practicar técnicas de documentación de programas.
5. Conocer y practicar uso de `Makefile` para compilación de programas.
6. Simular y apreciar el impacto de partículas en un material.

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 * const argv[], const char *optstring)
```

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

IV. Enunciado

En este laboratorio, se busca simular la energía depositada en un material por partículas de alta energía que lo impactan (ej: partículas en un satélite). El programa a desarrollar calculará y registrará la energía en Joules que cada partícula va depositando en algún punto del material.

Para facilitar el trabajo, se utilizará un modelo de una dimensión (1D). Entonces, se usará un arreglo para llevar registro de las energías acumuladas.

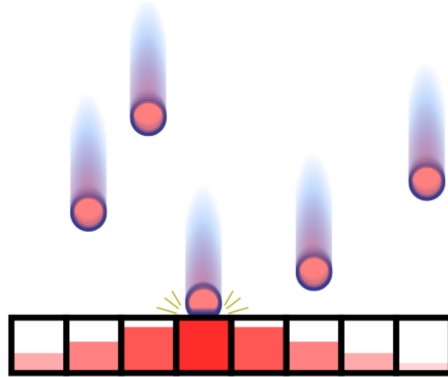


Figure 1. Acumulación de energías por partículas.

Por cada partícula simulada, se entrega su posición de impacto j y la energía potencial E_p que trae. La energía depositada en cada celda i del material es:

$$E_i = E_i + \frac{10^3 * E_p}{N \sqrt{|j - i| + 1}}, \forall i = 0, 1, \dots, N - 1 \quad (1)$$

Para este caso N representa el número de celdas que dispone el material. Cabe destacar que una partícula puede impactar en una posición mayor a N , y aunque impacte fuera del material, su energía igual puede afectar en alguna zona del material. En caso de que la energía depositada sea muy pequeña, esta no se registrará, para lo cual se utilizará un umbral que se calcula como $MIN_ENERGY = 10^{-3}/N$, por lo que se depositará energía solo en el caso que $E_i \geq MIN_ENERGY$.

IV.A. Archivos de entrada y salida

Un bombardeo de partículas se especifica en un archivo de entrada, el cual contiene el número de partículas, y para cada partícula, la posición de impacto y la energía. El siguiente es un ejemplo de un archivo de bombardeo:

```
6
4 81
8 10
10 100
7 35
11 12
5 8
```

La primera línea indica la cantidad de partículas y en las siguientes entrega la descripción de cada una de ellas. Es importante destacar que en este archivo **NO** se especifica el número de celdas, ya que este parámetro será ingresado por la terminal.

Para el archivo de salida, después de haber impactado todas las partículas, se debe escribir en la primera línea la posición del material con máxima energía y la cantidad acumulada. En las siguientes líneas, se debe indicar cada posición en orden con su respectiva energía acumulada (se deben escribir todas las celdas del material). Por ejemplo, haciendo uso de la entrada anterior, teniendo un material con 35 celdas, el resultado sería el siguiente, donde se logra ver en la celda 10 la máxima energía y luego la energía acumulada en todas las posiciones del material.

```
10 4732.568359
0 2537.520752
1 2745.225830
2 3027.479004
3 3456.707031
...
```

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`. La ejecución del programa tendrá los siguientes parámetros que deben ser procesados por `getopt()`:

- `-N` : número de celdas
- `-i filename` : archivo con el bombardeo
- `-o filename` : archivo de salida
- `-D`: bandera que indica si se entregan resultados por `stdout` (opcional).

Por ejemplo:

```
$ ./lab1 -N 5 -i input.txt -o output.txt -D
```

El programa debe imprimir por `stdout` el tiempo de ejecución, para lo cual se recomienda el uso de la función `clock()` u otra que estime pertinente. En caso que se utilice el flag `-D` para debug, se debe imprimir por `stdout` un gráfico simple y normalizado con las energías. Esto se hará mediante una función externa que debe ser enlazada y declarada en el código:

```
extern void niceprint(int N, float *Energy);
```

Donde los argumentos son el número de celdas y el arreglo con las energías. Para el caso anterior, utilizando un `N=10` y el flag `D` presente se obtiene:

```

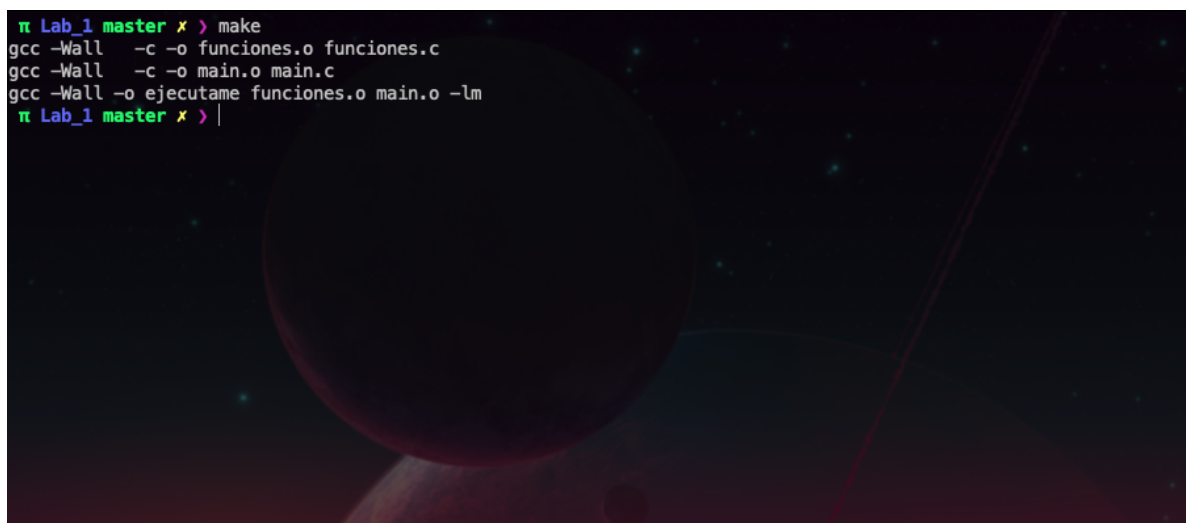
0 8881.3223 |oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
1 9608.2910 |oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
2 10596.1777|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
3 12098.4746|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
4 15066.8076|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
5 13584.3311|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
6 13256.4805|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
7 14255.6436|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
8 13870.8066|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
9 14156.3018|oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo

```

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. Una vez terminada la compilación debe crear el ejecutable llamado lab1.



```

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 2. 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, pueden ser de distintas secciones. Si es así, por favor avisar a los ayudantes Benjamín Muñoz y Camila Norambuena.

VII. Fecha de entrega

Domingo 31 de Octubre 2021, hasta las 23:55 hrs.