

PRÁCTICA 1

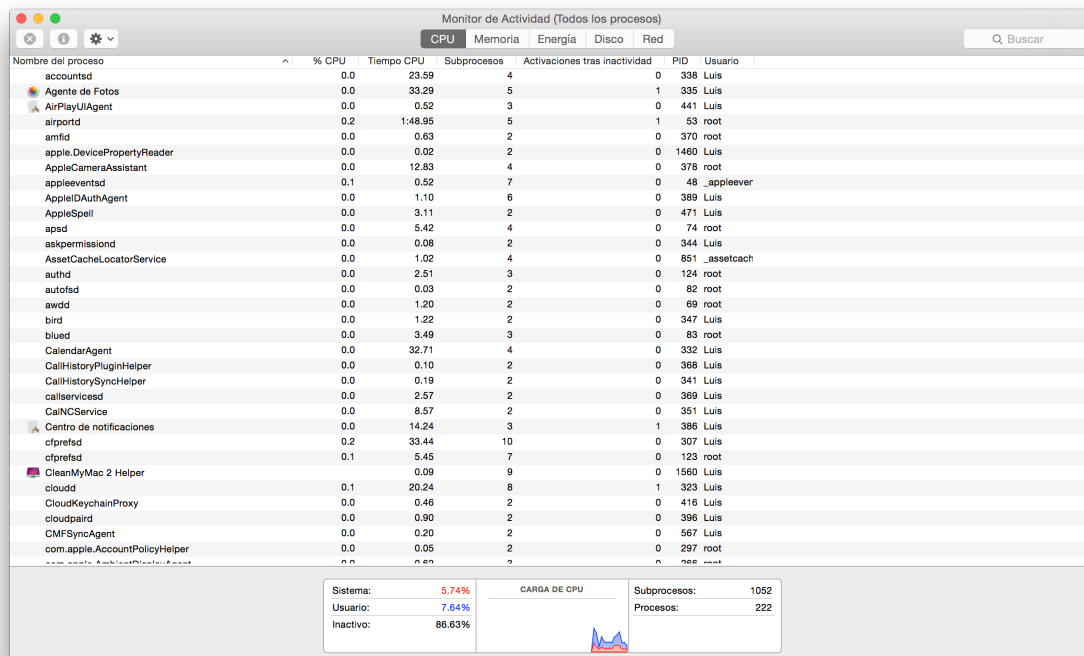
Chávez Soto Luis Armando

18 de Abril del 2015

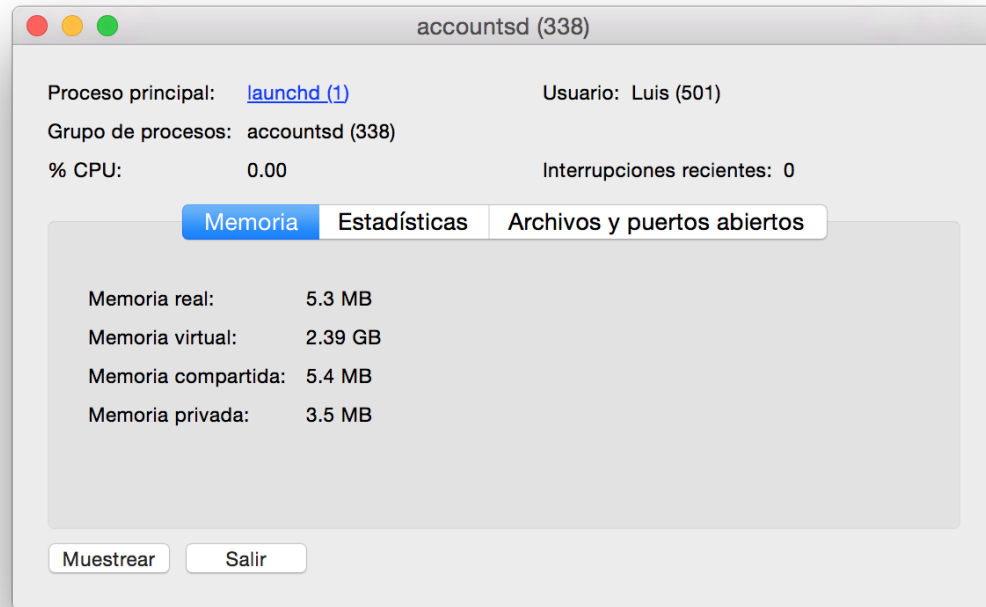
1.- Herramientas

El Monitor de Actividad (Process Explorer) de Mac OS X 10.10.3, cuenta con los registros de todas las actividades del CPU, memoria, energía, disco y red.

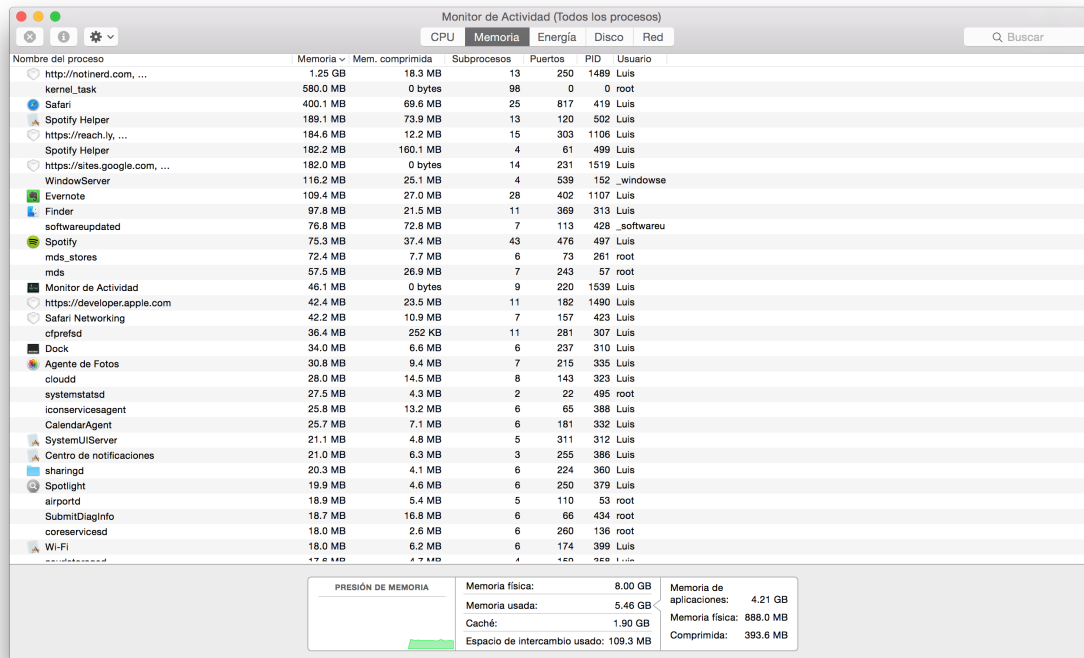
Dentro de la sección de CPU podemos decir que tiene un listado de los procesos que se ejecutan, seccionado a nivel de tipo de usuario, root, system, etc, los cuales podemos clasificar por el porcentaje de CPU asignado a cada proceso, el tiempo en el CPU, subprocesos, el número de activaciones desde alguna inactividad y por su puesto su identificador, el PID (Process Identifier).



Podemos ver que al seleccionar algún proceso, nos despliega una ventana con información aún más detallada como la memoria real y virtual empleada para la ejecución de dicho proceso.



En cuanto a la sección de Memoria podemos apreciar que los procesos se encuentran organizados mediante la memoria que se les es asignada, la memoria compartida, subprocesos asignados a dicho proceso, puertos, el PID y por su puesto el nivel de usuario en cual se encuentran trabajando.



Mientras que en la sección de ENERGÍA podemos apreciar las aplicaciones con sus procesos y subprocesos que consumen más energía dentro del computador, en la sección de DISCO observamos los bytes escritos en la unidad secundaria de memoria así como los leídos, el tipo de proceso de 32 o 64 bits y en cuanto a RED vemos los procesos que consumen recursos en red, los bytes de entrada y salida. El monitor de actividad también nos presenta una sección en donde podemos apreciar mejor estos datos en forma de gráficas.

2.- Señales en Linux y Windows

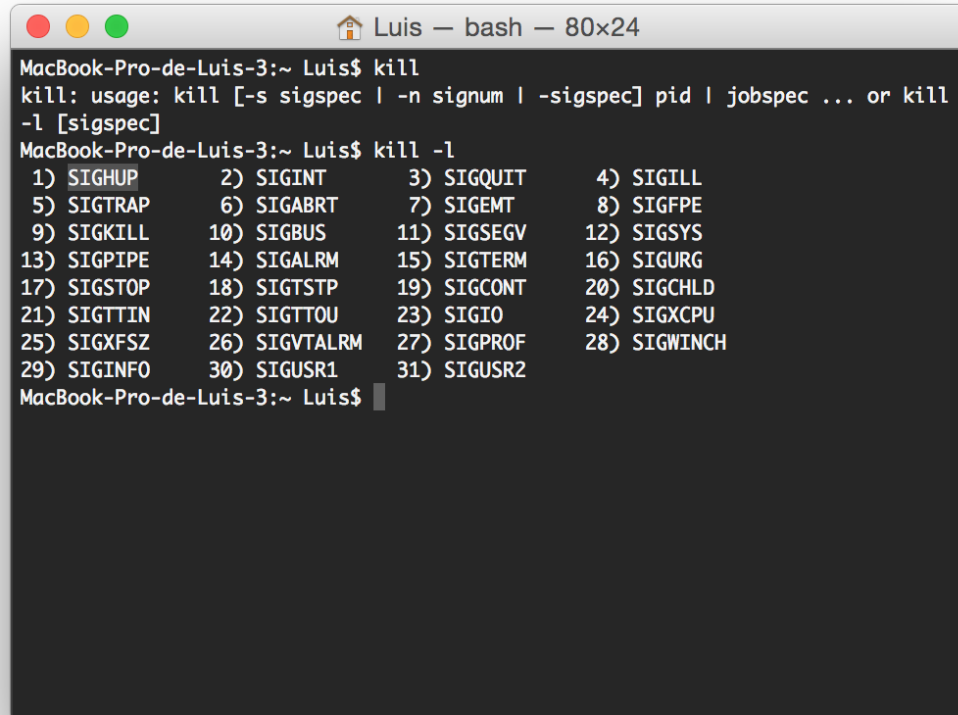
Primero que nada deberemos aclarar que una señal es una forma limitada de comunicación entre procesos empleada en Unix y otros sistemas operativos compatibles con POSIX, la notificación es asíncrona enviada a un proceso para notificarle sobre un evento. Estas señales o notificaciones se encuentran declarada sobre la librería `<signal.h>`. Por defecto el mensaje que se envía en una señal es la de terminación `SIGTERM`, el cual solicita al proceso limpiar su estado y salir. Hay que dejar en claro que el comando Kill no tiene que ver con terminar un proceso.

- **SIGHUP** - Hangup, al salir de la sesión se envía a los procesos en Background. Tratamiento por defecto: exit. Reprogramable.

- SIGINT - Interrupción, se genera al pulsar "Ctrl-C" durante la ejecución. Tratamiento por defecto: exit. Reprogramable.
- SIGQUIT - Terminar Pau.
- SIGILL - Instrucción ilegal.
- SIGTRAP - Trace/breakpoint trap.
- SIGABRT - PROCESO abortado.
- SIGFPE - Excepción de coma flotante – "Erroneous arithmetic operation" (SUS).
- SIGKILL - Destrucción inmediata del proceso. Tratamiento: exit. No reprogramable, no ignorable.
- SIGBUS - Error en el bus "Access to undefined portion of memory object" (SUS).
- SIGSEGV - segmentation violation. Salta con dirección de memoria ilegal. Tratamiento por defecto: exit + volcado de memoria. Reprogramable.
- SIGSYS - Error de argumentos al realizar una llamada al sistema llamada al sistema.
- SIGPIPE - Se genera al escribir sobre la pipe sin lector. Tratamiento por defecto: exit. Reprogramable.
- SIGALRM - Señal de alarma, salta al expirar el timer. Reprogramable.
- SIGTERM - Terminación. Tratamiento por defecto: exit. Reprogramable.
- SIGURG - datos importantes disponibles en socket
- SIGSTOP - Detiene el proceso. Se genera al pulsar "Ctrl-Z" durante la ejecución. No reprogramable, no ignorable.

Entre otras señales.

En cuanto a Microsoft Windows, opera bajo un comando taskkill para finalizar procesos.



```
MacBook-Pro-de-Luis-3:~ Luis$ kill
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill
-l [sigspec]
MacBook-Pro-de-Luis-3:~ Luis$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL
5) SIGTRAP     6) SIGABRT     7) SIGEMT      8) SIGFPE
9) SIGKILL     10) SIGBUS     11) SIGSEGV    12) SIGSYS
13) SIGPIPE    14) SIGALRM    15) SIGTERM    16) SIGURG
17) SIGSTOP    18) SIGTSTP    19) SIGCONT    20) SIGCHLD
21) SIGTTIN    22) SIGTTOU    23) SIGIO      24) SIGXCPU
25) SIGXFSZ    26) SIGVTALRM 27) SIGPROF    28) SIGWINCH
29) SIGINFO    30) SIGUSR1    31) SIGUSR2
MacBook-Pro-de-Luis-3:~ Luis$
```

3.- Wait()

Existe la llamada `wait()` para que un proceso padre espere la respuesta de su proceso hijo, o hasta que se produce una señal cuya acción es terminar el proceso actual o llamar a la función manejadora de la señal. Si un hijo ha salido cuando se produce la llamada (lo que se entiende por proceso "zombie"), la función vuelve inmediatamente. Todos los recursos del sistema reservados por el hijo son liberados.

El valor que tenga el parámetro PID tiene los siguientes significados:

- `<-1`: Esperar por cualquier hijo cuyo ID de grupo sea igual al valor del pid (con el signo cambiado).
- `-1`: Esperar por el primer hijo que termine.
- `0`: Esperar por cualquier hijo cuyo identificador de grupo sea igual al pid del proceso llamador (padre).

- 0: Esperar por los hijos cuyo pid es el indicado.

El estándar POSIX original estableció como indefinido el comportamiento de tratar SIGCHLD con SIG_IGN. Estándares posteriores, incluyendo SUSv2 y POSIX 1003.1-2001 especifican este comportamiento describiéndolo tan solo como una opción conforme con XSI. Linux no es conforme con el segundo de los dos puntos recién descritos: si se hace una llamada a `wait()` mientras SIGCHLD está siendo ignorada, la llamada se comporta como si SIGCHLD no estuviera siendo ignorada, es decir, se bloquea hasta que el siguiente hijo termina y luego devuelve el PID y el estado de ese hijo.

4.- Programas con fork()

(A) Programa 1 (C en Linux)

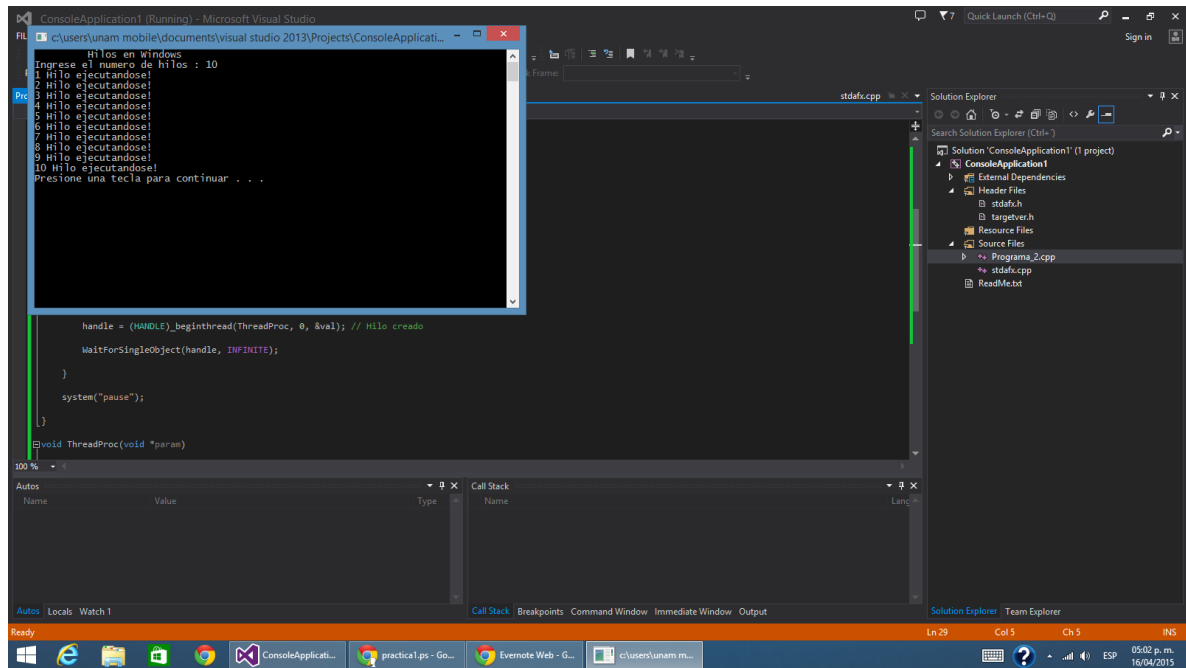
En resumen el programa solo estar el proceso padre que genera al proceso hijo, indicando su PID, después salvamos el PID del proceso hijo al ejecutar la función `fork()`, a continuación tenemos un método preventivo en caso de que falle la creación del nuevo proceso. A continuación ejecutamos un comando del sistema, `Process Status`, para ver los procesos activos mediante su PID, TTY, su tiempo de ejecución y su jerarquía. Inmediatamente los procesos comienzan a ejecutar el resto del código al mismo tiempo, por consiguiente separamos al proceso padre del hijo mediante una condicional, cada procesos ejecutara una serie de letras en pantalla y dormir un determinado tiempo en caso del proceso padre pondrá ,a, en caso del proceso hijo será b.

[illegible]

(B) Programa 2 (Microsoft Visual C++ Express 2010)

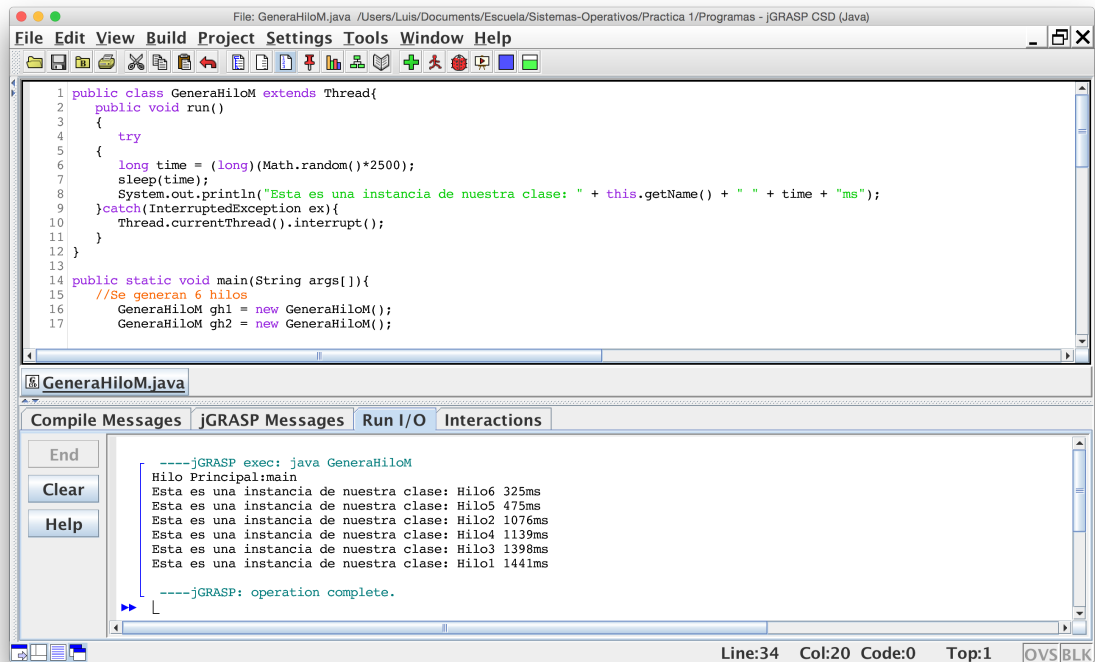
En resumen el programa solo genera hilos a petición del usuario. Y muestra el número de hilos que se están ejecutando, este espera a que termine y enseguida sigue

iterando el ciclo for.



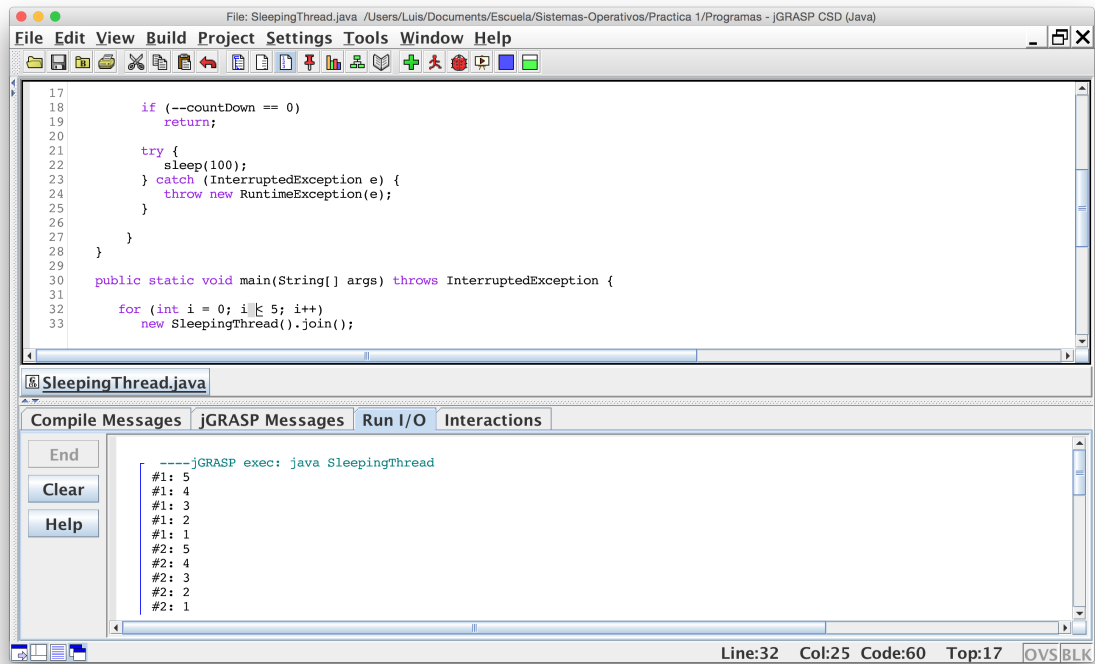
(B) Programa 3 (Java)

El código se divide en dos partes esenciales, la primera es una clase llamada *GeneraHiloM* que extiende de la clase *Thread*, dicha clase se encarga de crear un hilo con una ejecución en donde crea un tiempo aleatorio en el cual dormir el hilo e imprimir su nombre y el tiempo que durmió. La siguiente parte será el programa principal, en donde se crean 6 objetos instancias de la clase *GeneraHiloM*, se les asigna un nombre y a continuación se inician. Pero dependiendo el tiempo que se le fue asignado para dormir al hilo, será como terminen pues no terminaran de forma consecutiva.



(B) Programa 4 (Java)

Un programa muy similar al pasado, esta vez la clase es una extensión de Thread, donde existe una sobrecarga de los métodos nativos de la clase Thread. Se crean 5 hilos pero con em método join(), en donde este tiene la función de esperar la ejecución de un hilo para comenzar el siguiente. Cada hilo tiene como actividad una cuenta de 5 hasta 1, y duerme durante 100 ms. Al terminar cada hilo, por consiguiente se ejecuta en siguiente.



5.- Programas con fork()

```

1  //
2  //  programa_fork.c
3  //
4  //
5  //  Created by Luis Armando Chávez Soto on 17/04/15.
6  //
7  //
8
9  #include <stdio.h>
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include <sys/wait.h> /* waitpid */
13 #include <math.h>
14
15 #define clear() printf("\033[H\033[J")
16
17 void serieFibonacci(int iteraciones);
18 void raices();
19 void suma();
20
21 int main(){

```

```

23 clear();

25 // Identificador
pid_t pid;
27 int i;

29 // Verificamos proceso padre
printf("Proceso padre: %d\n\n",getppid());

31 // Ciclo para la creación del los procesos, asi mismo la asignación
de tareas para cada proceso
33 for (i = 0; i < 3; ++i) {
pid = fork();
35 if (pid) {

37     // Ponemos al padre a esperar cada proceso hijo
int status;
39 wait(&status);

41     continue;
} else if (pid == 0) {
43     //printf("Hijo (%d): %d\n", i + 1, getpid());
switch (i) {
45         case 0:
sleep(2);
47         clear();
printf("——> Raiz (1er Proceso PID: %d)\n",getpid());
49         raices();
break;

51         case 1:
sleep(3);
53         clear();
printf("——> Fibonacci (2do Proceso PID: %d)\n",
getpid());

55         int iteraciones = obtenerNumeroFibo();
serieFibonacci(iteraciones);
57         break;

59         case 2:
sleep(3);
clear();
61         printf("——> Suma de los elementos de un arreglo (3er
Proceso PID: %d)\n",getpid());
suma();
63         break;

65     }

67     break;
} else {

```

```

69         printf("Error Fork\n");
        exit(1);
71     }
    }
73 }

75 // Respectivas funciones para Fibonacci, Raices y suma de los elementos
    de un arreglo.

77 int obtenerNumeroFibo(){
    int num;
79     do{
        printf("Necesitamos que nos des un numero de iteraciones para la
        serie de Fibonacci (0-99): \n");
81         scanf("%d",&num);

83         if (num < 0 || num > 99) {
            //clear();
85             printf("\n\n *** Error con numero de iteraciones deseadas ***
            \n\n\n");
        }

87     }while (num < 0 || num > 99);

89     return num;
91 }

93 void serieFibonacci(int iteraciones){

95     int i;
    int prim = 0;
97     int segu = 1;
    int sig;

99     printf("Serie: \n");

101     for (i = 0; i < iteraciones; i++) {
103         if (i <= 1) {
            sig = i;
105         }else{
            sig = prim + segu;
107             prim = segu;
            segu = sig;
109         }
        printf("%d - ",sig);
111     }

113     printf("\n\n\n");

```

```

115 }
117 void raices(){
119     int tamano;
121     do {
123         printf("Tamaño del arreglo (1 - 20): \n");
125         scanf("%d",&tamano);
127         if (tamano < 1 || tamano > 20) {
129             //clear();
131             printf("\n\n *** Error con el tamaño del arreglo deseado ***
133             \n\n\n");
135         }
137     } while (tamano < 1 || tamano > 20);
139
141     int arreglo [tamano];
143     int i;
145     printf("\nLlene el arreglo con valores entre 0 y 99\n");
147
149     int dato;
151     for (i = 0; i < tamano ; i++) {
153         do {
155             printf("\nArreglo[%d] - ",i);
157             scanf("%d",&dato);
159             if (dato < 0 || dato > 99) {
161                 //clear();
162                 printf("\n\n *** Error con el dato, fuera de rango ***\n\
163                 n\n");
164             }else{
165                 arreglo[i] = dato;
166             }
167         } while (dato < 0 || dato > 99);
168     }
169
170     printf("\nArreglo lleno.\n");
171
172     printf("Iteracion \tValor \tRaiz\n");
173     for (i = 0; i < tamano; i++) {
174         if (arreglo [i] < 0) {
175             printf("Arreglo[%d] \t%d \tMath ERROR",i , arreglo[i]);
176         }else{

```

```

163         printf("Arreglo[%d] \t %d \t %f", i, arreglo[i], sqrt(arreglo[i])
164     );
165     }
166     printf("\n");
167 }
168
169 printf("\n\n\n");
170
171 }
172
173 void suma(){
174     int tamanio;
175     do {
176         printf("Tamaño del arreglo (1 - 20): \n");
177         scanf("%d",&tamanio);
178
179         if (tamanio < 1 || tamanio > 20) {
180             //clear();
181             printf("\n\n *** Error con el tamaño del arreglo deseado ***
182 \n\n\n");
183         }
184
185     } while (tamanio < 1 || tamanio > 20);
186
187     int arreglo [tamanio];
188     int i;
189
190     printf("\nLlene el arreglo con valores entre 0 y 99\n");
191
192     int dato;
193     for (i = 0; i < tamanio ; i++) {
194
195         do {
196             printf("\nArreglo[%d] - ", i);
197             scanf("%d",&dato);
198
199             if (dato < 0 || dato > 99) {
200                 //clear();
201                 printf("\n\n *** Error con el dato, fuera de rango ***\n\
202 n\n");
203             }else{
204                 arreglo[i] = dato;
205             }
206
207         } while (dato < 0 || dato > 99);

```

```

    }
209     printf("\nArreglo lleno.\n");

211     int dat = 0;

213     for (i = 0; i < tamaño; i++) {
        dat = dat + arreglo[i];
215     }

217     printf("Sumatoria de los elementos del arreglo: %d",dat);

219     printf("\n\n\n");
}

```

```

//
2 //  programa_thread.c
//
4 //
//  Created by Luis Armando Chávez Soto on 18/04/15.
6 //
//
8
#include <pthread.h>
10 #include <stdio.h>
#include <stdlib.h>
12 #include <time.h>
#include <stdint.h> /* 64 bits */
14
#define clear() printf("\033[H\033[J")
16

18 void *helloWorld(void *numHilo)
{
20     int i,hilo;

22     hilo = (int) numHilo;

24     for (i=0; i<5; i++){
        printf("Hola Mundo [%d]\n",hilo);
26     }
    pthread_exit((void*) numHilo);
28 }

30 int main()
{
32     // Limpiamos Pantalla
    clear();
34

```

```

36 // Arreglo de hilos
pthread_t thread[10];
pthread_attr_t attr;

38

40 // Contador y rc
int rc, i;
void *status;

42

44 // Inicializador y tipo de Hilo
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

46

48 // Ciclo para crear hilo con su respectiva función y paso de argumento
for(i=0;i<10;i++)
{
50
52     int *numHilo;
    numHilo = i+1;
    rc = pthread_create(&thread[i], &attr, helloWorld, (void *)
numHilo);
54     if (rc){
56         printf("ERROR: pthread_create() -> %d\n", rc);
        exit(EXIT_FAILURE);
58     }

60 // Destrucción del hilo
pthread_attr_destroy(&attr);

62

64 // Estado final del hilo
for(i=0;i<10;i++)
{
66     rc = pthread_join(thread[i], &status);

68     if (rc){
70         printf("ERROR: pthread_join() -> %d\n", rc);
        exit(EXIT_FAILURE);
72     }

74 pthread_exit(NULL);
}

```

Bibliografía

- Señales (Wikipedia) - [http://es.wikipedia.org/wiki/Señal_\(informática\)](http://es.wikipedia.org/wiki/Señal_(informática))

- Signals and Traps (TutorialPoints) - <http://www.tutorialspoint.com/unix/unix-signals-traps.htm>
- Kill (Wikipedia) - <http://es.wikipedia.org/wiki/Kill>
- TaskKill (Microsoft) - <http://www.microsoft.com/resources/documentation/windows/xp/all/producers/taskkill.mspx?mfr=true>
- Ps, Unix (Wikipedia) - [http://en.wikipedia.org/wiki/Ps_\(Unix\)](http://en.wikipedia.org/wiki/Ps_(Unix))
- Windows Data Types (Microsoft) - <https://msdn.microsoft.com/en-us/library/windows/desktop/a>
- _BeginThread Doc (Microsoft) - <https://msdn.microsoft.com/es-mx/library/kdzttddb.aspx>
- WaitForSingleObject Doc (Microsoft) - <https://msdn.microsoft.com/en-us/library/windows/desktop/>
- Join (Oracle) - <https://docs.oracle.com/javase/tutorial/essential/concurrency/join.html>