UA

ProgramaciónConcurrente

DanielAsensiRochDNI : **48776120C**

7 de octubre de 2022

Índice

1. Procesos en Unix/C

1.1. Ejercicio 1

Comenta qué se espera que ocurra en cada porción de código y la salida. ¿Qué crees que hace WEXITSTATUS ? ¿Es una función o una macro del preprocesador de C?

El funcionamiento del programa se basa en la creación de 5 procesos, además de un proceso padre que es el que los genera, este proceso padre espera que sus hijos terminen y imprime información sobre estos mediante la siguiente linea:

```
printf("Proceso(pid=%d) con id = %x terminado y status = %d \n", pid,
```

Además los procesos hijos se imprimen 50 veces con su información:

```
printf("Proceso(id=%d): i = %d, I = %d\n", id, i, I++);
```

La sentencia WEXITSTATUS es la macro encargada de evaluar la variable de salida, es decir, devuelve el codigo de salida especificado por el proceso hijo.

1.2. Ejercicio 2 y 3

Para evitar la redundancia en la práctica he unificado los ejercicios 2 y 3 en el siguiente código.

```c
/* PROCESOS */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Cambiamos la cantidad de procesos
#define NUM_PROCESOS 2


//Cambiamos lo que debe de imprimir cada uno de los hijos ejercicio 3
void codigo_del_proceso(int id)
{
    printf("Proceso(id=%d): %d \n", id,id);
}


int main()
{
    int p;
    int id[NUM_PROCESOS] = {1, 2};
    int pid;
    int salida;

    for (p = 0; p < NUM_PROCESOS; p++)
    {
        pid = fork();
        if (pid == -1)
        {
            perror("Error al crear un proceso: ");
            exit(-1);
        }
        else if (pid == 0){ // Codigo del hijo
            //ejercicio 3
            codigo_del_proceso(id[p]);
            //Nos salimos del proceso
            exit(id[p]);
        }
    }

    // Codigo del padre este no cambia
```

```
    for (p = 0; p < NUM_PROCESOS; p++)
    {
        pid = wait(&salida);
        printf("Proceso(pid=%d) con id = %x terminado y status = %d \n", pid,
               salida >> 8, WEXITSTATUS(salida));
    }
    return 0;
}
```

1.3. Ejercicio 4

Los cambios realizados en el programa proporcionado por el profesor han sido descritos en el código

```
/* PROCESOS */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

// Cambiamos el numero de procesos
#define NUM_PROCESOS 3
#define VECES_IMPRIME 5

// Cambiamos el valor pasado por parametro
void codigo_del_proceso(char c)
{
    for (int i = 0; i < VECES_IMPRIME; i++)
    {
        printf("%c", c);
    }
    printf("\n");
}

int main()
{
    int p;
    //Cambiamos lo que deben de imprimir los procesos
    int id[NUM_PROCESOS] = {'A', 'B', 'C'};
    int pid;
    int salida;

    for (p = 0; p < NUM_PROCESOS; p++)
    {
        pid = fork();
        if (pid == -1)
        {
            perror("Error al crear un proceso: ");
            exit(-1);
        }
        else if (pid == 0)
        { // Codigo del hijo
            codigo_del_proceso(id[p]);
            exit(id[p]);
        }
    }

    // Codigo del padre
    for (p = 0; p < NUM_PROCESOS; p++)
    {
        pid = wait(&salida);
        printf("Proceso(pid=%d) con id = %x terminado y status = %d \n", pid,
               salida >> 8, WEXITSTATUS(salida));
    }
    return 0;
}
```

2. Hilos POSIX

2.1. Ejercicio 1

El programa genera 5 hilos, mediante la siguiente sentencia:

```
pthread_create( &hilos[h], NULL, codigo_del_hilo, &id[h]);
```

una vez creado el hilo ejecuta una función, en este caso imprimir 50 veces su información, mediante el bucle siguiente:

```
void *codigo_del_hilo (void *id){
   int i;
   for( i = 0; i < 50; i++)
      printf("Hilo %d: i = %d, I = %d\n", *(int *)id, i, I++);
   pthread_exit (id);
}
```

Además imprime la cantidad de iteraciones que lleva mediante la variable I, la cual lleva el conteo y debería imprimir el valor 250 ya que 50 * 5 = 250.

Una vez realizada la función el segundo bucle espera a que terminen los hilos creados y imprime su información y muestra la salida correspondiente en este caso la siguiente:

```
printf ("Hilo %d terminado\n", *salida);
```

La gran diferencia entre los procesos y los hilos es que los procesos llaman a una función, y los hilos se crean con un una función u comportamiento específico que realizará el propio hilo.

2.2. Ejercicio 2

Para este ejercicio he modificado el código proporcionado por el profesor, quedando el mismo de la siguiente manera:

```
#include <pthread.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define NUM_HILOS 3

typedef struct
{
    char caracter;
    int veces;
} estructura;

void imprimirValores(estructura e)
{

    for (int i = 0; i < e.veces; i++)
    {
        printf("%c", e.caracter);
    }
    printf("\n");
}

void *codigo_del_hilo(void *id)
{

    estructura e;

```

```
29      switch (*(int *)id)
30      {
31      case 1:
32          e.caracter = 'A';
33          e.veces = 50; // Hilo 1
34          break;
35      case 2:
36          e.caracter = 'B';
37          e.veces = 100; // Hilo 2
38          break;
39      case 3:
40          e.caracter = 'C';
41          e.veces = 150; // Hilo 3
42          break;
43      }
44      imprimirValores(e);
45      pthread_exit(id);
46  }
47
48  int main()
49  {
50      int h;
51      pthread_t hilos[NUM_HILOS];
52      int id[NUM_HILOS] = {1, 2, 3};
53      int error;
54      int *salida;
55
56      for (h = 0; h < NUM_HILOS; h++)
57      {
58          error = pthread_create(&hilos[h], NULL, codigo_del_hilo, &id[h]);
59          if (error)
60          {
61              fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
62              exit(-1);
63          }
64      }
65      for (h = 0; h < NUM_HILOS; h++)
66      {
67          error = pthread_join(hilos[h], (void **)&salida);
68          if (error)
69          {
70              fprintf(stderr, "Error: %d: %s\n", error, strerror(error));
71          }
72          else
73          {
74              printf("Hilo %d terminado\n", *salida);
75          }
76      }
77  }
```