

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 1</p>

INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Laboratorio de Programación de Sistemas				
TÍTULO DE LA PRÁCTICA:	Procesos y Señales				
NÚMERO DE PRÁCTICA:	Guía 8	AÑO LECTIVO:	2023	NRO. SEMESTRE:	V
FECHA DE PRESENTACIÓN	13/07/2023	HORA DE PRESENTACIÓN	14		
INTEGRANTE (s) <i>Calcina Puma Esteven Antonio</i> <i>Galvez Quilla Henry Isaias</i> <i>Paredes Quispe Jose Andre</i>				NOTA (0-20)	
DOCENTE(s): Mg. Edith Giovanna Cano Mamani					

RESULTADOS Y PRUEBAS
<p>I. EJERCICIOS RESUELTOS:</p> <p>EJERCICIO 1.</p> <p>Crear un programa en C++ donde utilizando una llamada a sistema fork() se puedan crear tres hijos.</p> <pre> 1 #include <stdio.h> 1 #include<sys/types.h> 2 #include<unistd.h> 3 int main(){ 4 fork(); 5 fork(); 6 printf("Hola Mundo\n"); 7 return 0; 8 }</pre> <p>Aquí tenemos un proceso padre y tres procesos hijos.</p>

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 2</p>

```
stv@stv-Satellite-P55t-A:~/PS/lab08$ gcc ej1.c -o ej1c
stv@stv-Satellite-P55t-A:~/PS/lab08$ ./ej1c
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
stv@stv-Satellite-P55t-A:~/PS/lab08$
```

EJERCICIO 2.

Crear un programa en C++ donde utilizando la biblioteca csignal se pueda imprimir un mensaje de “hola” utilizando la señal SIGABRT

```
17 #include <iostream>
16 #include <csignal>
15
14 void signalHandler(int signal) {
13     if (signal == SIGABRT) {
12         std::cout << "Hola desde la señal SIGABRT" << std::endl;
11     }
10 }
9
8 int main() {
7     // Asignar el manejador de señales
6     std::signal(SIGABRT, signalHandler);
5
4     // Generar una señal SIGABRT
3     std::abort();
2
1     return 0;
0 }
```

```
usuario@usuario-ThinkCentre-M93p:~$ ./ej
Hola desde la señal SIGABRT
Abortado ('core' generado)
usuario@usuario-ThinkCentre-M93p:~$
```

EJERCICIO 3.

Modifica el código10.c para que mediante la utilización de la función “sleep()”, la frecuencia a la que el proceso hijo escribe en los ficheros sea menor que la del proceso padre. Es decir que realice menos escrituras por unidad de tiempo.

```

1 // codigo10.c modificado
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <fcntl.h>
6
7 int main() {
8     int i;
9     int fd1, fd2;
10    const char string1[10]= "*****";
11    const char string2[10]= "-----";
12    pid_t rf;
13    fd1 = creat("ficheroA", 0666);
14    fd2 = creat("ficheroB", 0666);
15    rf = fork(); /* Crea el proceso hijo */
16    switch (rf) {
17        case -1:
18            printf("\nNo he podido crear el proceso hijo");
19            break;
20        case 0: /* Ejecuta el proceso hijo */
21            for (i = 0; i < 10; i++) { /* Escribe ----- */
22                write(fd1, string2, sizeof(string2));
23                write(fd2, string2, sizeof(string2));
24                sleep(5); /* Abandonamos voluntariamente el procesador */
25            }
26            break;
27        default: /* Ejecuta el proceso padre */
28            for (i = 0; i < 10; i++) { /* Escribe ***** */
29                write(fd1, string1, sizeof(string1));
30                write(fd2, string1, sizeof(string1));
31                sleep(1); /* Abandonamos voluntariamente el procesador */
32            }
33    }
34    printf("\nFinal de ejecucion de %d \n", getpid());
35    exit(0);
36 }

```

```

(andr3@kali)-[~/Documents/PS]
$ gcc -Wall -o codigo10 codigo10.c

```

```

(andr3@kali)-[~/Documents/PS]
$ ./codigo10

```

Final de ejecucion de 1505

```

(andr3@kali)-[~/Documents/PS]
$ ps
  PID TTY          TIME CMD
 1066 pts/0    00:00:00 bash
 1506 pts/0    00:00:00 codigo10
 1514 pts/0    00:00:00 ps

```

```

(andr3@kali)-[~/Documents/PS]
$
Final de ejecucion de 1506

```

```

(andr3@kali)-[~/Documents/PS]
$ cat ficheroA
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

(andr3@kali)-[~/Documents/PS]
$ cat ficheroA
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

(andr3@kali)-[~/Documents/PS]
$ cat ficheroA
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

EJERCICIO 4.

Crear un programa en C++ o C que permita comprobar que el proceso nieto se ejecuta antes que el proceso hijo y el proceso hijo se ejecuta antes que el proceso padre

```
17 #include<iostream>
16 #include <stdio.h>
15 #include <sys/types.h>
14 #include <sys/wait.h>
13 #include <unistd.h>
12 using namespace std;
11
10 int main(){
9     pid_t hijo =fork();
8     if(hijo==0){
7         pid_t nieto = fork();
6         if(nieto==0){
5             cout<<"Proceso nieto"<<endl;
4         }
3         else{
2             wait(NULL);
1             cout<<"termino Proceso nieto"<<endl;
8             cout<<"Proceso hijo"<<endl;
1         }
2     }
3     else{
4         wait(NULL);
5         cout<<"termino proceso hijo"<<endl;
6         cout<<"El proceso padre"<<endl;
7     }
8     return 0;
9 }
```

```
stv@stv--pc:[~/PS/lab08]$ ./ej4
Proceso nieto
termino Proceso nieto
Proceso hijo
termino proceso hijo
El proceso padre
```

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 5

II. PRUEBAS

¿Con que valores comprobaste que tu práctica estuviera correcta? ¿Qué resultado esperabas obtener para cada valor de entrada? ¿Qué valor o comportamiento obtuviste para cada valor de entrada?

Cada ejercicio se probó con los ficheros generados y se comprobaba con las indicaciones de cada uno de los ejercicios

III. CUESTIONARIO:

1) ¿Cómo usted definiría la bomba fork?

La "bomba fork" es una técnica utilizada en programación para generar una gran cantidad de procesos o bifurcaciones de procesos en un corto período de tiempo. Esta técnica puede ser utilizada para crear una carga excesiva en el sistema, consumir recursos o causar una denegación de servicio.

2) ¿Para qué sirve la señal SIGXCPU?

La señal SIGXCPU es una señal de tiempo de CPU (CPU time limit exceeded) que se envía a un proceso cuando ha excedido su límite de tiempo de CPU asignado. En sistemas Unix, los procesos pueden estar sujetos a límites de tiempo de CPU para evitar que utilicen recursos del sistema de manera excesiva o maliciosa.

CONCLUSIONES

Procesos:

Los procesos son entidades fundamentales en sistemas operativos que permiten la ejecución de programas de manera concurrente e independiente.

En C y C++, la creación de procesos se puede lograr mediante llamadas al sistema como fork() en sistemas Unix.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 6</p>

Los procesos creados mediante `fork()` son copias exactas del proceso original, pero tienen diferentes identificadores de proceso (PID).

Cada proceso tiene su propio espacio de memoria y recursos, lo que permite la ejecución paralela de programas y evita conflictos de memoria.

La comunicación entre procesos puede lograrse a través de mecanismos como pipes, colas de mensajes o memoria compartida.

Señales:

Las señales son mecanismos que permiten a los procesos recibir notificaciones asíncronas sobre eventos específicos.

En C y C++, se pueden manejar señales usando la función `signal()` o estableciendo un manejador de señales. Algunas señales comunes incluyen SIGINT (generada por Ctrl+C en la terminal) y SIGTERM (señal de terminación).

La recepción de una señal puede alterar el flujo de ejecución del programa, como finalizarlo o realizar ciertas acciones antes de finalizar.

El manejo adecuado de señales es importante para evitar comportamientos inesperados o para liberar recursos adecuadamente antes de finalizar un programa.

Consideraciones adicionales:

Al trabajar con procesos y señales, es fundamental garantizar la sincronización y la gestión adecuada de los recursos compartidos para evitar condiciones de carrera y bloqueos.

Es importante manejar errores y excepciones correctamente al trabajar con procesos y señales para asegurar un comportamiento robusto y predecible del programa.

Al programar en entornos multiplataforma, ten en cuenta que las llamadas al sistema y el manejo de señales pueden variar entre sistemas operativos. Utiliza bibliotecas y abstracciones adecuadas para garantizar la portabilidad del código.

METODOLOGÍA DE TRABAJO

Se siguieron los siguientes pasos:

1. *Lectura del enunciado.*
2. *Análisis del ejercicio.*
3. *Propuesta de solución.*
4. *Diseño de la solución.*
5. *Propuesta de casos de prueba.*
6. *Validación del diseño con los casos de prueba.*
7. *Implementación de la solución.*
8. *Pruebas unitarias.*
9. *Validación de la solución.*
10. *Refactorización en funciones atómicas.*

REFERENCIAS Y BIBLIOGRAFÍA

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 7</p>

- [1] P.J. Deitel and H.M. Deitel, "Cómo Programar en C++", México, Ed. Pearson Educación, 2009
[2] B. Stroustrup, "El Lenguaje de Programación C++", Madrid, Adisson Pearson Educación, 2002
[3] B. Eckel, "Thinking in C++", Prentice Hall, 2000