



**UNIVERSIDAD NACIONAL
SAN AGUSTIN DE AREQUIPA**



Facultad de Ingeniería, Producción y Servicios

Escuela Profesional de Ciencia de la Computación

Laboratorio 2

Presentado por:

Parizaca Mozo, Paul Antony

CUI:

20210686

Curso:

Sistemas Operativos - Grupo A

Github:

<https://github.com/PaulParizacaMozo/SistemasOperativos>

Docente:

Yessenia Yari Ramos

Arequipa, Perú

2023

Todos los códigos en github:

<https://github.com/PaulParizacaMozo/SistemasOperativos/tree/main/Laboratorio3%20Analisis%20Codigo>

Ejercicio 1

-Código comentado

```
21 //Ejercicio 1
22 #include <stdio.h>
23 #include <sys/types.h>
24 #include <unistd.h>
25
26 int main(void) {
27     pid_t pid; // Declaración de una variable para almacenar el ID del proceso
28     int var = 0; // Declaración e inicialización de una variable entera "var"
29
30     printf("PID antes de fork(): %d\n", (int) getpid()); // Imprimir el ID del proceso antes de crear un proceso hijo format:
31     if ((pid = fork()) > 0) { // Crear un proceso hijo con fork() y comprueba si se está ejecutando en el proceso padre
32         printf("PID del padre: %d\n", (int) getpid()); // Imprimir el ID del proceso padre format:
33         var++; // Incrementar la variable "var" en el proceso padre
34     } else {
35         if (pid == 0) // Comprueba si se está ejecutando en el proceso hijo
36             printf("PID del hijo: %d\n", (int) getpid()); // Imprimir el ID del proceso hijo además la variable var no es modificada format:
37         else
38             printf("Error al hacer fork()\n"); // Mensaje de error al crear el proceso hijo format:
39     }
40     printf("Proceso [%d] -> var = %d\n", (int) getpid(), var); // Imprime el ID del proceso actual y el valor de "var" format:
41     getchar();
42 }
```

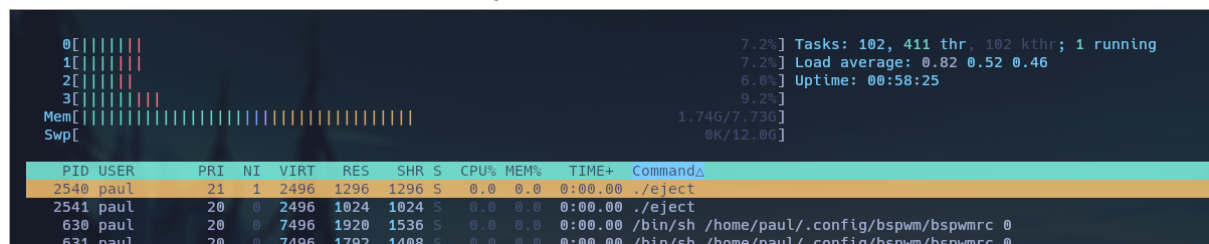
Lo que hace el programa es crear una variable `pid_t` para almacenar el ID de un proceso. Luego creamos un proceso hijo usando `fork()` y tenemos 3 casos si es mayor a 0 se está ejecutando el proceso padre, si es igual a 0 se está ejecutando el proceso hijo, caso contrario hay un error y no se pudo ejecutar de forma correcta el `fork()`. Y se irá imprimiendo cual caso sucede haciendo uso de la función `getpid()` que retorna el identificador(PID) del proceso actual.

-Ejecución del código

```
> gcc -o eject main.c && ./eject
PID antes de fork(): 2540
PID del padre: 2540
Proceso [2540] -> var = 1
PID del hijo: 2541
Proceso [2541] -> var = 0
```

solicitamos ingresar un enter para finalizar el programa y así poder observar los procesos creados.

si vemos los procesos que están en ejecución en ese momento con `htop`



The screenshot shows the htop interface. At the top, system statistics are displayed: Tasks: 102, 411 thr, 102 kthr; 1 running; Load average: 0.82 0.52 0.46; Uptime: 00:58:25; Mem: 1.746/7.736; Swp: 0K/12.06. Below this, a table of running processes is shown.

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
2540	paul	21	1	2496	1296	1296	S	0.0	0.0	0:00.00	./eject
2541	paul	20	0	2496	1024	1024	S	0.0	0.0	0:00.00	./eject
630	paul	20	0	7496	1920	1536	S	0.0	0.0	0:00.00	/bin/sh /home/paul/.config/bspwm/bspwmrc 0
631	paul	20	0	7496	1792	1408	S	0.0	0.0	0:00.00	/bin/sh /home/paul/.config/bspwm/bspwmrc 0

podemos observar que se encuentran los dos procesos creados por el `./eject` que es el ejecutable de nuestro programa.

Ejercicio 2

Código Implementado

```
15 //Ejercicio2
14 #include <stdio.h>
13 #include <sys/types.h>
12 #include <unistd.h>
11
10 int main(void) {
9     pid_t pid; //Declaracion de una variable tipo pid_t que almacenara el Identificador del proceso
8     int i, n= 4; // Variable i sera el contador del bucle, y la variable n la cantidad maxima del bucle
7     for (i=0; i<n; i++){
6         if ( (pid = fork()) < 0 ){//Se llama al proceso fork() las n veces ejecutandose como proceso hijo o padre
5             break;} //Si el proceso es menor que 0 se detiene el bucle.
4     }
3     printf ("Proceso: %d / Padre: %d\n", (int) getpid(), (int) getppid()); // Imprime el Id del proceso actual y del padre format:
2     //El printf se ejecutara por cada proceso creado es decir 2 a la n veces.
1     getchar();
16 }
```

```
> gcc -o eject main.c && ./eject
Proceso: 3138 / Padre: 1992
Proceso: 3140 / Padre: 3138
Proceso: 3139 / Padre: 3138
Proceso: 3147 / Padre: 3140
Proceso: 3143 / Padre: 3138
Proceso: 3141 / Padre: 3138
Proceso: 3145 / Padre: 3139
Proceso: 3148 / Padre: 3139
Proceso: 3144 / Padre: 3140
Proceso: 3149 / Padre: 3145
Proceso: 3146 / Padre: 3141
Proceso: 3150 / Padre: 3144
Proceso: 3142 / Padre: 3139
Proceso: 3152 / Padre: 3142
Proceso: 3151 / Padre: 3142
Proceso: 3153 / Padre: 3151
|
```

0[|||||]

1[|||||]

2[|||||]

3[|||||]

Mem[|||||]

Swp[|||||]

9.5%

9.2%

7.9%

7.8%

1.87G/7.73G

0K/12.0G

Tasks: 119, 439 thr, 101 kthr; 1 running

Load average: 0.09 0.35 0.47

Uptime: 01:12:02

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3138	paul	20	0	2496	1228	1228	S	0.0	0.0	0:00.00	./eject
3139	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3140	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3141	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3142	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3143	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3144	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3145	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3146	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3147	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3148	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3149	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3150	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3151	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3152	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject
3153	paul	20	0	2496	896	896	S	0.0	0.0	0:00.00	./eject

Ejercicio 3

Ejercicio 4

Ejercicio 5