
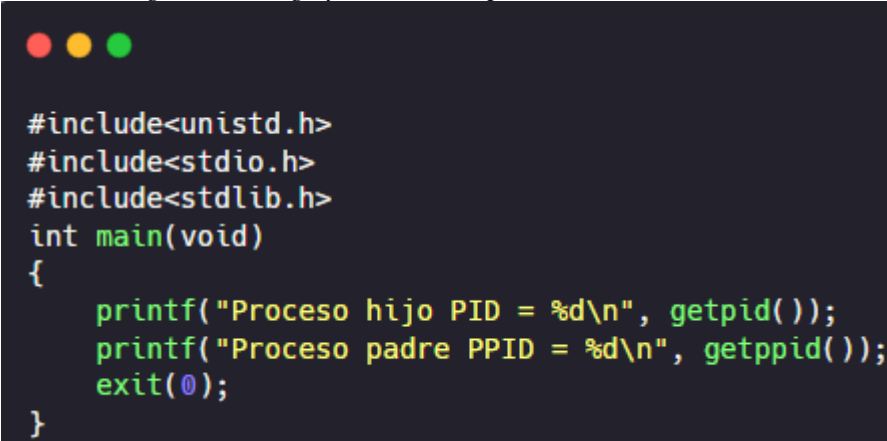
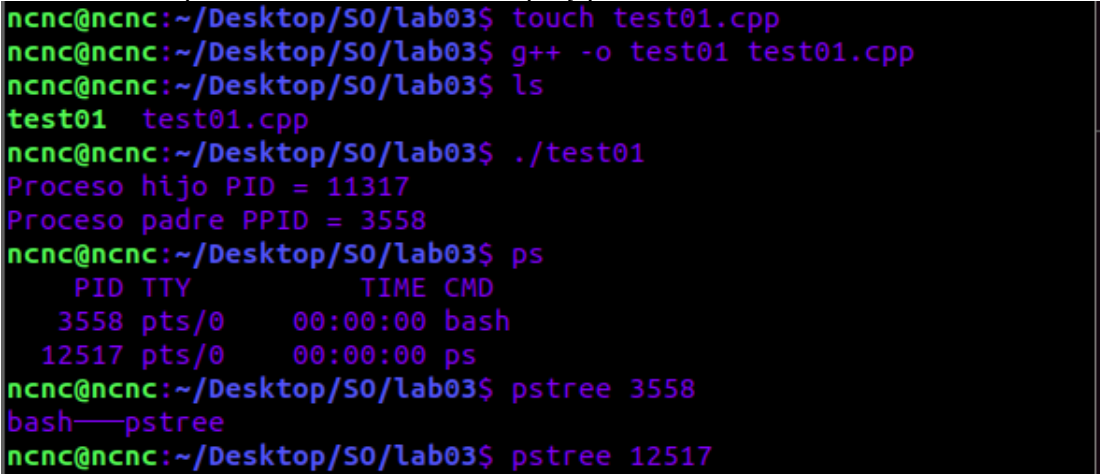


	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 1

INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Sistema Operativos				
TÍTULO DE LA PRÁCTICA:					
NÚMERO DE PRÁCTICA:	03	AÑO LECTIVO:	2022-B	NRO. SEMESTRE:	VI (sexto)
FECHA DE PRESENTACIÓN	12/10/2022	HORA DE PRESENTACIÓN	23:59		
INTEGRANTE (s): Yoset Cozco Mauri				NOTA:	
DOCENTE(s): <i>ROLANDO JESUS CARDENAS TALAVERA</i>					

SOLUCIÓN Y RESULTADOS	
I. ACTIVIDADES	
1. Analice el siguiente código y de una interpretación del resultado obtenido de acuerdo con el marco teórico.	
 <pre> #include<unistd.h> #include<stdio.h> #include<stdlib.h> int main(void) { printf("Proceso hijo PID = %d\n", getpid()); printf("Proceso padre PPID = %d\n", getppid()); exit(0); } </pre>	
Revision de los procesos mediante comando <i>ps</i> y <i>pstree</i> :  <pre> ncnc@ncnc:~/Desktop/SO/lab03\$ touch test01.cpp ncnc@ncnc:~/Desktop/SO/lab03\$ g++ -o test01 test01.cpp ncnc@ncnc:~/Desktop/SO/lab03\$ ls test01 test01.cpp ncnc@ncnc:~/Desktop/SO/lab03\$./test01 Proceso hijo PID = 11317 Proceso padre PPID = 3558 ncnc@ncnc:~/Desktop/SO/lab03\$ ps PID TTY TIME CMD 3558 pts/0 00:00:00 bash 12517 pts/0 00:00:00 ps ncnc@ncnc:~/Desktop/SO/lab03\$ pstree 3558 bash--pstree ncnc@ncnc:~/Desktop/SO/lab03\$ pstree 12517 </pre>	

	<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: 2</p>

El método *getpid()*: Nos muestra el ID de un proceso padre.

El método *getppid()*: Nos muestra el ID de un proceso hijo.

En este caso tenemos el proceso padre *bash* con el PID: 3558

Y otro proceso hijo con PID: 11317:

Adicional a eso al listar procesos con el comando *ps*, se inicia este proceso hijo llamado *ps* con PID: 12517.

II. SOLUCIÓN DE EJERCICIOS/PROBLEMAS

EJERCICIOS PROPUESTOS

- El siguiente código crea un proceso hijo, realice un seguimiento de la variable *value* y describa el porque tiene ese comportamiento.

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

int value = 5;
int main()
{
    pid_t pid;
    pid = fork();
    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("PARENT: value = %d", value); /*LINE A*/
        return 0;
    }
}
```

```
ncnc@ncnc:~/Desktop/S0/lab03$ g++ -o prop_ejer01 prop_ejer01.cpp
ncnc@ncnc:~/Desktop/S0/lab03$ ./prop_ejer01
```

```
PARENT: value = 5ncnc@ncnc:~/Desktop/S0/lab03$ █
```

Value empieza con un valor de 5, *pid_t* crea un nuevo proceso, *fork()* crea un proceso duplicado, siendo el *pid* actualmente de valor 0, entra en la condicional para crear un proceso hijo, aumentando el valor de *value* en 15, valiendo 20, ahora el *pid* después de su creación tiene asignado un nuevo valor PID, reiniciando el valor de *value* a su valor original 5, mostrando así en la impresión 5(valor inicial).

2. En el siguiente código, detalle que parte del código es ejecutada por el proceso padre y que porción del código es ejecutada por el proceso hijo. Describa la actividad de cada uno.

```

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

int main(void)
{
    pid_t pid;
    /* fork a child process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete \n");
    }
    return 0;
}

```



```

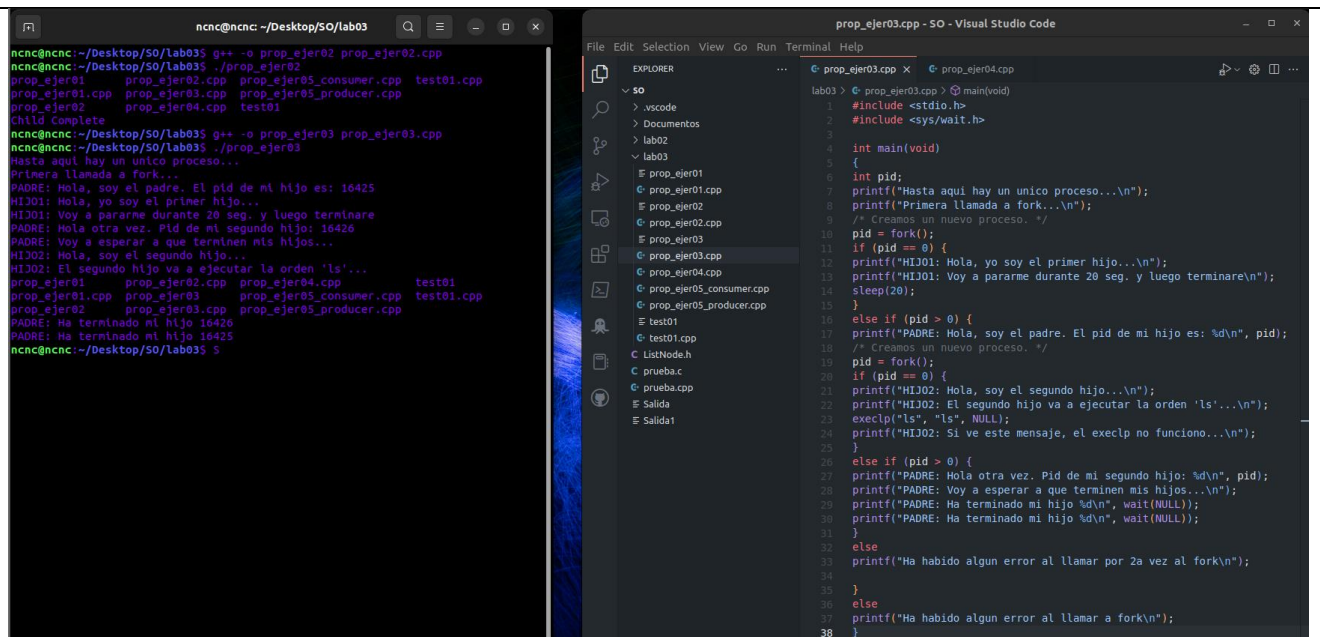
ncnc@ncnc:~/Desktop/S0/lab03$ g++ -o prop_ejer02 prop_ejer02.cpp
ncnc@ncnc:~/Desktop/S0/lab03$ ./prop_ejer02
prop_ejer01      prop_ejer02.cpp  prop_ejer05_consumer.cpp  test01.cpp
prop_ejer01.cpp  prop_ejer03.cpp  prop_ejer05_producer.cpp
prop_ejer02      prop_ejer04.cpp  test01
Child Complete
ncnc@ncnc:~/Desktop/S0/lab03$

```

El proceso padre es iniciado en `pid_t pid`, a partir de esto se crea un `fork()` para iniciar al proceso hijo, este proceso llama al proceso `ls` del sistema, una vez ejecutado este proceso el padre acaba la espera e imprime "child complete".

3. Analice el siguiente código, explique como se da el flujo del código desde el proceso padre y como procede con los procesos hijos.

	<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: 4</p>



```

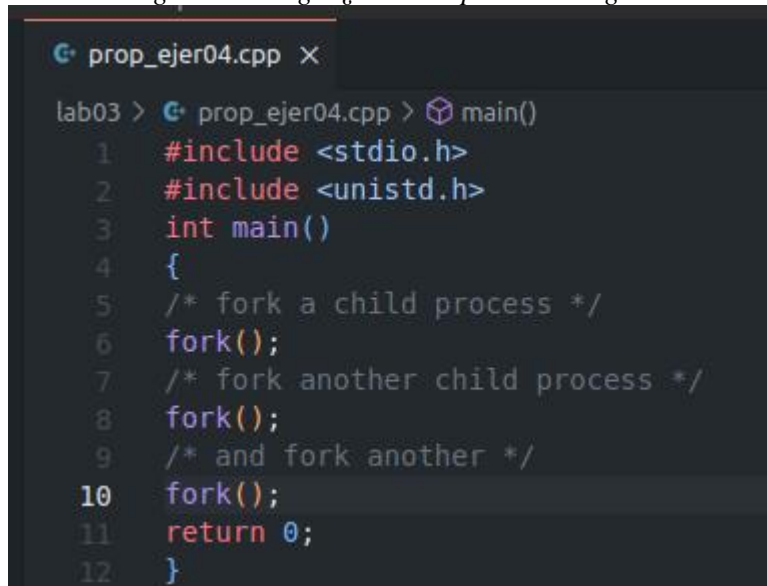
ncnc@ncnc: ~/Desktop/SO/lab03
ncnc@ncnc:~/Desktop/SO/lab03$ g++ -o prop_ejer02 prop_ejer02.cpp
ncnc@ncnc:~/Desktop/SO/lab03$ ./prop_ejer02
prop_ejer01 prop_ejer02.cpp prop_ejer05_consumer.cpp test01.cpp
prop_ejer01.cpp prop_ejer03.cpp prop_ejer05_producer.cpp
prop_ejer02 prop_ejer04.cpp test01
child Complete
ncnc@ncnc:~/Desktop/SO/lab03$ g++ -o prop_ejer03 prop_ejer03.cpp
ncnc@ncnc:~/Desktop/SO/lab03$ ./prop_ejer03
Hasta aqui hay un unico proceso...
Primera llamada a fork...
PADRE: Hola, soy el padre. El pid de mi hijo es: 16425
HIJ01: Hola, yo soy el primer hijo...
HIJ01: Voy a pararme durante 20 seg. y luego terminare
PADRE: Hola otra vez. Pid de mi segundo hijo: 16426
PADRE: Voy a esperar a que terminen mis hijos...
HIJ02: Hola, soy el segundo hijo...
HIJ02: El segundo hijo va a ejecutar la orden 'ls'...
prop_ejer01 prop_ejer02.cpp prop_ejer04.cpp test01
prop_ejer01.cpp prop_ejer03 prop_ejer05_consumer.cpp test01.cpp
prop_ejer02 prop_ejer03.cpp prop_ejer05_producer.cpp
PADRE: Ha terminado mi hijo 16426
PADRE: Ha terminado mi hijo 16425
ncnc@ncnc:~/Desktop/SO/lab03$ s

prop_ejer03.cpp - SO - Visual Studio Code
lab03 > C: prop_ejer03.cpp > main(void)
1 #include <stdio.h>
2 #include <sys/wait.h>
3
4 int main(void)
5 {
6     int pid;
7     printf("Hasta aqui hay un unico proceso...\n");
8     printf("Primera llamada a fork...\n");
9     /* Creamos un nuevo proceso. */
10    pid = fork();
11    if (pid == 0) {
12        printf("HIJ01: Hola, yo soy el primer hijo...\n");
13        printf("HIJ01: Voy a pararme durante 20 seg. y luego terminare\n");
14        sleep(20);
15    }
16    else if (pid > 0) {
17        printf("PADRE: Hola, soy el padre. El pid de mi hijo es: %d\n", pid);
18        /* Creamos un nuevo proceso. */
19        pid = fork();
20        if (pid == 0) {
21            printf("HIJ02: Hola, soy el segundo hijo...\n");
22            printf("HIJ02: El segundo hijo va a ejecutar la orden 'ls'...\n");
23            execlp("ls", "ls", NULL);
24            printf("HIJ02: Si ve este mensaje, el execlp no funciono...\n");
25        }
26        else if (pid > 0) {
27            printf("PADRE: Hola otra vez. Pid de mi segundo hijo: %d\n", pid);
28            printf("PADRE: Voy a esperar a que terminen mis hijos...\n");
29            printf("PADRE: Ha terminado mi hijo %d\n", wait(NULL));
30            printf("PADRE: Ha terminado mi hijo %d\n", wait(NULL));
31        }
32    }
33    printf("Ha habido algun error al llamar por 2a vez al fork\n");
34
35 }
36 else
37     printf("Ha habido algun error al llamar a fork\n");
38 }

```

Al ejecutar la línea pid, sea crea un nuevo proceso, seguido se hace un fork() creación de un proceso hijo. Siendo el pid mayor que 0, se ejecuta la sentencia que imprime el primer mensaje del padre, seguido se vuelve a la primera condicional cumpliendo la condicional se ejecuta los prints por parte del primer hijo, adicional tiene la instrucción sleep(20). Se crea el segundo hijo con un fork() adicional que se imprime el pid del nuevo proceso hijo, el segundo hijo ejecuta la instrucción ls, continuando ya que el pid es mayor que 0, entra en la condición y se ejecuta el comando wait(null) terminando con los procesos hijos.

4. Analice el siguiente código. ¿Cuántos procesos se generan?



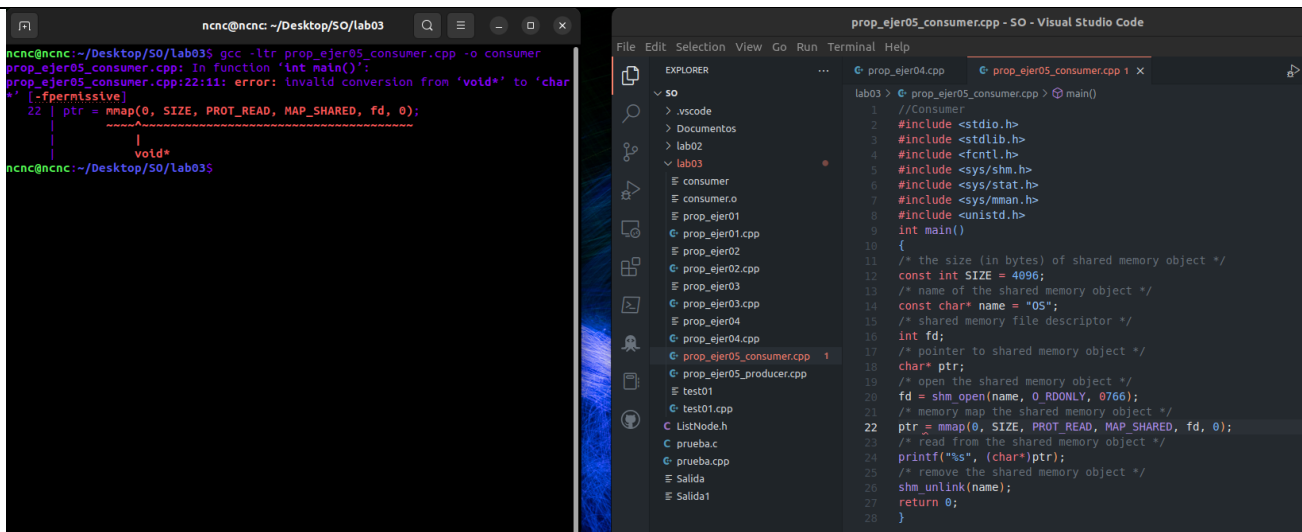
```

prop_ejer04.cpp X
lab03 > C: prop_ejer04.cpp > main()
1 #include <stdio.h>
2 #include <unistd.h>
3 int main()
4 {
5     /* fork a child process */
6     fork();
7     /* fork another child process */
8     fork();
9     /* and fork another */
10    fork();
11    return 0;
12 }

```

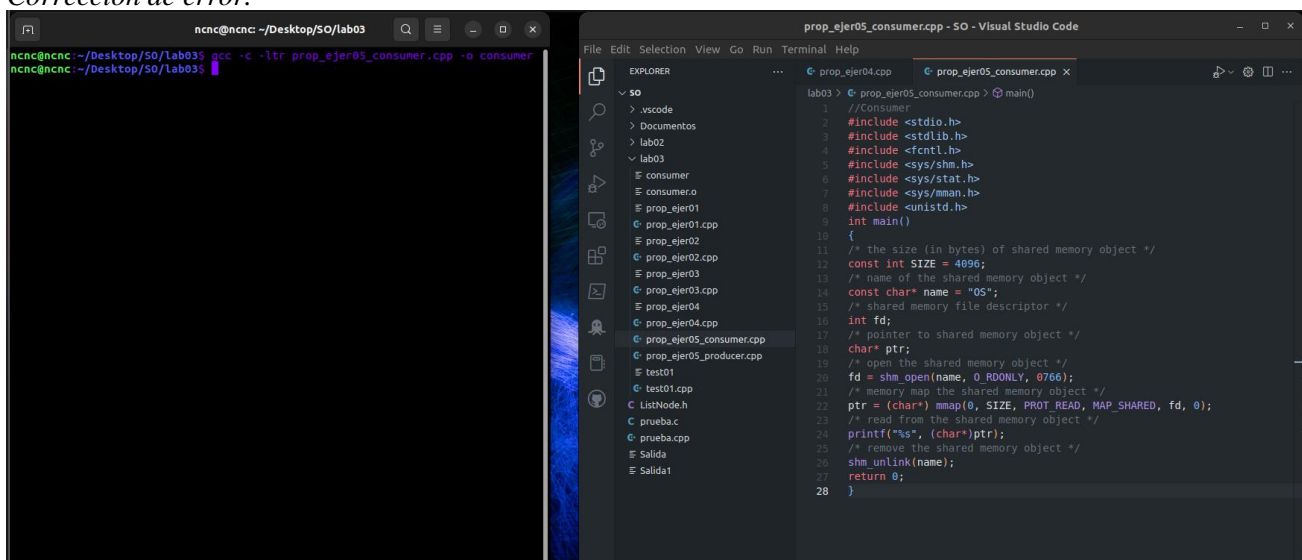
Genera 3 procesos hijo.

5. Ejecución código consumer.cpp



```
ncnc@ncnc: ~/Desktop/SO/lab03
ncnc@ncnc:~/Desktop/SO/lab03$ gcc -c -ltr prop_ejer05_consumer.cpp -o consumer
prop_ejer05_consumer.cpp: In function 'int main()':
prop_ejer05_consumer.cpp:22:11: error: invalid conversion from 'void*' to 'char*' [-fpermissive]
   22 | ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, fd, 0);
      |           ^~~~~
      |           |
      |       void*
ncnc@ncnc:~/Desktop/SO/lab03$
```

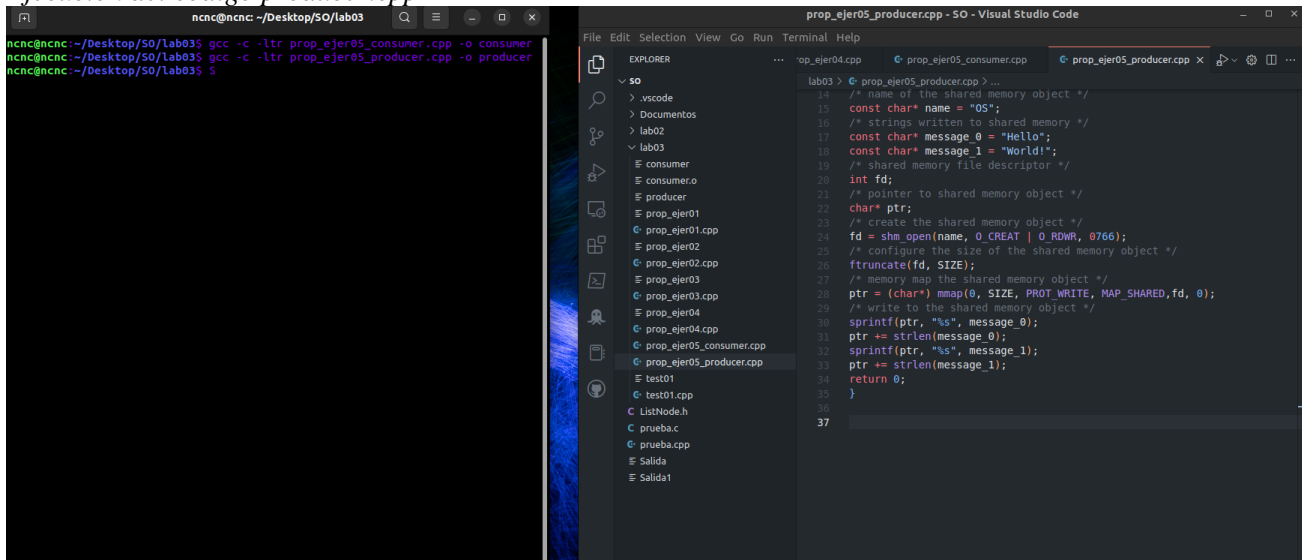
Corrección de error.



```
ncnc@ncnc:~/Desktop/SO/lab03$ gcc -c -ltr prop_ejer05_consumer.cpp -o consumer
ncnc@ncnc:~/Desktop/SO/lab03$
```



Se hace un cast (char)*

Ejecución del código producer.cpp



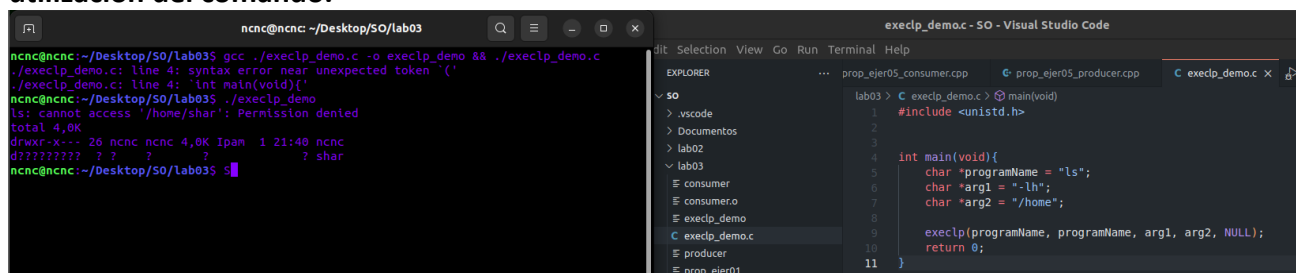
```
ncnc@ncnc:~/Desktop/SO/lab03$ gcc -c -ltr prop_ejer05_producer.cpp -o producer
ncnc@ncnc:~/Desktop/SO/lab03$
```

Incluye cast (char)*

	<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>

III. SOLUCIÓN DEL CUESTIONARIO

1. ¿Cuál es la principal característica de crear un proceso utilizando la función FORK?
Crea un duplicado del proceso actual, que comparte todos los valores actuales de las variables, ficheros y otras estructuras de datos. La llamada a fork devuelve al padre el pid del proceso hijo y devuelve un 0 al proceso hijo.
2. ¿Cuántos procesos FORK se pueden crear de forma secuencial? De manera ilimitada, El limite generalmente lo establece los recursos del sistema, el tamaño de memoria existente. Un error EANOMEM. ¿Existe algún límite establecido por el sistema operativo?
El limite lo establece el sistema operativo, en Linux podemos encontrar la configuración de esta opción en la ubicación /etc/security/limits.conf.
3. ¿Qué trabajo realiza la función EXECLP? Explique utilizando un ejemplo de utilización del comando.



```

ncnc@ncnc: ~/Desktop/SO/lab03
ncnc@ncnc:~/Desktop/SO/lab03$ gcc ./execlp_demo.c -o execlp_demo && ./execlp_demo.c
./execlp_demo.c: line 4: syntax error near unexpected token '('
./execlp_demo.c: line 4: `int main(void){'
ncnc@ncnc:~/Desktop/SO/lab03$ ./execlp_demo
ls: cannot access '/home/shar': Permission denied
total 4,0K
drwxr-x--- 26 ncnc ncnc 4,0K Ipam 1 21:40 ncnc
d???????? ? ? ? ? ? shar
ncnc@ncnc:~/Desktop/SO/lab03$

```

```

lab03 > C execlp_demo.c > main(void)
1 #include <unistd.h>
2
3
4 int main(void){
5     char *programName = "ls";
6     char *arg1 = "-lh";
7     char *arg2 = "/home";
8
9     execlp(programName, programName, arg1, arg2, NULL);
10    return 0;
11 }

```



La función execlp() reemplaza la imagen del proceso actual con una nueva imagen de proceso especificada por el archivo. Esta nueva imagen se construye a partir de un archivo ejecutable normal denominado archivo de imagen de proceso nuevo "new process image file". No se realiza ningún return porque la imagen de proceso de llamada se reemplaza por la nueva imagen del proceso.

IV. CONCLUSIONES

El laboratorio fue retador el tener partes del código que tenían errores el cual el compilador me marcaba, los manuales en línea son de gran ayuda. Comprendí que los sistemas operativos establecen sus propios límites y que también como usuarios administradores podemos modificar no tan libremente estos límites, los verdaderos limites siempre lo establecen la parte física, específicamente la memoria.

RETROALIMENTACIÓN GENERAL

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>