Universidad Nacional de San Agustín de Arequipa

Escuela Profesional de Ciencia de la Computación

Sistemas Operativos

Laboratorio 3: Procesos

Presentado por:
Rushell Vanessa Zavalaga Orozco

Docente :
Dra. Yessenia D. Yari R.

Generar un documento en donde incluirá la explicación de cada código (línea a línea) y mostrar el resultado(screen de lo ejecutado en el SO).

1. Ejercicio 1

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

pid_t wait(int *status);
int status;
pid_t fork_return;
fork_return = fork();

if (fork_return == 0) /* child process */
{
    printf("\n I'm the child!");
    exit(0);
}
else if (fork_return > 0) /* parent process */
{
    wait(&status);
    printf("\n I'm the parent!");
    if (WIFEXITED(status))
        printf("\n Child returned: %d\n",
    WEXITSTATUS(status));
}
```

1.1. Explicación del código linea por línea:

Primero se incluyen las librerías necesarias, las cuales son las siguientes:

- `stdio.h` para las funciones de entrada/salida como `printf()`
- `sys/types.h` para definir tipos como `pid_t`
- `sys/wait.h` para usar `wait()` y manejar la finalización de procesos
- `unistd.h` para `fork()` y otras llamadas del sistema
- `stdlib.h` para `exit()`

Se inicia la función principal del programa (main), como variables necesarias para declarar procesos hijos:

- `pid_t fork_return;`: Declaración de una variable para almacenar el valor de retorno de `fork()`
- `int status;`: Declaración de una variable para guardar el estado del proceso hijo al usar `wait()`.
- `pid_t wait(int *status);`: Declaración de la función `wait()`.

Con nuestras variables inicializadas se crea un proceso hijo con la función `fork()`, la cual puede retornar tres valores.

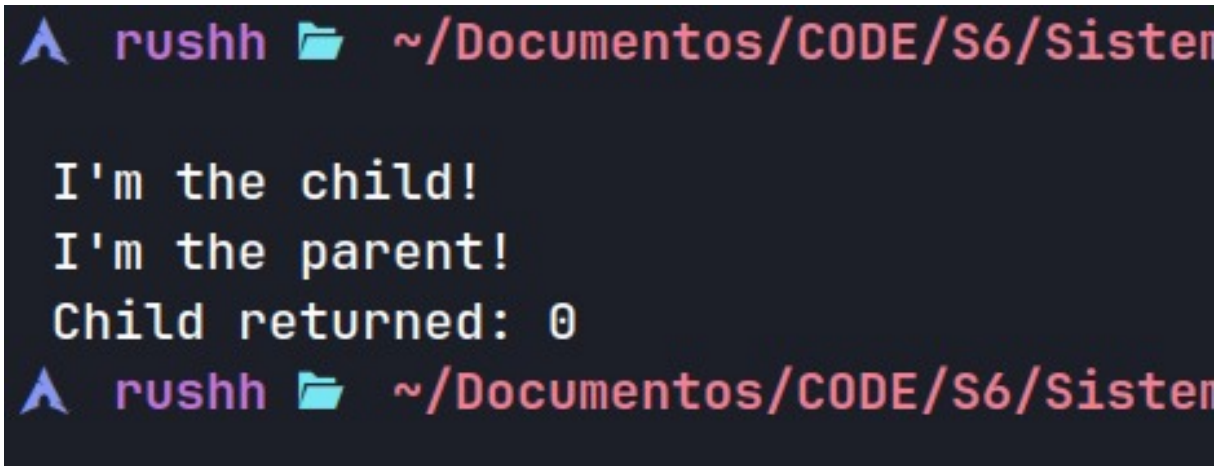
- 0 en el proceso hijo.
- Un valor mayor que 0 (PID del hijo) en el proceso padre.
- -1 en caso de fallo.

Ahora si `if (fork_return == 0) {` : Si el valor retornado por `fork()` es 0, significa que este es el proceso hijo, lo cual imprimiría un mensaje indicando que es el hijo. `printf("\n I'm the child!");`. Luego el proceso hijo finaliza su ejecución. `exit(0);`

En cambio si estamos en el proceso padre, entonces:
`} else if (fork_return >0) {` :

- `wait(&status);`: El proceso padre espera a que el hijo termine y captura su estado de salida.
- `printf("\n I'm the parent!");`: El proceso padre imprime un mensaje indicando que es el padre.
- `if (WIFEXITED(status))`: Verifica si el hijo terminó normalmente.
- `printf("\n Child returned: %d\n", WEXITSTATUS(status));`: Muestra el código de salida del proceso hijo.

1.2. Resultado:



2. Ejercicio 2

```
#include <stdio.h>
#include <sys/types.h>
#define MAX_COUNT 200

void ChildProcess(void); /* child process prototype */
void ParentProcess(void); /* parent process prototype */
void main(void)
{
    pid_t pid;
    pid = fork();
    if (pid == 0)
        ChildProcess();
    else
        ParentProcess();
}

void ChildProcess(void)
{
    int i;
    for (i = 1; i <= MAX_COUNT; i++)
        printf("This line is from child, value = %d\n", i);
    printf("Child process is done\n");
}

void ParentProcess(void)
{
    int i;
    for (i = 1; i <= MAX_COUNT; i++)
        printf("This line is from parent, value = %d\n", i);
    printf("Parent is done\n");
}
```

2.1. Explicación del código linea por línea:

Primero se incluyen las librerías necesarias, las cuales son las siguientes:

- `#include <stdio.h>`: Se incluye el encabezado estándar de entrada/salida para usar funciones como `printf()`.
- `#include <sys/types.h>`: Se incluye para definir tipos de datos como `pid_t`.

Ahora definimos algunas variables importantes como algunas funciones de ayuda como:

- `#define MAX_COUNT 200`: Se define una constante llamada `MAX_COUNT` con valor 200, que será usada en los bucles de impresión.
- `void ChildProcess(void);`, `void ParentProcess(void);`: Declaraciones de las funciones `ChildProcess()` y `ParentProcess()`, que son llamadas según si el proceso es el hijo o el padre.
- `void main(void) {` : Se define la función `main()` que es el punto de entrada del programa.

Ahora en las funciones que tenemos declaramos nuevas variables y su ejecución.

- `pid_t pid;`: Se declara una variable de tipo `pid_t` para almacenar el valor retornado por `fork()`.
- `pid = fork();`: Se crea un nuevo proceso. La llamada a `fork()` retorna:
  - 0 en el proceso hijo.
  - Un valor mayor que 0 (el PID del hijo) en el proceso padre.
- `if (pid == 0) {` : Si `pid` es igual a 0, significa que estamos en el proceso hijo.
- `ChildProcess();`: Si es el proceso hijo, se llama a la función `ChildProcess()`.
- `else {` : Si el valor de `pid` es mayor que 0, significa que estamos en el proceso padre.
- `ParentProcess();`: Si es el proceso padre, se llama a la función `ParentProcess()`.

Ahora en la función ChildProcess:

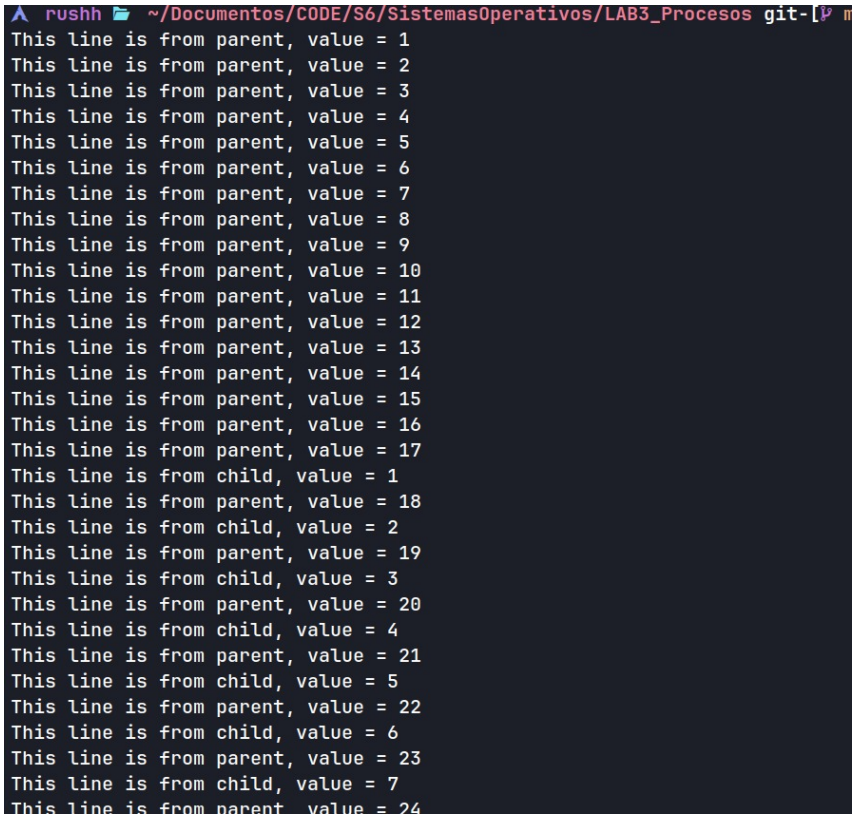
- `void ChildProcess(void) {` : Definición de la función `ChildProcess()`.
- `for (i = 1; i <= MAX_COUNT; i++) {` : Bucle que imprime un mensaje 200 veces, desde 1 hasta `MAX_COUNT`.
- `printf("This line is from child, value =%d", i);`: Imprime el valor de `i` para cada iteración del bucle, indicando que el mensaje es del proceso hijo.
- `printf(Çhild process is done");`: Imprime un mensaje indicando que el proceso hijo ha terminado su ejecución.

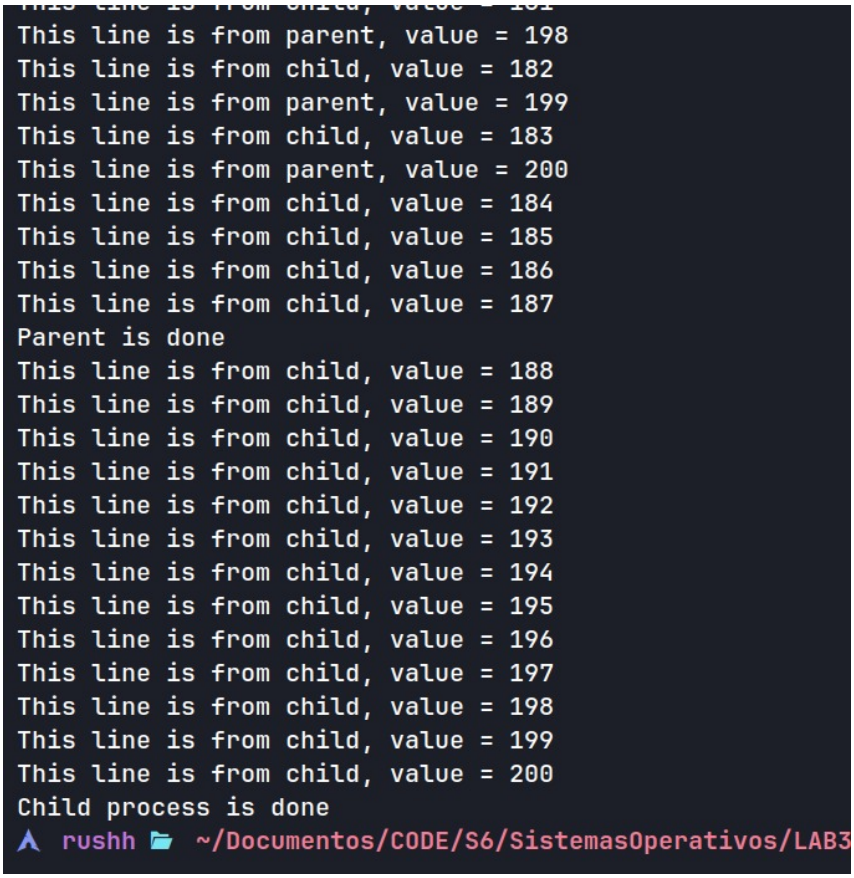
Ahora en la función ParentProcess:

- `void ParentProcess(void) {` : Definición de la función `ParentProcess()`.
- `for (i = 1; i <= MAX_COUNT; i++) {` : Bucle que imprime un mensaje 200 veces, desde 1 hasta `MAX_COUNT`, igual que en el proceso hijo.

- `printf("This line is from parent, value =%d", i);`: Imprime el valor de `i` para cada iteración del bucle, indicando que el mensaje es del proceso padre.
- `printf("Parent is done");`: Imprime un mensaje indicando que el proceso padre ha terminado su ejecución.

2.2. Resultado:




This line is from parent, value = 198
This line is from child, value = 182
This line is from parent, value = 199
This line is from child, value = 183
This line is from parent, value = 200
This line is from child, value = 184
This line is from child, value = 185
This line is from child, value = 186
This line is from child, value = 187
Parent is done
This line is from child, value = 188
This line is from child, value = 189
This line is from child, value = 190
This line is from child, value = 191
This line is from child, value = 192
This line is from child, value = 193
This line is from child, value = 194
This line is from child, value = 195
This line is from child, value = 196
This line is from child, value = 197
This line is from child, value = 198
This line is from child, value = 199
This line is from child, value = 200
Child process is done
rushh ~/Documentos/CODE/S6/SistemasOperativos/LAB3