

Universidad Francisco Marroquín
Etson Saúl Guerrero Hernández
Sistemas Operativos



004

Michelle Dardon - 20210534
Stephanie Grotewold - 20210567

Creación y Comunicación de Procesos en Linux

1. Creación de Procesos

La creación de procesos en sistemas operativos basados en Linux se realiza con la llamada al sistema `fork()`. Este sistema crea un nuevo proceso hijo duplicando el proceso padre. Los pasos principales son:

1. Llamar a `fork()`: Crea un proceso hijo que hereda el código y el entorno del padre.
2. Diferenciar padre e hijo:
 - a. `fork()` devuelve un `PID > 0` en el padre (el ID del hijo).
 - b. `fork()` devuelve 0 en el proceso hijo.
3. Ejecutar código específico: Se puede utilizar `if-else` para asignar tareas distintas al padre y al hijo.
4. Finalizar el proceso: Cada proceso finaliza su ejecución con `exit()` o ejecutando el código completamente.

2. Sincronización de Procesos

Dado que los procesos pueden ejecutarse de forma paralela, es importante sincronizarlos para garantizar que el padre no termine antes que el hijo o que los datos se procesen en orden correcto.

Método en Linux: `waitpid()`

La función `waitpid()` permite que el proceso padre espere a que su hijo termine antes de continuar. Los pasos principales son:

1. El padre ejecuta `waitpid(pid, NULL, 0)` para esperar la finalización del hijo.
2. El hijo ejecuta su código y finaliza con `exit(0)`.
3. El padre recibe la notificación de que el hijo terminó y continúa su ejecución.

3. Comunicación Entre Procesos (IPC)

La comunicación entre procesos (IPC) permite el intercambio de datos entre procesos independientes. Algunos de los métodos más comunes incluyen:

- Pipes (`pipe()`): Los pipes permiten que un proceso escriba datos en un canal y otro proceso los lea. Son unidireccionales (de padre a hijo o viceversa). Pasos clave:
 1. Se crea un pipe con `pipe(fd)`.
 2. El padre escribe en el `fd[1]` (escritura).
 3. El hijo lee desde `fd[0]` (lectura).
- Memoria Compartida (`shmget()` y `shmat()`): La memoria compartida permite que múltiples procesos accedan a una misma región de memoria. Pasos clave:
 1. Se crea un segmento de memoria con `shmget()`.
 2. El padre escribe en la memoria compartida usando `shmat()`.
 3. El hijo lee el mensaje desde la memoria compartida.

4. Finalmente, se elimina la memoria compartida con ``shmctl()``.

La creación de procesos se realiza con ``fork()``, generando procesos hijos. La sincronización se logra con ``waitpid()``, asegurando que el padre espere la finalización del hijo. La comunicación entre procesos (IPC) puede implementarse mediante pipes o memoria compartida, permitiendo el intercambio de datos entre procesos. Estos mecanismos son fundamentales en sistemas operativos para la gestión eficiente de procesos y recursos.

4. Screenshot

```
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % gcc -o 1_task 1_task.c
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % ./1_task
Parent Process: PID=7895
Child Process: PID=7904, Parent PID=7895
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % gcc -o 2_task 2_task.c
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % ./2_task
Child Process: PID=7966, Parent PID=7957
Parent Process: Child has finished execution.
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % gcc -o pipe pipe.c
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % ./pipe
Parent Process: Writing "Hello from Parent"
Child Process: Received "Hello from Parent"
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % gcc -o multi_process multi_process.c
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % ./multi_process
Parent Process: PID=8086
Child 1: PID=8087, Parent PID=8086
Child 2: PID=8088, Parent PID=8086
Child 3: PID=8089, Parent PID=8086
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % gcc -o shared_memory shared_memory.c
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % ./shared_memory
Parent Process: Writing "Shared Memory Example"
Child Process: Read "Shared Memory Example"
stephgrotewold@Stephs-MacBook-Pro OperatingSystems % █
```