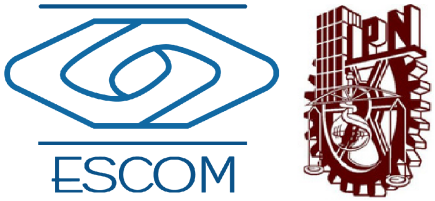


Escuela Superior de Cómputo
Instituto Politécnico Nacional

Sistemas Operativos
Procesos

Práctica 2

Proceso zombie, Kill y Waitpid

Hernández Herrera Daniel Alejandro
Ojeda Galván René
Sánchez Gutiérrez Luis Arturo
Torres Rodríguez Mauricio Alberto
2CV7

1. zombie.c

Muestre la pantalla de ejecución del programa y ejecute de manera adicional en una terminal diferente el comando ps -ae.

Realice lo siguiente

- Investigue en la bibliografía proporcionada para la asignatura, el diagrama de estados de procesos en LINUX y describa de manera cuáles son las condiciones que hacen que se pase a cada estado

Estos son los estados de los procesos de Linux[1]:

- **TASK_RUNNING**
  El proceso ya está siendo ejecutado o está en espera de ser ejecutado.
- **TASK_INTERRUPTIBLE**
  Proceso suspendido que puede reanudar a TASK_RUNNING dada una señal.
- **TASK_UNINTERRUPTIBLE**
  Proceso suspendido que no puede reanudar normalmente a TASK_RUNNING dada una señal.
- **TASK_STOPPED**
  Estado que indica que un proceso ha acabado.
- **TASK_ZOMBIE**
  Estado que indica que un proceso ha acabado, pero no puede ser removido apropiadamente pues el padre no puede ser reportado de datos del proceso que podría necesitar. (Según la bibliografía, esto sucede cuando el proceso padre no sale de *wait()*)

- Tomando como referenca el diagrama obtenido en el punto anterior, explique de manera detallada los estados por los que pasa el proceso padre y el proceso hijo durante la ejecución del programa

- El proceso padre empieza en estado TASK_RUNNING.
- El proceso padre llama la función *fork()* y obtiene un hijo de mismo código.
- Ambos procesos en estado TASK_RUNNING entran al switch case.
- El padre pasa a TASK_UNINTERRUPTIBLE con *sleep()*.
- El proceso hijo termina, pero pasa a TASK_ZOMBIE, pues su padre sigue en TASK_UNINTERRUPTIBLE.
- Ambos procesos pasan a TASK_STOPPED

- perror

  Esta función produce un mensaje que describe al último error obtenido durante una llamada al sistema o una función de biblioteca.

- fork

  Crea un proceso hijo duplicando el proceso padre.

2. kill.c

Muestre la pantalla de ejecución del programa.

Describa cuál es el propósito de las siguientes bibliotecas:

- signal.h

  > Esta biblioteca es utilizada para manejar señales.

- stdlib.h

  > Biblioteca estándar que utilizamos para manejar los procesos. Nos define las macros EXIT_SUCCESS y EXIT_FAILURE y la función *exit()*.

Para las siguientes funciones, mencione dónde están definidas, qué es lo que proporcionan de salida y qué argumentos necesitan de entrada:

- exit

  > Está definida en la biblioteca *stdlib.h*. No regresan nada y requieren un entero que representará el estado de salida del proceso. Termina el proceso.

- kill

  Está definida en la biblioteca *signal.h*. Regresa 0 si tuvo éxito, -1 si falló. Pide como parámetros la señal a enviar y el pid del proceso a enviar la señal. Si es ¿0, se enviará la señal al proceso específico. Igual a 0 enviará la señal a todos los procesos del grupo del emisor. -1 enviará la señal a todos los procesos.

Mencione el significado de los siguientes valores de salida en exit:

- EXIT SUCCESS

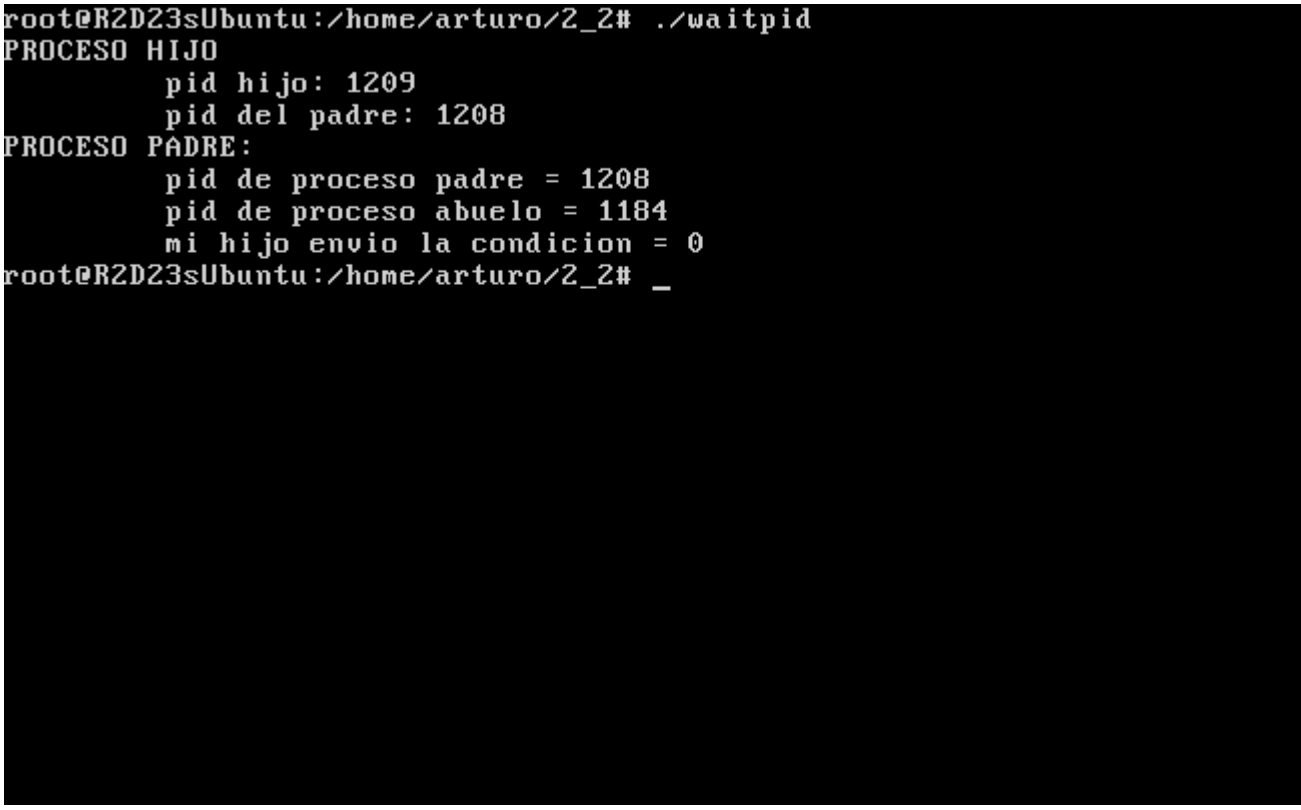
  El valor a regresar si el proceso ha acabado con éxito.

- EXIT FAILURE

  El valor a regresar si el proceso ha acabado con errores.

3. waitpid.c

Muestre la pantalla de ejecución del programa.



Describa cuál es el propósito de la siguiente biblioteca:

- sys/wait.h

  > Proporciona lo necesario para trabajar con *waitpid()*.

Mencione dónde está definida, qué es lo que proporciona de salida y qué argumentos necesita de entrada la función waitpid:

- waitpid

Se encuentra definido en sys/wait.h. Espera a que un proceso hijo acabe.

Referencias

[1] Bovet D. & Cesati M. (2000), *Understanding the Linux Kernel*, O'Reilly, Primera Edición, 65-66, 1994.