Taller de *syscalls* y señales

Sistemas Operativos

Departamento de Computación, FCEyN,
Universidad de Buenos Aires, Buenos Aires, Argentina

21 de Abril de 2020

Detrás de escena

¿Y `strace` cómo funciona?

Detrás de escena

¿Y `strace` cómo funciona? `strace strace`.

Detrás de escena

¿Y `strace` cómo funciona? `strace strace`.

El secreto es la *syscall* `ptrace()`.

Detrás de escena

¿Y `strace` cómo funciona? `strace strace`.

El secreto es la *syscall* `ptrace()`. Veamos qué tiene para decir el manual.

`man 2 ptrace`

```
NOMBRE
  ptrace - rastreo de un proceso

SINOPSIS
  #include <sys/ptrace.h>

  long ptrace(enum __ptrace_request request, pid_t pid,
              void *addr, void *data);

DESCRIPCIÓN
  La llamada al sistema ptrace proporciona un medio por el que un proceso
  padre puede observar y controlar la ejecución de un proceso hijo y examinar
  y cambiar su imagen de memoria y registros. Se usa principalmente en la
  implementación de depuración con puntos de ruptura y en el rastreo de
  llamadas al sistema.
```

Usando ptrace()

Vamos a usar ptrace() para monitorear un proceso.

Prototipo de ptrace()

```
long ptrace(enum __ptrace_request request, pid_t pid,
            void *addr, void *data);
```

request puede ser alguno de estos:

- PTRACE_TRACEME, PTRACE_ATTACH, PTRACE_DETACH,
- PTRACE_KILL, PTRACE_CONT,
- PTRACE_SYSCALL, PTRACE_SINGLESTEP,
- PTRACE_PEEKDATA, PTRACE_POKEDATA,
- PTRACE_PEEKUSER, PTRACE_POKEUSER,
- ...y más[1].

[1] Ver man 2 ptrace.

Usando `ptrace()`

- **Situación**:
  - Proceso **padre**.
  - Proceso **hijo** que queremos monitorear.
- **Inicialización**. Dos alternativas:
  1. El proceso hijo solicita ser monitoreado por su padre haciendo una llamada a `ptrace(PTRACE_TRACEME)`. Por ejemplo, después de un `fork()` y antes de un `execve()`.
  2. El proceso padre se engancha al proceso hijo con la llamada `ptrace(PTRACE_ATTACH, pid_child)`. Esto permite engancharse a un proceso que ya está corriendo (si se tienen permisos suficientes).
- **Finalización**:
  - Con la llamada `ptrace(PTRACE_DETACH, pid_child)` se deja de monitorear.

Usando `ptrace()`

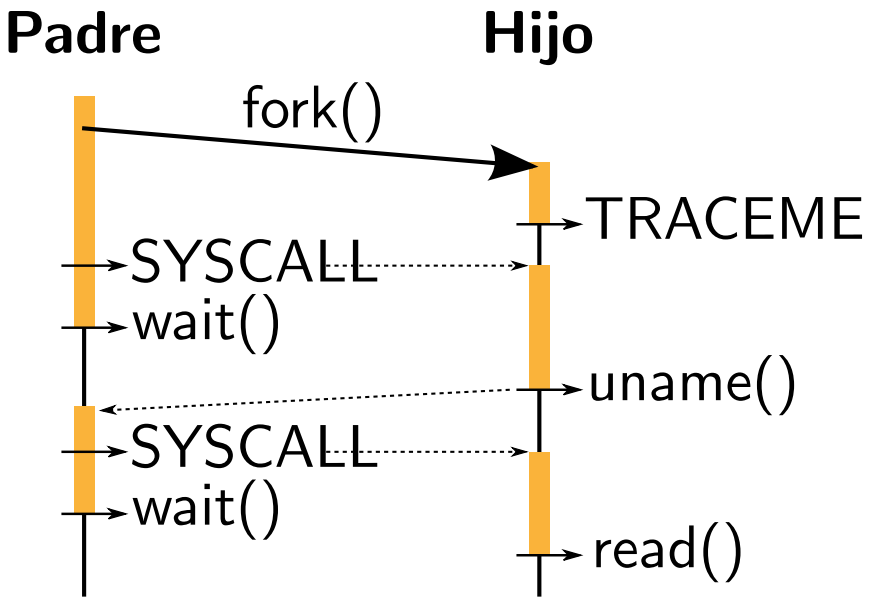
- `ptrace()` permite monitorear tres tipos de **eventos**:
  1. Señales: cuando el proceso hijo recibe una señal.
  2. *Syscalls*: cada vez que el proceso hijo entra o sale de la llamada a una *syscall*.
  3. Instrucciones: cuando el proceso hijo ejecuta **una** instrucción.

Usando `ptrace()`

- `ptrace()` permite monitorear tres tipos de **eventos**:
  1. Señales: cuando el proceso hijo recibe una señal.
  2. *Syscalls*: cada vez que el proceso hijo entra o sale de la llamada a una *syscall*.
  3. Instrucciones: cuando el proceso hijo ejecuta **una** instrucción.

- Cada vez que se genera un **evento**, el proceso hijo se detiene. El padre se entera mediante una llamada (bloqueante) a la *syscall* `wait()`, que retorna al producirse el evento. Luego, puede **reanudar** al hijo hasta:
  1. la siguiente señal recibida, llamando a `ptrace(PTRACE_CONT)`.
  2. la siguiente señal recibida o *syscall* ejecutada, llamando a `ptrace(PTRACE_SYSCALL)`.
  3. solo por una instrucción, llamando a `ptrace(PTRACE_SINGLESTEP)`.

ptrace(): Esquema de uso (simplificado)

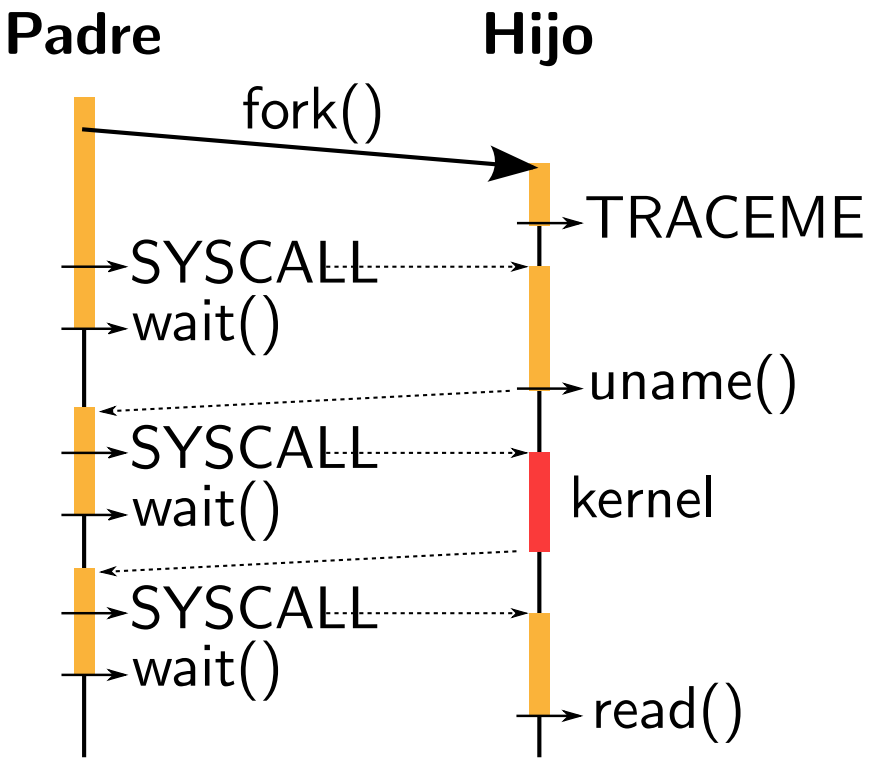


Ejemplo **simplificado** del mecanismo de bloqueo de `ptrace()`.

El hijo se detiene cada vez que llama a una *syscall*.

El padre lo reanuda con una llamada a `ptrace(PTRACE_SYSCALL)`.

ptrace(): Esquema de uso



En realidad, el padre recibe **dos** eventos, al entrar y salir de la *syscall*.

ptrace(): Esquema de comunicación

1. Se inicializa el mecanismo de `ptrace()` (`PTRACE_TRACEME` o `PTRACE_ATTACH`).
2. **padre**: Llama a `wait()`; espera el próximo evento del hijo.
3. **hijo**: Ejecuta normalmente hasta que se genere un evento (recibir una señal, hacer una *syscall* o ejecutar una instrucción).
4. **hijo**: Se genera el evento y el proceso se detiene.
5. **padre**: Vuelve de la *syscall* `wait()`.
6. **padre**: Puede inspecionar y modificar el estado del hijo: registros, memoria, etc.
7. **padre**:
   - Reanuda el proceso hijo con `PTRACE_CONT`, `PTRACE_SYSCALL` o `PTRACE_SINGLESTEP` y vuelve a 2,
   - o bien termina el proceso con `PTRACE_KILL` o lo libera con `PTRACE_DETACH`.

Ejemplo: `launch`

A modo de ejemplo, consideremos un programa, `launch.c`, que permite poner a ejecutar otro programa.

`launch.c` - `main()`

```
/* Fork en dos procesos */
child = fork();
if (child == -1) { perror("ERROR fork"); return 1; }
if (child == 0) {
  /* Solo se ejecuta en el hijo */
  execvp(argv[1], argv + 1);
  /* Si vuelve de exec() hubo un error */
  perror("ERROR child exec(...)"); exit(1);
} else {
  /* Solo se ejecuta en el padre */
  while(1) {
    if (wait(&status) < 0) { perror("wait"); break; }
    if (WIFEXITED(status)) break; /* Proceso terminado */
  }
}
```

Ejemplo: launch + ptrace()

launch.c + ptrace

```c
/* Fork en dos procesos */
child = fork();
if (child == -1) { perror("ERROR fork"); return 1; }
if (child == 0) {
  /* Solo se ejecuta en el hijo */
  if (ptrace(PTRACE_TRACEME, 0, NULL, NULL)) {
    perror("ERROR child ptrace(PTRACE_TRACEME, ...)"); exit(1);
  }
  execvp(argv[1], argv + 1);
  /* Si vuelve de exec() hubo un error */
  perror("ERROR child exec(...)"); exit(1);
} else {
  /* Solo se ejecuta en el padre */
  while(1) {
    if (wait(&status) < 0) { perror("wait"); break; }
    if (WIFEXITED(status)) break; /* Proceso terminado */
    ptrace(PTRACE_SYSCALL, child, NULL, NULL); /* Contin a */
  }
  ptrace(PTRACE_DETACH, child, NULL, NULL); /* Liberamos al hijo */
}
```

ptrace(): Modificando el estado del proceso hijo

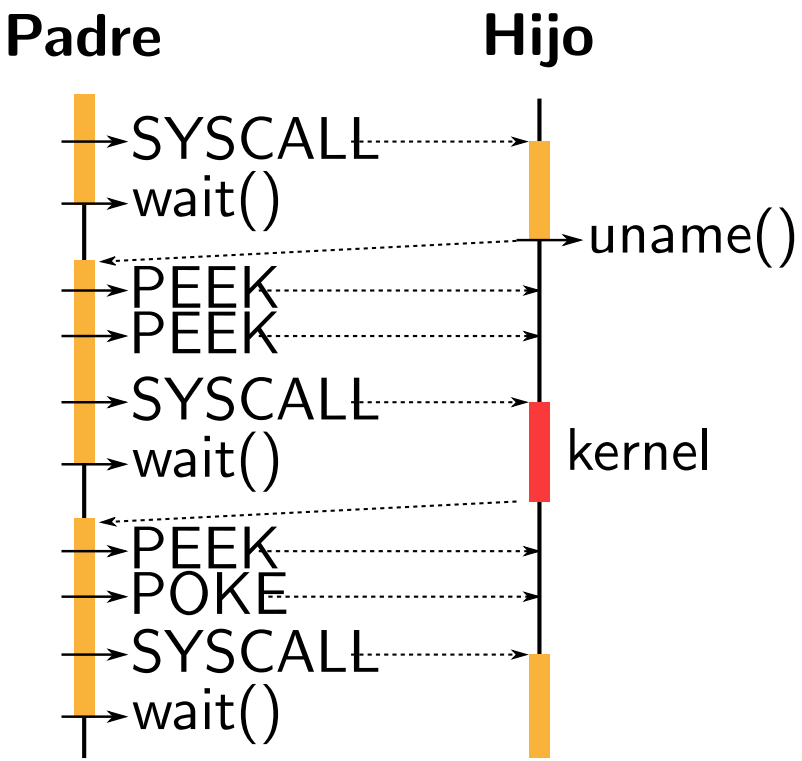
ptrace() permite acceder a la memoria del proceso hijo.

- PTRACE_PEEKDATA y PTRACE_POKEDATA: leer (PEEK) o escribir (POKE) cualquier dirección de memoria en el proceso hijo.
- PTRACE_PEEKUSER, PTRACE_POKEUSER: leer o escribir la memoria de usuario que el sistema guarda al iniciar la *syscall* (registros y estado del proceso).

Ejemplos

- Obtener el número de *syscall* llamada:
```
int sysno = ptrace(PTRACE_PEEKUSER, child, 4 * ORIG_EAX, NULL);
```
- Leer la dirección addr (memoria del proceso hijo):
```
unsigned int valor = ptrace(PTRACE_PEEKDATA, child, addr, NULL);
```
- Escribir otro valor en la direccion addr:
```
ptrace(PTRACE_POKEDATA, child, addr, valor + 1);
```

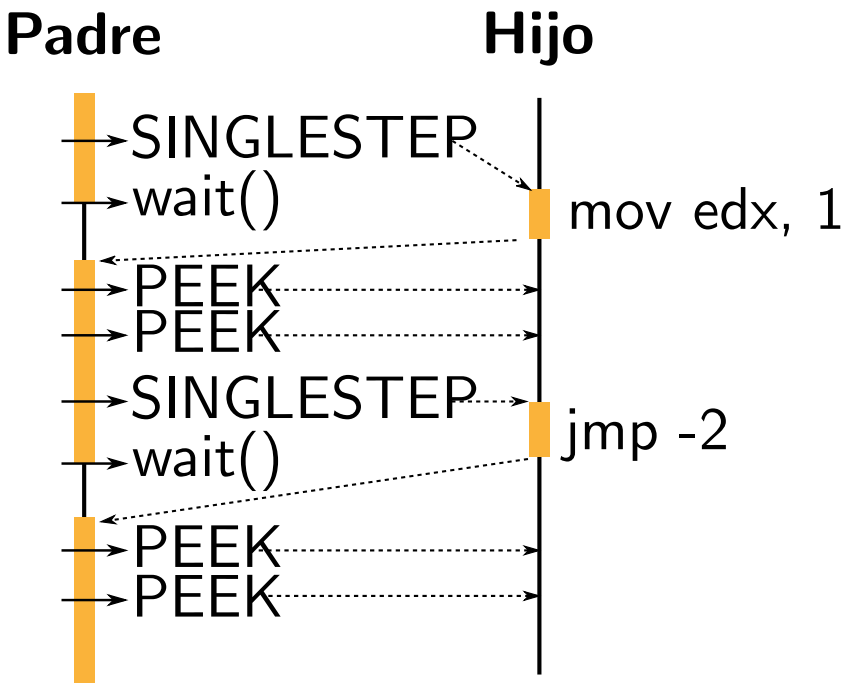
ptrace(): Esquema de uso - Obteniendo datos



Mientras el proceso hijo está detenido, se pueden obtener y modificar datos con

- `PTRACE_PEEKDATA`,
- `PTRACE_POKEDATA`,
- `PTRACE_PEEKUSER` y
- `PTRACE_POKEUSER`.

ptrace(): Esquema de uso - *Debugger*



Un debugger puede usar `PTRACE_SINGLESTEP` para ejecutar paso a paso cada instrucción.