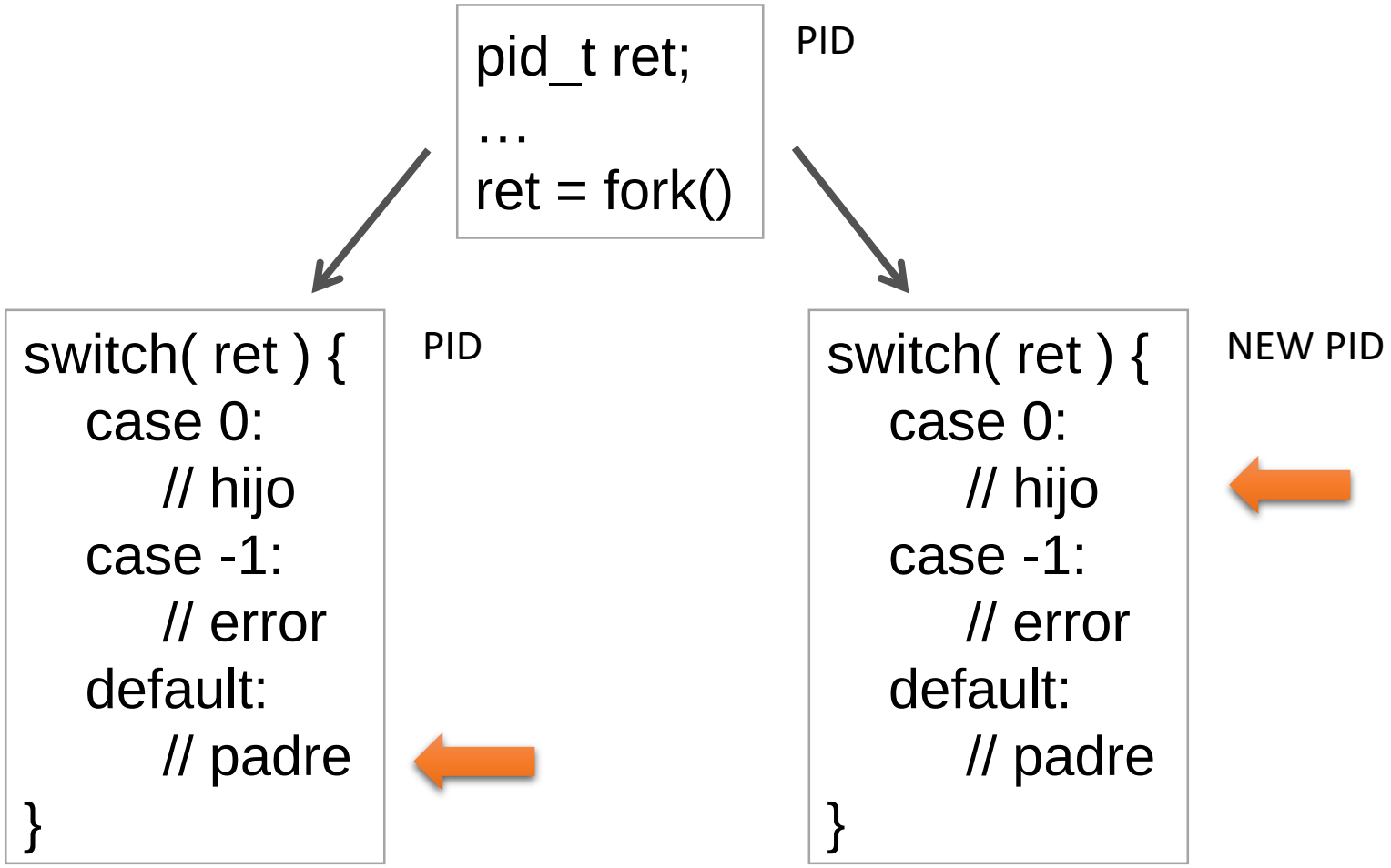
Creación de procesos

- En Unix: system call fork().
- Genera un nuevo proceso que es exactamente una copia del proceso que lo invoca.
- El nuevo proceso (hijo) tiene un nuevo PID.
- Al proceso ya existente (padre) fork() le retorna el PID del nuevo proceso.
- Al nuevo proceso hijo, fork() le retorna 0.
- En caso de error, retorna -1 y setea errno. No hay hijo.

fork()


pid_t ret;
…
ret = fork()
PID
switch(ret) {
case 0:
// hijo
case -1:
// error
default:
// padre
}
PID
switch(ret) {
case 0:
// hijo
case -1:
// error
default:
// padre
}
NEW PID

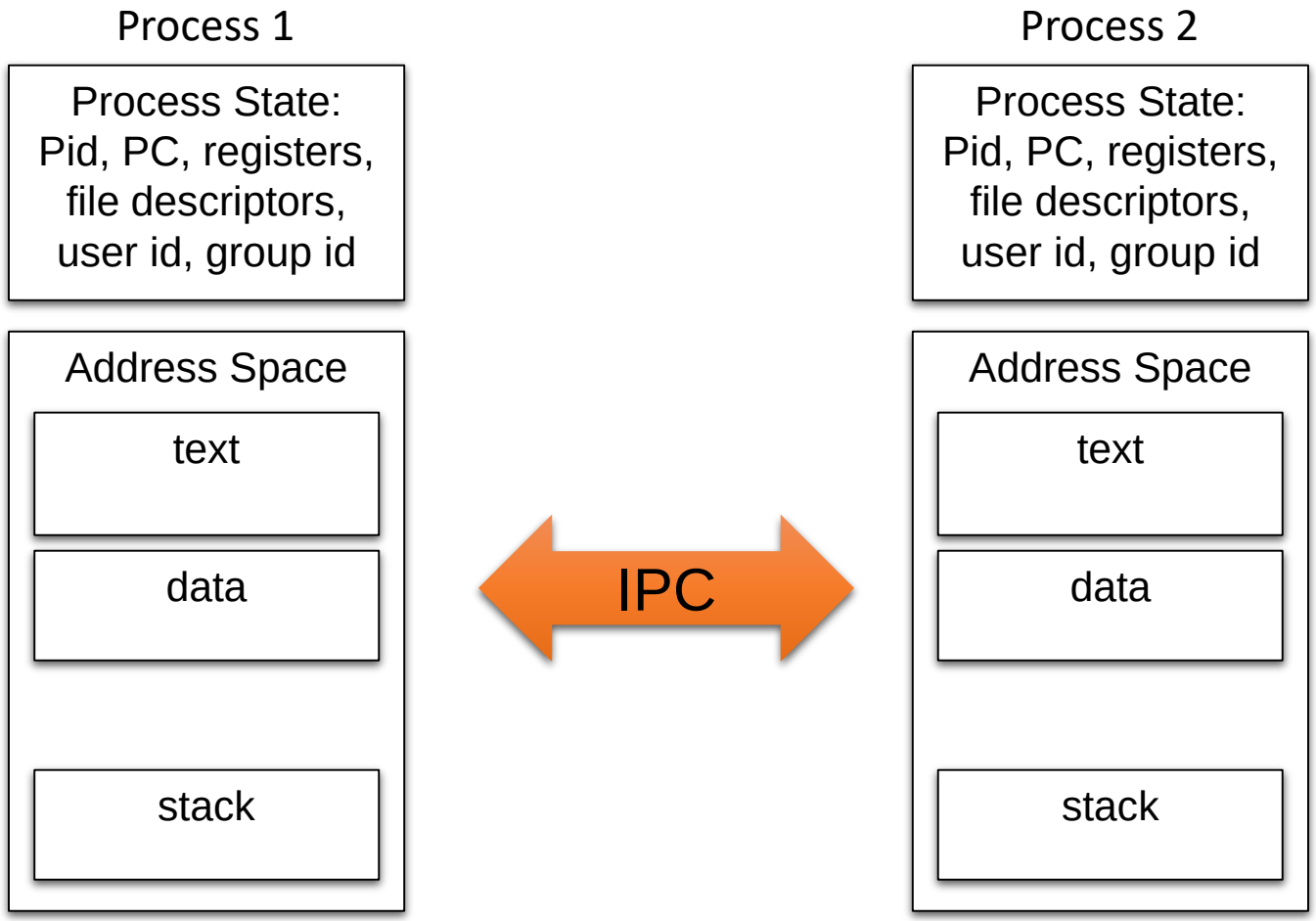
fork()

- Cuando un hijo termina, informa a su padre el estado de salida del proceso. El valor de retorno de main, “maquillado”.
- Si el padre no se da por enterado de la muerte de un hijo, el hijo queda en estado “zombie”.
- El padre espera por eventos de sus hijos con el syscall wait().

execve()

- Reemplaza el proceso que se está ejecutando por un nuevo programa, especificado por nombre de archivo.
- En caso de error, retorna -1.
- En caso de éxito, no retorna.
- El programa tiene que ser un binario de formato reconocido o un script.
- Los script empiezan con:
  - #! interprete [argumentos opcionales]
- Familia de funciones convenientes en libc:
  - execl, exclp, execle, execv, execvp

Distintos procesos



signals

- Mecanismo muy básico de IPC.
- Se envía una señal de un proceso a otro. La señal está definida por un entero positivo chico.
- Números de señales predefinidos.
- Muchas señales son enviadas automáticamente por el sistema operativo ante ciertos eventos.
- Un proceso puede enviar señales a otros programáticamente, usando el syscall kill().
- Los procesos pueden elegir que hacer cuando les llega una señal determinada (no todas) con el syscall sigaction(). Handling asincrónico.

signals

- No hay que usar el syscall signal() para cambiar el handling de las señales. Tiene comportamiento distinto según el flavor de unix. Usar sigaction().
- Para enviar señales, además de kill() existen:
  raise, killpg, pthread_kill, tgkill, sigqueue
- pause() y sigsuspend() sirven para esperar por señales.
- Las señales se pueden bloquear (enmascarar) con sigprocmask() y pthread_sigmask().
- No todas las funciones pueden ser llamadas desde un handler de señales.
- Los syscalls pueden ser interrumpidos por señales.

pipes

- pipe: canal de comunicación unidireccional.
- pipe() retorna 2 file descriptors que son la puntas del caño.
- Por un fd se escribe, por el otro se lee.
- El buffer está implementado en el kernel.
- Tratamiento standard sobre los 2 file descriptors: read, write, close, fcntl, select.
- SIGPIPE signals.

pipes

```
int p[2];
…
ret = pipe(p);
```

pipes

- Poco sentido usar pipe dentro de un único proceso (aunque puede convertirse en una buena forma, selectable, de comunicar comandos entre distintos threads.)

- Muy usado para comunicar un proceso padre con un proceso hijo. (pipe4.c)

- Establece una comunicación unidireccional. Para tener bidireccionalidad, hay que crear 2 pipes. (pipe5.c)