Shell con control de tareas

v.20220504

Sistemas Operativos

Grados Ingeniería de Computadores, Informática y Software - Grupos C

Depto. de Arquitectura de Computadores
Universidad de Málaga

El terminal

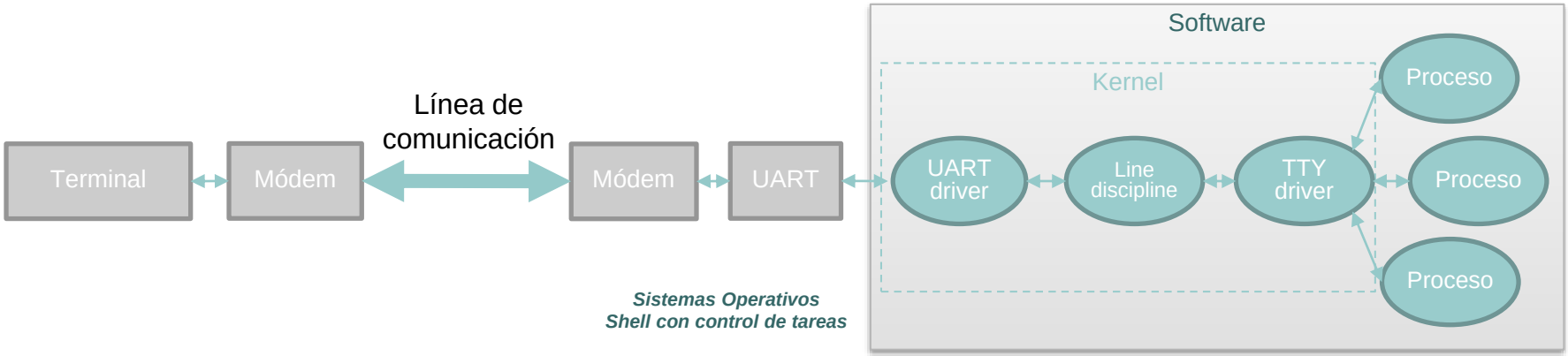
- **Historia**
  - 1869: *stock ticker* → precursor del teletipo
    - Máquina de escribir conectada por cable a una impresora
    - Propósito: distribuir precios de acciones a larga distancia en tiempo real
  - Teletipo (TTY): comienzos del siglo XX
    - Basado en ASCII
    - Conectados por todo el mundo:
      - Red Telex: red conmutada similar a la telefónica
    - Usados para comunicación de información:
      - Prensa
      - Interna de gobierno, policía, uso militar
      - Industria

Fuente: **The TTY demystified,** Linus Åkesson, http://www.linusakesson.net/programming/tty/index.php

El terminal

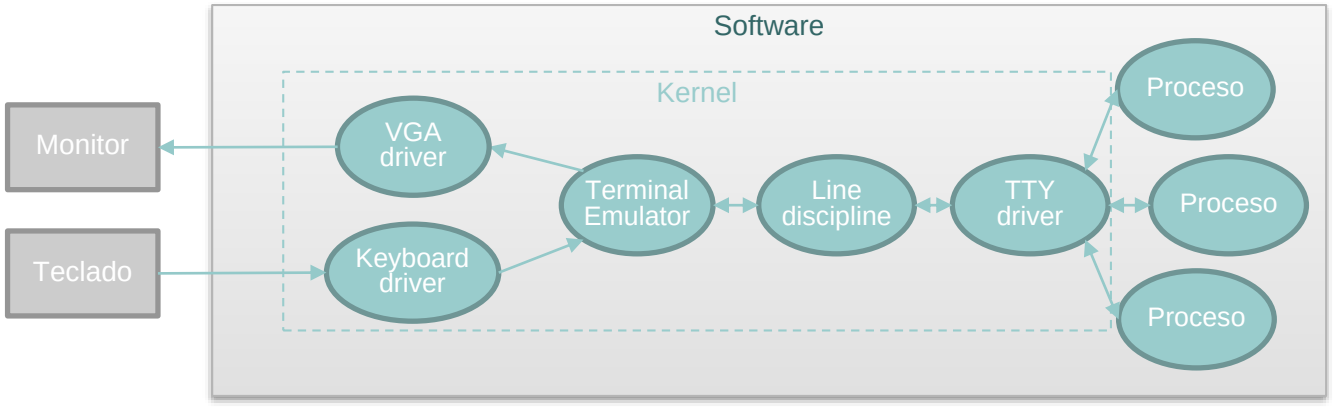
- **Historia**
  - Con la aparición de la 3ª generación de computadores (1965-1971) empieza a introducirse la interacción con usuarios en tiempo real
  - Primero se utilizan teletipos y luego terminales con pantalla y teclado
    - *UART* (*Universal Asynchronous Receiver-Transmitter*): Se utiliza para la comunicación serie entre el terminal y el computador
    - *Line discipline*: interpreta ciertos caracteres (borrar, imprimir, ^C, ^Z,…)
    - *TTY driver*: manda señales a procesos,…

Línea de comunicación serie



El terminal

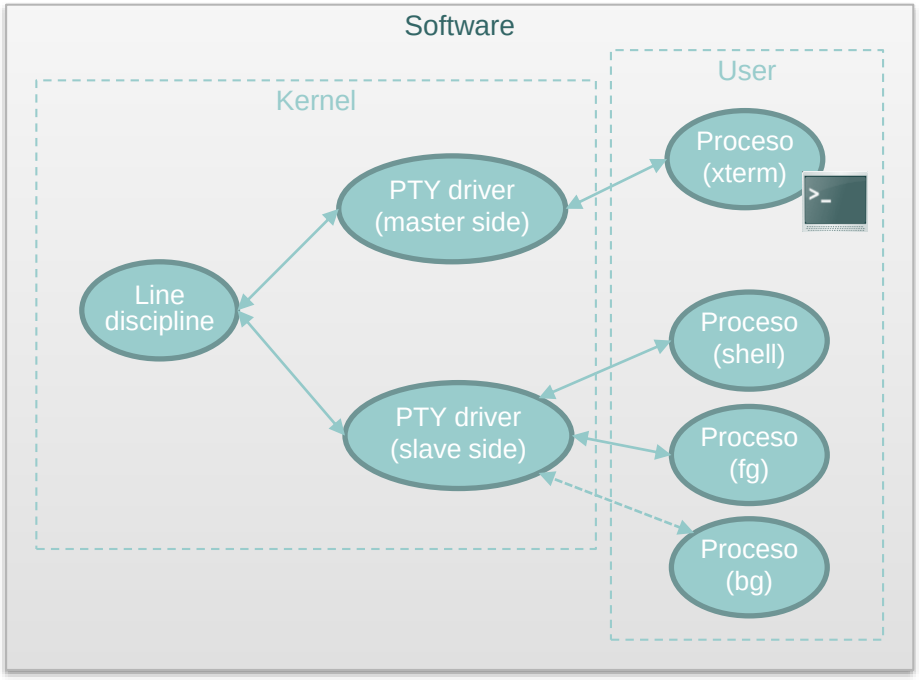
- **En Linux:**
  - Normalmente sin terminal serie físico → Ahora se **emula** (*frame buffer* + máquina de estados)
  - Line discipline y TTY driver se mantienen. En un emulador de terminal la UART pierde sentido (aunque se pueden ver los baudios: `stty -a`)
  - **TTY device**: Terminal emulator + Line discipline + TTY driver
    - Ejemplo: La consola de linux número <N> se activa con `Ctrl+Alt+F<N>` y está ligada a `/dev/tty<N>` con un proceso `agetty` asociado para realizar el login
  - Comando `tty:` para conocer el *TTY/PTY* del terminal actual



Fuente: **The TTY demystified,** Linus Åkesson, http://www.linusakesson.net/programming/tty/index.php

El terminal

- **En un GUI linux:**
  - Usamos una aplicación de emulación de terminal (`xterm`, `tilix`, `gnome-terminal`, `lxterminal`, ...)
  - **Pseudo-terminal (PTY)**: Terminal llevado al espacio de usuario
    - Lado *master*: conectado al emulador de terminal
    - Lado *slave* (`/dev/pts/#`): conectado generalmente a un *shell (*y a sus *jobs)*
  - Señal SIGHUP enviada al líder de sesión cuando se cierra un terminal (o cuando se desconecta la UART de un tty) (véase comando nohup)



Fuente: **The TTY demystified,** Linus Åkesson, http://www.linusakesson.net/programming/tty/index.php

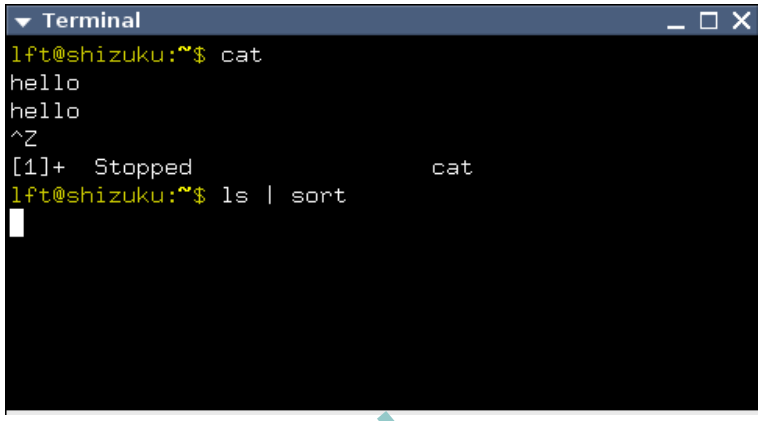
Sesiones y grupos de procesos

- Cuando se abre un (emulador de) terminal:
  - Los procesos tienen asociado un ID de sesión (SID) y un ID de grupo de proceso (***process group***) (PGID).
    - Estos identificadores son abstracciones para gestionar el control de trabajos en un terminal por parte de un *shell.*
    - Al crear un proceso con `fork()`, el proceso creado hereda ambos identificadores de su proceso padre.
    - El identificador de sesión o de grupo es el pid de alguno de los procesos miembros. A ese proceso se le denomina **líder** de sesión o de grupo.
  - El identificador de sesión (sid) debe ser el mismo para todos los procesos que compartan terminal (lado *slave*).
  - En nuestra práctica el líder de sesión es el *shell* y todos sus *jobs* pertenecen a la misma sesión.
  - No obstante los procesos hijos y descendiente del shell podrán pertenecer a diferentes grupos.

Grupos de procesos

- El grupo de procesos se introduce para facilitar el control de tareas:
  - ***Job o tarea****:* conjunto de procesos (descendientes del *shell*) con el mismo PGID
  - Ejemplo: cuando el usuario pulsa ^Z se envía SIGTSTP al grupo/tarea/job
- Los procesos pueden cambiar de grupo (`setpgid`):
  - El proceso líder puede expulsar a un proceso de su grupo a un grupo propio. El expulsado forma su propio grupo y será su líder.
  - Cualquier proceso puede irse se un grupo y formar grupo propio convirtiéndose en su líder.
- Asignación del terminal:
  - Decimos que el terminal está asignado a un grupo de procesos.
  - Sólo el grupo propietario de ese terminal puede hacer operaciones con él (leer, escribir, asignar, …).
  - Si un proceso de un grupo que NO posee el terminal intenta hacer una operación recibirá SIGTTIN/SIGTTOU, que por defecto hará que se suspenda

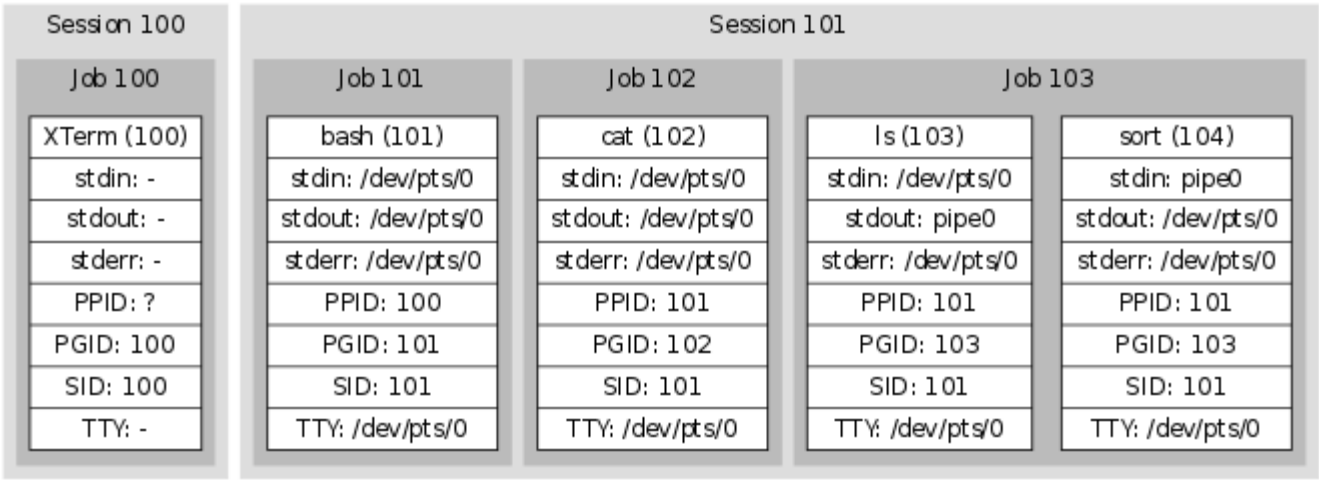
Terminal, Sesión y SHELL



Estructuras del kernel

- TTY Driver (/dev/pts/0):
  - Size: 45x13
  - Controlling process group: (101)
  - Foreground process group: (103)
  - UART configuration (ignored, since this is an xterm): Baud rate, parity, word length and much more.
  - Line discipline configuration: cooked/raw mode, linefeed correction, meaning of interrupt characters etc.
  - Line discipline state: edit buffer (currently empty), cursor position within buffer etc.
- Pipe0:
  - Readable end (connected to PID 104 as file descriptor 0)
  - Writable end (connected to PID 103 as file descriptor 1)
  - Buffer

Fuente: **The TTY demystified,** Linus Åkesson, http://www.linusakesson.net/programming/tty/index.php



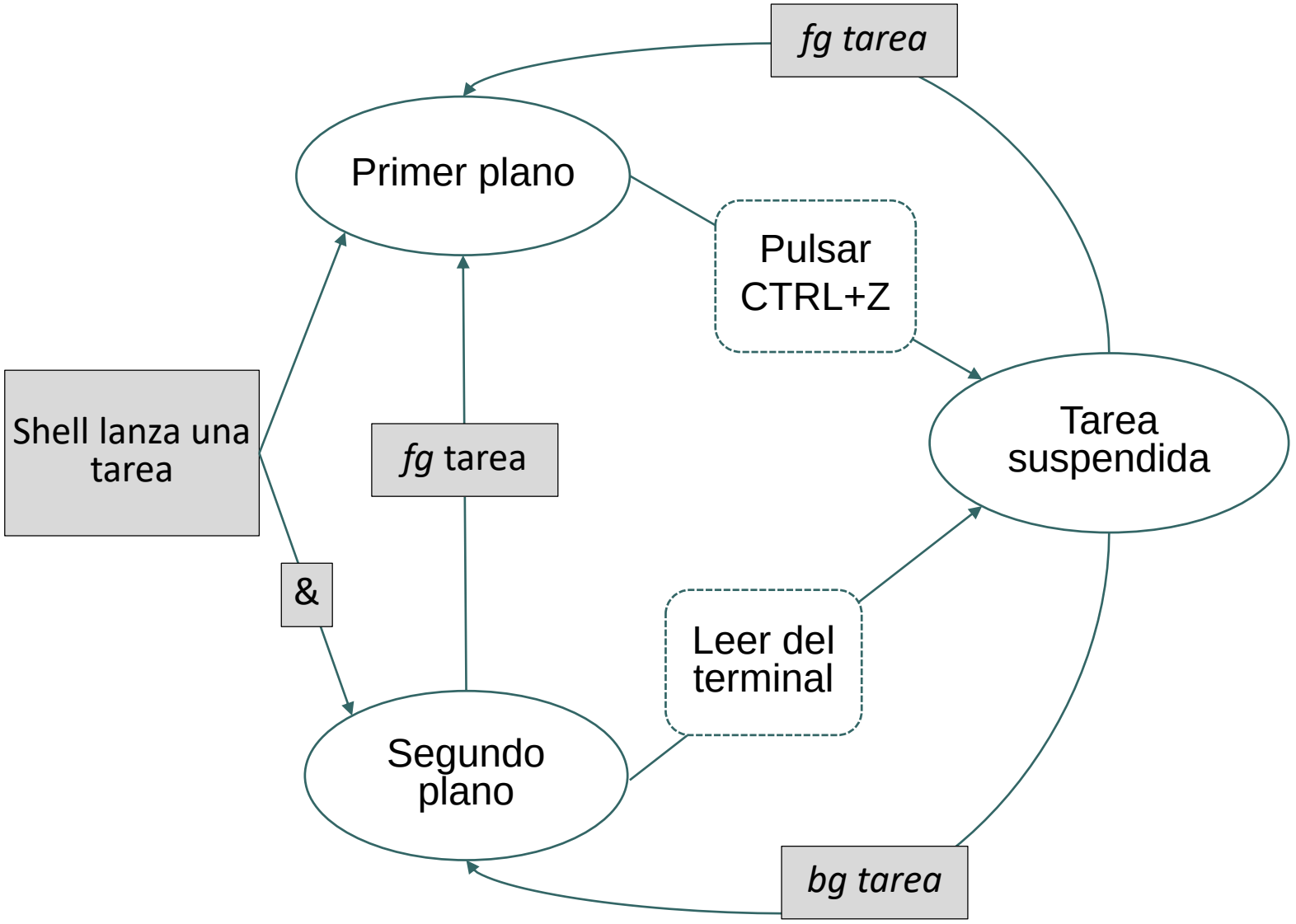
Control del terminal

- El **terminal** se asigna a un grupo de procesos (*job*)
- El *shell* **controla** qué tarea (*job*) accede al terminal en cada momento (`tcsetpgrp`)
- El grupo de procesos con el terminal es la tarea en **primer plano** (*fg*)
- Las demás que se ejecutan sin terminal se conocen como tareas en **segundo plano** (*bg*)
- El *shell* es la tarea en primer plano mientras no hay proceso en primer plano, es decir mientras lee los comandos
- El *shell* debe identificar a cada tarea (`setpgid`)

Control de Tareas

- El *shell* debe:
  - Mantener una **lista de tareas** (*job list*)
    - Corriendo en segundo plano: pueden ser varias
    - Suspendidas (*stopped*): pueden ser varias
      - Puede ser la de primer plano (^Z)
      - Pueden suspenderse las de segundo plano (`kill -STOP`)
  - Controlar el estado de las tareas
    - Sistema de señales: permite controlar los cambios de estado
      - SIGCHLD: notifica al *shell* si un hijo se suspende, continúa o termina
        - Instalar manejador (`signal(SIGCHLD,manejador)`)
    - Comandos internos:
      - `fg` y `bg`: permiten cambiar de plano las tareas
      - `jobs`: permite listar las tareas en segundo plano

Diagrama de control de tareas



Terminal y Señales

- Señales generadas por el *TTY driver*:
  - Provocadas tras el *parsing* del *line discipline*:
    - **SIGINT**: carácter INTR (^C). *Interrupt* desde terminal
    - **SIGQUIT**: carácter QUIT (^\). Como ^C + *core dump*
    - **SIGTSTP**: carácter STOP (^Z). Suspender desde terminal
  - Provocadas por procesos:
    - **SIGTTIN**: si un proceso de un *job* en segundo plano intenta leer del TTY, el TTY driver envía esta señal a todo el *job*
      - Acción por defecto: suspensión (*stopped*)
    - **SIGTTOU**: si un proceso de un *job* en segundo plano intenta escribir en el TTY, el TTY driver envía esta señal a todo el *job*
      - Acción por defecto: suspensión (*stopped*)
      - Se puede desactivar (`stty -tostop`)

Llamadas al Sistema

- A usar por el *shell* (se proporcionan *wrappers* para un uso más sencillo)
  - `setpgid(pid, pgid):`

```
#define new_process_group(pid)  setpgid (pid, pid)
```

    - Asigna un id de grupo (pgid) a un proceso
    - Se usa su propio pid para un nuevo pgid
    - Uso: siempre que creemos una tarea nueva
  - `tcsetpgrp(fd, pgid):`

```
#define set_terminal(pid)  tcsetpgrp(STDIN_FILENO,pid)
```

    - Asigna el terminal a un id de grupo
    - El terminal se identifica como un file descriptor
    - Uso:
      - Siempre que pasemos una tarea a *fg*
      - Siempre que una tarea *fg* termine o se suspenda

Llamadas al Sistema

- A usar por el *shell* (se proporcionan *wrappers* para un uso más sencillo)
  - `sigprocmask(how,set,oldset):`

```
#define block_SIGCHLD()       block_signal(SIGCHLD, 1)
#define unblock_SIGCHLD()     block_signal(SIGCHLD, 0)
void block_signal(int signal, int block)
{
    sigset_t block_sigchld;
    sigemptyset(&block_sigchld );
    sigaddset(&block_sigchld,signal);
    if(block) {
        sigprocmask(SIG_BLOCK, &block_sigchld, NULL);
    } else {
        sigprocmask(SIG_UNBLOCK, &block_sigchld, NULL);
    }
}
```

    - Enmascara señales
    - Uso: para proteger la modificación de la lista de *jobs*

Llamadas al Sistema

- A usar por el *shell* (para las ampliaciones del *shell* básico)
  - `fileno(FILE *stream):`
    - Devuelve el número de descriptor de fichero correspondiente al *stream*
    - Uso: para pasar los parámetros de `dup2` y `pipe`
  - `dup2(int oldfd, int newfd):`
    - Hace que la entrada `newfd` de la tabla de ficheros del proceso apunte al fichero `oldfd`
    - Uso: para implementar la redirección y el pipe
      - Ej. `ls -la > listado.txt` Si `newfd` es `fileno(stdout)` y `oldfd` apunta a `listado.txt`, tras hacer `dup2` podemos hacer `fork` y `exec` de `ls` y el hijo heredará la tabla de descriptores de fichero vertiendo el resultado en el archivo en lugar de en `stdout`
  - `pipe(int pipefd[2]):`