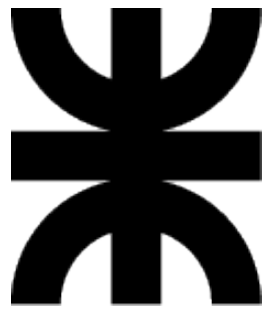
Informática II
La shell de Linux (4/4)

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Procesos

Un proceso es una instancia de un programa en ejecución

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el Kernel:

- carga el código del programa en memoria,

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el Kernel:

- carga el código del programa en memoria,
- reserva espacio para las variables del programa, e

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el Kernel:

- carga el código del programa en memoria,
- reserva espacio para las variables del programa, e
- inicializa estructuras de datos propias del Kernel para guardar información del proceso (PID, estado, IDs de usuario y grupo, etc.).

Procesos

Un proceso es una instancia de un programa en ejecución

Cuando se ejecuta un programa (`./a.out`, `ls`, `gcc`, etc.), el Kernel:

- carga el código del programa en memoria,
- reserva espacio para las variables del programa, e
- inicializa estructuras de datos propias del Kernel para guardar información del proceso (PID, estado, IDs de usuario y grupo, etc.).

El estándar UNIX (IEEE Std 1003.1-2004) define un proceso como:

> *"an address space with one or more threads executing within that address space, and the required system resources for those threads"*

Procesos

- Cada proceso tiene asociado un identificador único (entero positivo), PID (process identifier)
  `> ps -u`
- Cada proceso tiene asociado un identificador del proceso padre, PPID (parent process ID)
- El nro. de proceso 1 está reservado para el proceso `init/systemd`
  Ver el comando:
  `> ps -e | head`

Procesos

- Cada proceso tiene asociado un identificador único (entero positivo), PID (process identifier)
  `> ps -u`
- Cada proceso tiene asociado un identificador del proceso padre, PPID (parent process ID)
- El nro. de proceso 1 está reservado para el proceso `init/systemd`
  Ver el comando:
  `> ps -e | head`

Probar los comandos:

- `ps` con opciones `-u`, `-e`, `-f`, `-H`
- `man ps`
- `top`
- `htop`
- `pidof` (p.e. `> pidof bash`)
- `pstree`

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

heap : área desde la cual un programa puede asignar memoria adicional de forma dinámica

Procesos

Un proceso se divide lógicamente en las siguientes partes, conocidas como *segmentos*:

text : instrucciones de un programa

data : variables estáticas utilizadas por un programa

heap : área desde la cual un programa puede asignar memoria adicional de forma dinámica

stack : porción de memoria que crece o se encoje a medida que las funciones se llaman y retornan (almacenar las variables locales de funciones)

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Segundo plano: Permite la ejecución de otros comandos mientras se ejecuta el proceso (`&`)

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Segundo plano: Permite la ejecución de otros comandos mientras se ejecuta el proceso (`&`)

Combinación de teclas:

- `Ctrl+C`: interrumpe la ejecución del proceso
- `Ctrl+Z`: suspende la ejecución del proceso

Procesos

Ejecución de procesos

Primer plano: Ejecución normal, la shell no admite otro comando hasta que haya terminado el proceso en ejecución

Segundo plano: Permite la ejecución de otros comandos mientras se ejecuta el proceso (`&`)

Combinación de teclas:

- `Ctrl+C`: interrumpe la ejecución del proceso
- `Ctrl+Z`: suspende la ejecución del proceso

Probar lanzar procesos en segundo plano (kate, kcalc, etc.).

Procesos

- Ejecutar el comando `yes` y detener con `Ctrl+C`
- Luego ejecutar con redirección (`yes >/dev/null`) y:

Procesos

- Ejecutar el comando `yes` y detener con `Ctrl+C`
- Luego ejecutar con redirección (`yes >/dev/null`) y:
  - Suspender, `Ctrl+Z`
  - Enviar a 2do plano, `bg`
  - Enviar a 1er plano, `fg`
  - Detener, `Ctrl+C`

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones
- Ejemplo de señal: *caracter de interrupción* Ctrl-C

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones
- Ejemplo de señal: *caracter de interrupción* Ctrl-C

Ver comando `kill` (`man 7 signal`)

Señales

Es un mecanismo de comunicación entre procesos, IPC (Interprocess Communication)

- Es un evento generado por un sistema UNIX o Linux
- Se describen a menudo como *interrupciones de software*
- El arribo de una señal le indica al proceso que ha ocurrido un evento o condición excepcional
- Hay varios tipos de señales, cada una de las cuales identifican diferentes eventos o condiciones
- Ejemplo de señal: *caracter de interrupción* Ctrl-C

Ver comando `kill` (`man 7 signal`)

Terminar/matar un proceso:

1. Identificar el proceso utilizando el comando `ps`
2. Enviarle una señal con el comando `kill`

Señales en lenguaje C

Archivo fuente `signal1.c`

```c
#include <stdio.h>
#include <unistd.h>

int main(void)
{
  printf("Iniciando bucle...\n");
  while(1)
  {
    printf("Corriendo.\n");
    sleep(1);
  }
  printf("Terminando bucle...\n");

  return 0;
}
```

Señales en lenguaje C

Archivo fuente `signal2.c`

```c
#include <stdio.h>
#include <unistd.h>


unsigned int flag = 1;
void handler(int );

int main(void) {

  printf("Iniciando bucle...\n");
  while(flag)
  {
    printf("Corriendo.\n");
    sleep(1);
  }
  printf("Terminando bucle...\n");

  return 0;
}

void handler(int sig)
{ flag = 0; }
```

Señales en lenguaje C

Archivo fuente `signal2.c`

```c
#include <stdio.h>
#include <unistd.h>
#include <signal.h>

unsigned int flag = 1;
void handler(int );

int main(void) {
  signal(SIGINT, handler);
  printf("Iniciando bucle...\n");
  while(flag)
  {
    printf("Corriendo.\n");
    sleep(1);
  }
  printf("Terminando bucle...\n");

  return 0;
}

void handler(int sig) // manejador de la señal
{ flag = 0; }
```