Sistemas Operativos (75.08)

Primer cuatrimestre 2022 - FIUBA

Temas administrativos
(la parte aburrida)

Checklist

- ✓ Página de la materia: fisop.github.io
  - Completar el formulario de alta!
  - Adquirir el calendario de la materia
- ✓ 2 labs + 4 TPs + 1 parcial
- ✓ Calendario con fechas importantes en la página
- ✓ Lista de correo:
  - fisop-consultas@googlegroups.com
- ✓ Consultas administrativas a:
  - fisop-doc@googlegroups.com
- ✓ **Discord de la materia!**
  - https://discord.gg/2xSsTKdbsq
  - Al unirse, cambiar el nick para que los podamos reconocer y agregarlos.

Régimen de cursada

- ✓ 2 labs **individuales**
- ✓ 4 TPS **grupales (dos personas)**
  - Acompañado de un parcialito individual por TP
- ✓ 1 parcial individual
  - En la semana 12 de la cursada

- ✓ Régimen **regular**
  - Aprobar todas las instancias
  - Rendir un **coloquio**
- ✓ Régimen **promoción**
  - Calificación de tp y lab >= 7
  - Promedio de la cursada >= 8
  - Nota de cursada como nota de la materia
- ✓ Final alternativo
  - Reemplaza el coloquio con desafíos

Intro a entorno Unix

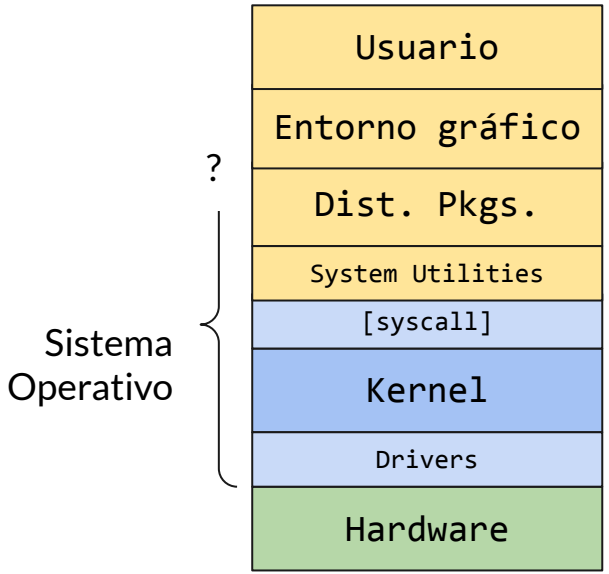
Un muy breve recorrido por la terminal

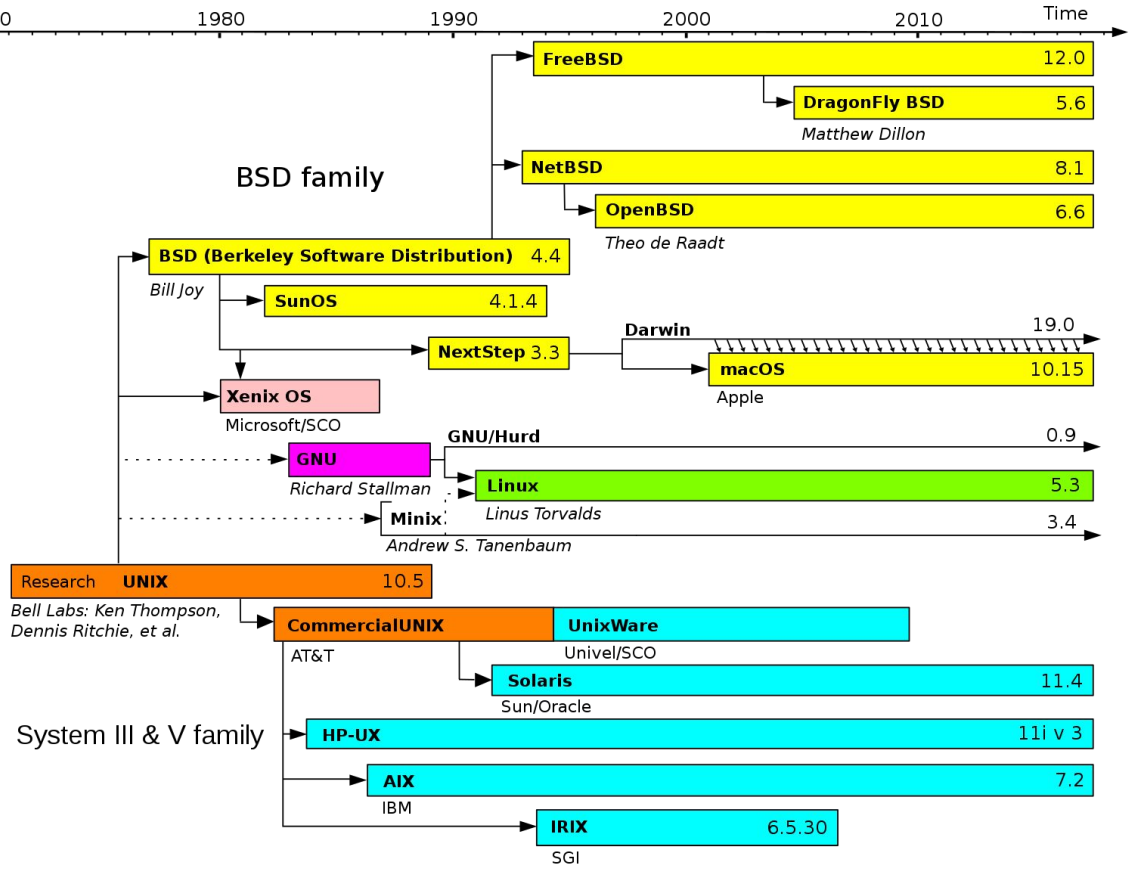
- Sistema Operativo
- Kernel
- UNIX
- Linux
- GNU
- BSD

- Ubuntu
- Debian
- Windows
- POSIX
- Línea de comandos/Shell
- Drivers

¿Qué quiere decir Unix?

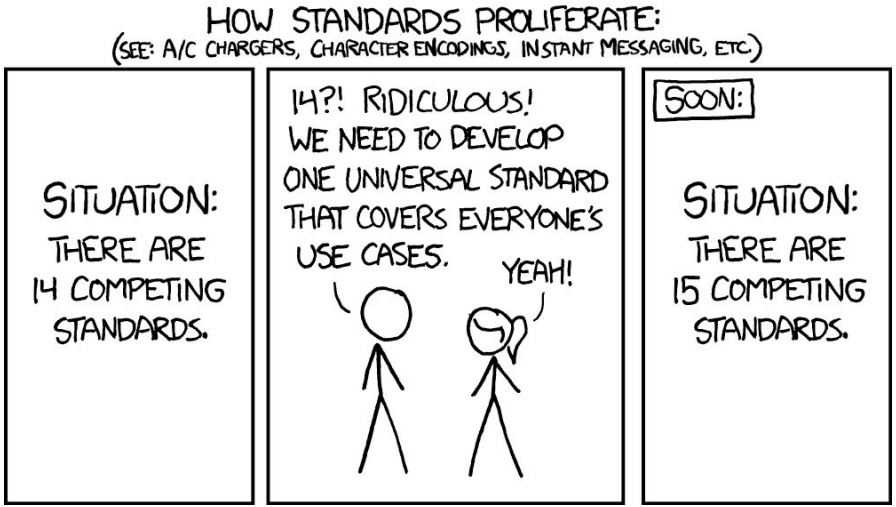
- Es una *familia* de sistemas operativos
  - **Descendientes de UNIX**
  - De propósito general
- Varios componentes:
  - Un kernel
  - Una línea de comandos (*shell*)
  - Librerías y *headers*
  - Utilidades del sistema: **ls, ps, find, grep, sed**
- Filosofía Unix




1970
1980
1990
2000
2010
Time
FreeBSD 12.0
DragonFly BSD 5.6
Matthew Dillon
BSD family
NetBSD 8.1
OpenBSD 6.6
Theo de Raadt
BSD (Berkeley Software Distribution) 4.4
Bill Joy
SunOS 4.1.4
Darwin
19.0
NextStep 3.3
macOS 10.15
Apple
Xenix OS
Microsoft/SCO
GNU/Hurd
0.9
GNU
Richard Stallman
Linux 5.3
Linus Torvalds
Minix
3.4
Andrew S. Tanenbaum
Research UNIX 10.5
Bell Labs: Ken Thompson,
Dennis Ritchie, et al.
CommercialUNIX
UnixWare
Univel/SCO
AT&T
Solaris 11.4
Sun/Oracle
System III & V family
HP-UX 11i v 3
AIX 7.2
IBM
IRIX 6.5.30
SGI

Estándares POSIX y SUSv4

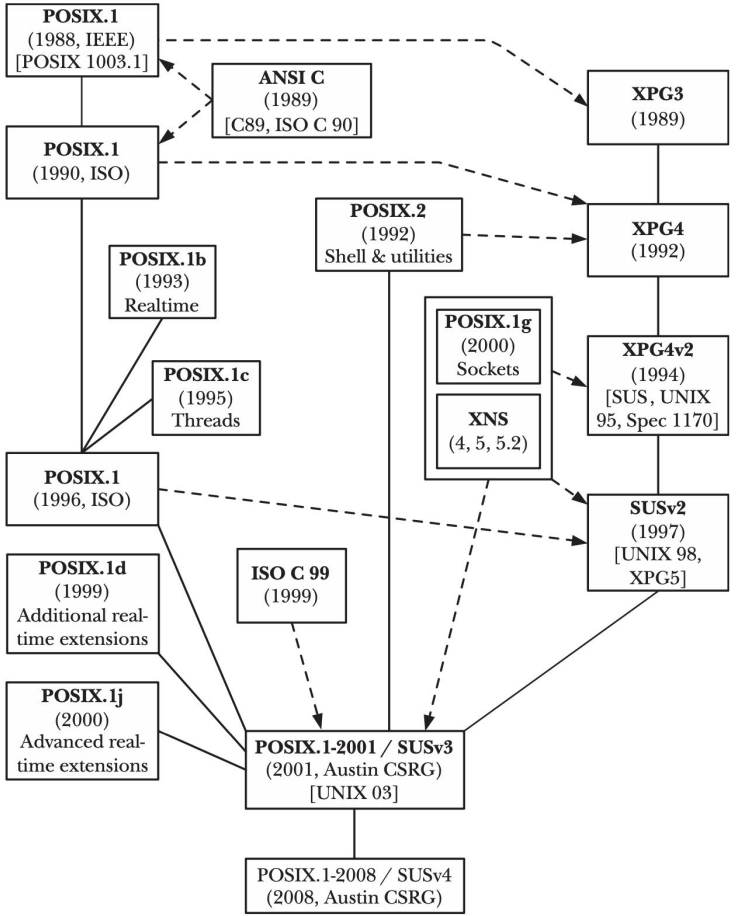
- Portable Operating System Interface y Single Unix Specification v4
- Principios
  - Application-Oriented
  - Interface, Not Implementation
  - **Source, Not Object, Portability**
  - Minimal Interface, Minimally Defined
- Unificados en el mismo estándar
  - SUSv4 es un superset de POSIX



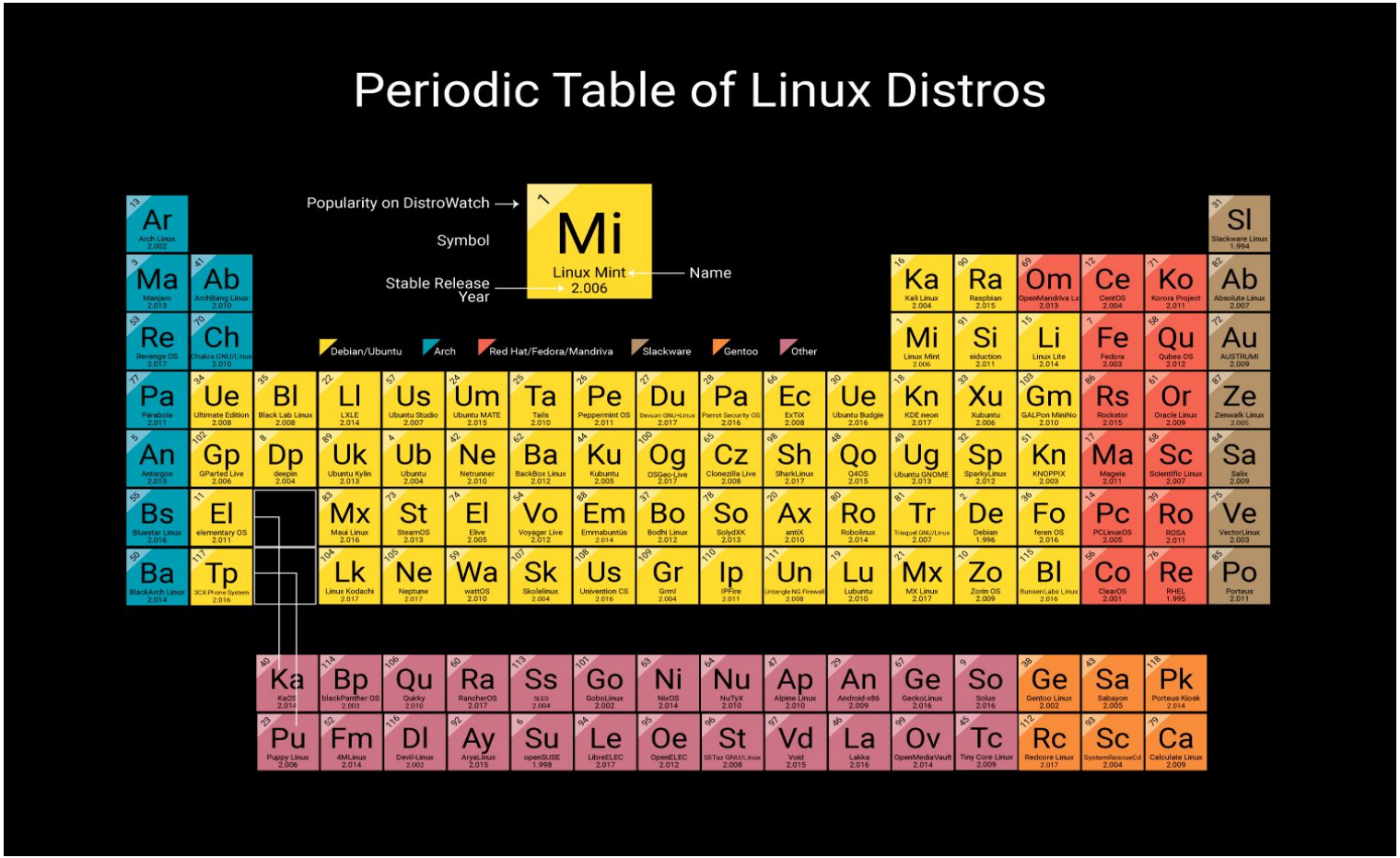
POSIX y SUSv4

- Evolución de los estándares
- *Incorporan* el estándar de C
- *The Open Group* es el dueño del “nombre UNIX”

THE Open GROUP

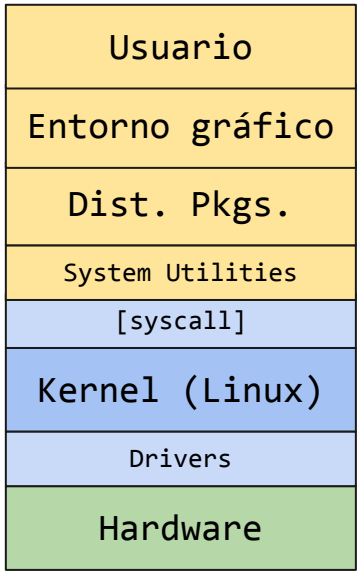


Periodic Table of Linux Distros



Distribuciones basadas en Linux

- Basados en el **kernel de Linux**
  - Incluyendo utilidades en común
  - ***mayormente*** POSIX
- Diferentes aplicaciones
  - Manejadores de paquetes
  - Entornos gráficos
- Ej: Ubuntu, Debian, Fedora, RedHat, LinuxMint, y muchas más
- *If* Stallman *then* GNU/Linux



¿Por qué línea de comandos?

Pros:

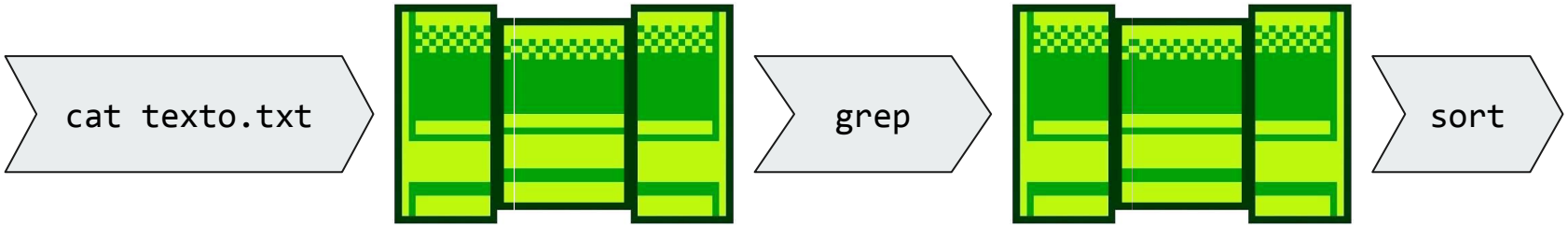
- Es **estándar** en cualquier sistema Unix-like (POSIX)
- Facilita **scripting y automatización**
- Muchas utilidades
  - *“do one thing and do it well”*
- Más **rápido** con menos recursos
- Extensible y configurable

Cons:

- Curva de aprendizaje
- Estética
- No se puede hacer todo
  - Browsers, IDE

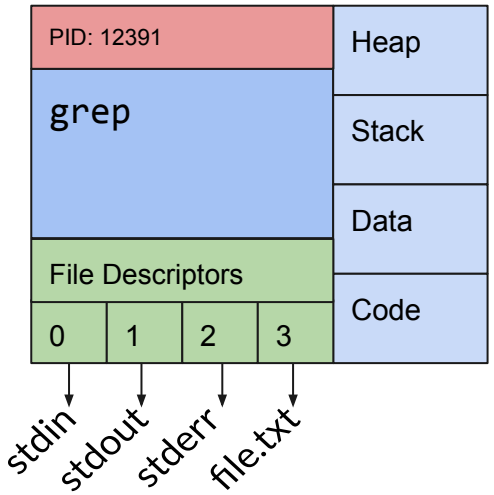
Filosofía Unix

- Utilidades **pequeñas** que realizan tareas **específicas**
- **Piping**: La **salida** de un programa es la **entrada** de otro
- Una **línea de comandos** que **crea procesos**



El proceso

- Una “instancia” de un programa
- Tiene su propia memoria
  - Heap, stack, code, etc
- Tiene un identificador único
  - PID
- Tiene un conjunto de “archivos abiertos”
  - Descriptores de archivo



¿Dónde estoy? ¿Quién soy?

`pwd`	Muestra directorio actual de trabajo
`whoami`	Muestra el usuario actual
`uname`	Muestra información del sistema operativo
`whereis`	Encuentra dónde reside un programa
`who`	Muestra quién está logueado

Navegación y archivos

`ls`	Muestra contenidos del directorio actual
`cd`	Cambia el directorio actual
`mkdir`	Crea un directorio
`touch`	Crea un archivo
`file`	Determina el tipo de un archivo
`mv`	Mueve o renombra archivos (y directorios)

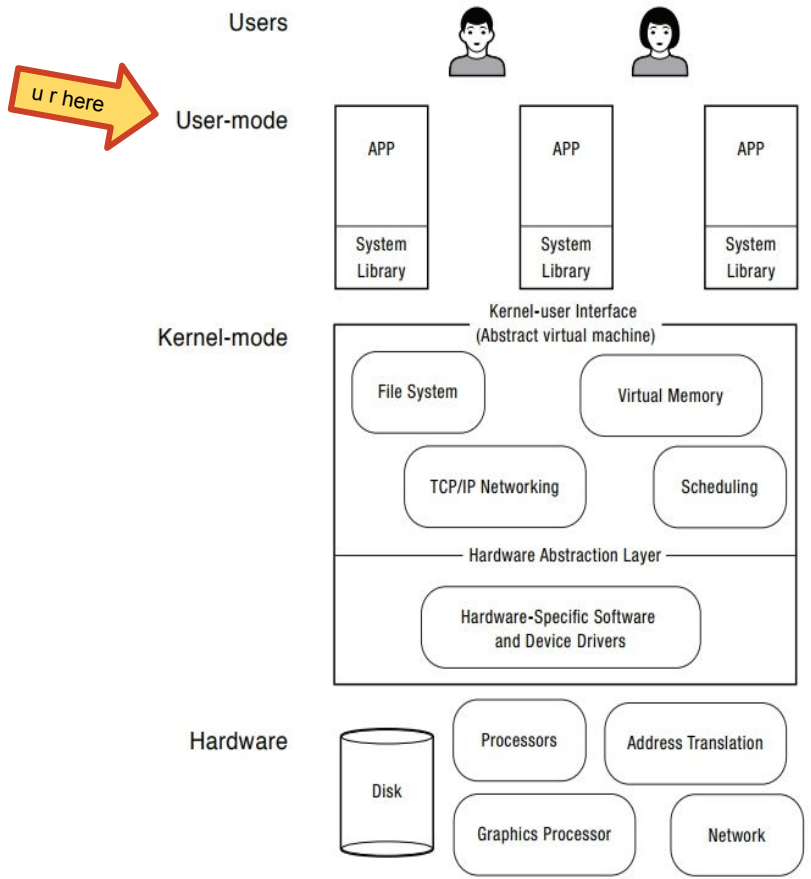
¡Ayuda!

`man`	Páginas de manual (referencia)
`info`	Páginas de info (más completas)
`whatis`	Breve descripción de lo que hace un comando
`apropos`	Búsqueda de páginas de manual por palabras clave
`history`	Muestra historial de comandos

Lab fork
Primera parte

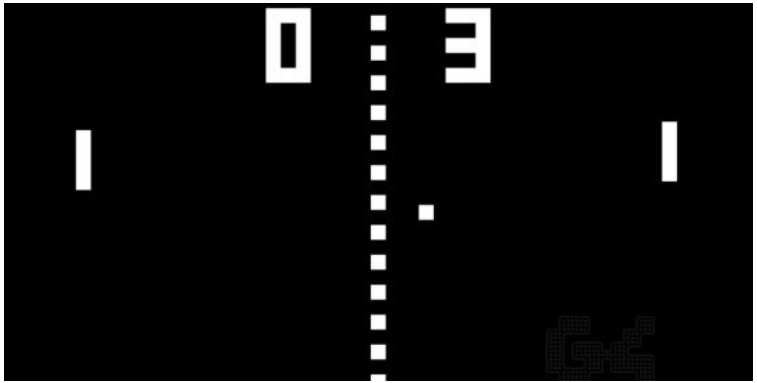
Lab fork

- Creación de utilidades Unix like
- Punto de vista de un usuario (user-mode)
- Utilizaremos **funcionalidades provistas por el kernel**



Tarea: ping-pong

- Dos procesos: padre e hijo
- Proceso padre envía un mensaje al proceso hijo
- Proceso hijo recibe y contesta
- Ambos procesos terminan



- ¿Cómo se crea un nuevo proceso?
- ¿Cómo se comunican?

fork(2)

- Crea un nuevo proceso
- Idéntico **en todo sentido** al proceso que llama a fork
- **Excepto** por el valor que **retorna fork**
- Puede fallar!

fork(2) - ejemplo 0

- ¿Qué imprime este programa?
- ¿Qué *includes* son necesarios?

```
int main(int argc, char* argv[]) {
  printf("Mi PID es: %d\n", getpid());

  int i = fork();

  if (i < 0) {
    printf("Error en fork! %d\n", i);
    exit(-1);
  }

  if (i == 0) {
    printf("Soy el proceso hijo y mi pid es: %d\n", getpid());
  } else {
    printf("Soy el proceso padre y mi pid es: %d\n", getpid());
  }

  printf("Terminando\n");
  exit(0);
}
```

fork(2) - ejemplo 1

- ¿Qué imprime este programa?
- ¿Cuál es el valor de a *"a"* lo largo de la ejecución?
- ¿Y si imprimimos la dirección de la variable *"a"*?
- Ejercicio: hacer *malloc* y comparar punteros

```
int main(int argc, char* argv[]) {
  int a = 4;
  int i = fork();

  a = 5;

  if (i < 0) {
    printf("Error en fork! %d\n", i);
    exit(-1);
  }

  if (i == 0) {
    printf("[hijo] mi pid es: %d\n", getpid());
    printf("[hijo] a=%d\n", a);
  } else {
    a = 6;
    printf("[padre] mi pid es: %d\n", getpid());
    printf("[padre] a=%d\n", a);
  }

  printf("Terminando\n");
  exit(0);
}
```

fork(2) - ejemplo 2

- ¿Qué hace este programa?
- ¿Queda algún archivo abierto?
  - ¿Se “filtra” algún file descriptor?
- ¿Podría el proceso padre escribir luego de que el proceso hijo llame a *close*?

```
int main(int argc, char* argv[]) {
  char* msg = "fisop\n";

  // Abro un archivo y si no existe lo creo
  int fd = open("hola.txt", O_CREAT | O_RDWR, 0644);
  int i = fork();

  if (i < 0) {
    printf("Error en fork! %d\n", i);
    exit(-1);
  }

  if (i == 0) {
    printf("[hijo] mi pid es: %d\n", getpid());
    write(fd, msg, 6);
    close(fd);
  } else {
    printf("[padre] mi pid es: %d\n", getpid());
  }

  printf("Terminando\n");
  exit(0);
}
```

fork(2) - ejemplo 3

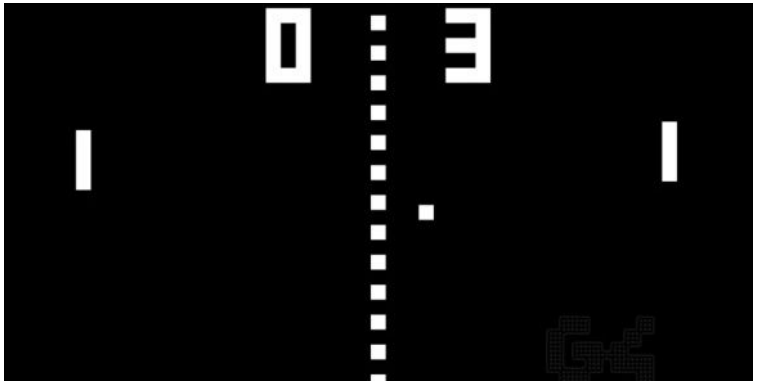
- ¿Cuántos procesos en total genera este código?
- ¿Fallará **fork()** en algún momento?

```
int main(int argc, char* argv[]) {
  printf("Mi PID es: %d\n", getpid());

  for (int i = 0; i < 12; i++) {
    int r = fork();

    if (r < 0) {
      perror("Error en fork");
      exit(-1);
    }

    printf("[%d] Hola!\n", getpid());
  }

  printf("Terminando\n");
  exit(0);
}
```

Tarea: ping-pong

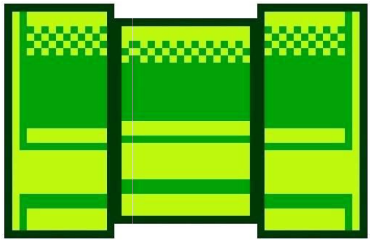
✅ Dos procesos: padre e hijo

- Proceso padre envía un mensaje al proceso hijo
- Proceso hijo recibe y contesta
- Ambos procesos terminan



- ~~¿Cómo se crea un nuevo proceso?~~
- ¿Cómo se comunican?

pipe(2)



- Crea un par de **descriptores de archivos** que están **conectados**
  - Archivo “virtual”
- El pipe es **unidireccional**
- La **lectura bloquea** hasta que haya algo que leer
- La **escritura bloquea** hasta que se pueda escribir (e.g. el pipe está lleno!)

pipe(2) - ejemplo 0

- ¿Qué hace este programa?
- ¿Qué ocurrirá si descomentamos el primer *read*?
- ¿Qué valores imprime el primer *print*? ¿Serán siempre los mismos?

```
int main(int argc, char* argv[]) {
  int fds[2];
  int msg = 42;

  int r = pipe(fds);
  if (r < 0) {
    perror("Error en pipe");
    exit(-1);
  }

  printf("Lectura: %d, Escritura: %d\n", fds[0], fds[1]);

  // read(fds[0], %msg, sizeof(msg)); // ????

  // Escribo en el pipe
  write(fds[1], &msg, sizeof(msg));

  int recibido = 0;
  read(fds[0], &recibido, sizeof(recibido));
  printf("Recibi: %d\n", recibido);

  close(fds[0]);
  close(fds[1]);
}
```

pipe(2) - ejemplo 1

- ¿Qué hace este programa?
- ¿Por qué se cierran algunos fds?
- ¿Qué ocurre si el proceso hijo llega a llamar a *read* antes que el padre llame a *write*?
- ¿Y al revés?

```
int main(int argc, char* argv[]) {
  int fds[2];
  int msg = 42;

  pipe(fds);
  int i = fork();

  if (i == 0) {
    printf("[hijo] mi pid es: %d\n", getpid());
    // El hijo no va a escribir
    close(fds[1]);

    int recv = 0;
    read(fds[0], &recv, sizeof(recv));
    printf("[hijo] lei: %d\n", recv);

    close(fds[0]);
  } else {
    printf("[padre] mi pid es: %d\n", getpid());
    // El padre no va a leer
    close(fds[0]);

    // Esperamos dos segundos, el hijo no deberia seguir
    sleep(2);
    write(fds[1], &msg, sizeof(msg));

    close(fds[1]);
  }
}
```

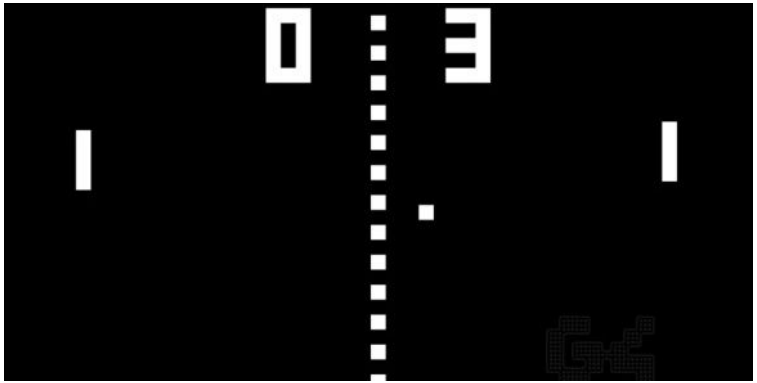
pipe(2) - ejemplo 2

- ¿Qué hace este programa?
- ¿Termina el algún momento? ¿Cómo?
- ¿Que contiene **pipe(7)**?

```
int main(int argc, char* argv[]) {
  int fds[2];

  pipe(fds);

  printf("Lectura: %d, Escritura: %d\n", fds[0], fds[1]);

  int msg = 42;
  int escritos = 0;
  while (1) {
    r = write(fds[1], &msg, sizeof(msg));
    if (r >= 0) {
      printf("Total escrito: %d\n", r, escritos);
      escritos += sizeof(msg);
    } else {
      printf("write fallo con %d\n", r);
      printf("errno was: %d\n", errno);
      perror("perror en write");
      break;
    }
  };

  close(fds[0]);
  close(fds[1]);
}
```

Tarea: ping-pong

- ✅ Dos procesos: padre e hijo
- ✅ Proceso padre envía un mensaje al proceso hijo
- ✅ Proceso hijo recibe y contesta
- ✅ Ambos procesos terminan



- ~~¿Cómo se crea un nuevo proceso?~~
- ~~¿Cómo se comunican?~~

pipe(2) - probar

- ¿Qué hace este programa?
- ¿Termina el algún momento?
- ¿Imprime algo?
- Tratar de explicar el comportamiento, con ayuda de **pipe(7)**
- Ayuda: correr el programa con **valgrind**

```
int main(int argc, char* argv[]) {
  int fds[2];

  pipe(fds);

  printf("Escritura: %d, lectura: %d\n", fds[0], fds[1]);

  close(fds[0]);

  int msg = 42;
  int escritos = 0;
  while (1) {
    printf("Voy a intentar escribir\n");
    r = write(fds[1], &msg, sizeof(msg));
    if (r >= 0) {
      printf("Total escrito: %d\n", r, escritos);
      escritos += sizeof(msg);
    } else {
      printf("write fallo con %d\n", r);
      printf("errno was: %d\n", errno);
      perror("perror en write");
      break;
    }
  };

  close(fds[1]);
}
```

La criba de eratóstenes - calculando primos

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	**3**	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	**3**	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	**3**	4	**5**	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	**3**	4	**5**	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	**3**	4	**5**	6	**7**	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	**3**	4	**5**	6	**7**	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

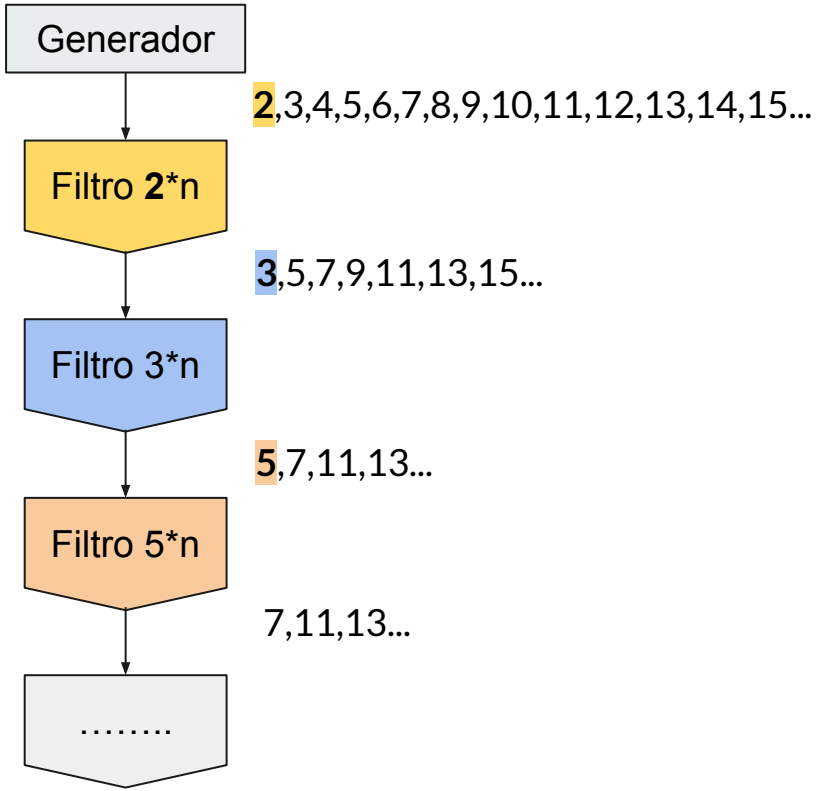
1	**2**	**3**	4	**5**	6	**7**	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

La criba de eratóstenes - calculando primos

1	**2**	**3**	4	**5**	6	**7**	8	9	10
11	12	**13**	14	15	16	**17**	18	**19**	20
21	22	**23**	24	25	26	27	28	**29**	30
31	32	33	34	35	36	**37**	38	39	40
41	42	**43**	44	45	46	**47**	48	49	50
51	52	**53**	54	55	56	57	58	**59**	60
61	62	63	64	65	66	**67**	68	69	70

Tarea: primes

- Imprime los números primos menores o iguales a N
- El primer proceso genera una lista de números consecutivos
  - Se los envía a un proceso “filtro”
- Cada filtro toma el primer valor que recibe, y filtra los múltiplos
- **Tiene que escalar con el valor de N y no leakear file descriptors**



Más recursos

- Páginas de manual (`man`)
- *The Linux Programming Interface*
  de Michael Kerrisk
- The missing semester
  Mastering the tools
- Linux Journey