udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Procesos

Adín Ramírez
adin.ramirez@mail.udp.cl

Sistemas Operativos (CIT2003-1)
1er. Semestre 2015

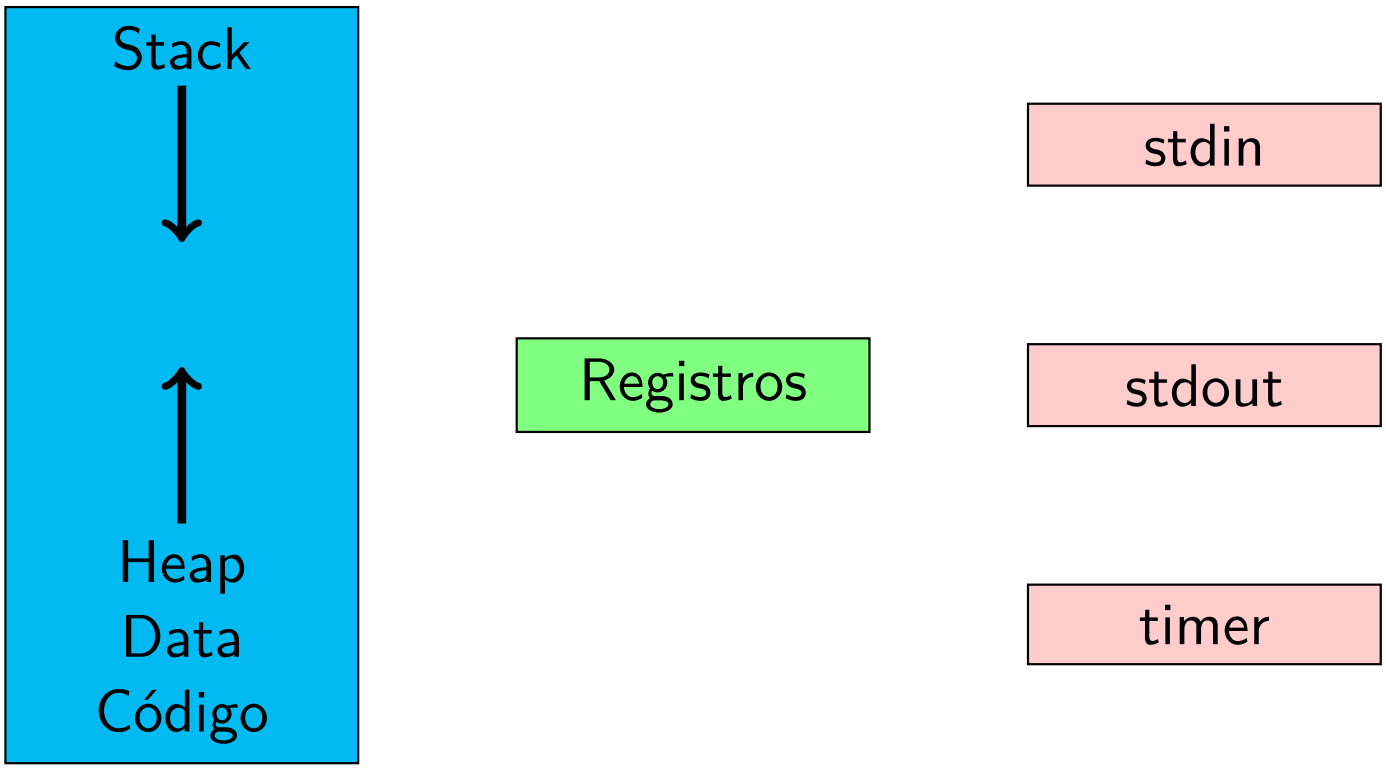
Adaptado de Dave Eckhardt

Objetivos de la clase

- Proceso como una pseudo máquina
- Ciclo de vida de un proceso
- Estados de un proceso de kernel

La computadora

El proceso



Ciclo de vida

- Ciclo de vida
  - Nacimiento: fisión
  - Escuela
  - Trabajo
  - Muerte
- Nomenclatura
  - De los abuelos (1988)

Nacimiento

- ¿De donde vienen los procesos?
  - No de la cigüeña
- ¿Que necesitamos?
  - Contenido de la memoria
    - Texto, datos, stack
  - Contenidos de los registros del CPU (n de ellos)
  - Puertos de entrada y salida (I/O)
    - Descriptores de archivos, e.g., `stdin`, `stdout`, `stderr`
  - Lo oculto
    - estado del timer, directorio actual, `umask`

¿Intimidante?

- ¿Cómo especificamos todas esas cosas?
- Escojamos un proceso que nos guste
  - Clonémoslo!

`fork()`

- `fork` es la llamada del sistema original de Unix para la creación de procesos
- Memoria
  - La copiamos toda
  - Para después: trucos de VM pueden hacer la copia más barata
- Registros
  - Los copiamos todos
    - Excepto, el padre aprende el ID del proceso del hijo, y el hijo obtiene 0

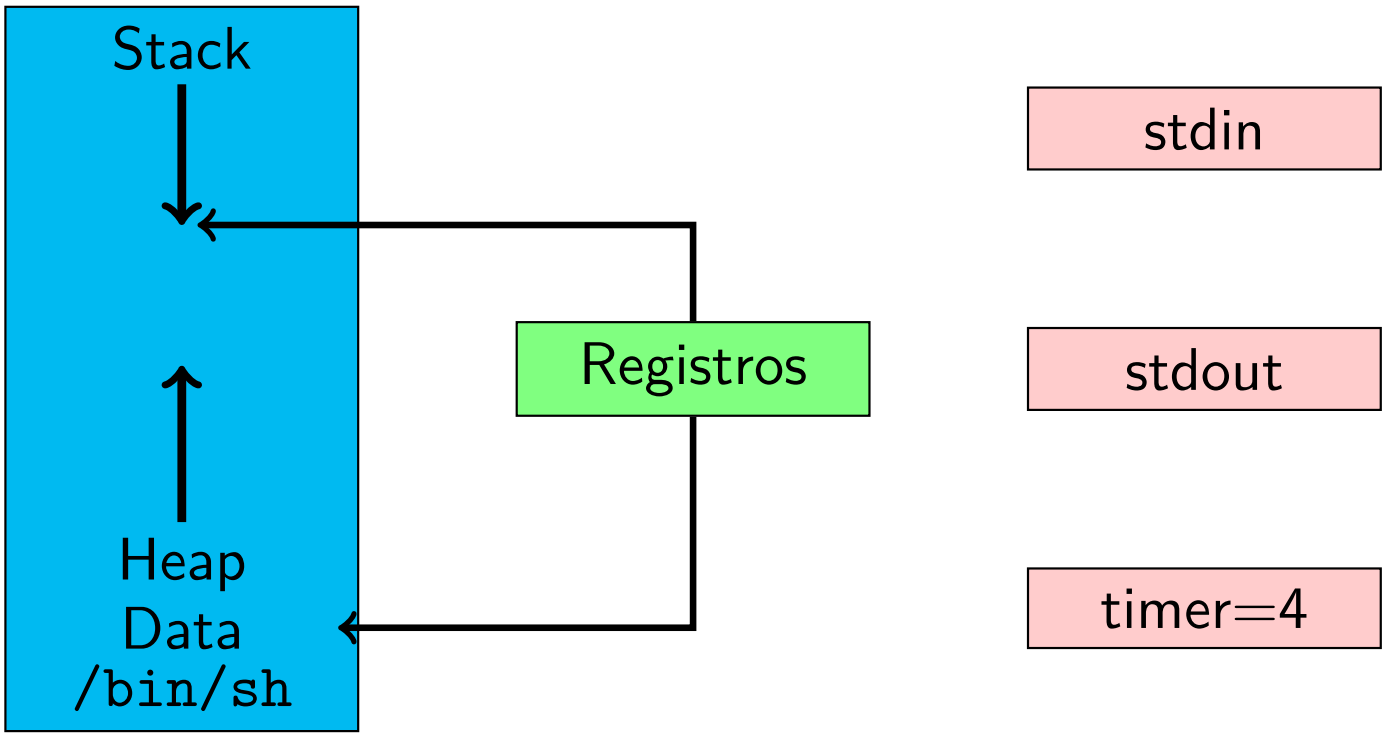
`fork()`

- Descriptores de archivos
  - Los copiamos todos
  - No podemos copiar los archivos mismos
  - Copiamos las referencias al estado de los archivos abiertos
- Lo oculto
  - Se hace lo que es obvio
- Resultado
  - Original, es el proceso padre
  - El hijo es un proceso totalmente especificado, a pesar de tener `0` en los parámetros del `fork`

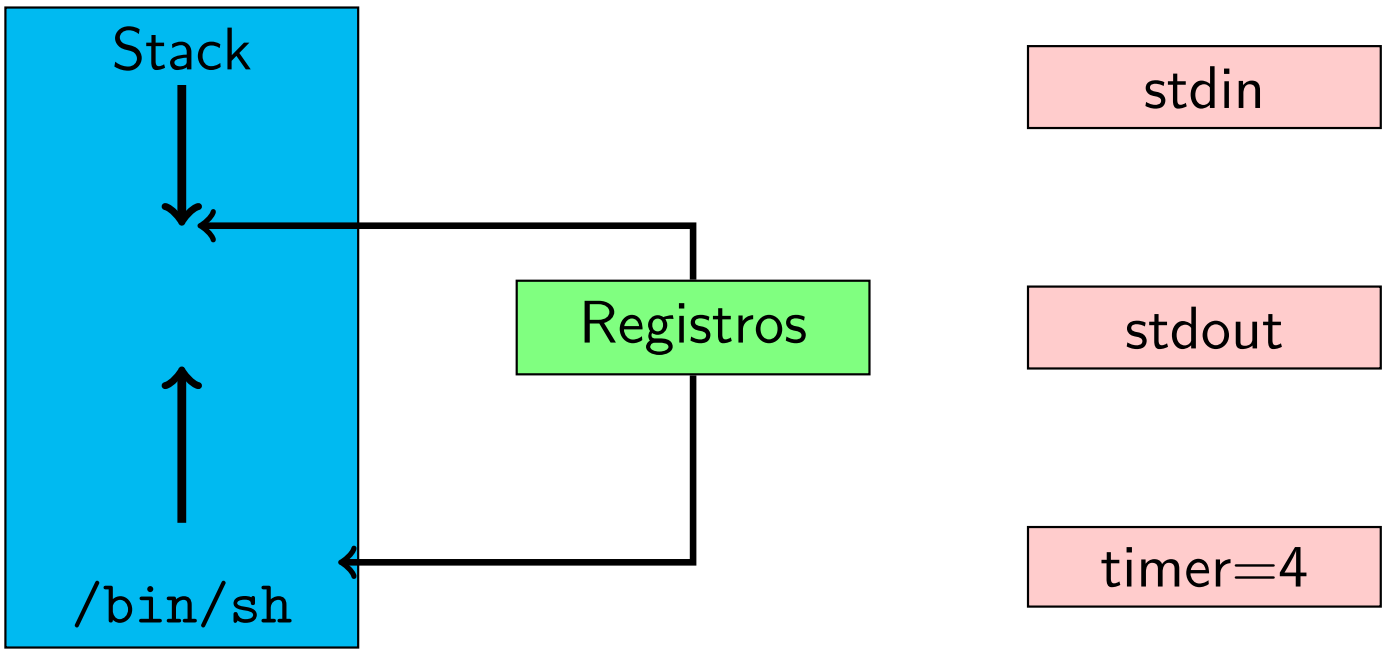
¿Y ahora?

- ¿Cuál es el punto de tener dos procesos idénticos?
- Hacemos cirugía plástica al proceso
  - Implantamos nueva memoria: nuevo texto de programa
  - Implantamos nuevos registros: los antiguos no apuntan correctamente a la nueva memoria
  - Mantenemos la mayoría de los descriptores de archivos
    - Bueno para cooperar y delegar
  - Lo oculto
    - Hacemos lo que es obvio

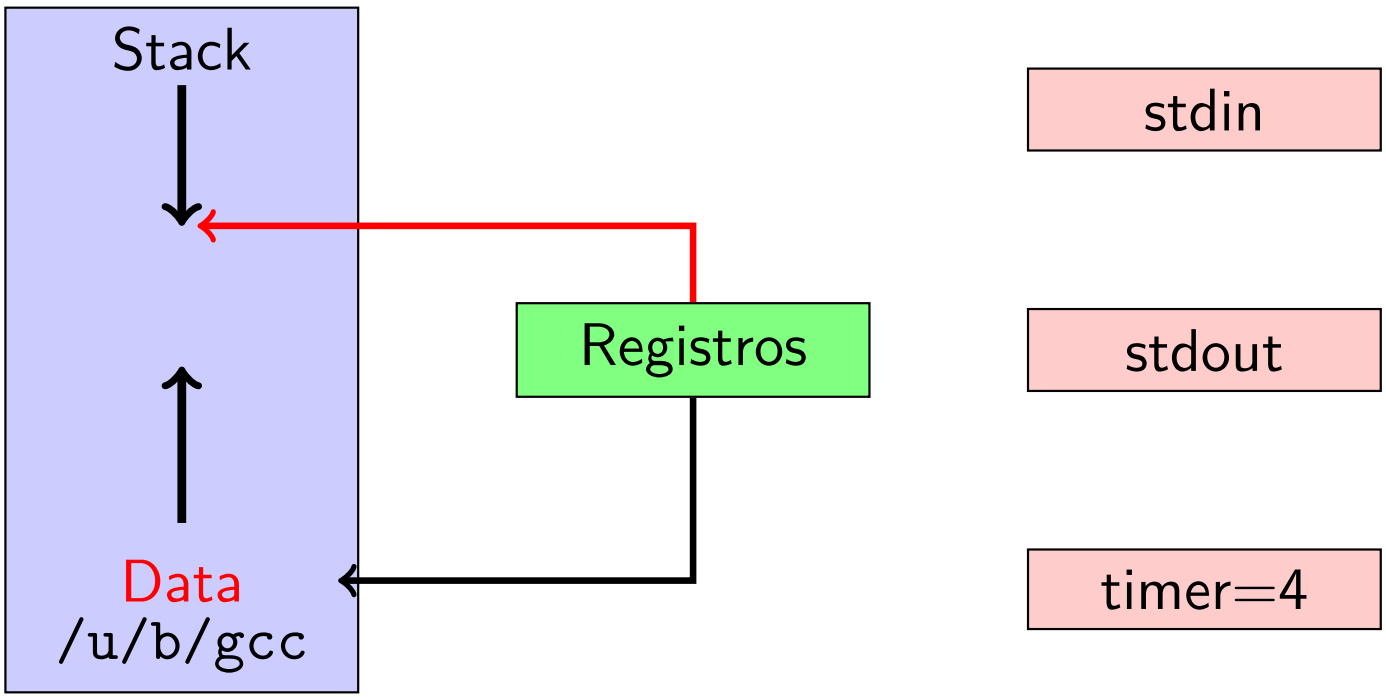
El proceso original



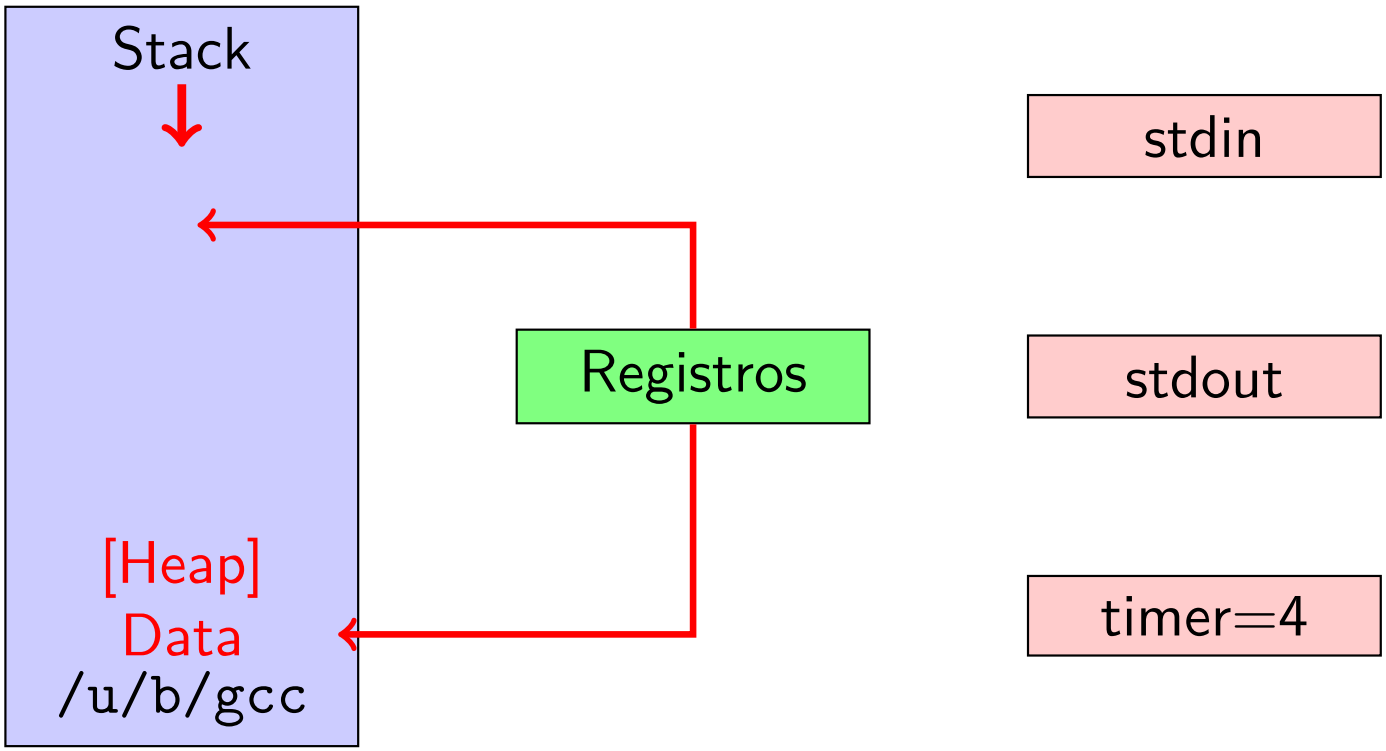
Eliminamos heap y los datos



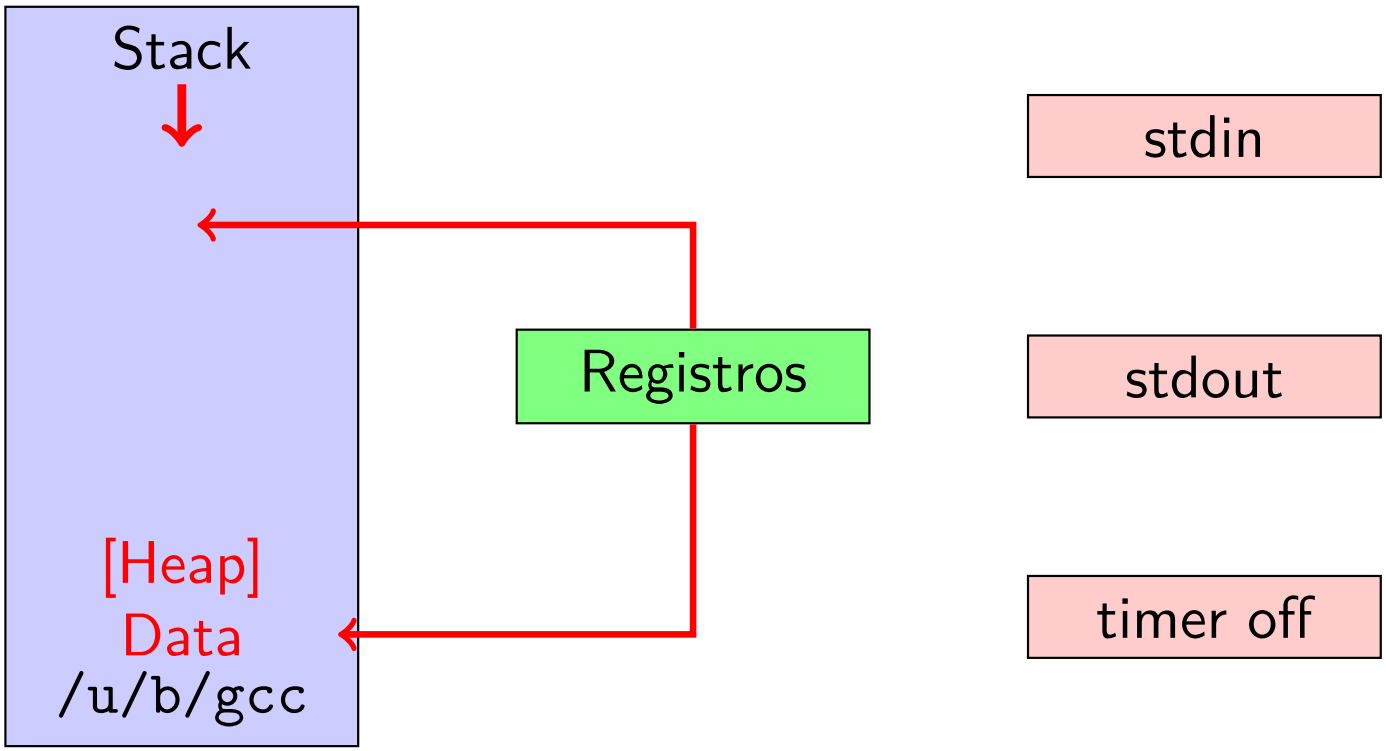
Cargamos el nuevo código y datos del archivo



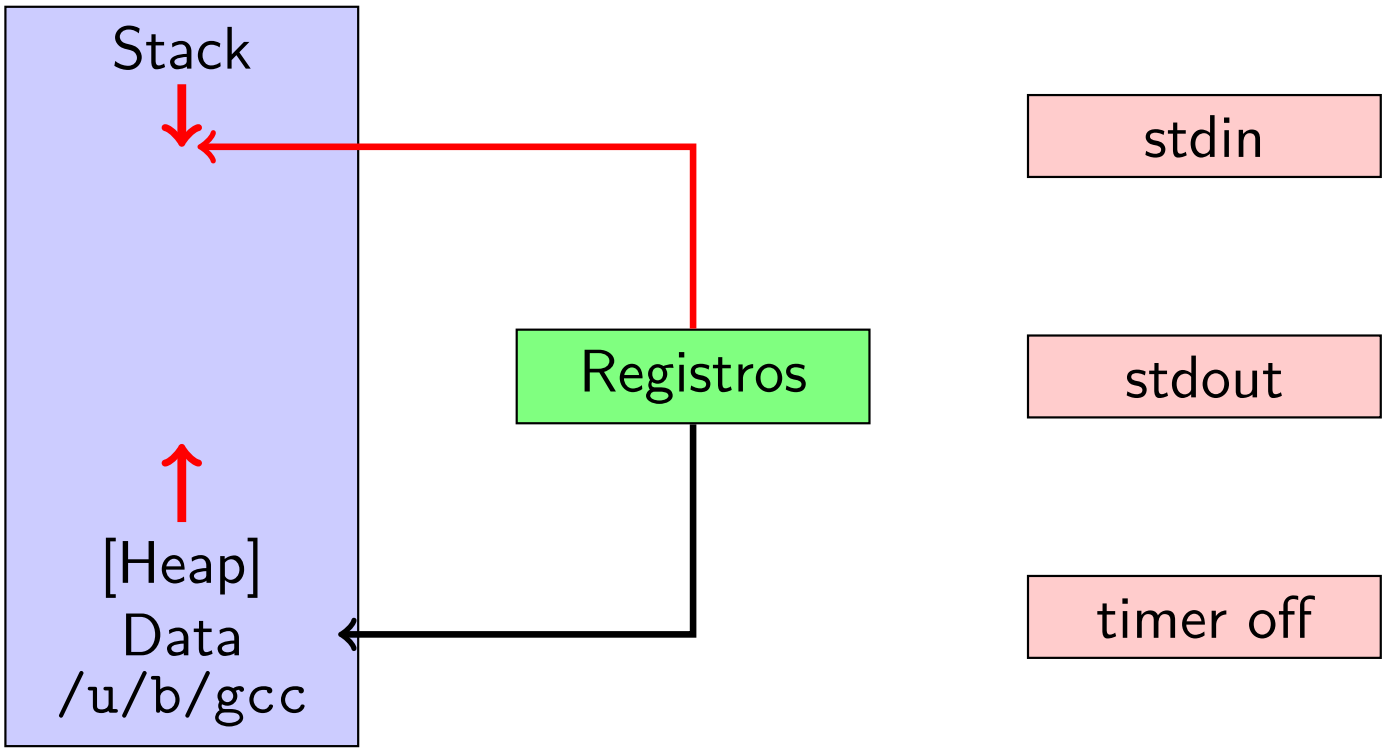
Reset al stack y al heap



Corregimos las cosas



Inicializamos los registros



Formas de ejecutar este procedimiento

- `execve`

```
int execve(
  char *path,
  char *argv[],
  char *envp[])
```

- `spawn()`
  - Necesitamos especificar todas las características del nuevo proceso (complicado)
  - Lo bueno, no tenemos que copiar cosas que vamos a eliminar al momento
- `fork()`, `rfork()`, `clone()` [1]
  - Construye el nuevo proceso a través del viejo
  - Especifica que cosas que copian y cuales se comparten

[1] Revisar: `http://stackoverflow.com/q/4856255/424986`

fork()

```c
int child_pid = fork();
if(child_pid == 0){ // se ejecuta en el proceso hijo
    solamente
  printf("Soy el proceso hijo %d\n",getpid());
  return 0;
} else { // se ejecuta en el proceso padre solamente
  printf("Soy el proceso padre de %d\n",child_pid);
  return 0;
}
```

Salidas posibles

```
Soy el proceso padre de 456
Soy el proceso hijo 456

Soy el proceso hijo 456
Soy el proceso padre de 456
```

La magia

- El setup del kernel
  - Tirar los datos en memoria antiguos
  - Tirar el stack en memoria antiguo
  - Cargar el archivo ejecutable
- El kernel construye el stack para el nuevo proceso
  - Transfiere `argv[]` y `envp[]` al tope del stack del nuevo proceso
  - Construye el frame del stack para `__main()` (`main` del `main`)
  - Establece los registros
    - Puntero al stack (al tope de éste)
    - Contador del programa (al inicio de `__main()`)

Estados del proceso

- Ejecutandose
  - En modo de usuario o de kernel
- Bloqueado
  - Esperando algún evento
    - Finalización de I/O, salida de otro proceso, mesajes, etc.
    - Dormido (suspendido) por un periodo de tiempo definido
  - Scheduler: *no te ejecutes*

  Q: Modo de usuario? de kernel? ambos? ninguno?
- Ejecutable

  Q: Modo de usuario? de kernel? ambos? ninguno?

Otros estados

- Forking
  - Obsoleto, usado alguna vez para un tratamiento especial
- Zombie
  - El proceso llamo a `exit()`, el padre no se ha dado cuenta
- Ejercicio para ustedes: dibujar el diagrama de transición de estados

Muerte

- Voluntaria

```
void exit(int reason);
```

- Excepción de hardware
  - `SIGSEGV`: no hay memoria aquí para tí (gracias por participar)
  - Segmentation fault
- Excepción de software
  - `SIGXCPU`: has usado mucho CPU

Llamado del sistema `kill(pid, sig);`

- Entregar `sig` al proceso `pid`
  - Los valores negativos de `pid` tienen comportamientos interesantes
- Equivalente al teclado `^C`
  - `kill(getpid(), SIGINT);`
- Iniciar o detener logging
  - `kill(daemon_pid, SIGUSR1);`
  - `% kill -USR1 33`
  - `% kill -USR2 33`
  - Es un uso de `kill` que no mata

Limpieza del proceso

- Liberación de recursos
  - Archivos abiertos: `close()` cada uno
    - TCP: 2 minutos o más
    - Disco offline: para siempre (nadie puede pasar)
  - Liberar memoria
- Auditoría
  - Llevar registro del uso de los recursos en un archivo mágico
- ¿Terminó?

Zombies

- El estado del proceso se redujo al código de salida
- Espera mientras el padre llamó a `wait()`
  - El código de salida es copiado a la memoria del padre
  - PCB (process control block) es borrado del kernel

Estado del proceso del kernel

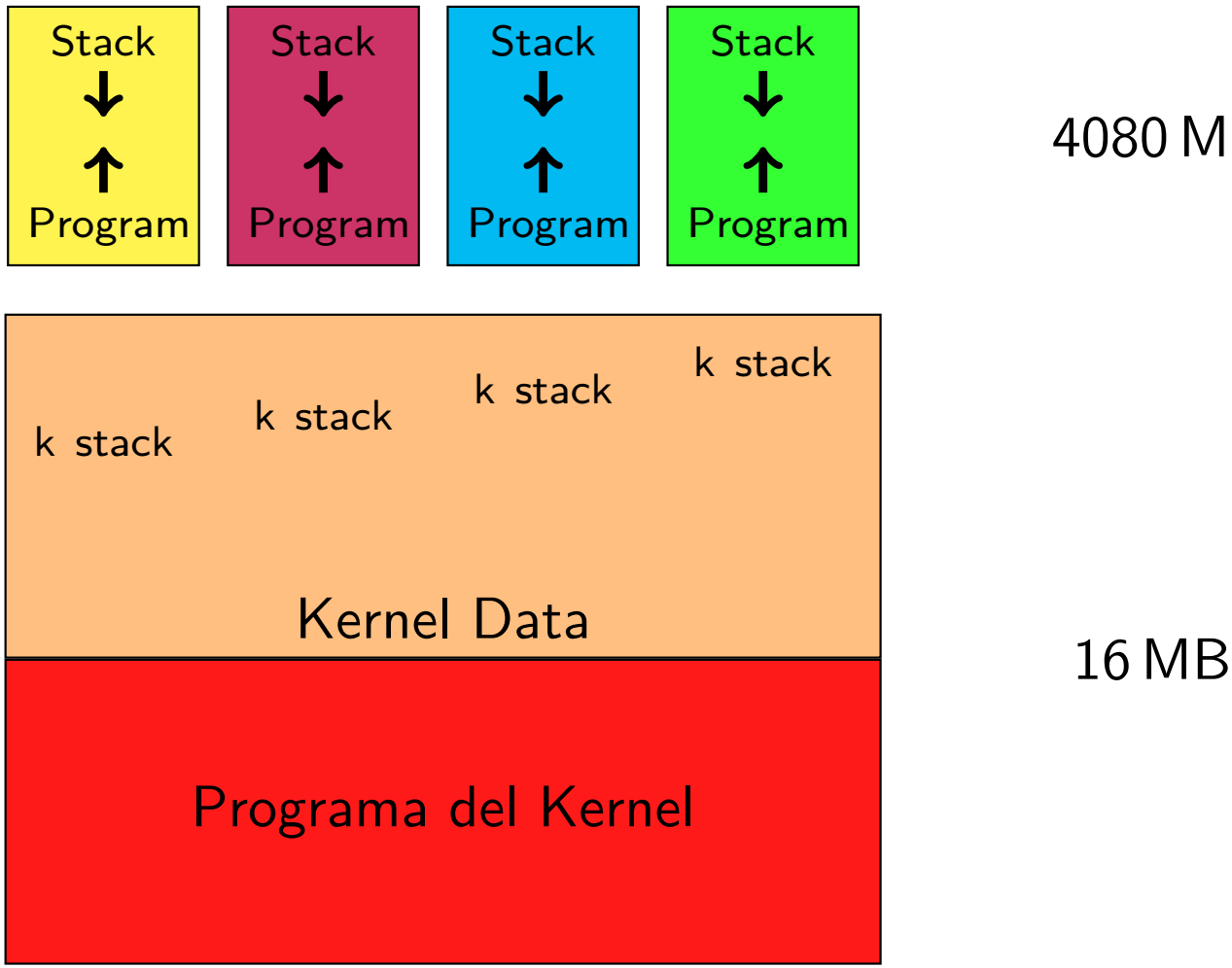
- El temido PCB[2]
  - Printer circuit board —no
  - Polychlorinated biphenyl —tampoco
- Process control block
  - Cualquier cosa sin una dirección de memoria visible para el usuario
  - Información para la administración del kernel
  - Estado del scheduler
  - Más cosas

[2] `http://lmgtfy.com/?q=pcb`

Ejemplo del contenido de PCB

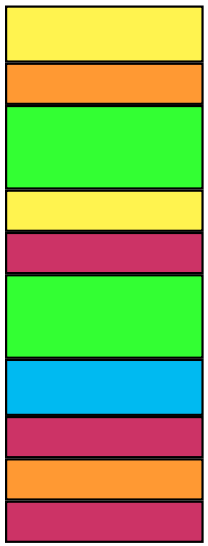
- Puntero al área de almacenamiento de registros del CPU
- Número del proceso, y número del proceso padre
- Valor del timer de cuenta regresiva
- Información del segmento de memoria
  - Lista de segmentos de la memoria de usuario
  - Referencia del stack del kernel
- Información del scheduler
  - Posición en la lista enlazada, prioridad, canal donde duerme

Plano de memoria virtual



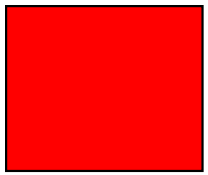
Plano de memoria física

Memoria de usuario



240 MB

Memoria de kernel



16 MB

Implementación

- `getpid()`
- `fork()`
- `exec()`
- `wait()`
- `exit()`
- A leer los manuales!!

Puntos importantes

- Partes de un proceso
  - Físico: páginas de memoria, registros, dispositivos I/O
  - Virtual: regiones de memoria, registros, puertos I/O
- Nacimiento, trabajo, y muerte de un proceso
- Escuela sobre procesos
- *Big picture* de los procesos
  - Dos vistas: virtual y física