1. El Proceso

1.1. De Programa a Proceso

```c
#include <stdio.h>

int main() {
printf("hello, world\n");
}

```

Script 1: hola mundo.

La Compilación:

1. **La fase de procesamiento**. El preprocesador (cpp) sustituye las macros (#) y se eliminan los comentarios del código fuente. El resultado es un archivo de código C preprocesado con extensión .i .

2. **La fase de compilacion**. El compilador (cc) traduce el programa .i a un archivo de texto .s que contiene un programa en lenguaje assembly.

3. **La fase de ensablaje**. A continuación el ensamblador (as) traduce el archivo .s en instrucciones de lenguaje de máquina (binario) empaquetándolas en un formato conocido como programa objeto realocable. Este es almacenado en un archivo con extensión .o

4. **La fase de link edicion**. Generalmente los programas escritos en lenguaje C hacen uso de funciones que forman parte de la biblioteca estandar de C que es provista por cualquier compilador de ese lenguaje. Por ejemplo la función printf(), la misma se encuentra en un archivo objeto pre compilado que tiene que ser mezclado con el programa que se esta compilando, para ello el linker realiza esta tarea teniendo como resultado un archivo objeto ejecutable.

   Es decir se combinan los archivos objeto con las bibliotecas y dependencias necesarias para formar el archivo ejecutable final.

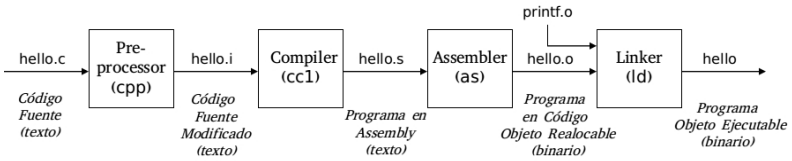

Figura 1: Proceso de compilacion.

1.2. Un Programa en Unix

Un programa es un archivo que posee toda la información de como construir un proceso en memoria [KER](cap. 6). Un programa contiene:

1. **Formato de Identificación Binaria:** Cada archivo ejecutable posee META información describiendo el formato ejecutable. Esto permite al kernel interpretar la información contenida en el mismo archivo.

   Formatos en Unix:

   - **OUT** Assembler Output → Salida del compilador de C
   - **COFF** Common Object File Format → Utilizado en las versiones de System V compartidas, y símbolos de depuración en sistemas Unix.
   - **ELF** Executable and Linking Format → Utilizado en la actualidad

2. **Instrucciones de Lenguaje de Máquina:** Almacena el código del algoritmo del programa.

3. **Dirección del Punto de Entrada del Programa:** Identifica la dirección de la instrucción con la cual la ejecución del programa debe iniciar.

4. **Datos:** El programa contiene valores de los datos con los cuales se deben inicializar variables, valores de contantes y de literales utilizadas en el programa.

5. **Simbolos y Tablas de Realocación:** Describe la ubicación y los nombres de las funciones y variables de todo el programa, así como otra información que es utilizada por ejemplo para debugg.

6. **Bibliotecas Compartidas:** describe los nombres de las bibliotecas compartidas que son utilizadas por el programa en tiempo de ejecución así como también la ruta del linker dinámico que debe ser usado para cargar dicha biblioteca.

7. **Otra información:** El programa contiene además otra información necesaria para terminar de construir el proceso en memoria.

El Sistema Operativo más precisamente el Kernel se encarga de:

1. Cargar instrucciones y Datos de un programa ejecutable en memoria.

2. Crear el Stack y el Heap.

3. Transferir el Control al programa.

4. Proteger al SO y al Programa.

1.3. El Proceso

Un proceso es sólo un programa en ejecución. Un proceso incluye:

- Los Archivos abiertos.
- Las señales(signals) pendientes.
- Datos internos del kernel.
- El estado completo del procesador.
- Un espacio de direcciones de memoria.
- Uno o más hilos de Ejecución. Cada thread contiene
  - Un único contador de programa.
  - Un Stack.
  - Un Conjunto de Registros.
  - Una sección de datos globales

El proceso:

- Una "instancia" de un programa
- Tiene su propia memoria: código, datos, stack, heap
- Tiene un identificador único: PID.
- Tiene un conjunto de "archivos abiertos": Descriptores de Archivos.

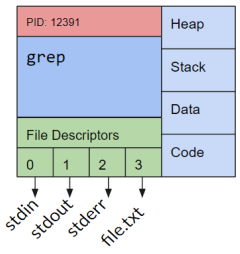

Figura 2: Proceso.

1.4. La Virtualización

Crear una abstracción que haga que un dispositivo de hardware sea mucho más fácil de utilizar.

- Virtualización de memoria.
- Virtualizacion de procesador.

Virtualización de Memoria

La virtualización de memoria le hace creer al proceso que este tiene toda la memoria disponible para ser reservada y usada como si este estuviera siendo ejecutado sólo en la computadora (ilusión).

Todos los procesos en Linux, está dividido en 4 segmentos:

- **Text Segment:** Contiene el código del programa.
- **Data:** Almacena las Variables Globales (extern o static en C).
- **Heap:** Memoria Dinámica Alocable.
- **Stack:** Almacena las Variables Locales y trace de llamadas.

Protección de Memoria Para que un proceso se ejecute tiene que estar residente en memoria, pero a su vez el sistema operativo tiene que estar residente en memoria.

- *El proceso tiene que estar en memoria para poder ejecutarse.*
- *El sistema operativo tiene que estar ahí para:*
  - *iniciar la ejecución del programa*
  - *manejar las interrupciones.*
  - *y/o atender las systems call..*

Es más, otros procesos podrían estar simultáneamente en memoria para poder compartir la memoria de forma segura, para ello el sistema operativo tiene que poder configurar el hardware de forma tal que cada proceso pueda leer y escribir solo su propia memoria (No la memoria del sistema operativo tampoco la de otros procesos. Ya que sino el proceso en cuestión podría incluso modificar al Kernel del sistema operativo. Para ello el Hardware debe proveer un mecanismo de protección de memoria, (que se verán detalladamente mas adelante).

Uno de estos mecanismos es denominado Memoria Virtual, la memoria virtual es una asbtracción por al cual la memoria física puede ser compartida por diversos procesos.

Un componente clave de la memoria virtual son las direcciones virtuales, con las direcciones virtuales, para cada proceso su memoria inicia en el mismo lugar, la dirección 0.

Cada proceso piensa que tiene toda la memoria de la computadora para si mismo, si bien obviamente esto en la realidad no sucede. El hardware traduce la dirección virtual a una dirección física de memoria.

Figura 3: mmu.

Traducción de Direcciones Se traduce una Dirección Virtual (emitida por la CPU) en una Dirección Física (la memoria). Este mapeo se realiza por hardware, más específicamente por Memory Management Unit (MMU).

Virtualización de Procesador

La virtualización de procesamiento es la forma de virtualización más primitiva, consiste en dar la ilusión de la existencia de un único procesador para cualquier programa que requiera de su uso. De esta forma, se provee:

Simplicidad en la programación:

- Cada proceso cree que tiene toda la CPU.
- Cada proceso cree que todos los dispositivos le pertenecen.
- Distintos dispositivos parecen tener el mismo nivel de interfaces.
- Las interfaces con los dispositivos son más potentes que el bare metal.

Aislamiento frente a Fallas:

- Los procesos no pueden directamente afectar a otros procesos.
- Los errores no colapsan toda la máquina.

¿Cómo se provee la ilusión de tener varios CPUs?: El SO crea esta ilusión mediante la virtualización de la CPU a través del kernel.

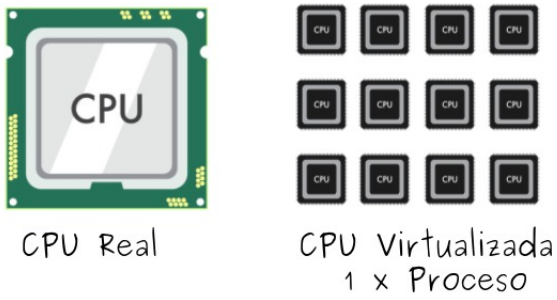

Figura 4: Virtualizacion Cpu.

Viéndolo desde el punto de vista de la abstracción y virtualización:

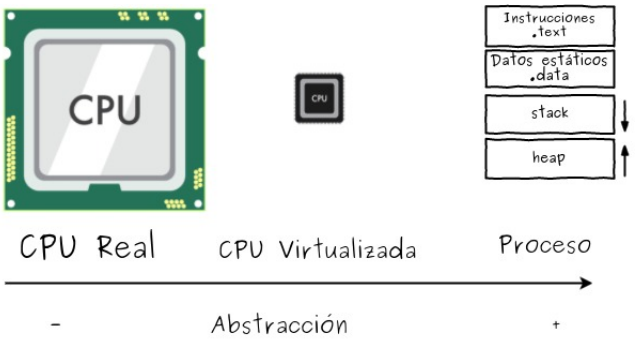

Figura 5: Virtualizacion Cpu.

Entonces

"un proceso es básicamente una abstracción de un programa en ejecución."

Se ha de tener en cuenta que el Kernel en sí mismo también es un proceso y que la abstracción del proceso provee ejecución, aislamiento y protección. Estos tres conceptos pueden merecer varios capítulos de un libro. El sistema operativo lleva la contabilidad de todos los procesos que se están ejecutando en la computadora mediante la utilización de una estructura llamada Process Control Block o PCB. La PCB almacena toda la información que un sistema operativo debe conocer sobre un proceso en particular:

- Donde se encuentra almacenado en memoria.
- Donde la imagen ejecutable esta en el disco.
- Que usuario solicito su ejecución.
- Que privilegios tiene ese proceso.

1.5. El Proceso: por dentro

La idea detras de la abstracción es como virtualizar una CPU, es decir, como hacer para que una única procesador pueda ejecutar múltiples procesos.

Un Proceso necesita permisos del Kernel del SO para:

- Acceder a memoria perteneciente a otro proceso.
- Antes de escribir o leer en el disco.
- Antes de cambiar algún seteo del hardware del equipo.
- Antes de enviar información a otro proceso.

1.6. El API de Procesos

Que debe incluir cualquier interfaz de un SO:

- Creación (Create): todo sistema operativo debe incluir una forma de crear un nuevo proceso.
- Destrucción(Destroy): así como existe una interface para crear un proceso debe existir una interface para destruirlo por la fuerza.
- Espera (wait): A veces es útil esperar a que un proceso termine su ejecución por ende algún tipo de interface de espera debe ser provista.
- Control Vario (Miscellaneous Control): Además de esperar o matar a un proceso otros tipos de operaciones deben poder realizarse. Por ejemplo, suspender su ejecución por un tiempo y luego reanudarla.
- Estado (Status) : Tiene que existir una forma de saber sobre la situación del proceso y su estado. Cuánto hace que se está ejecutando, en que estado se encuentra, etc.

Estas son las acciones básicas que todo SO debe proveer sobre la **abstracción de la CPU.**