**APUNTES SOA**

**4: The Abstraction: The process**

>>PROCESS CREATION

**1.**El primer paso consiste en **cargar** el código en **memoria**, en el espacio de direcciones del proceso

**2.** El **stack** del proceso en tiempo de ejecución es **alocatado**. El stack se emplea para **variables locales, parámetros de función y direcciones de retorno.**

**3.** El **heap** del programa se crea. Es empleado para alocatar información de forma dinámica.

**4.** El sistema operativo realiza otras tareas de inicialización. Configuración de entrada/salida. Cada proceso tiene por defecto tres ficheros: entrada estándar, salida estándar y salida de error.

**5. Correr el programa** en el main().

Diagrama

Descripción generada automáticamente

>>PROCESS STATES

Un proceso puede estar en alguno de los siguientes estados.

-**Running:** un proceso está corriendo en la CPU.

-**Ready:** un proceso está listo para correr pero por alguna razón el SO ha escogido no correrlo en ese momento.

-**Blocked:** El proceso ha realizado una determinada operación.

>>DATA STRUCTURES

El SO tiene unas **estructuras de datos especiales** para controlar algunas piezas de información.

-**Process list:** ready processes, blocked processes, running processes.

-**Register context.**

**PCB (process control Block):** estructura de C que contiene información sobre cada proceso.

**5. Interlude: Process API**

>>FORK() SYSEM CALL

Texto

Descripción generada automáticamenteCon la llamada al sistema **fork(),** podemos crear un nuevo proceso. El nuevo proceso que se acaba de crear, dispone de una **copia del espacio de direcciones**, **registros y PC.**

**Texto

Descripción generada automáticamente**

**6. Mechanism: Limited Direct Execution**

A los procesos deben darse la ilusión de tener toda la CPU para ellos mismos.

El sistema operativo necesita compartir la CPU mediante **reparto de tiempo**.

Surgen una serie de problemas.

-**Rendimiento:** cómo podemos hacer la virtualización sin que se forme un overhead excesivo para el sistema?

-**Control:** cómo podemos correr procesos de forma eficiente y seguir manteniendo control sobre la CPU?

*Veremos a continuación, algunos problemas que pueden surgir para implementar lo que queremos y de qué forma se resuelven*.

**Problema 1:** ¿Qué sucede si un proceso desea ejecutar un cierto tipo de operación, como petición de entrada/salida al disco, o acceder a otros recursos del sistema como CPU o memoria?

La respuesta es usar **un control de transferencia protegido**:

-**User mode:** las aplicaciones no tienen acceso completo a los recursos hardware.

-**Kernel mode:** El SO tiene acceso a todos los recursos hardware del sistema.

>>SYSTEM CALLS

Permite a que el kernel **revele de forma cuidadosa** ciertas piezas de funcionalidad al progama de usuario. Por ejemplo: acceso al sistema de ficheros, crear y destruir procesos, comunicarse con otros procesos, alocatar más memoria.