VisualOS

Diseño

1. Introducción

Este documento intenta describir la estructura interna de la aplicación a un nivel abstracto. Esto permitirá luego una comprensión más fácil del manual del programador y del propio código.

Para entender el funcionamiento de la aplicación se recomienda leer primero este documento y, a continuación, el manual del programador junto con el código.

2. Visión Global

La aplicación consta de distintos subsistemas (Procesador, Memoria, Disco) interconectados a través del sistema de mensajería que se encarga de gestionar la comunicación.

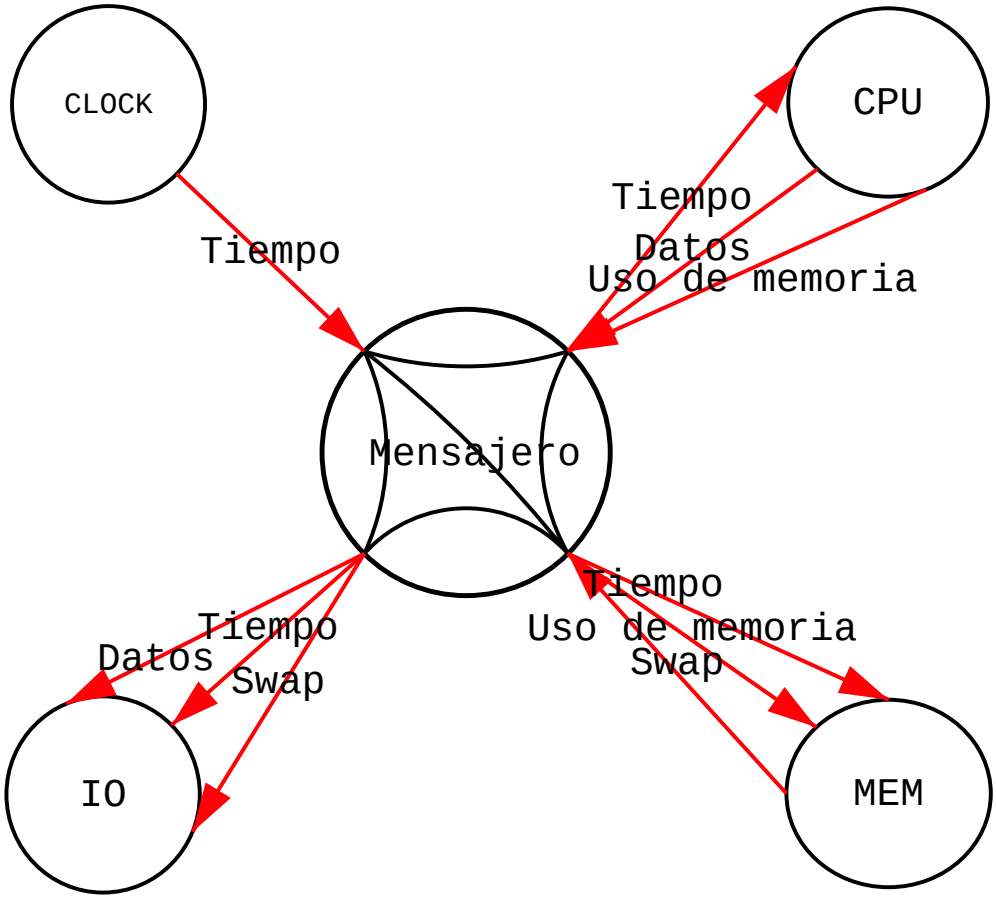
En todos los casos en que aparecen algoritmos y representaciones, el patrón es el mismo. Existen unas estructuras de datos, gestionadas por el algoritmo elegido, que representan el estado del subsistema. De esta manera, el código que dibuja cada representación no tiene más que utilizar esta misma estructura de datos como referencia.

Cada subsistema es un objeto que ejemplifica cada una de las cuatro partes de la arquitectura Von Newmann, a la vez que los métodos de gestión impuestos por un sistema operativo moderno.

Dichos objetos implementan una interfaz bien definida, comunicándose a través del sistema de mensajería y ofrecen unos servicios que generalmente pueden solicitar, de

igual manera, cualquiera de los demás.

Figura 1. Diagrama de bloques



2.1. Reloj

Este subsistema marca el ritmo de ejecución de los demás. Su simulación es importante porque viene a representar el carácter síncrono del ordenador y permite al usuario elegir la velocidad a la que transcurre la simulación.

2.1.1. Servicios que ofrece

- Señala el paso del tiempo.
- Se detiene a petición del cliente.

2.2. Procesador

Simboliza la gestión que el SO hace de la capacidad de proceso de la máquina.

2.2.1. Servicios que ofrece

- Informa de la creación y destrucción de procesos.
- Termina anticipadamente los procesos a petición del cliente.

2.2.2. Servicios que utiliza

- Produce fallos de página y accesos a la memoria principal.
- Solicita bloques de datos del disco.
- Recibe la referencia de tiempo del reloj
- Detiene el reloj en ciertas situaciones

2.3. Memoria

Representa la gestión de memoria principal con las técnicas más comúnmente empleadas para aumentar su eficiencia (paginación, memoria. virtual, swap)

2.3.1. Servicios que ofrece

- Recibe fallos de página.
- Comunica cuándo un fallo de página ha sido resuelto.
- Recibe accesos a páginas disponibles.
- Informa cuándo cambia la disponibilidad de páginas de un proceso.

2.3.2. Servicios que utiliza

- Realiza accesos al disco para escribir en el área de swap las páginas que "roba" a los distintos procesos y poder devolvérselas después.
- Recibe la referencia de tiempo del reloj.
- Recibe notificación del procesador cuando se crean o destruyen procesos
  .
- Solicita la terminación anticipada de procesos al procesador cuando no hay recursos suficientes.

2.4. Disco

En un sistema real, la memoria principal (rápida y volátil) se ve complementada por una memoria secundaria (comúnmente en forma de disco magnético) que es

persistente, más lenta y barata, y que le confiere al sistema capacidad de almacenar la información necesaria para funcionar (sistema operativo y datos de usuario).

2.4.1. Servicios que ofrece

- Acepta peticiones de acceso a los distintos bloques de datos y swap.
- Informa (al cliente que lo solicitó) cuándo un acceso ha sido completado.

2.4.2. Servicios que utiliza

- Recibe la referencia de tiempos del reloj.

2.5. Mensajero

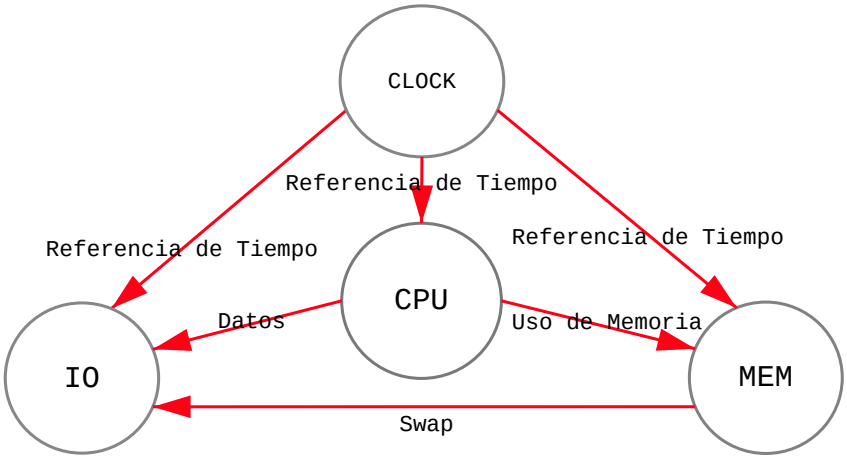
Este elemento se encarga de gestionar la comunicación entre los demás.

2.5.1. Servicios que ofrece

- Reinicia todo el sistema a petición del cliente

En la figura se muestra un diagrama simplificado del sistema, suponiendo que la comunicación ocurre directamente sin tener en cuanta al mensajero.

Figura 2. Diagrama de bloques simplificado



3. Algoritmos

En todos los casos en que entraban en juego distintos algoritmos estos se han implementado en forma de objetos con su interfaz bien definido. En cada caso, tienen datos y métodos distintos (Ver el Manual del programador) pero en todos los casos todo queda encapsulado detrás del interfaz.

De esta manera, para anadir un algoritmo nuevo sólo es necesario anadir un archivo nuevo con el código específico de este y una llamada a su función de inicialización (donde se tendrá que registrar como algoritmo) en el archivo "main.c"del mismo directorio.

4. Representaciones

También en el caso de las representaciones se ha tratado de mantener la encapsulación. Son objetos cuyo único método es "actualizar". Para el resto del código, es totalmente transparente la representación que ha elegido el usuario o si está viendo una o varias a la vez.

En este caso, al igual que en el anterior, para anadir una representación nueva no hay más que anadir un archivo con el código que la dibuja y una llamada a su función de inicialización (donde se tendrá que registrar como representación) en el archivo "main.c"del mismo directorio.

5. Eventos del sistema

Para que cualquier parte del código pueda "saber"cuando ocurren ciertas cosas que le son interesantes (ej. el cambio de estado de un proceso para la representación de estados del procesador), se ha implementado una infraestructura para la difusión de eventos.

El código donde se produce el evento lo hace público para quien lo quiera y el código que necesita conocer del evento sólo tiene que registrar una función para ser llamada siempre que ocurra el evento elegido (Ver el manual del programador).