01. Introducción

a. *Arquitecturas de Computadoras*

➢ **Arquitectura Von Neumann:**

En la arquitectura Von Neumann toda la memoria es capaz de almacenar todos los elementos del programa, datos e instrucciones.

➢ **Arquitectura Harvard**

En el caso de la arquitectura Harvard, la memoria está dividida en dos, una parte donde almacenamos los datos y la otra donde almacenamos las instrucciones, esto permite su acceso simultáneo.

➢ **Resumen**

Con esto, podemos ver, que para ejecutar la misma operación, dado que solo podemos acceder a la memoria una vez por ciclo de reloj, la arquitectura Von Neumann necesita de al menos dos ciclos para ejecutar dicha operación, una para acceder a los datos y otra para ejecutar la instrucción.

Con la arquitectura Harvard sólo necesitamos de un ciclo para todo ello, ya que puede acceder a ambos compartimentos de memoria en el mismo ciclo.

b. *Características básicas de los sistemas operativos*

➢ **Gestión de procesos**

Controla la ejecución de programas y la asignación de recursos a dichos programas. Los procesos son las entidades que el sistema operativo utiliza para controlar el uso de los recursos.

➢ **Gestión de memoria**

Administra la memoria RAM y virtual asignada a los procesos que la solicitan.

➢ **Sistema de archivos**

Sistema de almacenamiento de un dispositivo de memoria, el cual estructura y organiza la escritura, búsqueda, lectura, almacenamiento, edición y eliminación de archivos de una manera concreta.

➢ **Interfaz de usuario**

Aquello que conecta a los usuarios con el sistema, medio por el cual controla el sistema.

➢ **Gestión de dispositivos**

El SO se encarga de administrar los periféricos de E/S que utilizaran los programas independientemente del modelo y tipo de periférico.

c. *POSIX (Portable Operating System Interface)*

POSIX se refiere a un conjunto de estándares que definen la interfaz de programación y comportamientos comunes para el software del sistema operativo

en SO de tipo UNIX. Esto proporciona portabilidad entre diferentes sistemas UNIX, consiguiendo que se puedan escribir programas que se ejecuten en diferentes SO sin tener que reescribir el código para cada plataforma.

POSIX define una serie de interfaces de programación que sean compatibles con cualquier sistema operativo que cumpla con el propio estándar, entre las funciones incluye la gestión de ficheros, la comunicación entre procesos, la gestión de memoria, etc.

d. Tipos de SO

- **Tiempo compartido**

Aquel SO que utiliza técnicas de planificación y programación concurrente para dar la apariencia de ejecutarse de manera simultánea múltiples procesos.

- **Tiempo real**

Considerados un subconjunto de los sistemas de control, son sistemas operativos livianos utilizados para desarrollar cosas íntimas e integrar tareas de diseño con recursos y tiempos específicos de manera óptima

- **Red**

Permite la interconexión de equipos para acceder a los servicios y recursos creando redes de computadoras.

- **Distribuidos**

Conjunto de equipos independientes que actúan de forma transparente actuando como un único equipo

- **Embebidos**

Considerados un subconjunto de los sistemas de control, son sistemas operativos livianos utilizados para desarrollar cosas íntimas e integrar tareas de diseño con recursos y tiempos específicos de manera óptima

- **Máquinas virtuales**

- **Teléfonos móviles**

Aquellos utilizados en dispositivos móviles, como Android y iOs.

e. SO según el tipo de Kernel

El kernel o núcleo, es una parte fundamental del sistema operativo y se define como la parte que se ejecuta en modo privilegiado o modo núcleo

- **Monolíticos**

SO que trabaja en su totalidad desde el espacio del núcleo

- **Microkernel**

El micronúcleo provee de un conjunto de llamadas mínimas al sistema para implementar servicios básicos, todos los otros servicios se ejecutan como procesos en el espacio de usuario

➢ **Kernel Híbrido**

Micronúcleo con algo de código no esencial en espacio de núcleo para que la ejecución sea más rápida

➢ **Exokernel**

Sistema creado con fines de investigación.

➢ **Nanokernel**

Más pequeño incluso que el microkernel, representa la capa más cercana de abstracción al hardware del sistema operativo interconectado a la CPU, maneja las interrupciones e interactúa con la unidad de manejo de memoria

02. Gestión de procesos

a. Concepto de proceso

Un proceso es un programa en ejecución, o de manera técnica, una abstracción técnica para lograr gestión de recursos del equipo. Se compone de los siguientes puntos:

- *Una imagen binaria de un programa*, la cual está formada por las instrucciones y los datos del programa.
- *Una pila* o zona de memoria para almacenar datos temporales
- *Una tabla de páginas* para traducir las direcciones virtuales generadas por el proceso en direcciones físicas en las cuales se encuentra almacenado
- *Una estructura de control*, conocida como PCB, con la cual el sistema operativo puede controlar su ejecución

b. Estados de los procesos

➢ **Nuevo**

Un proceso recién creado, pero no admitido todavía entre los procesos ejecutables del sistema operativo

➢ **Listo**

Esperando a ser asignado para su ejecución.

➢ **En ejecución**

El proceso en CPU

➢ **En espera**

Esperando un suceso de E/S

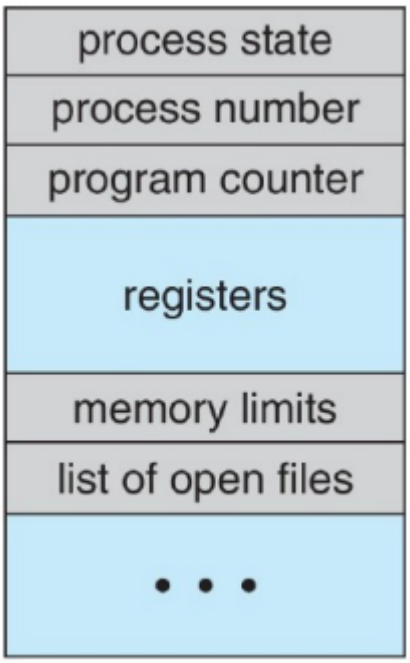
➢ **Terminado**

Ejecución del proceso acabada, se liberan los recursos asignados al mismo.

c. Planificación de los procesos

Para la planificación de los procesos se necesita de un PCB ó Process Control Block, el cual es una estructura de datos que permite al sistema operativo controlar los diferentes aspectos de la ejecución de un proceso.

El PCB se organiza en un conjunto de campos en los que se almacena información de diversos tipos



Estructura típica de un PCB

El mecanismo que decide qué proceso se ejecuta el siguiente se denomina Scheduling. El planificador como tal encargado de asignar los procesos es el Scheduler, el cual decide hacia qué cola de prioridad deberá ir dicho proceso. Por último el Dispatcher decide qué proceso pasa a ejecución o se retira de la misma y realiza un cambio de contexto.

Las colas de procesos tienen asignados diferentes procesos en su estado listo a la espera de ser asignados a la CPU para ser ejecutados, dentro de la CPU, pueden ocurrir diversos eventos que hagan que vuelvan a la cola de espera, desde un evento de E/S, finalización de su tiempo asignado, se crea un proceso hijo o se espera una interrupción, una vez estos eventos terminan se les vuelve a asignar el estado listo al proceso y aguardan para volver a ser ejecutados.

El cambio de contexto es la acción de cambiar entre procesos de ejecución donde se salva el estado del proceso ejecutado actualmente en su PCB y cargar el estado desde el PCB del nuevo proceso el cual vamos a ejecutar.

Además cuando el proceso es muy grande, y no se es capaz de almacenarlo en la memoria real del equipo, lo que ocurre es que se carga una parte del proceso a ejecutar y en la memoria física del equipo queda el resto del proceso, cuando se necesitan otras partes del mismo proceso, ocurre lo que se denomina como

swapping, que es el cambio de la zona cargada actualmente y que se está ejecutando por otra página de memoria que vamos a ejecutar a partir de ese momento.

➢ **Algoritmos de planificación de procesos**
- First In - First Out (FIFO)
- Shortest Job First
- Shortest Remaining Job First
- Round Robin ~por turnos
- Colas de Prioridad

d. Problemas de la programación concurrente

Cuando se programa a alto nivel, la visión que se tiene es determinista o lo que es lo mismo, esperamos que la salida de nuestro programa sea siempre la misma para lo que hemos programado.

Con el uso de la programación concurrente rompemos este determinismo, ya que en dos momentos diferentes, el uso de una misma instrucción puede tener diferentes resultados, al acceder el proceso desde varios puntos a la misma instrucción de manera simultánea o casi simultánea.

Esto hace, que no siempre el resultado esperado sea el mismo y por tanto a la hora de programar una solución debemos tener en cuenta que pueden existir estos problemas de concurrencia.

Estas secciones las cuales pueden ser utilizadas por varios procesos de manera casi simultánea, se denominan secciones críticas, y lo que se produce al acceder a dicha sección compartida al mismo tiempo es una condición de carrera, haciendo que los procesos modifiquen dicha variable y que al final prevalezca el valor del proceso que la modifica último.

e. Creación de procesos en C

Para crear procesos en Linux y otros SO a través de C, tenemos tres operaciones diferentes:

- system(): El cual ejecutará un comando del sistema operativo o un subproceso. Podemos por ejemplo escribir *system("pause")* y esperar a que el usuario presione alguna tecla para continuar el programa
- fork(): Usada para crear nuevos procesos en sistemas Linux y Unix, el proceso creado se denomina proceso hijo.
- execl(): Reemplaza el proceso actual con uno nuevo, ejecutando las instrucciones que le demos como argumentos.

f. Comunicación entre procesos

Para la comunicación entre procesos en sistemas operativos similares a Linux / Unix se disponen de varias formas:

➢ **Tuberías con nombre**

Mecanismo de comunicación entre procesos que permite la transferencia de datos entre ellos a través de un archivo especial en el sistema de archivos

➢ **Tuberías sin nombre**

Este mecanismo se establece directamente en la memoria RAM del sistema sin necesidad de crear un archivo en el sistema de archivos del sistema operativo.

➢ **Colas de mensajes**

Este mecanismo permite que se compartan datos de manera estructurada y asíncrona.

➢ **Semáforos**

Herramienta de sincronización entre procesos utilizada en programación concurrente para evitar condiciones de carrera y gestionar el acceso a recursos compartidos entre procesos o hilos.

➢ **Segmentos de memoria compartida**

Regiones de memoria que pueden ser accedidas por múltiples procesos de manera simultánea

03. Programación Multihilo

a. Concepto de hilo

Un hilo es conocido como un proceso ligero o secuencia de código dentro del contexto de un proceso.

Los hilos tienen unos recursos asociados que son:

- Estado de ejecución
- Contexto propio
- Pila para albergar la ejecución
- Espacio para almacenar las variables locales
- El acceso a la memoria y los recursos del proceso es compartido por todo los hilos del proceso.

Un hilo tiene una mejor respuesta a las solicitudes de usuarios, un mejor acceso a los recursos comunes y ahorrando así en métodos de comunicación, además se ahorra en recursos de memoria ya que se trabaja con los reservados para el proceso. La programación multihilo hace que nos aproximemos al paralelismo real en programación.

Los diferentes estados por los que pasa un hilo son los siguientes:

- *Creado*: Un hilo se crea cuando se crea un proceso, además este hilo puede crear otros dentro del mismo proceso. Los recursos asociados también se crean a la vez que se crea el hilo
- *Bloqueado*: Un hilo necesita esperar a que ocurra un suceso, salva sus registros y se bloquea, esto hace que el procesador pueda ejecutar otro hilo mientras este permanece bloqueado.
- *Desbloqueado:* Cuando el suceso se produce, el hilo pasa a la cola de listos.
- *Terminado:* Un hilo finaliza y se liberan los recursos asociados a él.

b. Hilos a nivel Usuario vs nivel Kernel

➢ **Nivel usuario**

La administración de los hilos se hace a nivel de la aplicación o del usuario, son ligeros y eficientes, pero limitados en funcionalidades.

➢ **Nivel kernel**

La administración de los hilos se hace a nivel del kernel del sistema operativo, más robustos y versátiles pero con más coste de rendimiento.

c. Hilos POSIX

Con el estándar POSIX podemos crear aplicaciones portables y tiene una biblioteca para el manejo de hilos que es *pthread.* Con el manejo de múltiples hilos podemos llevar a cabo varias tareas dentro de un proceso. Esto nos permite:

- El manejo de eventos asíncronos
- Comparten espacio de direcciones y descriptores de archivos entre ellos
- Se reducen los tiempos de espera al realizar las tareas en diferentes hilos
- Se consumen menos recursos en la creación de un hilo a diferencia de crear un proceso.

Además un proceso es único en el sistema, mientras que el hilo es único dentro del contexto de un proceso dado, un proceso es identificable fácilmente con un PID pero el hilo no es tan fácil de localizar.

d. Creación de hilos

Los hilos POSIX se identifican por el struct pthread_t. Para la creación de hilos y su identificación tenemos las siguientes estructuras asociadas:

- ***pthread_equal(pthread_t tid1, pthread_t tid2);*** → Comprueba si los IDs de los hilos son iguales o no, devolviendo distinto de cero o cero respectivamente.
- ***pthread_t pthread_self*** → Usado para que el hilo imprima su propia ID.
- ***pthread_create(pthread_t *restrict tidp, const pthread_attr_t *restrict attr, void *(*start_rtn)(void), void *restrict arg)*** → Con esta función

crearemos un hilo nuevo.

pthread_t *restrict tidp → El primer argumento una dirección tipo pthread_t, la cual contendrá el ID del hilo recien creado.

const pthread_attr_t *restrict attr → El segundo argumento son los atributos que queremos que contenga el hilo.

void *(*start_rtn)(void) → El tercer argumento es un puntero de función, la cual es donde el hilo iniciará su ejecución

void *restrict arg → El último argumento son los argumentos que queremos pasarle a la función del tercer argumento.

- ***pthread_join(pthread_t thread, void **rval_ptr)*** → Con esta función nos aseguramos de que el hilo padre no termine antes que el hijo. Esta función la llamamos desde el hilo padre y el primer argumento es el ID del hilo al cual esperar y un puntero donde almacenar el valor de retorno si hay alguno.
- ***pthread_exit (void *rval_ptr)*** → Con esta función el hilo puede terminar por sí mismo su ejecución.