**Trabajo Práctico N°2 – ISO.**

1. **Conceptos**
2. Un sistema operativo es un **software** diseñado para controlar y ejecutar otro software y que el mismo utilice el hardware.

* Actúa como intermediario entre el usuario y el hardware.
* Gestiona el hardware
* Brinda comodidad, eficiencia y una interfaz entre las aplicaciones y el hardware
* Controla la ejecución de los **procesos**

1. Los **componentes/aspectos** del hardware que se precisan para cumplir los objetivos de un S.O son:

* **CPU** (Unidad Central de Procesamiento) -> Procesador
  + Es el "cerebro" de la computadora. El SO debe gestionar y asignar tiempo de CPU a los diferentes procesos y programas (planificación de procesos), manejar interrupciones y proporcionar multiprocesamiento
* Memoria **RAM**
  + Es el espacio de trabajo **volátil** donde se cargan los programas y datos en ejecución. Uno de los objetivos clave del SO es la **gestión de memoria**: **asignar** y **liberar** espacio, **proteger** la memoria de un proceso de otro, y utilizar **técnicas** como la memoria virtual.
* Memoria **Secundaria** (HDD, SSD, Etc…)
  + Proporcionan almacenamiento **persistente** (**no volátil**). El SO los gestiona a través de su sistema de archivos, que se encarga de almacenar, organizar, recuperar y proteger la información (archivos y directorios) a largo plazo
* Dispositivos de **E/S**
  + Son la interfaz para la **comunicación** con el **mundo** **exterior**. El SO debe controlar estos dispositivos a través de controladores (drivers), manejar las solicitudes de E/S y abstraer su complejidad para el usuario
* BUS del Sistema
  + Es el canal de comunicación que **conecta todos los componentes internos del hardware** (CPU, memoria, dispositivos de E/S). El SO debe interactuar con este bus para coordinar la transferencia de datos entre los distintos componentes.

1. Componentes:
2. **Kernel**
   * Es el corazón del SO. Es la parte más crítica y privilegiada, que se carga en memoria al iniciar la computadora y permanece allí. Gestiona directamente el hardware y proporciona servicios esenciales a los demás componentes y a las aplicaciones
3. **Shell**: GUI / CUI o CLI
   * Es la interfaz entre el usuario y el kernel
4. **Herramientas**: Editores, compiladores, librerias, etc…
5. **Administrador de memoria**: Módulo (a menudo parte del kernel) dedicado exclusivamente a la memoria
   * Lleva un registro de cada byte en memoria, decidiendo qué procesos se cargan en memoria y dónde. Se encarga de técnicas como paginación, segmentación y memoria virtual
6. **Sistema de Archivos:** El componente responsable de la organización y manipulación de los datos almacenados
   * Gestiona cómo se almacenan y recuperan los datos en los discos. Crea la estructura de directorios (carpetas) y se encarga de los permisos de acceso, el nombre de los archivos y sus atributos
7. **Administrador de Procesos:** El módulo que trabaja con la creación, ejecución y eliminación de procesos e hilos
   * Asigna recursos a los procesos, sincroniza la comunicación entre ellos (IPC - Comunicación entre Procesos) y evita condiciones de carrera o interbloqueos (**deadlocks**)
8. **Drivers**: Programas especiales que actúan como traductores entre el hardware específico y el sistema operativo
   * Permiten que el kernel se comunique con un dispositivo de hardware (una impresora, una tarjeta gráfica, etc.) sin necesidad de conocer sus detalles específicos. Cada dispositivo requiere su propio driver
9. Una llamada al sistema (**System** **Call**) es la forma en la que los programas de usuario acceden a los servicios del sistema operativo (Es una RUTINA o FUNCIÓN).
   * Los **parámetros** asociados a las llamadas pueden pasarse de varias maneras: por registros, bloques, tablas de memoria o stack (pila)
   * Se ejecutan en **modo kernel**o **supervisor**
   * Los S.O cambian su SYSTEM CALLs dependiendo la arquitectura del procesador

Es posible implementar los llamados al sistema mediante instrucciones de código. Ejemplo: El **read** en assembler es una librería que ejecuta un proceso, el cuál dentro produce una interrupción para pasar al modo kernel. Allí se verifica quien precisa del uso del kernel mediante el uso de un registro y se procede a ejecutar la instrucción (SysCall Handler). *\*Falta detallar el retorno al llamado… Está en carpeta*

1. Un **programa** es un archivo con código dentro que está almacenado en memoria secundaria esperando a ser ejecutado.

* Es estático
* No tiene PC (Program Counter)
* Existe desde que se edita hasta que se borra **(Ciclo de vida)**

Un **proceso** es un programa en **ejecución**. Es una entidad de abstracción, es decir, tiene un “algo” en CPU pero falta algo…

* Es dinámico, es quién ejecuta el programa
* Tiene un PC (Program Counter)
* Existe desde que se solicita hasta que se termina **(Ciclo de vida)**

1. La información mínima que debe tener el Kernel sobre un proceso es:

* **PID** del proceso (Identificador)
* **Estado** del proceso (Si está en *listo****,*** *ejecutándose, bloqueado o esperando*)
* **PC**
* **Registros** de la CPU
* **Espacio de direcciones de memoria** (Memoria asignada al proceso)
* **Prioridad** o **Información de planificación** (Tiempo dedicado al proceso)
* **Información de E/S y archivos**
* **Privilegios** (Información de su propiedad. UID, GID, usuario, etc..)

La estructura de datos asociada al proceso donde se almacena dicha información es la **PCB** (Process Contol Block). El PCB es la "*cédula de identidad*" completa de un proceso dentro del sistema operativo. Contiene toda la información mínima y necesaria para que el kernel pueda detener un proceso y luego reanudarlo más tarde sin que este se percate de la interrupción, permitiendo así la multitarea.

1. Los **algoritmos de planificación (scheduling)** son el componente del kernel responsable de decidir qué proceso en estado "Listo" se asignará a la CPU a continuación.

Sus objetivos se pueden clasificar en dos categorías principales: **orientados al usuario** (visibles desde fuera del sistema) y **orientados al sistema** (internos y de eficiencia).

* Orientados al usuario:
  + Minimizar el Tiempo de Retorno (Turnaround Time)
  + Minimizar el Tiempo de Espera
  + Minimizar el Tiempo de Respuesta
* Orientados al sistema
  + Maximizar el rendimiento (Throughput)
  + Maximizar la utilización del CPU
  + Equidad
  + Previsibilidad

1. Los algoritmos **apropiativos (Preemptive)** existen en situaciones en los que el proceso en ejecución es expulsado por la **CPU**. Los algoritmos **no apropiativos (Non- Preemptive)** son cuando los procesos se ejecutan hasta que el mismo, por **cuenta** **propia**, abandona la **CPU**.

* El **SHORT-TERM SCHEDULER** determina cuál de todos los procesos que estén listos para ejecutarse se ejecutará a continuación en un ambiente multrporgramado.
* El **LONG-TERM SCHEDULER** controla el grado de multi-programación, es decir, la cantidad de procesos en memoria. *\*Puede no existir y absorber esta tarea el short-term.*
* El **MEDIUM-TERM SCHEDULER** reduce el grado de multiprogramación (Solo si ES NECESARIO). “Saca” temporalmente de la memoria los procesos que sea necesario para mantener el equilibro del sistema.
* Los NOMBRES de los SCHEDULER provienen de la frecuencia de ejecución

1. El **DISPATCHER** se encarga de realizar el cambio de contexto, cambio de modo de ejecución, “Despacha” el proceso elegido por el **short term**.

El **LOADER** carga en memoria el proceso elegido por el **long term**.

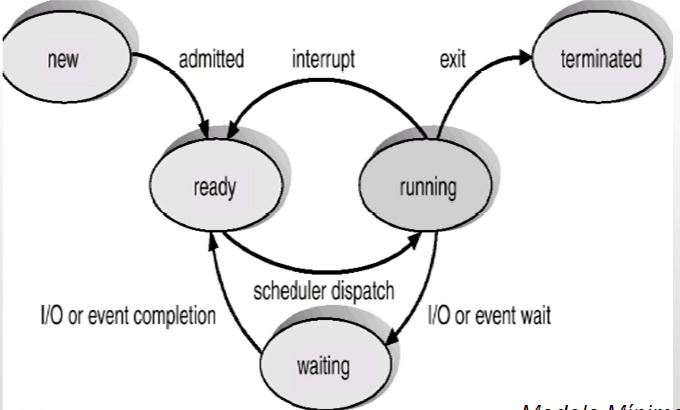
1. Que un proceso sea **CPU-Bound** significa que el mismo se pasa la mayor parte del tiempo utilizando la CPU. Que un proceso sea **I/O-Bound** significa que pasa la mayor parte del tiempo esperando por I/O (E/S)
2. Los estados posibles por los que puede atravesar un proceso son:

* **NEW:** Un usuario “dispara” el proceso. Un proceso es creado por otro, su padre

**En este estado** se crean las estructuras asociadas (PCB), el proceso queda en la *cola de procesos*, normalmente en espera de ser cargado en memoria

* **READY:** El proceso queda en estado listo, solo necesita que se le asigne CPU
* **RUNNING:** Se le asignó CPU al proceso y este lo tendrá hasta que se termine el período de tiempo asignado (**quantum o time slice**), termine o hasta que se necesite alguna operación de E/S
* **WAITING**: El proceso precisa que se cumpla un evento esperado para continuar. Este puede ser una E/S solicitada o la llegada de una señal por parte de otro proceso.

**Sigue en memoria** pero sin CPU. Al cumplirse el evento para a LISTO (Ready)



**Transiciones:**

* **New-Ready**: Por elección del scheduler de largo plazo, el **loader** carga en memoria el programa.
* **Ready-Running**: Por elección del scheduler de corto plazo, el **dispatcher** asigna la CPU.
* **Running-Waiting**: El proceso se “pone a dormir”, esperando por un evento.
* **Waiting**-**Ready**: Terminó la espera y compite nuevamente por la CPU.

1. De los schedulers anteriormente mencionados el que se encarga de las transiciones en los estados es el **LONG-TERM** (Encargado de elegir el proceso para cargarlo en memoria, queda en LISTO) y el **SHORT-TERM** (Elige el proceso para asignarle CPU, queda en RUNNING)
2. **Tiempo de Retorno (Tr):** Tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución.

**Tiempo de Espera (Te):** Tiempo que el proceso se encuentra en el sistema esperando, es decir, que pasa sin ejecutarse (Tr - Tcpu).

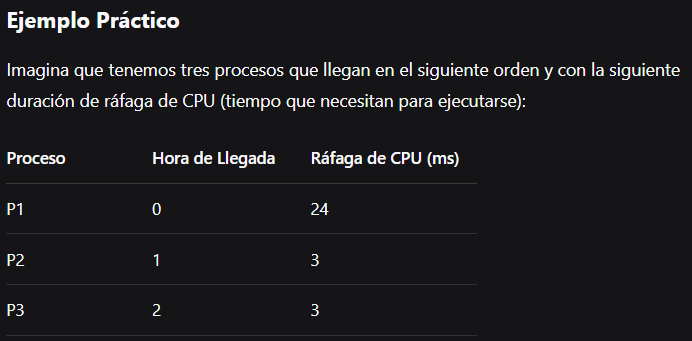
1. **Tiempo Promedios (TPR y TPE):** Tiempos promedio de retorno y espera. Promedio calculado de los tiempos individuales de cada proceso de lote.
2. **Tiempo de Respuesta:** Es el LAPSO DE TIEMPO que transcurre desde que un usuario (o proceso) emite una solicitud o comando hasta que se produce la primera respuesta perceptible o útil.

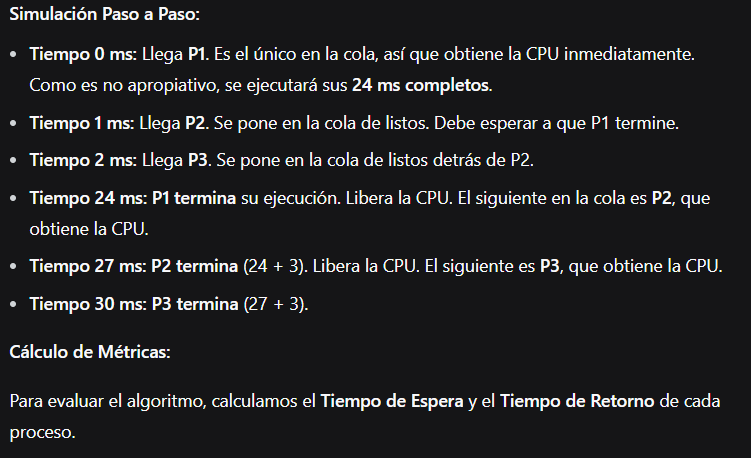
2) **Algoritmos de Scheduling**

1. **First Come First Served (FCFS):** Su funcionamiento se basa en una cola **FIFO** (First In, First Out).

* Los procesos se atienden en el orden estricto de llegada a la cola de LISTO
* Quien llega primero obtiene la CPU primero, la mantiene hasta que termina
* Solo cuando el proceso en ejecución libera la CPU, se asigna la CPU al siguiente proceso de la cola
* No son apropiativos

**Ejemplo:**





**SJF – Shortest Job First**: Es un algoritmo de planificación que **da prioridad al proceso con la ráfaga de CPU (tiempo de ejecución) más corta siguiente**

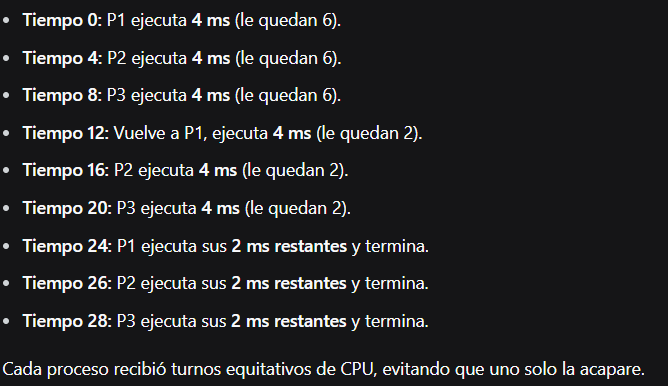
* Cuando la CPU se libera, se selecciona del estado “READY” el proceso que **tenga el tiempo de ráfaga siguiente más corto**
* No se interrumpe el proceso hasta que termine su tiempo en la CPU
* No apropiativo

**Ejemplo:** Llegan P1 (duración 6), P2 (duración 8), P3 (duración 7). El planificador SJF elige el orden: P1 (6), P3 (7), P2 (8), porque **6 < 7 < 8.**

**Round Robin:** Es un **algoritmo apropiativo** diseñado para sistemas de tiempo compartido. Asigna la CPU a cada proceso por un **quantum** **de** **tiempo** **fijo** (ej: 10-100 ms). Si un proceso no termina en ese tiempo, **es expulsado de la CPU** y se vuelve a colocar al final de la cola de procesos listos.

* El QUANTUM (Q) es una porción de tiempo fijo. Es el **parámetro** **más importante** del algoritmo
* Los nuevos procesos se agregan al final de la cola. Un proceso, al ser expulsado, se agrega al final
* Su **objetivo** es proporcionar buenos **tiempos de respuesta y equidad (fairness)**, ya que todos los procesos reciben una porción de CPU de forma cíclica

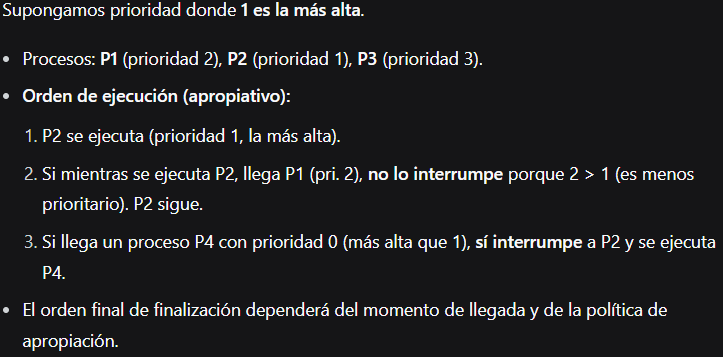
**Ejemplo:** Tres procesos (P1, P2, P3) con **ráfaga de 10 ms** cada uno y un **quantum** de **4 ms**.



**Prioridades:** A cada proceso se le asigna un **número de prioridad** (entero). La CPU se asigna al proceso que esté en la cola de listos y que tenga **la prioridad más alta** (que puede ser el número mayor o el menor, dependiendo de la implementación del sistema).

* La asignación de prioridades puede ser **estática** (Fijas durante toda la vida del proceso) o **dinámicas** (Cambian durante la ejecución)
  + Ejemplo dinámica: Prioridad baja si usa mucha CPU, alta si está esperando por E/S
* Pueden ser **No Apropiativos**, el proceso con mayor prioridad obtiene la CPU y la mantiene hasta terminar
* Pueden ser **Apropiativos**, un proceso con mayor prioridad llega al estado “READY” y **le quita inmediatamente la CPU** al proceso de menor prioridad en ejecución
* Si varios procesos tienen la misma prioridad, se suele usar otro algoritmo para desempatar (**FCFS**)

**Ejemplo**:



1. De los 4 algoritmos quien tiene el **parámetro** más claro y crítico a la hora de su funcionamiento es el **Round Robin (RR)** con el **Quantum** (Time Slic o Quanta). FCFS, SJF **no tienen un parámetro directo de configuración**. El algoritmo de prioridades tiene como parámetro la **propia prioridad asignada a cada proceso** (Valor específico para cada proceso).
2. No existe un único algoritmo adecuado para todos los casos. La elección del más adecuado depende del tipo de sistema y de los objetivos que se prioricen:

* **FCFS**: Es más adecuado para sistemas por lotes (Batch) simples donde la predictibilidad y la simplicidad son más importantes que el rendimiento interactivo
* **SJF**: Más adecuado para entornos especializados donde se puede predecir con alta precisión el tiempo de ejecución de los procesos
* **PRIORIDADES:** Es más adecuado para sistemas donde los procesos tienen **claros niveles de importancia o urgencia** distintos. Muy usado en **sistememas de tiempo real y kernels modernos**
* **Round Robin (RR):** Más adecuado para **sistemas de tiempo compartido e interactivos** (Escritorios, servidores de terminales, shells multiusuario). Es el **algoritmo por excelencia para este fin**. Proporciona un buen equilibrio entre el **TR** y **EQUIDAD**.

El “Más común” para sistemas de propósitos general (Windows, Linux, MacOs) es la combinación de **Round Robin y Prioridades** con **envejecimiento**. Los sistemas modernos **nunca usan un solo algoritmo**, utilizan esquemas **híbridos** o de **múltiples** colas:

1. **Ventajas** **FCFS**:

* Muy simple de implementar. Solo requiere COLA FIFO
* Facil de entender
* Justo por orden de llegada

**Desventajas FCFS:**

* Bajo rendimiento para métricas de tiempo de espera promedio
* No es apropiativo, lo que hace que no sea adecuado para entornos interactivos
* Poco eficiente si los procesos tienen grandes diferencias en sus tiempos de ejecución
* Efecto “**convoy”**; Un proceso largo retrasa al resto

\*

**Ventajas** **SJF**:

* Minimiza el tiempo de espera promedio, es **optimo**
* **Eficiente** para reducir el tiempo total de ejecución de un lote de procesos

**Desventajas SJF:**

* **Inanición (Starvation):** Los procesos largos pueden postergarse indefinidamente
* **Impredecible**: Es imposible saber con certeza la duración de la siguiente ráfaga
* Dificultad en la práctica

\*

**Ventajas** **Round Robin**:

* **Excelente** tiempo de respuesta
* **Equitativo**, todos los procesos reciben una porción de CPU
* **Apropiativo**, evita que un proceso robe la CPU

**Desventajas Round Robin:**

* **Alto overhead** (sobrecarga de consumo) por cambios de contexto si el quantum es pequeño
* **TR** puede ser alto para procesos largos
* El rendimiento depende **criticamente** de la elección del **quantum**

\*

**Ventajas Prioridades:**

* **Flexible**; Permite tratar procesos de distinta importancia de forma diferenciada
* Puede priorizar tareas críticas
* Se puede combinar fácilmente con otros algoritmos, ejemplo RR

**Desventajas Prioridades:**

* Los procesos de baja prioridad pueden no ejecutarse nunca
* Si no se usa **envejecimiento**, el sistema puede volverse injusto
* La asignación de prioridades puede ser compleja o subjetiva

3)