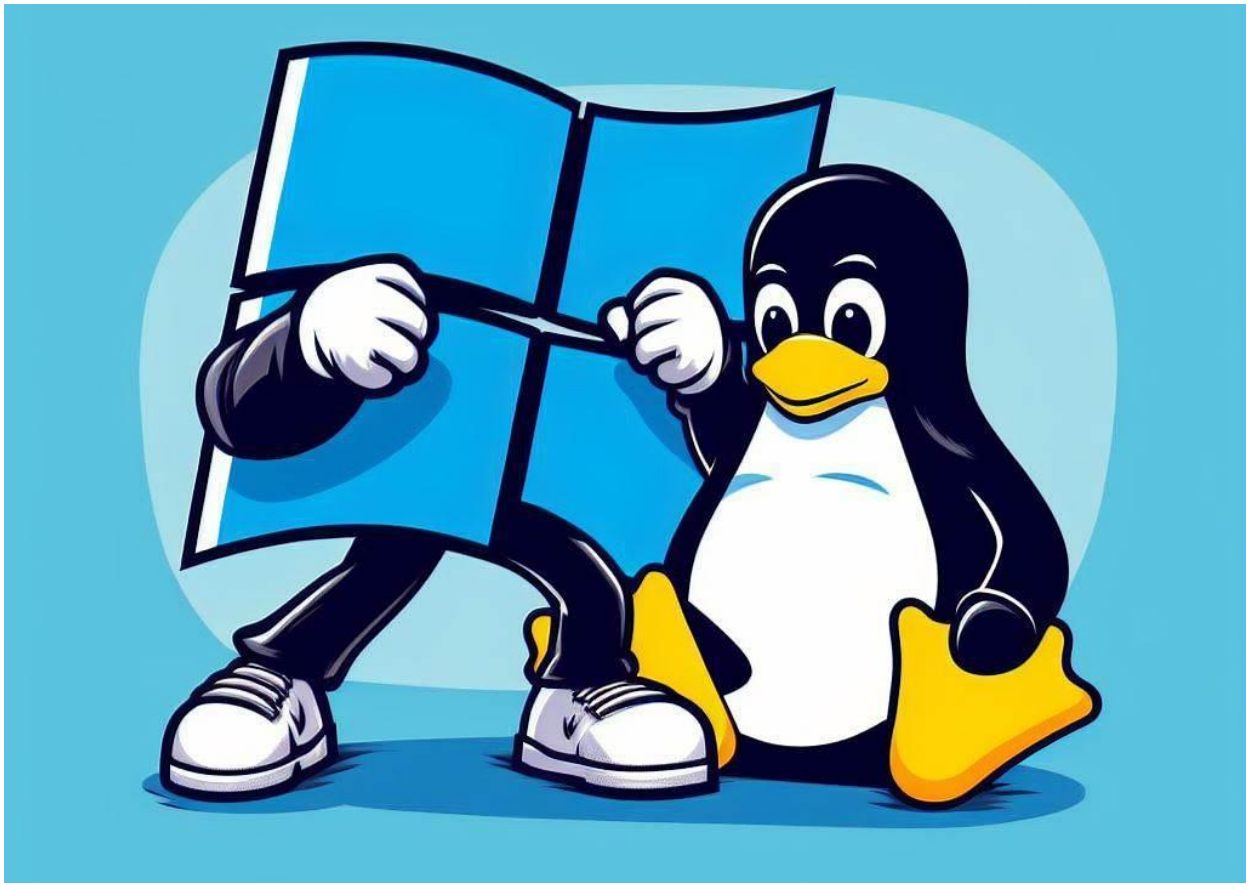
INTRODUCCIÓN A LOS SISTEMAS OPERATIVOS

PROCESOS

Resumen teoría



Caporal Nicolás

Facultad de Informática
UNLP

2023

ÍNDICE

Clase 1

¿QUÉ ES UN PROCESO?

Un proceso (también job o tarea) es un programa en ejecución.

Es importante diferenciar el concepto de **proceso** con el de **programa**.

Un **programa** es algo estático, no tiene program counter. Existe desde que se compila y se genera el .exe, hasta que se borra. Está en memoria secundaria.

Un **proceso** es un programa en ejecución. Es dinámico, y tiene Program Counter. Existe desde que se lo "dispara" hasta que termina su ejecución. En ese tiempo puede cambiar de estado y su contenido.

Visto desde el SO, el proceso es una entidad que sirve para abstraer la ejecución

COMPONENTES DE UN PROCESO

Como mínimo, un proceso para poder ejecutarse incluye:

- Sección de Código (texto)
- Sección de Datos (variables globales)
- Stack(s) (Datos temporarios: parametros, variables temporales y direcciones de retorno)

STACKS

Un proceso cuenta con 1 o más Stacks. En general son 2, una pila para cuando se ejecuta en modo Usuario y otra para cuando se ejecuta en modo Kernel.
(Ver Enfoque 2 de diseño del Kernel para comprender el porqué)

Las Stacks se crean automáticamente y su tamaño se ajusta de manera dinámica en tiempo de ejecución (run-time).

ATRIBUTOS DE UN PROCESO

- Identificación del proceso (PID), y del proceso padre (PPID)
- Identificación del usuario que lo disparó (UID)
- Si hay estructura de grupos, grupo que lo disparó (GID)
- En ambientes multiusuario, desde que terminal y quien lo ejecutó

PROCESS CONTROL BLOCK (PCB)

El PCB es una estructura de datos asociada al proceso (abstracción) donde se almacenan los atributos recién mencionados e información relacionada a la memoria que está usando.

Existe una PCB por proceso.

Es lo primero que el SO tiene que crear cuando se crea un proceso, ya que es donde se empiezan a inicializar los datos necesarios, y lo último que se borra cuando termina.

Contiene la información asociada con cada proceso, en particular:

- PID, PPID, etc
- Valores de los registros de la CPU (PC, AC, etc, para guardar contexto)
- Planificación (estado, prioridad, tiempo consumido, etc)
- Ubicación (representación) en memoria
- Accounting (contabilidad, estadisticas que lleva el SO para tomar decisiones)
- Entrada salida (estado, pendientes, etc)

¿QUÉ ES EL ESPACIO DE DIRECCIONES DE UN PROCESO?

Es el conjunto de direcciones que ocupa el proceso y delimita hasta donde puede usar.

No incluye su PCB o tablas asociadas, son sólo sus datos.

En modo **Usuario** un proceso puede acceder sólo a su espacio de direcciones.

En modo **Kernel** se puede acceder a estructuras internas, como a la PCB, o a espacios de direcciones de otros procesos. (Recordemos que en modo Kernel se puede hacer todo)

EL CONTEXTO DE UN PROCESO:

Es toda la información que el SO necesita mantener para administrar el proceso, y la CPU para ejecutarlo correctamente.

Son parte del contexto, los registros de cpu, inclusive el contador de programa, prioridad del proceso, si tiene E/S pendientes, que archivo se abrió, etc.

CAMBIO DE CONTEXTO (CONTEXT SWITCH)

Se produce cuando la CPU “saca” a un proceso que se estaba ejecutando para darle el tiempo a otro.

El proceso saliente pasa a espera y después retornará a la CPU. Para poder seguir la ejecución desde el mismo punto donde se lo sacó, es que se debe resguardar el contexto.

Cuando se le otorga la CPU a un proceso que estaba en espera, se debe cargar el contexto de este nuevo proceso entrante, y se comienza desde la siguiente instrucción a la última ejecutada en dicho contexto.

Realizar el cambio de contexto lleva tiempo, por tanto es tiempo no productivo de CPU.

El tiempo que consume depende del soporte de hardware

¿Y EL KERNEL? ENFOQUES DE DISEÑO:

El Kernel es un conjunto de módulos de software, pero **NO** es un proceso.
El concepto de proceso se asocia solo a programas del usuario.

Pero si, se ejecuta en el procesador como cualquier otro proceso.

Para manejar esto, se plantean dos enfoques básicos que utilizan los sistemas operativos.

ENFOQUE 1: EL KERNEL COMO ENTIDAD INDEPENDIENTE

Esta arquitectura es utilizada por los primeros sistemas operativos.

Se ve al Kernel como una entidad independiente que está “a la par” de los procesos y se ejecuta en modo privilegiado (modo Kernel).

El kernel tiene su propia región de memoria, y su propio Stack.
Finalizada su actividad, le devuelve el control al proceso (o a otro diferente).
*Por más que se maneje de manera similar, el Kernel **NO** es un proceso.*

Cada vez que ocurre una interrupción (por ej. el Clock) hay que resguardar el contexto del proceso que estaba en ejecución, y ahí pasar a la entidad “Kernel” que maneja la interrupción.

Como ya se dijo, el cambio de contexto lleva tiempo, y la interrupción de Clock ocurre todo el tiempo de manera constante, por tanto este enfoque tiene mucho tiempo ocioso que la CPU no está ejecutando.

ENFOQUE 2: EL KERNEL “DENTRO” DEL PROCESO

Se ve al Kernel como una colección de rutinas que el proceso utiliza.

El “Código” del Kernel se encuentra dentro del espacio de direcciones de cada proceso y se ejecuta en el **MISMO** contexto que cada proceso de usuario.

Entonces, en este caso, una interrupción por Clock no genera un cambio de contexto. Porque el Kernel ya está “dentro” del proceso. En vez de realizar un cambio de contexto de la CPU con todo lo que ello conlleva, solo hay que saltar a una dirección de memoria que pertenece al código del Kernel y cambiar el bit de modo de la CPU.

Lo malo es que un proceso tiene un espacio de direcciones limitado, y al estar código y datos el kernel ahí adentro, cada proceso tiene menos espacio para ocupar.

Dentro de un proceso se encuentra el código del programa (user) y el código de los módulos de software del SO (kernel).

*Por esto cada proceso tiene 2 pilas, una en modo Usuario y otra en modo Kernel, porque se considera que es el **mismo** proceso que ejecuta en ambos modos.*

El proceso es el que se Ejecuta en Modo Usuario y el kernel del SO se ejecuta en Modo Kernel (cambio de modo).
*Por más que se vean juntos, el Kernel **NO** es un proceso.*

En resumen:

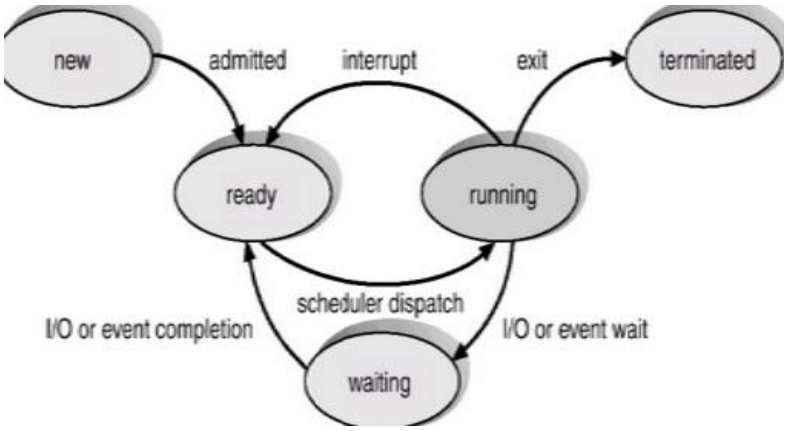
- El código del Kernel es compartido por todos los procesos
  (En administración de memoria veremos el “como”)

- Cada interrupción (incluyendo las de System Call) es atendida en el contexto del proceso que se encontraba en ejecución
  - Pero en modo Kernel!!! (se pasa a este modo sin necesidad de hacer un cambio de contexto completo)
  - Si el SO determina que el proceso debe seguir ejecutándose luego de atender la interrupción, cambia a modo usuario y devuelve el control. Es más económico y performante

Clase 2

ESTADOS DE UN PROCESO

En su ciclo de vida, un proceso pasa por diferentes estados.



New: El proceso se acaba de crear, se crea la PCB y se inicializan sus datos, pero que aún no ha sido cargado en memoria.

Ready: El proceso que está cargado y preparado en memoria, listo para ejecutar. En este estado el procesos compite por CPU (en la cola de readys).

Running: Luego del cambio de contexto, se le da la CPU y el proceso utiliza la CPU y está ejecutando.

En este capítulo se suponen computadores con un único procesador, de forma que solo un proceso, a lo sumo, puede estar en este estado en un instante dado.

Hay 3 posibles salidas de este estado:

- Si el proceso termina lo que tenía que hacer, pasa al estado de Terminated.
- Si la CPU saca al proceso (por ejemplo porque se acabó su Quantum, o porque llegó un proceso con mayor prioridad), se realiza un cambio de contexto y el proceso pasa nuevamente al estado de Ready.
- Si el proceso quiere realizar E/S, el SO saca al proceso de la CPU para que no quede ociosa, y este pasa a un estado de Waiting, esperando por el dispositivo de E/S

Waiting: El proceso que no puede ejecutar hasta que se produzca cierto suceso, como la terminación de una operación de E/S. Una vez realizada, se pasa nuevamente al estado de Ready

Terminated: El proceso termina y ejecuta la System Call ‘exit’. Por lo que pasa al estado terminated. Este estado es el opuesto al New, se empiezan a des-inicializar los datos asociados para finalmente liberar y eliminar la PCB.

Suspend: Ver en

COLAS EN LA PLANIFICACIÓN DE PROCESOS

Para realizar la planificación, el SO utiliza la PCB de cada proceso como una abstracción del mismo. Las colas de planificación son estructuras de datos, que relacionan PCB’s.

Las PCB se enlazan en Colas siguiendo un orden determinado, en función del estado del proceso.

Ejemplos:

Cola de procesos:
Contiene todas las PCB de procesos en el sistema

Cola de procesos listos:
Contiene PCB de procesos residentes en memoria principal esperando para ejecutarse

Cola de dispositivos:
Contiene PCB de procesos esperando por un dispositivo de I/O

Si bien se llaman “colas de planificación”, se pueden implementar otras estructuras como una lista ordenada por prioridad, o un arbol para busqueda más eficientemente. El nombre “cola” es teórico.

MÓDULOS DE LA PLANIFICACIÓN

Son módulos (software) del Kernel que realizan distintas tareas asociadas a la planificación.

Se ejecutan y llaman ante determinados eventos que así lo requieren:

- Creación/Terminación de procesos
- Eventos de Sincronización o de E/S
- Finalización de lapso de tiempo
- Etc…

Los principales módulos son:

- Scheduler de long term
- Scheduler de short term
- Scheduler de medium term

Su nombre proviene de la frecuencia de ejecución. El Short se ejecuta de manera muy frecuente, el Long se ejecuta poco.

DISPATCHER

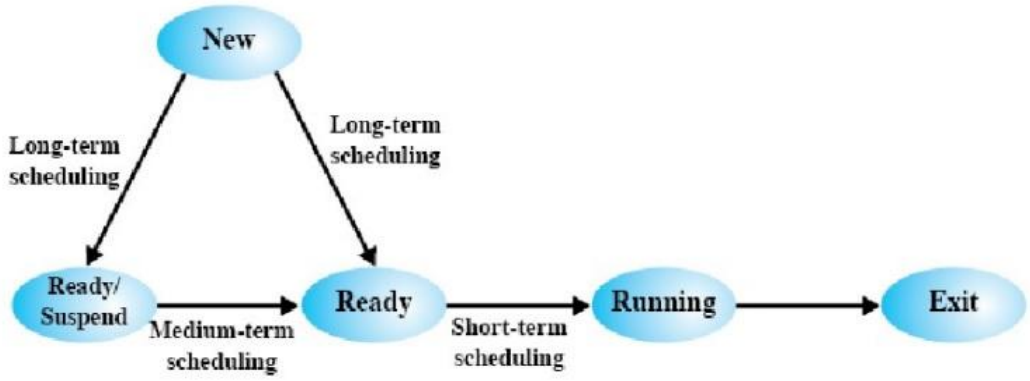
Hace el cambio de contexto, cambio de modo de ejecución…"despacha" el proceso elegido por el Short Term (es decir, "salta" a la instrucción a ejecutar).

Está ligado al Short Therm Scheduller.

LOADER

Carga en memoria el proceso elegido por el long term.

Está ligado al Long Term Scheduller.



LONG TERM SCHEDULLER

Este planificador es quien se encarga de admitir procesos del estado de “New” al estado de “Ready”.

Controla el grado de multiprogramación, es decir, la cantidad de procesos en memoria. Puede no existir este scheduler y absorber esta tarea el de short term.

New -> Ready *or* ***New -> Suspend***

MEDIUM TERM SCHEDULLER (SWAPPING)

Si es necesario, reduce el grado de multiprogramación

Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema.

Términos asociados: swap out (sacar de memoria), swap in (volver a memoria).

Ready -> Suspend

SHORT TERM SCHEDULLER

Decide a cuál de los procesos en la cola de listos se elige para que use la CPU.

Términos asociados: apropiativo, no apropiativo, algoritmo de scheduling.

Ready -> Running

Clase 3

COMPORTAMIENTO DE LOS PROCESOS

Los procesos alternan entre ráfagas de CPU y IO.

Es decir, en los SO de uso general, la CPU está multiplexada (compartida) en el tiempo.

Por este motivo es que el Short Term Scheduller (quien se encarga de seleccionar procesos de Ready para pasarlos a Running) es el que más se ejecuta.

Según su comportamiento, los podemos clasificar en:

- CPU Bound: La mayor parte de tiempo del proceso es utilizando la CPU
- I/O Bound: La mayor parte de tiempo del proceso es esperando por E/S

Tener en cuenta, que la velocidad de la CPU es mucho más rápida que la de los dispositivos de E/S. Es una buena estrategia atender rápidamente a los procesos I/O Bound para mantener el dispositivo de E/S ocupado y aprovechar la CPU para procesos CPU/Bound

PLANIFICACIÓN

Entonces ¿Qué es planificar la CPU?

Planificar la CPU es determinar cuál proceso de los que están listos para ejecutarse, es el que se va a ejecutar a continuación (en un ambiente multiprogramado).

Un **algoritmo de planificación** es el algoritmo utilizado para realizar la planificación del sistema. Es decir, el cómo y por qué el algoritmo toma esa desisión.

ALGORITMOS APROPIATIVOS Y NO APROPIATIVOS

En los algoritmos **Apropiativos** (preemtive) existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU. (Por ejemplo, que llegue un proceso de mayor prioridad)

En los algoritmos **No Apropiativo** (nonpreemptive) los procesos se ejecutan hasta que el mismo (por su propia cuenta) abandone la CPU

– Se bloquea por E/S o finaliza

– No hay decisiones de planificación durante las interrupciones de reloj

CATEGORÍAS DE LOS ALGORITMOS DE PLANIFICACIÓN

Según el ambiente es posible requerir algoritmos de planificación diferentes, con diferentes metas:

- **Equidad:** Otorgar una parte justa de la CPU a cada proceso.
- **Balance:** Mantener ocupadas todas las partes del sistema.

PROCESOS BATCH

En los procesos Batch, no hay un usuario esperando una respuesta del proceso.

Entonces se pueden utilizar algoritmos no apropiativos, no hay motivos para sacarlo de la CPU.

Metas propias de este tipo de algoritmos:

- **Rendimiento:** Maximizar el número de trabajos por hora
- **Tiempo de Retorno:** Minimizar los tiempos entre el comienzo y la finalización El Tiempo es espera se puede ver afectado
- **Uso de la CPU:** Mantener la CPU ocupada la mayor cantidad de tiempo posible

Ejemplos de algoritmos de este tipo son el FCFS (First Come First Served) o SJF (Shortest Job First). SJF requiere tiempo de CPU para estimar cuánto le falta a cada proceso, por lo tanto se desperdicia tiempo productivo.

PROCESOS INTERACTIVOS

Son procesos en los que hay alguien o algo esperando una respuesta. Puede ser un usuario humano, que hace un click y quiere la respuesta ahora, o por ejemplo, otro proceso que necesita de la respuestas para hacer un cálculo.

Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU

Metas propias de este tipo de algoritmos:

- **Tiempo de Respuesta:** Responder a peticiones con rapidez
- **Proporcionalidad:** Cumplir con expectativas de los usuarios.
  Si el usuario le pone STOP al reproductor de música, que la música deje de ser reproducida en un tiempo considerablemente corto.

Ejemplos de algoritmos de este tipo son el RR (Round Robin), Algoritmo de Prioridades, Colas Multinivel, o SRTF (Shortest Remaining Time First).

El de Prioridades o el SRTF podrían generar inanición.

El SRTF al igual que el SJF, requiere tiempo de CPU para estimar cuánto le faltaría a cada proceso.

POLÍTICA VS MECANISMO

Existen situaciones en las que es necesario que la planificación de uno o varios procesos se comporte de manera diferente .

El algoritmo de planificación debe estar parametrizado, de manera que los procesos/usuarios pueden indicar los parámetros para modificar la planificación.

El Kernel implementa el mecanismo.

El usuario/proceso/administrador utiliza los parámetros para determinar la Política.

Por ejemplo:

> En **Windows**, el mecanismo es usar el algoritmo de Colas Multinivel por prioridad, con 32 colas (o sea 32 prioridades), y cada una de esas colas se planifica con un Quantum. La **política** en la versión Windows Home es darle un Quantum de 3 a cada proceso, mientras que en la versión de Windows Server que no necesita interactividad del usuario, se le da un Quantum de 12, evitando cambios de contexto. En ambas versiones se utiliza el mismo mecanismo, pero cambia la política.

Clase 4

CREACIÓN DE PROCESOS

Un proceso es SIEMPRE creado por otro proceso (a excepción del proceso 0)

Un proceso padre tiene cero, uno o más procesos hijos.

Se forma un árbol de procesos

ACTIVIDADES EN LA CREACIÓN

1) Crear la PCB
2) Asignar PID (Process IDentification) único
3) Asignarle memoria para regiones (Stack, Text y Datos)
4) Crear estructuras de datos asociadas (Fork: copiar el contexto, regiones de datos, text y stack)

RELACIÓN ENTRE PROCESOS PADRE E HIJO

Con respecto a la Ejecución:

Hay dos modelos diferentes:

- El padre puede continuar ejecutándose concurrentemente con su hijo. Es decir, compiten por igual por la CPU.

- El padre puede esperar a que el proceso hijo (o los procesos hijos) terminen para continuar la ejecución. Le da prioridad al hijo en el uso de la CPU.
  (Este modelo no se utiliza, aunque hay System Calls que permiten hacerlo voluntariamente)

Con respecto a la Ejecución:

- En el caso de Unix, cuando un proceso crea a otro, se crea un nuevo espacio de direcciones para el hijo y es un duplicado de el del padre. Es decir, el contenido y la organización es una copia exacta del padre.

- En el caso de Windows cuando se crea el proceso hijo, se crea un nuevo espacio de direcciones vacío y se le carga adentro el programa.

EJEMPLOS CREACIÓN DE PROCESOS

En UNIX: (2 System Calls)

- **fork()**: crea nuevo proceso igual al llamador
- **execve()**: generalmente usada después del fork, carga un nuevo programa en el espacio de direcciones.

En Windows: (1 System Call)

- **CreateProcess()**: crea un nuevo proceso y carga el programa para ejecución.

FORK()

La System Call Fork crea un nuevo proceso, que es una copia exacta del proceso padre, copiando incluso los valores de los registros, incluyendo el PC (Program Counter). Es decir, el proceso hijo se comienza a ejecutar desde el mismo punto en el que estaba el proceso padre.

Se dice que es “el único proceso que retorna 2 valores”:

- Porque el proceso hijo retorna un 0.
- Y el proceso padre retorna un numero positivo o negativo. Si es negativo significa que hubo un error (y el proceso hijo no se creó), si es positivo significa que el proceso hijo se creó correctamente, y ese número retornado es el PID del hijo.

PROCESOS COOPERATIVOS E INDEPENDIENTES

Coooperativo:

Proceso que trabaja en conjunto con otros, comunicándose entre sí o compartiendo espacio de memoria (el So debe proveer las herramientas para ello).

Un qué proceso afecta o es afectado por la ejecución de otros procesos en el sistema.

Sirven para compartir información (por ejemplo, un archivo), para acelerar el cómputo (separar una tarea en sub-tareas que cooperan ejecutándose paralelamente) o para planificar tareas de manera tal que se puedan ejecutar en paralelo

Independiente:

El proceso no afecta ni puede ser afectado por la ejecución de otros procesos.

No comparte ningún tipo de dato.