ChatGPT

Procesos y Planificación Round-Robin: de Concepto a Simulación en Python

Programa, Proceso y Hilo

- **Programa**: es un conjunto de instrucciones estáticas almacenadas en disco. No consume recursos por sí solo hasta que se ejecuta. – Es sólo código y datos en reposo.
- **Proceso**: es la instancia activa de un programa en ejecución. Incluye la información dinámica asociada (espacio de direcciones, registros, pila, etc.). Un programa "se convierte" en proceso únicamente cuando se le asigna memoria y tiempo de CPU [1]. Cada proceso recibe un identificador único (PID) y su propia tabla de páginas virtuales.
- **Hilo (thread)**: es una hebra de ejecución dentro de un proceso. Varios hilos pueden coexistir en un proceso compartiendo el mismo espacio de direcciones y recursos abiertos, pero cada hilo mantiene su propio contador de programa y pila de ejecución [2]. En resumen, el proceso es la unidad de aislamiento de memoria, mientras que los hilos son unidades más ligeras que comparten ese espacio protegido (Tanenbaum & Bos; Silberschatz et al.).

Bloque de Control de Proceso (PCB)

El **PCB (Process Control Block)** es la estructura de datos donde el sistema operativo registra toda la información relevante de un proceso [3]. Típicamente incluye:
- **PID** (identificador único del proceso).
- **Estado del proceso** (Nuevo, Listo, Ejecutando, Bloqueado, Terminado).
- **Contador de Programa (PC)**: la dirección de la siguiente instrucción a ejecutar.
- **Registros de CPU**: valores de registros generales y de propósito especial, que se salvan/restauran en cambios de contexto.
- **Información de planificación**: prioridad, cola a la que pertenece, otros datos de scheduling.
- **Información de memoria**: mapeo de memoria (páginas o segmentos), límites y permisos del espacio de direcciones.
- **Información de contabilidad**: uso acumulado de CPU (tiempo de usuario/sistema), tiempo de llegada, etc.
- **Estado de E/S**: lista de dispositivos o archivos abiertos por el proceso [4].

Este diseño compacto permite al núcleo suspender un proceso (guardando su PCB) y reanudarlo más tarde cargando nuevamente sus campos.

Estados de un Proceso y Transiciones

Cada proceso alterna entre varios estados durante su vida [5]:
- **Nuevo**: se ha solicitado la creación del proceso y aún se asignan sus recursos iniciales.
- **Listo**: el proceso está preparado para ejecutarse (tiene memoria y demás recursos) pero espera que el CPU esté libre.
- **Ejecutando**: el proceso está actualmente en CPU realizando cómputo.
- **Bloqueado (o Esperando)**: el proceso espera un evento externo (por ejemplo, la finalización de una E/S) y no puede continuar, incluso si hay CPU disponible.

- **Terminado**: el proceso completó su ejecución. Su PCB permanece hasta que el sistema operativo realiza la limpieza final (liberando memoria, cerrando archivos, etc.).
- *(A veces se considera un estado intermedio Zombie: un proceso terminado cuyo PCB aún no se ha liberado totalmente.)*

Las transiciones típicas son: Nuevo → Listo (admisión por planificador a largo plazo), Listo → Ejecutando (el planificador a corto plazo asigna CPU), Ejecutando → Bloqueado (solicita E/S), Bloqueado → Listo (evento de E/S completado), Ejecutando → Terminado (fin natural) [5].

Fundamentos de Planificación de CPU y Objetivos

La **planificación de procesos** decide cómo repartir el CPU entre procesos listos. Los principales objetivos suelen ser [6] [7]:
- **Alta utilización del CPU**: mantener la CPU ocupada tanto como sea posible (ideal cercano al 100%).
- **Alto rendimiento (throughput)**: maximizar el número de procesos terminados por unidad de tiempo [6].
- **Bajo tiempo de respuesta**: para procesos interactivos, minimizar el tiempo hasta la primera respuesta. Se suele definir el **tiempo de respuesta** de un proceso como el tiempo total desde su llegada hasta su finalización (incluye tiempos de espera) [8].
- **Justicia (fairness)**: tratar procesos similares de manera equitativa y evitar la inanición. Un buen algoritmo de scheduling debe dar oportunidades razonables a todos los procesos sin posponer indefinidamente a ninguno [9].
- **Minimizar la sobrecarga**: reducir el costo de los cambios de contexto.

Estos objetivos a veces entran en conflicto, por lo que el diseño de algoritmos busca un balance. Por ejemplo, Silberschatz et al. destacan maximizar CPU y throughput mientras se minimizan turnaround y tiempos de espera [6].

Planificación Round-Robin (RR)

El algoritmo **Round-Robin** (RR) es un esquema sencillo de planificación preventiva. Sus características principales son [10] [11]:
- **Cola FIFO de listos**: los procesos listos se mantienen en una cola. El planificador a corto plazo toma siempre el primer proceso de la cola para ejecutarlo.
- **Quantum o intervalo de tiempo fijo**: al asignar CPU, cada proceso recibe un *quantum* (q) de tiempo. Si el proceso no termina en ese intervalo, es interrumpido al final del quantum.
- **Cambio de contexto y reencolado**: cuando expira el quantum, el proceso en ejecución sufre un *cambio de contexto*: sus registros se salvan en el PCB y se coloca al final de la cola de listos para esperar su siguiente turno. Esto se implementa típicamente con una interrupción periódica del temporizador [11].
- **Ventajas**: ofrece un buen tiempo de respuesta promedio, especialmente cuando hay muchos procesos interactivos. Cada proceso recibe porciones cortas de CPU de forma cíclica, de modo que ninguno monopoliza el procesador.
- **Parámetros críticos**: el tamaño del quantum q influye en el rendimiento. Quantum grande reduce la sobrecarga de cambios de contexto pero puede parecerse a FCFS (afavor de procesos largos), mientras que quantum demasiado pequeño aumenta la sobrecarga [12]. Silberschatz recomienda q menor que la duración media del 80% de los procesos [13].

En resumen, RR itera sobre los procesos listos, asignándoles turnos de CPU de duración limitada y rotándolos en la cola, garantizando así cierta equidad.

Memoria Virtual por Proceso

Cada proceso se ejecuta como si tuviera su propio espacio de direcciones privado. La memoria virtual (MMU) gestiona estos espacios virtuales separados [14]. En la práctica:
- **Espacio de direcciones aislado**: cada proceso tiene una tabla de páginas propia [15]. De este modo, direcciones iguales en procesos distintos apuntan a marcos de memoria diferentes. Esto evita que un proceso lea o modifique la memoria de otro.
- **Carga dinámica**: un programa cargado en memoria constituye un proceso real. Como señala la bibliografía, "los programas sólo se vuelven procesos cuando se les asigna memoria y tiempo de cómputo" [1]. Mientras el proceso existe, sus páginas se mapean a memoria física (o incluso a disco) sin solaparse con las de otros procesos.
- **Protección y seguridad**: esta separación evita fallos de seguridad (por ejemplo, sobreescritura accidental de otra memoria) y permite que cada proceso crea que tiene un sistema aislado. La tabla de páginas del proceso se activa al programar el proceso (via registros PTBR/PTLR), asegurando que cada cambio de contexto actualice el espacio de direcciones en uso [15].

Así, la memoria virtual es clave para que varios procesos puedan coexistir "como si cada uno tuviera su propia máquina", brindando aislamiento y protección mutua.

Uso Conjunto de PCB, Planificador y Memoria

Para ejecutar múltiples procesos de forma segura, el sistema operativo combina **PCB**, **planificación** y **MMU**. La multiprogramación hace que un solo CPU atienda varios procesos alternadamente [14]: cada cierto intervalo el temporizador genera una interrupción y el núcleo realiza un cambio de contexto. En ese momento guarda los registros del proceso actual en su PCB y carga los registros del siguiente. La figura ilustra este esquema: cada proceso tiene sus secciones de código, datos, pila y registros salvados, y el CPU se alterna entre ellos [14].

En detalle, al cambiar de proceso el SO: (1) actualiza el **estado** de procesos en los PCB correspondientes (por ejemplo, de Ejecutando a Listo/Bloqueado) [5], (2) salva/restaura los registros CPU desde el PCB, y (3) actualiza los punteros de memoria virtual a la tabla de páginas del nuevo proceso [15]. El planificador a corto plazo escoge el siguiente proceso listo (siguiendo la política RR en este caso) y lo pone en ejecución. Gracias a los PCB y la memoria virtual, cada proceso cree tener el CPU "exclusivo" y su propio espacio de direcciones, a pesar de la concurrencia real [2] [15]. En este esquema, el kernel supervisa y aísla a cada proceso, garantizando que se ejecuten múltiples programas simultáneamente sin interferir.

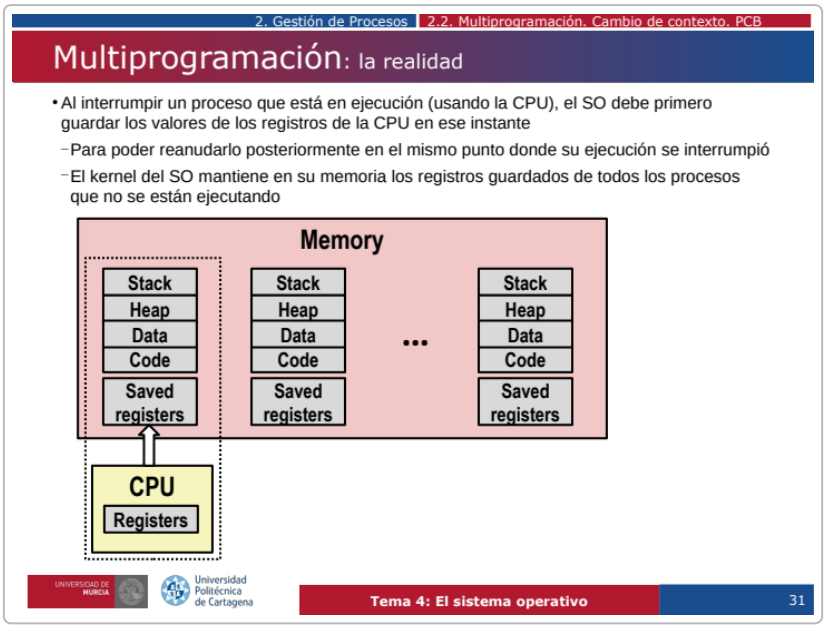

Figura: En un sistema multiprogramado de un solo CPU, la memoria virtual asigna a cada proceso su espacio de direcciones (código, datos, pila, etc.) y el sistema guarda/restaura los registros de cada proceso al hacer cambios de contexto [14] [15] .

Simulación en Python

En Python se puede modelar esta teoría mediante clases y estructuras de datos sencillas. Por ejemplo:
- Definir una clase `Proceso` con atributos como `pid`, `estado` (Nuevo, Listo, Ejecutando, etc.), contador de programa, registros de CPU simulados (por ejemplo, enteros), y un objeto de memoria virtual (como una lista de páginas o un diccionario que represente la tabla de páginas).
- Mantener colas de procesos: una lista para los procesos en estado Listo, otra para los bloqueados, etc.
- Implementar un bucle de tiempo “tick a tick”: en cada ciclo el scheduler extrae de la cola el primer proceso listo y lo ejecuta durante un quantum. Al finalizar el quantum (o si el proceso pide I/O), se actualiza su estado y se realiza el cambio de contexto —se “salva” su PC y registros en el objeto `Proceso` y se coloca de nuevo en la cola.
- Gestionar eventos de E/S: cuando un proceso simulado solicita E/S, se mueve al estado Bloqueado; tras un tiempo (simulado o manual), se devuelve a Listo.

En este simulador, cada objeto `Proceso` representa el PCB+memoria de un proceso real. El planificador (por ejemplo, una función RR) decide el orden de ejecución, y un objeto de memoria asociada simula la paginación. De este modo, la asignación de CPU y la separación de espacios de direcciones quedan reflejadas por la lógica de la clase `Proceso` y las estructuras de las colas: cada proceso avanza en su propio espacio simulado y no interfiere con los demás, tal como en un sistema operativo real.

Referencias: Principios de sistemas operativos (Silberschatz, Galvin & Gagne; Tanenbaum & Bos) y apuntes modernos sobre gestión de procesos [5] [3] [10] [7] .

---

[1] [2] [3] [4] [5] [7] [8] [9] [10] [11] [12] [13] [15] Calaméo - Sistemas Operativos Cc By Sa 3 0
https://www.calameo.com/books/006853334dcc96e923ea3

6 2.01

https://www.cs.hunter.cuny.edu/~sweiss/course_materials/csci340/slides/chapter05.pdf

14 Diapositiva 1

https://ocw.bib.upct.es/mod/resource/view.php?id=14996&redirect=1