Sistemas Operativos I
Trabajo Práctico Obligatorio 2
2025

Objetivos

- Analizar diferentes algoritmos de planificación de procesos.
- Examinar los estados de un proceso en un sistema operativo.

Referencias

[1] Tanenbaum, Bos – Modern Operating Systems - Prentice Hall; 4 edition (March 10, 2014) - ISBN-10: 013359162X
[2] Douglas Comer - Operating System Design - The Xinu Approach. CRC Press, 2015. ISBN : 9781498712439
[3] Silberschatz, Galvin, Gagne - Operating Systems Concepts - John Wiley & Sons; 10 edition (2018) – ISBN 978-1-119-32091-3

Software y Hardware

La versión de Xinu que utilizamos es para arquitectura PC (x86). Ejecutamos el sistema operativo Xinu en una máquina virtual llamada QEMU, que emula una PC básica.
El trabajo puede realizarse sobre las máquinas de los laboratorios (RECOMENDADO).
Quienes tengan Linux en sus casas, podrían intentar instalar todo lo necesario y llevarlo a cabo ahí también. Una tercera posibilidad es el acceso remoto RDP comentado en la web de la materia.

Ejercicio 1.

Estudie en XINU para qué sirven las siguientes system calls (puede revisar el código fuente de XINU, o el libro XINU):

suspend(pid); // Suspende proceso que está en ejecución o listo, y los pone en estado suspendido.
parametro (pid) // es un entero32 o pid32 (mejor!!(mas especifico)), devuelve: la prioridad que es de tipo pri 16
resume(pid); // deja de estar en estado suspendido el proceso y pasa a estar en estado LISTO
parametro (pid) // es un entero32 o pid32 (mejor!!(mas especifico))
getprio(pid); // devuelve: la prioridad que es de tipo pri 16
parametro (pid) // entero32 o pid32 (mejor!!(mas especifico)), return: prioridad que es de tipo pri 16chprio(pid, newprio);
parametro (pid) // entero32 o pid32 (mejor!!(mas especifico))
parametro (newprio) //pri 16 es la nueva prioridad, devuelve: la prioridad antigua, antes de que se actualice
getpid(); // devuelve: id del proceso, que es de tipo pid 32
sleepms(ms); // tiempo en milisegundos que espera antes de ejecutar al proceso

suspend(pid); **// Suspende el proceso con id == pid, colocándolo en hibernación**
resume(pid); **// Quita suspensión a un proceso con id == pid, colocándolo en cola de listos**
getprio(pid); **// Retorna la prioridad de ejecución/planificación de un proceso con id == pid**
chprio(pid, newprio); **// Establece la prioridad de ejecución/planificación "newprio" de un proceso con id = pid**
getpid(); **// Retorna el id del proceso en ejecución actualmente**
sleepms(ms); **// Demora al proceso en ejecución durante n milisegundos, ubicando dicho proceso en la lista delta de procesos dormidos (delta list of sleeping processes) (Global variable sleepq contains the queue ID of the delta list for sleeping processes)**

a. **¿Se puede hacer un suspend(pid) del proceso pid estando el proceso pid en estado "sleeping"?**

No es posible, para que suspend(pid) se ejecute, el proceso con pid debe encontrarse en uno de los dos estados válidos (listo o en ejecución). (comment en la definición del system call: /* Only suspend a process that is current or ready */) **(Page 107. Operating System Design The Xinu Approach)** Because suspension is only valid for a process that is ready or current, the code verifies that the process is in one of the two valid states. If an error is detected, suspend restores interrupts and returns SYSERR to the caller.

b. **Repase el TP anterior: ¿cuánto vale el quantum en XINU?. ¿Qué problema o comportamiento puede haber si usted cambia el quantum a 100ms?**

El quantum en XINU vale 2ms.
Podría ocurrir que los procesos se terminen en el primer momento que utilizan la cpu, haciendo que otros procesos esperen mucho más y no logren realizar su ejecución.

c. **Investigue cómo se llaman y cómo funcionan los algoritmos de planificación de procesos del sistema operativo Windows y Linux. Redacte una respuesta-aprendizaje.** Esto implica: **no buscar en google/chatgpt y sólo copiar y pegar. Intente dar una explicación general**.

Ayuda: **Abraham Silberschatz, Operating System Concepts 10th-2018) (planificación-cpu.pdf)**

FCFS (First-Come, First-Served): El algoritmo más simple, donde el proceso que solicita la CPU primero es el primero en ser servido. Se implementa fácilmente con una cola FIFO. Sin embargo, un proceso largo puede bloquear a procesos más cortos, lo que lleva a un efecto convoy y a una menor utilización de la CPU y los dispositivos. FCFS es un algoritmo no apropiativo.

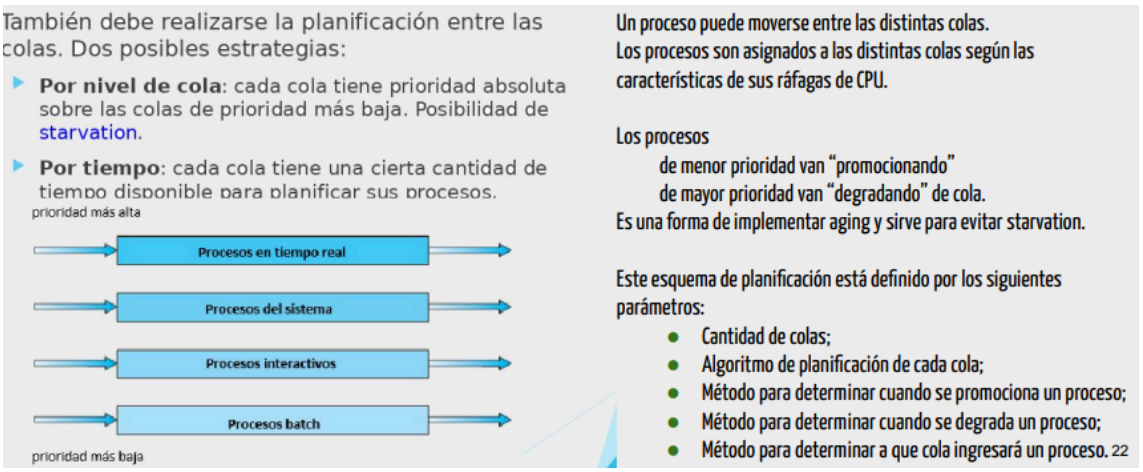
SJF (Shortest Job First): (ES TEÓRICO, porque no hay una forma de saber de cuanto es la rafaga). Este algoritmo asigna la CPU al proceso con la siguiente ráfaga de CPU más corta. SJF puede ser no apropiativo o apropiativo (en cuyo caso se denomina Trabajo Restante Más Corto Primero - SRTF). SJF es óptimo para minimizar el tiempo de espera promedio. Sin embargo, no se puede implementar a nivel de planificación de la CPU ya que no hay forma de conocer la duración de la siguiente ráfaga de CPU. Se pueden utilizar predicciones basadas en el historial de las ráfagas anteriores para aproximar SJF.
SRTF (Shortest Remaining Time First): Una versión apropiativa de SJF. Si un nuevo proceso llega con una ráfaga de CPU restante más corta que la del proceso en ejecución, el CPU se le asigna al nuevo proceso. También requiere conocer la duración de las ráfagas futuras.
SJF y SRTF son algoritmos teóricos pues no se puede saber exactamente cuál es la longitud de la próxima ráfaga de CPU. Se puede implementar realizando una estimación en base a las longitudes de las ráfagas de CPU previas. Se elige el proceso con la siguiente ráfaga de CPU que se ha estimado como más corta.

RR (Round Robin): Este algoritmo está diseñado para sistemas de tiempo compartido. A cada proceso se le asigna una pequeña unidad de tiempo de CPU llamada **quantum**. Si un proceso no termina su ráfaga de CPU dentro del quantum asignado, se interrumpe y se coloca al final de la cola de listos. RR es un algoritmo apropiativo que proporciona una respuesta más rápida a los usuarios. La elección del tamaño del quantum es crucial; un quantum demasiado **pequeño** puede aumentar la **sobrecarga** debido a los **cambios de contexto**, mientras que un quantum demasiado **grande** tiende a comportarse como **FCFS**.

Por Prioridades: A cada proceso se le asocia una prioridad, y la CPU se asigna al proceso con la prioridad más alta. Los procesos con igual prioridad se planifican mediante **FCFS**. La planificación por prioridades puede ser apropiativa o no apropiativa. Un problema importante es la inanición (starvation), donde los procesos de baja prioridad pueden esperar indefinidamente. Una solución común es el envejecimiento (AGING), que aumenta gradualmente la prioridad de los procesos que han esperado durante mucho tiempo. SJF puede considerarse un algoritmo de prioridad donde la prioridad es inversamente proporcional a la duración de la siguiente ráfaga de CPU predicha (es decir, más ráfagas, menos prioridad).

Combinado (Round Robin y Prioridades): Se pueden combinar diferentes algoritmos. Por ejemplo, en Xinu, la planificación por prioridades se combina con Round Robin.

Colas Multinivel: Este algoritmo utiliza múltiples colas de listos, cada una con su propio algoritmo de planificación. Los procesos se asignan a una cola en función de sus propiedades (por ejemplo, prioridad, tipo de proceso). Puede haber planificación entre las colas, a menudo con prioridad fija apropiativa, donde una cola de mayor prioridad tiene precedencia sobre las colas de menor prioridad. También es posible asignar una porción del tiempo de CPU a cada cola y luego planificar los procesos dentro de esa cola utilizando un algoritmo específico (por ejemplo, RR para colas de primer plano y FCFS para colas de fondo). La planificación con retroalimentación multinivel permite que los procesos se muevan entre las colas en función de su comportamiento (por ejemplo, un proceso con ráfagas de CPU cortas puede moverse a una cola de mayor prioridad).



La planificación de la CPU está estrechamente relacionada con el concepto de proceso. Un proceso es un programa en ejecución. Para gestionar los procesos, el sistema operativo utiliza una estructura de datos llamada Bloque de Control de Proceso (PCB), que contiene información sobre el estado del proceso, el contador de programa, los registros de la CPU, información de gestión de memoria y de E/S, entre otros. Los procesos atraviesan diferentes estados durante su ciclo de vida: nuevo, listo, en ejecución, esperando y terminado. El planificador es el encargado de seleccionar un proceso del estado listo para asignarle la CPU y llevarlo al estado en ejecución.
Cuando el sistema operativo decide cambiar la CPU de un proceso a otro, se produce un cambio de contexto. Esto implica guardar el estado del proceso que se interrumpe y restaurar el estado del proceso que se va a ejecutar. El cambio de contexto genera una sobrecarga en el sistema, ya que la CPU dedica tiempo a la administración en lugar de a la ejecución de las aplicaciones.
La planificación de la CPU es fundamental para la multiprogramación, que busca maximizar la utilización de la CPU al tener múltiples procesos en memoria, de modo que si un proceso se bloquea esperando una operación, otro pueda utilizar la CPU. También es esencial para los sistemas de tiempo compartido, que permiten que múltiples usuarios interactúen con sus programas simultáneamente al asignarles pequeños intervalos de tiempo de CPU.
Diferentes sistemas operativos implementan sus propias estrategias de planificación. Por ejemplo, Linux utiliza el algoritmo Completely Fair Scheduling (CFS), que considera la prioridad de los procesos y busca una distribución justa del tiempo de CPU. Windows utiliza un planificador basado en prioridades con diferentes niveles de prioridad(Gestionado por dispatcher ?)). Xinu utiliza una planificación por prioridades combinada con Round Robin, donde los procesos con la misma prioridad se ejecutan en orden de llegada dentro de su nivel de prioridad, pero pueden ser interrumpidos por procesos de mayor prioridad.

Implícitamente, se deduce que la elección del algoritmo de planificación es un compromiso entre diferentes objetivos de rendimiento. No existe un algoritmo único que sea óptimo para todas las situaciones. La decisión depende de las características de la carga de trabajo, los requisitos del sistema (por ejemplo, si es un sistema interactivo, en tiempo real o por lotes) y los objetivos de diseño (por ejemplo, priorizar la capacidad de respuesta, el rendimiento o la equidad). Los diseñadores de sistemas operativos deben analizar cuidadosamente estos factores para seleccionar o diseñar un algoritmo de planificación que satisfaga las necesidades específicas del sistema.

Ejercicio 2. Estado sleeping y prioridades

a. **Desarrollar un programa en XINU que genere dos procesos para mostrar números primos. Un proceso debe mostrar los primos del 1 al 5000, y el segundo proceso del 5000 al 10000.**

Listo, en pc del lab

b. **Modifique el programa anterior para darle al primer proceso de cálculo de primos mayor prioridad que al segundo proceso. Verifique su ejecución e indique cómo se comporta el planificador de CPU de XINU al ejecutar estos dos procesos de este inciso.**

El planificador de XINU permite que se ejecute primero el proceso con mayor prioridad (hasta que finalice su ejecución) para luego dar permiso al de menor prioridad.

c. **XINU provee el system call sleepms() para que un proceso voluntariamente "delegue" la CPU durante un tiempo en milisegundos. Al delegar la CPU, XINU coloca al proceso en estado "DURMIENDO" (estado "bloqueado" en la literatura teórica). Cuando el tiempo de dormir finaliza, XINU coloca al proceso en estado de "LISTO".**
Utilice este system call para modificar el comportamiento del programa del inciso b., de manera tal que el proceso 1, luego de 100 números primos calculados, duerma durante 10ms. Verifique su ejecución e indique cómo se comporta el planificador de CPU de XINU al ejecutar los dos procesos (siendo el proceso 1 de mayor prioridad que 2).

Espera 10ms luego de 100 números primos (hasta el 541) y comienza el segundo proceso (desde 5003).

Ejercicio 3. Ejemplos de Planificación de CPU

Sea un sistema con la siguiente carga de procesos. El planificador de CPU del sistema es por prioridades, round-robin, apropiativo.

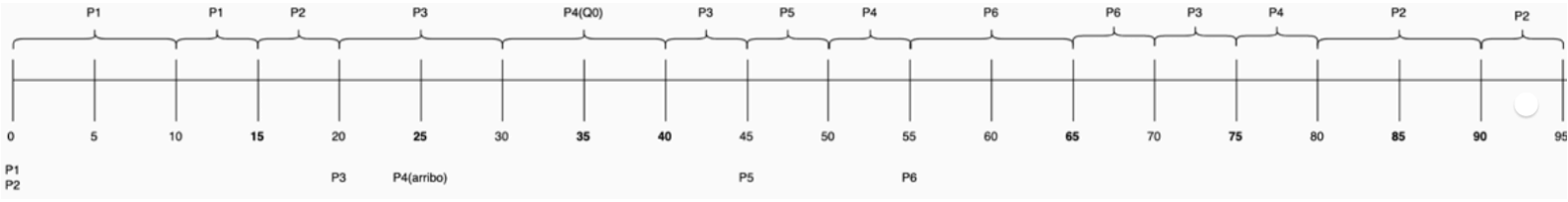
Proceso	Prioridad	Ráfaga	Arribo a la cola de listos
P1	8	15	0
P2	3	20	0
P3	4	20	20
P4	4	20	25
P5	5	5	45
P6	5	15	55

A más alto número de prioridad mayor prioridad. El quantum del sistema es de 10 unidades. A misma prioridad el planificador es round-robin.

a. Mostrar el orden de ejecución y las ráfagas de CPU de los procesos.

Q=10ms

P1 | P1 | P2 | P3 | P4 | P3 | P5 | P4 | P6 | P6 | P4 | P3 | P2 | P2 |
0 10 15 20 30 40 45 50 55 65 70 75 80 90 95



b. ¿Cuánto es el tiempo de turnaround para cada proceso? ¿Y el valor medio contando todos los procesos?

Tiempo desde que comienza ejecución hasta que termina (contando tiempo esperando). Promedio de turnarounds

c. ¿Cuál es el tiempo de espera para cada proceso? ¿Y el valor medio contando todos los procesos?

Suma del tiempo esperando. Promedio de los tiempos de espera

Scheduling Criteria (Albraham Silbeschatz)

Different CPU-scheduling algorithms have different properties, and the choice of a particular algorithm may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms.

Many criteria have been suggested for comparing CPU-scheduling algorithms. Which characteristics are used for comparison can make a substantial difference in which algorithm is judged to be best. The criteria include the following:

CPU utilization. We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system). (CPU utilization can be obtained by using the top command on Linux, macOS, and UNIX systems.)

Throughput. If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called throughput. For long processes, this rate may be one process over several seconds; for short transactions, it may be tens of processes per second.

Turnaround time. From the point of view of a particular process, the important criterion is **how long it takes to execute that process**. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting in the ready queue, executing on the CPU, and doing I/O.

Waiting time. The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O. It affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the **sum of the periods spent waiting in the ready queue**.

Response time. In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the **time it takes to start responding**, not the time it takes to output the response.

Ejercicio 4.

Sea un sistema con la siguiente carga de procesos.

Proceso	Ráfaga	Prioridad
P1	5	4
P2	3	1
P3	1	2
P4	7	2
P5	4	3

Los procesos arriban al sistema en el momento 0, en este orden: P1 , P2 , P3 , P4 , P5.

a. Mostrar el orden de ejecución y las ráfagas de CPU de los procesos si el planificador es FCFS, SJF, con prioridades no-apropiativo, round-robin (quantum = 2).
b. ¿Cuánto es el tiempo de turnaround para cada proceso? ¿Y el valor medio contando todos los procesos? (para cada algoritmo).
c. ¿Cuál es el tiempo de espera para cada proceso? ¿Y el valor medio contando todos los procesos? (para cada algoritmo).

d. ¿Cuál de los algoritmos produce el mínimo promedio de tiempo de espera?

Ejercicio 5.

a. Averigüe cómo puede ver el estado de cada proceso en ejecución de un sistema UNIX (Linux/MAC OS), etc.

Comandos

ps: nombre de cada proceso, su PID, el tiempo de CPU consumido y la terminal o usuario asociados.

htop, top: Interfaz interactiva para monitorear procesos en tiempo real.

Estados comunes:

R: Running — El proceso se está ejecutando o está listo para ejecutarse.

S: Sleeping — El proceso está inactivo, esperando algo (como E/S).

Z: Zombie — El proceso terminó, pero aún tiene una entrada en la tabla de procesos.

D: Uninterruptible sleep — El proceso está esperando una operación de E/S, no puede ser interrumpido.

T: Stopped — El proceso fue detenido (por una señal o el usuario).

Letras adicionales (modificadores del estado)

s: El proceso es un líder de sesión.

+: Proceso en primer plano (foreground, ligado a una terminal).

b. **¿Qué estados posibles existen en UNIX? ¿Qué significa el estado zombie en UNIX según Tanenbaum?**

El estado de los procesos en Linux (https://juncotic.com/procesos-en-linux-estados-y-prioridades/)

En Linux los procesos pueden estar en diferentes estados, simbolizados mediante una letra en la salida del comando htop/top. procesos en linux estados

Los estados principales en los que un proceso puede estar, y que en la salida de un htop están etiquetados en la columna «S«(status), son los siguientes:

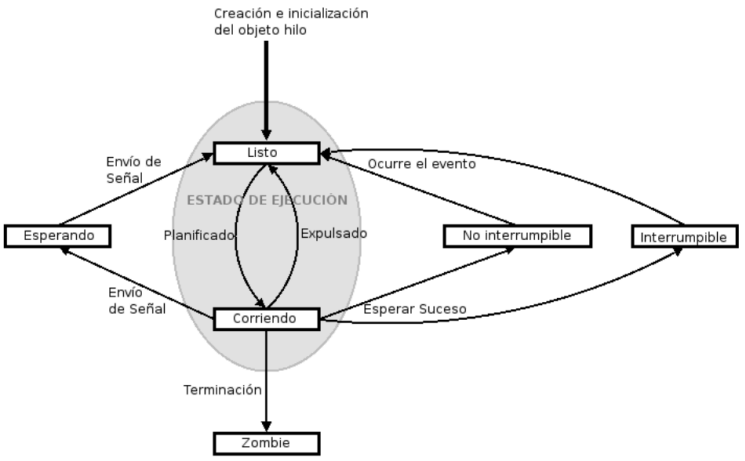


Figura 7.2: Estados de un procesos en Linux

D Uninterruptible sleep – Espera ininterrumpible, generalmente el proceso se encuentra esperando una operación de entrada/salida con algún dispositivo.

R Running – Corriendo, el proceso se encuentra corriendo en el procesador.

S Interruptible sleep, espera interrumpible, el proceso se encuentra esperando a que se cumpla algún evento, por ejemplo, que el planificador de procesos del kernel lo planifique para su ejecución.

T Stopped, detenido, un proceso que ha sido detenido mediante el envío de alguna señal generalmente.

Z Defunct ("zombie") process, proceso terminado, pero cuyo padre aún sigue «vivo» y no ha capturado el estado de terminación del proceso hijo, y por consiguiente, no lo ha eliminado de la tabla de procesos del sistema. En definitiva, un proceso zombie es un proceso que «murió», pero «sigue estando» en la tabla de procesos del sistema. E

(Abraham-Silberschatz-Operating-System-Concepts-10th-2018)

Zombie process: A process that has terminated, but whose parent has not yet called wait(), is known as a zombie process. All processes transition to this state when they terminate, but generally they exist as zombies only briefly. Once the parent calls wait(), the process identifier of the zombie process and its entry in the process table are released

Para XINU

Un proceso pasa por diferentes estados durante su ciclo de vida:

Listo (Ready): El proceso está esperando a ser asignado a un núcleo de CPU.

Ejecutando (Running): Las instrucciones del proceso se están ejecutando en la CPU.

Bloqueado o Esperando (Waiting): El proceso está esperando que ocurra algún evento (por ejemplo, la finalización de una operación de E/S o la recepción de una señal).

Nuevo (New): El proceso está siendo creado.

Terminado (Terminated): El proceso ha finalizado su ejecución.

Suspendido (Suspended): Un proceso en cualquiera de los estados anteriores puede ser suspendido (típicamente para liberar memoria) y luego reanudado.

Ejemplo la imagen, XINU

Listo Suspendido y Bloqueado Suspendido: Un estado suspendido en el contexto del sistema operativo Xinu. Un proceso puede ser movido al estado suspendido y luego reanudado. "listo suspendido" y "bloqueado suspendido", esto probablemente se refiere a procesos que se encuentran en los estados listo (esperando ser asignados a un procesador) o bloqueado (esperando algún evento, como finalización de E/S o recepción de una señal) pero que además han sido suspendidos. La suspensión es un mecanismo para pausar temporalmente un proceso y liberar recursos, como memoria, para otros procesos. Android también puede terminar procesos para reclamar recursos limitados, siguiendo una jerarquía de importancia. Si bien no se menciona explícitamente "suspendido" en la descripción general de estados de proceso, el concepto de suspender procesos para gestionar recursos existe.