

TEORÍA MÓDULO 1 - CSO

Sistema Operativo: *software* que actúa como intermediario entre el usuario de una computadora y su hardware. Como es software necesita procesador y memoria para ejecutarse.

Es el encargado de

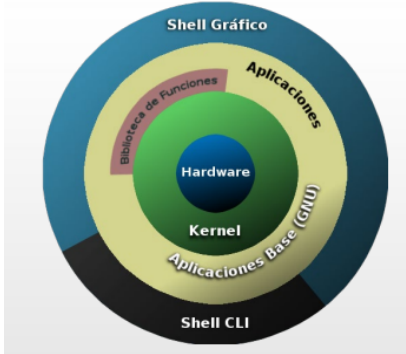
- Gestionar el HW
- Controlar la ejecución de los procesos
- Interfaz entre aplicaciones y HW
- Actúa como intermediario entre un usuario de una computadora y el HW de la misma

PERSPECTIVAS

DESDE EL USUARIO <i>(de arriba hacia abajo)</i>	DESDE EL SISTEMA <i>(de abajo hacia arriba)</i>
<ul style="list-style-type: none">• Abstracción con respecto a la arquitectura• El SO oculta el HW y presenta los programas abstracciones más simples de manejar• Los programas de aplicaciones son los clientes del SO• Comparación: uso de escritorio y uso de comandos• Comodidad	<ul style="list-style-type: none">• Administra los recursos de HW de uno o más procesos• Provee un conjunto de servicios a los usuarios del sistema• Maneja la memoria secundaria y dispositivos de I/O• Ejecución simultánea de procesos• Multiplexación en tiempo (CPU) y en espacio (memoria)

OBJETIVOS

- **Comodidad:** fácil uso del HW (PC, router, servidor, etc)
- **Eficiencia:** uso más eficiente de los recursos del sistema
- **Evolución:** introducción de nuevas funciones al sistema sin interferir con funciones anteriores

<u>COMPONENTES</u>	KERNEL
	<p>Es una porción de código la cual se encuentra en memoria principal y se encarga de la administración de los recursos</p> <p>Servicios que implementa</p> <ul style="list-style-type: none"> ➤ Manejo de memoria y CPU ➤ Administración de procesos ➤ Comunicación y Concurrencia ➤ Gestión de la E/S
SHELL <ul style="list-style-type: none"> - GUI /CUI o CLI 	HERRAMIENTAS <ul style="list-style-type: none"> - Editores, compiladores, librerías, etc
<u>SERVICIOS</u>	
<p>1. Administración y planificación del procesador</p> <ul style="list-style-type: none"> ○ Multiplexación de la carga de trabajo ○ Imparcialidad en la ejecución ○ Manejo de prioridades ○ Que no haya bloqueos 	<p>2. Administración de memoria</p> <ul style="list-style-type: none"> ○ Eficiencia ○ Memoria física vs virtual, jerarquías de memoria ○ Protección de programas que compiten o se ejecutan concurrentemente
<p>3. Administración del almacenamiento (File System)</p> <ul style="list-style-type: none"> ○ Acceso a medios de almacenamiento externos 	<p>4. Administración de dispositivos</p> <ul style="list-style-type: none"> ○ Ocultamiento de dependencias de HW ○ Administración de accesos simultáneos
<p>5. Detección de errores y respuestas</p> <ul style="list-style-type: none"> ○ Errores de HW internos y externos (de memoria/CPU o de dispositivos) ○ Errores de SW (aritméticos, acceso no permitido en direc de memorias) ○ Incapacidad del SO para conceder una solicitud de una aplicación 	<p>6. Contabilidad</p> <ul style="list-style-type: none"> ○ Recoger estadísticas del uso ○ Monitorear parámetros del rendimiento ○ Anticipar necesidades de mejoras futuras ○ Dar elementos si es necesario facturar tiempo de procesamiento
<p>7. Interacción del Usuario (SHELL)</p>	<p>8. Complejidad</p> <ul style="list-style-type: none"> ○ SO software extenso y complejo, desarrollado por partes

FUNCIONES PRINCIPALES	PROBLEMAS QUE DEBE EVITAR
<ul style="list-style-type: none"> • Brindar abstracciones de alto nivel a los procesos de usuario • Administrar eficientemente el uso de la CPU y memoria • Brindar asistencia para E/S por parte de procesos 	<ul style="list-style-type: none"> • Que un proceso: <ul style="list-style-type: none"> ◦ Se apropie de la CPU ◦ Intente ejecutar instrucciones de E/S ◦ Intente acceder a una pos de memoria fuera de su espacio declarado
APOYO DEL HW	PARA EVITAR PROBLEMAS
<ol style="list-style-type: none"> 1. <i>Modos de ejecución</i> 2. <i>Interrupción de Clock</i> 3. <i>Protección de la Memoria</i> 	<ul style="list-style-type: none"> • Gestionar uso de la CPU • Detectar intentos de ejecución de instrucciones de E/S ilegales • Detectar accesos ilegales a memoria • Proteger el vector de interrupciones

1. Modos de ejecución

Define limitaciones en el conjunto de instrucciones que se puede ejecutar en cada modo

El bit en la CPU indica el modo actual

- **Modo Supervisor o Kernel (MS)** → se ejecutan instrucciones del Kernel del SO
- **Modo Usuario (MU)** → se ejecutan las demás instrucciones del SO y los programas de usuario

Al arrancar el sistema, arranca con el bit en MS. Por lo tanto se debe cambiar a MU mediante una instrucción especial.

La única forma de pasar a MK es cuando hay una interrupción o trap. El bit se pone automáticamente en MK no existe un proceso de usuario que hace el cambio explícitamente.

Si el proceso de usuario intenta por sí mismo ejecutar instrucciones que pueden causar problemas (las privilegiadas), el HW lo detecta como una operación ilegal y produce un trap al SO

☑ **Modo kernel:**

- ☑ **Gestión de procesos:** Creación y terminación , planificación, intercambio, sincronización y soporte para la comunicación entre procesos
- ☑ **Gestión de memoria:** Reserva de espacio de direcciones para los procesos, Swapping, Gestión y páginas de segmentos
- ☑ **Gestión E/S:** Gestión de *buffers*, reserva de canales de E/S y de dispositivos de los procesos
- ☑ **Funciones de soporte:** Gestión de interrupciones, auditoría, monitoreo

☑ **Modo usuario:**

- ✓ Debug de procesos, definición de protocolos de comunicación gestión de aplicaciones (compilador, editor, aplicaciones de usuario)
- ✓ En este modo se llevan a cabo todas las tareas que no requieran accesos privilegiados
- ✓ En este modo no se puede interactuar con el hardware
- ✓ El proceso trabaja en su propio espacio de direcciones

2. Interrupción por Clock

Se debe evitar que un proceso se apropie de la CPU mediante la interrupción por Clock

- Se implementa a través de un clock y un contador
- El kernel le da valor al contador que se decrementa con cada tick de reloj y al llegar a cero puede expulsar al proceso para ejecutar otro
- Instrucciones que modifican al reloj son privilegiadas
- Se le asigna al contador el valor que se quiere que se ejecute un proceso
- Se la usa también para el cálculo de la hora actual, basándose en cantidad de interrupciones ocurridas cada tanto tiempo y desde una fecha y hora determinada

3. Protección de la memoria

Se debe definir límites de memoria a los que se pueda acceder cada proceso (registros base y límite)

- El kernel carga registros por medio de instrucciones privilegiadas. Esta acción solo puede realizarse en MK.
- La memoria principal aloja al SO y a los procesos de usuario
 - Kernel debe proteger
 - Para que los procesos de usuario no puedan acceder donde no les corresponde
 - El espacio de direcciones de un proceso del acceso de otros procesos
- Las instrucciones de E/S se definen como privilegiadas, es decir, deben ejecutarse en MK
- Procesos de usuarios realizan E/S a través de llamadas al SO

- **System calls** → forma en que los programas de usuario acceden a los servicios del SO
- Parámetros asociados a las llamadas pueden pasarse por registros, bloques de memoria o pila
 - `count = read(file, buffer, nbytes);`
- Se ejecutan en MK
- Categorías de system calls
 - Control de procesos
 - Manejo de archivos y dispositivos
 - Mantenimiento de información del sistema
 - Comunicaciones

Process management	
Call	Description
<code>pid = fork()</code>	Create a child process identical to the parent
<code>pid = waitpid(pid, &statloc, options)</code>	Wait for a child to terminate
<code>s = execve(name, argv, environp)</code>	Replace a process' core image
<code>exit(status)</code>	Terminate process execution and return status

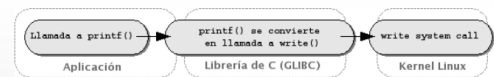
File management	
Call	Description
<code>fd = open(file, how, ...)</code>	Open a file for reading, writing or both
<code>s = close(fd)</code>	Close an open file
<code>n = read(fd, buffer, nbytes)</code>	Read data from a file into a buffer
<code>n = write(fd, buffer, nbytes)</code>	Write data from a buffer into a file
<code>position = lseek(fd, offset, whence)</code>	Move the file pointer
<code>s = stat(name, &buf)</code>	Get a file's status information

Directory and file system management	
Call	Description
<code>s = mkdir(name, mode)</code>	Create a new directory
<code>s = rmdir(name)</code>	Remove an empty directory
<code>s = link(name1, name2)</code>	Create a new entry, name2, pointing to name1
<code>s = unlink(name)</code>	Remove a directory entry
<code>s = mount(special, name, flag)</code>	Mount a file system
<code>s = umount(special)</code>	Unmount a file system

Miscellaneous	
Call	Description
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chmod(name, mode)</code>	Change a file's protection bits
<code>s = kill(pid, signal)</code>	Send a signal to a process
<code>seconds = time(&seconds)</code>	Get the elapsed time since Jan. 1, 1970

Directory and file system management	
Call	Description
<code>s = mkdir(name, mode)</code>	Create a new directory
<code>s = rmdir(name)</code>	Remove an empty directory
<code>s = link(name1, name2)</code>	Create a new entry, name2, pointing to name1
<code>s = unlink(name)</code>	Remove a directory entry
<code>s = mount(special, name, flag)</code>	Mount a file system
<code>s = umount(special)</code>	Unmount a file system

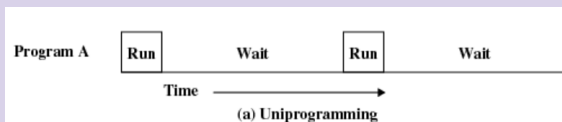
Miscellaneous	
Call	Description
<code>s = chdir(dirname)</code>	Change the working directory
<code>s = chmod(name, mode)</code>	Change a file's protection bits
<code>s = kill(pid, signal)</code>	Send a signal to a process
<code>seconds = time(&seconds)</code>	Get the elapsed time since Jan. 1, 1970



- ☑ Para activar iniciar la system call se indica:
 - el número de syscall que se quiere ejecutar
 - los parámetros de esa syscall
- ☑ Luego se emite un aviso al SO (trap/excepción) para pasar a modo Kernel y gestionar la system call
- ☑ Se evalúa la system call deseada y se ejecuta

ANEXO ARQUITECTURA DE COMPUTADORAS → no lo voy a resumir 🧑

EVOLUCIÓN HISTÓRICA	Los SO evolucionaron con el objetivo de soportar nuevos tipos de HW, brindar nuevos servicios y ofrecer mejoras y alternativas a problemas existentes en la planificación y el manejo de memoria, entre otras
Procesamiento en serie	Sistemas por Lotes Sencillos (<i>batch</i>)
<ul style="list-style-type: none"> No existía un SO Las máquinas utilizadas desde una consola que contenía luces, interruptores, dispositivos de entrada e impresoras <u>PROBLEMAS</u> <ul style="list-style-type: none"> Planificación, alto nivel de especialización, costos Configuración → carga del compilador, fuente, salvar prog compilado, carga y linkeo 	<ul style="list-style-type: none"> Monitor resistente. <ul style="list-style-type: none"> SW que controla la secuencia de eventos Trabajos se colocan juntos Programas vuelven al monitor cuando finaliza la ejecución No hay interacción con el usuario mientras se ejecutan los trabajos
Sistema batch	Multiprogramación
<ul style="list-style-type: none"> Baja utilización de la CPU Dispositivos de E/S mucho más lentos con respecto a la CPU Ante una instrucción de E/S el procesador permanece ocioso. Cuando se completa la E/S, se continúa la ejecución del programa que se estaba ejecutando 	<ul style="list-style-type: none"> La operación de los sistemas batch fue beneficiada debido a que al estar las tareas cargadas en el disco, no era necesario ejecutarlas en el orden que fueron cargadas El SO mantiene varias tareas en memoria simultáneamente La secuencia de programas es de acuerdo a la <i>prioridad</i> Cuando el proceso necesita realizar una op de E/S, la CPU no permanece ociosa sino que ejecuta otro proceso Luego de atender la interrupción, el control puede o no retornar al programa que se estaba ejecutando



Tiempo compartido	
<ul style="list-style-type: none"> • Utiliza la MP para manejar múltiples trabajos interactivos • El tiempo del procesador es compartido entre múltiples trabajos • Varios usuarios pueden acceder simultáneamente al sistema utilizado • Los procesos usan la CPU por un período máximo de tiempo. 	

ver anexo System Calls

Algoritmos Apropiativos y No Apropiativos

La apropiación está relacionada al recurso CPU

APROPIATIVOS (<i>preemptive</i>)	NO APROPIATIVOS (<i>non preemptive</i>)
Existen situaciones que hacen que el proceso en ejecución sea expulsado de la CPU por el planificador de corto plazo	Los procesos se ejecutan hasta que el mismo abandone la CPU (por su propia cuenta)
<ul style="list-style-type: none"> Round Robin SRTF Prioridades Apropiativo 	<ul style="list-style-type: none"> FCFS SJF Prioridad No apropiativo
El proceso puede ser expulsado de la CPU según la planificación implementada: Se le termina su quantum (<i>RR</i>), llega a la cola de listos un proceso de mayor prioridad (<i>prioridades</i>), llega a la cola de listos un proceso con menor tiempo restante (<i>SRTF</i>)	El procesos deja el estado de ejecución cuando: termina (<i>Syscall Exit</i>), se bloquea voluntariamente (<i>SysCall wait</i>), solicita una operación de E/S bloqueante (<i>Syscall Read, Write, etc</i>)

Pasos desde el momento que un proceso realiza un llamado a System Call

- User o Kernel Mode: modo de ejecución en el que se encuentra la CPU
- Hard o Soft: Si el que realiza la operación es el HW o el SW
- Stack utilizado: indica el stack que se está utilizando (Usuario o Kernel)

User o Kernel Mode	Hard o Soft	Stack Utilizado	Descripción
U	S	U	1. El proceso de Usuario llama a una Syscall por medio de la Glibc
U	S	U	2. La Glibc pone los parámetros para la syscall en el Stack y eleva una interrupción
U	H	U	3. Cambia a Kernel Mode
K	H	U	4. Coloca el PC y PSW en el stack (puede que se coloquen mas registros, dependiente de la arquitectura)
K	H	U	5. Se coloca en el PC la dirección de la rutina de atención de interrupción que se extrae de la IDT y se continua la ejecución
K	S	U	6. Se sacan los parámetros a la syscall del stack
K	S	U	7. De ser necesario se pueden guardar otros mas registros del proceso actual en el Stack o en la PCB, depende de la implementación del SO
K	S	U	8. Se cambia a kernel stack, guardando la dirección del stack en User Mode en la PCB
K	S	K	9. Se colocan los parámetros para la syscall en el Stack
K	S	K	10. Se ejecuta la Syscall
K	S	K	11. Si la Syscall bloquea el Proceso
			11.1. De ser necesario se guarda mas información sobre el proceso bloqueado (registros, estados, etc.)
			11.2. Se ejecuta el Short Term Scheduller para seleccionar un nuevo proceso
			11.3. Se realiza el context switch
			11.3.1. Se cargan los registros del nuevo proceso
			11.3.2. Se acomoda la dirección del Stack, dejando apuntando a la dirección que el HW dejo

			previo a que el proceso seleccionado sea suspendido.
			11.3.3. Se acomodan los datos necesarios en la PCB o estructuras utilizadas
K	S	U	12. Se cambia a User Mode
U	S	U	13. Se ejecuta RET
U	H	U	14. Se sacan de la pila el PSW y PC
U	S	U	15. Continúa la ejecución del proceso actual

PROCESOS
<p>Los procesos son programas en ejecución.</p> <p><i>tarea ≈ proceso ≈ job</i></p>

PROGRAMA	PROCESO
<ul style="list-style-type: none"> • Estático • No tiene PC • Existe desde que se edita hasta que se borra 	<ul style="list-style-type: none"> • Dinámico • Tiene PC • Su ciclo de vida comprende desde que se solicita ejecutar hasta que termina

MODELO	COMPONENTES
<ul style="list-style-type: none"> • Multiprogramación de 4 procesos • Modelo conceptual de 4 procesos secuenciales e independientes • Solo un proceso se encontrará activo en cualquier instante 	<ul style="list-style-type: none"> • Un proceso para poder ejecutarse debe tener una sección de código, una sección de datos (variables globales), <i>stacks</i> (datos temporarios: parámetros, variables temporales y direcciones de retorno)
STACKS	
<ul style="list-style-type: none"> • <i>Un proceso puede contar con uno o más stacks → modo Usuario y modo Kernel</i> • <i>Se crean automáticamente y su medida se ajusta en run-time</i> 	<ul style="list-style-type: none"> • Formado por <i>stack frames</i> los cuales se hace push y pop al llamar a la subrutina • <i>stack frame</i> → posee parámetros de la rutina y datos necesarios para recuperar el stack frame anterior (el PC y el valor del SP en el momento del llamado)
ATRIBUTOS	PCB
<ul style="list-style-type: none"> • Identificación de: <ul style="list-style-type: none"> ○ proceso y del proceso padre ○ del usuario que lo “disparó” ○ grupo que lo disparó (si lo hay) ○ desde qué terminal y quien lo ejecutó (en ambientes multiusuario) 	<ul style="list-style-type: none"> • <i>Process Control Block</i> • Estructura de datos asociada al proceso (abstracción) • Existe UNA por proceso • Es lo primero que se crea al crear un proceso y lo último que se borra cuando termina • Contiene info asociada al proceso

	(PID, PPID, valores de registros de la CPU, planificación, ubicación, accounting, E/S),
ESPACIO DE DIRECCIONES	
<ul style="list-style-type: none"> • Conjunto de direcciones de memoria que ocupa el proceso (<i>stack</i>, <i>text</i> y <i>datos</i>) • NO incluye su PCB o tablas asociadas • Proceso en MU puede acceder <i>solo a su espacio de direcciones</i> • Proceso en MK puede acceder a <i>estructuras internas</i> (PCB del proceso x ej) o <i>espacios de direcciones de otros procesos</i> 	
CONTEXTO DE DIRECCIONES	CAMBIO DE CONTEXTO
<ul style="list-style-type: none"> • Incluye toda la información que el SO necesita para administrar el proceso y la CPU para ejecutarlo correctamente • Contiene <i>registros de CPU, PC, prioridad, E/S pendientes</i>, etc 	<ul style="list-style-type: none"> • Se produce <i>cuando la CPU cambia de un proceso a otro</i> • Se resguarda el contexto del proceso saliente, que pasa a espera y retornará después a la CPU • Se carga el contexto del nuevo proceso y comienza desde la instrucción siguiente a la última ejecutada en dicho contexto • Es tiempo NO productivo de la CPU y el tiempo consumido depende del soporte del HW
Acerca del Kernel del SO	
<ul style="list-style-type: none"> • Conjunto de módulos de SW • Se ejecuta en el procesador como cualquier otro proceso • El kernel NO es un proceso → el concepto de proceso se asocia únicamente a los programas de usuario • Diferentes enfoques de diseño 	

Enfoque 1 <i>kernel como entidad independiente</i>	Enfoque 2 <i>kernel "dentro" del proceso</i>
<ul style="list-style-type: none"> • El kernel se ejecuta fuera de todo proceso, como una entidad independiente en modo privilegiado. • Es una arquitectura utilizada por los primeros SO • Cuando un proceso es interrumpido o realiza una System Call, el contexto del proceso se salva y el control se pasa al Kernel del SO (<i>resguardamos todo lo que venía ejecutándose antes de hacer el salto al Kernel</i>) • Kernel tiene su propia región de memoria y su propio Stack • Al finalizar su actividad, devuelve el control a OTRO proceso 	<ul style="list-style-type: none"> • El "código" del Kernel se encuentra dentro del espacio de direcciones de cada proceso • Kernel se ejecuta en el MISMO contexto que algún proceso de usuario • Kernel se puede ver como una colección de rutinas que el proceso utiliza • Dentro del proceso se encuentra el código del programa (user) y el código de los módulos de SW del SO (kernel) • Cada proceso tiene su propio stack uno en MU, en donde se ejecutará el proceso, y otro en MK, en donde se ejecutará el Kernel del SO • Kernel compartido por todos los procesos • Cada interrupción es atendida en el contexto del proceso que se encuentra en ejecución. <ul style="list-style-type: none"> ◦ Se cambia a MK, y al terminar de resolver debe volver al MU y saltar a la instrucción que estaba antes de que ocurra la interrupción
ENFOQUE	
En su ciclo de vida, un proceso pasa por diferentes estados	→ New (nuevo) → Ready (listo para ejecutar) → Running (ejecutándose) → Waiting (en espera) → Terminated (terminado)
COLAS en la planificación de procesos	Ejemplos
<ul style="list-style-type: none"> • Para realizar la planificación, el SO utiliza el PCB de cada proceso como una abstracción del mismo • Las PCB se enlazan en colas siguiendo un orden determinado 	→ Cola de procesos → Cola de procesos listos → Cola de dispositivos

Módulos de la planificación	Ejemplos
<ul style="list-style-type: none"> • Son <i>módulos</i> (SW) del kernel que realizan distintas tareas asociadas a la planificación • Se ejecutan ante determinados eventos que así lo requieren: <ul style="list-style-type: none"> ◦ creación/terminación de procesos ◦ eventos de sincronización o de E/S ◦ Finalización de lapso de tiempo ◦ Otras 	<p>→ Scheduler de:</p> <ul style="list-style-type: none"> ◆ Long term ◆ Short term ◆ Medium term <p>→ Dispatcher</p> <p>→ Loader</p>
DISPATCHER	LOADER
<ul style="list-style-type: none"> • Hace el cambio de contexto, cambio de modo de ejecución. “despacha” el proceso elegido por el <i>Short Term</i>, es decir que salta a la instrucción a ejecutar 	<ul style="list-style-type: none"> • Carga en memoria el proceso elegido por el <i>Long Term</i>
Long Term Scheduler	Medium Term Scheduler
<ul style="list-style-type: none"> • Controla el grado de multiprogramación (<i>controla la cantidad de procesos en memoria</i>) • Puede no existir y que el Short Term absorba esta tarea 	<ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> ◦ <i>swap out</i> → <i>saca de memoria</i> ◦ <i>swap in</i> → <i>vuelve a memoria</i>
Short Term Scheduler	
<ul style="list-style-type: none"> • Decide a cuál de los procesos en la cola de listos se elige para que use la CPU • Términos asociados <ul style="list-style-type: none"> ◦ <i>apropiativo</i> ◦ <i>no apropiativo</i> ◦ <i>algoritmo de scheduling</i> 	

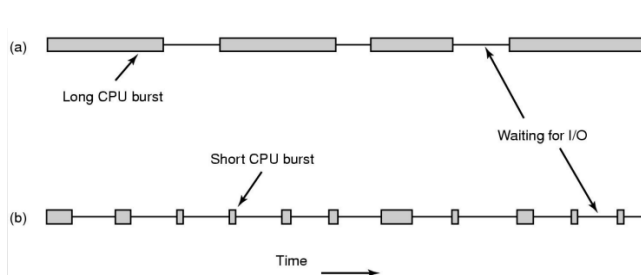
ESTADOS	
NEW	READY
<ul style="list-style-type: none"> • Un usuario “dispara” el proceso. • Un proceso es creado por otro proceso denominado <i>proceso padre</i> • En <i>new</i> se crean las estructuras asociadas, y el proceso queda en la cola de procesos para ser cargado en memoria 	<ul style="list-style-type: none"> • Luego que el long term scheduler eligió al proceso para cargarlo en memoria, el proceso queda en estado <i>listo</i> • el proceso sólo necesita que se le asigne CPU • Está en la cola de procesos listos
RUNNING	WAITING
<ul style="list-style-type: none"> • El proceso fue elegido por el short term scheduler para asignarle CPU • Por lo tanto, tendrá la CPU hasta que se termine el período de tiempo asignado (Quantum o time slice), termine o se necesite realizar una operación de E/S 	<ul style="list-style-type: none"> • El proceso necesita que se cumpla el evento esperado para continuar. <i>Este evento puede ser la terminación de una E/S solicitada o la llegada de una señal por parte de otro proceso</i> • Sigue en memoria, pero no tiene la CPU • Al cumplirse el evento, pasará al estado de listo
TRANSICIONES	
NEW-READY	READY-RUNNING
<ul style="list-style-type: none"> • Por elección del long term scheduler (carga en memoria) 	<ul style="list-style-type: none"> • Por elección del short term scheduler (asignación de CPU)
RUNNING-WAITING	WAITING-READY
<ul style="list-style-type: none"> • El proceso se encuentra en espera del evento 	<ul style="list-style-type: none"> • Termina la espera y compete nuevamente con la CPU
	<ul style="list-style-type: none"> • <u>CASO ESPECIAL</u> <ul style="list-style-type: none"> ◦ Cuando el proceso termina su quantum sin haber necesitado ser interrumpido por un evento, pasa al estado <i>ready</i> para competir por CPU ◦ El proceso es expulsado de la CPU contra su voluntad ◦ Esto se da en algoritmos APROPIATIVOS

EXPLICACIÓN POR ESTADO

1. Ejecución en MU
2. Ejecución en MK
3. Proceso listo para ser ejecutado cuando sea elegido
4. Proceso en espera en memoria principal
5. Proceso listo, pero el swapper debe llevar al proceso a memoria principal antes que el kernel lo pueda elegir para ejecutar
6. Proceso en espera en memoria secundaria
7. Proceso retornado desde el MK al usuario. El kernel se apropia, hace un context switch para darle la CPU a otro proceso
8. Proceso recientemente creado y en transición: existe pero aun no esta listo para ejecutar, ni está dormido
9. Proceso ejecutó la System Call exit y está en estado zombie. Ya no existe más, pero se registran datos sobre su uso, código resultante del exit. Es el estado final

COMPORTAMIENTO DE LOS PROCESOS

Procesos alternan ráfagas de CPU y de I/O



- CPU-Bound → mayor parte del tiempo utilizando la CPU
- I/O-Bound (E/S) → mayor parte dle tiempo esperando por I/O
- La velocidad de la CPU es mucho más rápida que la de los dispositivos de E/S

Planificación

- Necesidad de determinar cual de todos los procesos que están listos para ejecutarse, se ejecutará a continuación en un ambiente multiprogramado
- Algoritmo de Planificación → utilizado para realizar la planificación del sistema

Algoritmos APROPIATIVOS

Existen situaciones que hacen que el proceso en ejecución sea expulsado de la

Algoritmos NO APROPIATIVOS

Los procesos se ejecutan hasta que el mismo abandone la CPU

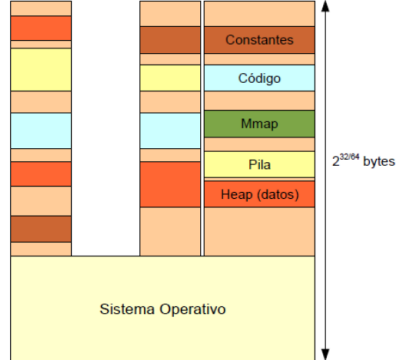
CPU (resumido antes)	<ul style="list-style-type: none"> - Se bloquea por E/S o finaliza - No hay decisiones de planificación durante las interrupciones de reloj
CATEGORÍAS	
<p>Según el ambiente es posible requerir algoritmos de planificación diferentes con diferentes metas:</p> <ul style="list-style-type: none"> - <u>Equidad</u>: se otorga una parte justa de la CPU a cada proceso - <u>Balance</u>: Mantener ocupadas todas las partes del sistema 	<p>Ejemplos</p> <ul style="list-style-type: none"> → Procesos por lotes (batch) → Procesos Interactivos → Procesos en Tiempo Real
Procesos Batch	Procesos Interactivos
<ul style="list-style-type: none"> • No existen usuarios que esperen una respuesta en una terminal • Se pueden utilizar algoritmos no apropiativos • Metas: <ul style="list-style-type: none"> ◦ Maximizar número de trabajos por hora ◦ Minimizar tiempos entre comienzo y finalización ◦ Tiempo de espera puede verse afectado ◦ Mantener la CPU ocupada la mayor cantidad de tiempo posible • Ejemplos: FCFS, SJF 	<ul style="list-style-type: none"> • No sólo interacción con los usuarios <ul style="list-style-type: none"> ◦ Servidor necesita varios procesos para dar respuesta a diferentes requerimientos • Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU • Metas: <ul style="list-style-type: none"> ◦ Responder a peticiones con rapidez ◦ Cumplir con expectativas de los usuarios • Ejemplos: RR, Prioridades, Colas Multinivel, SRTF
<p>Los procesos batch se ejecutan automáticamente sin la intervención directa del usuario y a menudo se utilizan para tareas automatizadas</p>	<p>Los procesos interactivos requieren la interacción directa del usuario y se utilizan en aplicaciones donde la entrada y la salida en tiempo real son fundamentales.</p>

POLÍTICA VS MECANISMO
<ul style="list-style-type: none"> • 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

CREACIÓN DE PROCESOS	Actividad en la creación
<ul style="list-style-type: none"> • Un proceso es creado por otro proceso • Un proceso padre tiene uno o más procesos hijos • Se forma un árbol de procesos 	<ul style="list-style-type: none"> → Crear la PCB → Asignar PID (único) → Asignar memoria para regiones (stack, text, datos) → Crear estructuras de datos asociadas (Fork → copiar contexto, regiones de datos, text, stack)
RELACIÓN PROCESOS PADRE-HIJO	
<ul style="list-style-type: none"> • Ejecución <ul style="list-style-type: none"> ◦ El padre puede continuar ejecutándose concurrentemente con su hijo ◦ El padre puede esperar a que el/los procesos hijos terminen para continuar la ejecución 	<ul style="list-style-type: none"> → Espacio de direcciones <ul style="list-style-type: none"> ◦ Caso Unix → <i>hijo es un duplicado del proceso padre</i> <ul style="list-style-type: none"> ■ Se crea un nuevo espacio de direcciones copiando el del padre ◦ Caso Windows → <i>se crea el proceso y se le carga dentro el programa</i> <ul style="list-style-type: none"> ■ Se crea un nuevo espacio de direcciones vacío
Creación de procesos	
<ul style="list-style-type: none"> • UNIX → 2 System Calls (SC) <ul style="list-style-type: none"> ◦ SC fork() → crea un nuevo proceso igual al llamador ◦ SC execve() → generalmente usada luego del fork, carga un nuevo programa en el espacio de direcciones 	<ul style="list-style-type: none"> → Windows <ul style="list-style-type: none"> ◆ SC CreateProcess() → crea nuevo proceso y carga el programa para ejecución
Terminación de procesos	
<ul style="list-style-type: none"> • Ante un exit se retorna el control al SO <ul style="list-style-type: none"> ◦ El proceso padre puede esperar recibir un código de retorno (via wait). Este se utiliza cuando se quiere que el padre espere a los hijos 	<ul style="list-style-type: none"> → Proceso padre puede terminar la ejecución de sus hijos (kill) <ul style="list-style-type: none"> ◦ La tarea asignada al hijo se terminó ◦ Cuando el padre termina su ejecución no permite a los hijos continuar ◦ Terminación es cascada

PROCESOS INDEPENDIENTES	PROCESOS COOPERATIVOS
<p>→ Independiente: proceso no afecta ni puede ser afectado por la ejecución de otros procesos</p>	<p>→ Cooperativo: afecta o es afectado por la ejecución de otros procesos</p> <p>Sirve para:</p> <ul style="list-style-type: none"> - Compartir información - Acelerar el cómputo (<i>separar una tarea en subtareas que cooperan ejecutandose paralelamente</i>) - Planificar tareas de manera tal que se puedan ejecutar en paralelo

MEMORIA	
<p>Organización y administración de memoria → factores importantes en el diseño del SO</p> <p>Programas y datos → deben estar en el almacenamiento principal para poder ejecutarlos y ser referenciados directamente</p>	
<p>Sistema Operativo encargado de:</p> <ul style="list-style-type: none"> Llevar un registro de las partes de memoria que se están utilizando o no Asignar en MP a los procesos cuando estos la necesitan Liberar espacio de memoria asignada a procesos que han terminado Lograr que el programador se abstraiga de la alocaión de programas 	<ul style="list-style-type: none"> Se espera del SO un uso eficiente de la memoria con el fin de alojar el mayor número de procesos Brindar seguridad entre los procesos para que unos no accedan a secciones privadas de otros Brindar la posibilidad de acceso compartido a determinadas secciones de la memoria (librerías, código común, etc) Garantizar la performance del sistema
ADMINISTRACIÓN DE MEMORIA	
División lógica de la memoria física para alojar múltiples procesos → garantiza protección. Depende del mecanismo provisto por el HW	Asignación eficiente → contener el mayor número de procesos para garantizar el mayor uso de la CPU por los mismo
REQUISITOS	
Reubicación	Protección
<p>El programador no debe ocuparse de conocer dónde será colocado el proceso en la memoria RAM.</p> <p>Mientras el proceso se ejecuta puede ser sacado o traído a la memoria (swap) y colocarse en diferentes direcciones.</p> <p>Las referencias a memoria se deben traducir según ubicación actual del proceso</p>	<p>Los procesos no deben referenciar/acceder a direcciones de memoria de otros procesos. Solo podrán hacerlo en el caso de que tengan permiso</p> <p>El chequeo se debe realizar durante la ejecución</p>
	Compartición
	<p>Permitir que varios procesos accedan a la misma porción de memoria. Por ejemplo: rutinas comunes, librerías, espacios explícitamente compartidos</p> <p>Permite un mejor aprovechamiento de la</p>

	memoria RAM, evitando copias repetidas de instrucciones
ABSTRACCIÓN - ESPACIO DE DIRECCIONES	
<p>Rango de direcciones posibles que un proceso puede utilizar para direcciones sus instrucciones y datos</p> <p>Tamaño → depende de la arquitectura del procesador</p> <p>Es independiente de la ubicación real del proceso en memoria RAM</p>	
DIRECCIONES	
LÓGICAS	FÍSICAS
<p>Referencia a una localidad de memoria independiente de la asignación actual de los datos en la memoria</p> <ul style="list-style-type: none"> - Representa una dirección en el “Espacio de Direcciones del Proceso” 	<p>Referencia a una velocidad en la memoria física (RAM) → dirección absoluta</p> <p>En caso de utilizar direcciones lógicas, es necesario algún tipo de conversión a direcciones físicas</p>
<p><i>Cuando un programa se ejecuta, utiliza direcciones lógicas para acceder a datos en su espacio de direcciones sin necesidad de preocuparse por la ubicación física real de esos datos en la memoria.</i></p> <p><i>Cada proceso que se ejecuta en un sistema operativo tiene su propio espacio de direcciones lógicas, lo que le permite pensar en la memoria como si fuera una secuencia lógica de direcciones, independientemente de la memoria física real.</i></p>	<p><i>Las direcciones físicas son direcciones absolutas que se utilizan para acceder directamente a la memoria física y recuperar los datos almacenados en esas ubicaciones.</i></p>
CONVERSIÓN DE DIRECCIONES	
<p>Se utilizan registros auxiliares para la conversión de direcciones.</p> <p>Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria</p> <p>Varían entre procesos → context switch</p>	

- **REGISTRO BASE** → dirección de comienzo del Espacio de Direcciones del proceso de la RAM
- **REGISTRO LÍMITE** → dirección final del proceso (tamaño de su Espacio de Direcciones)

Direcciones lógicas vs Físicas

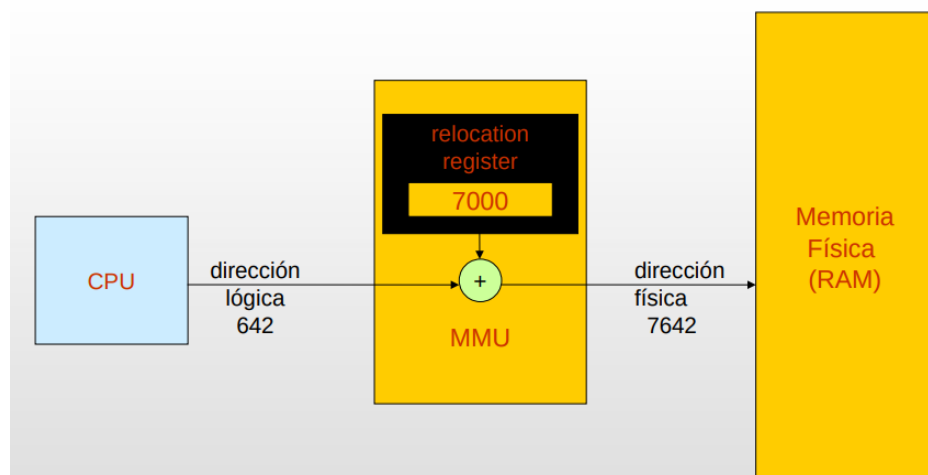
- Address-binding → si la CPU trabaja con direcciones lógicas, para acceder a MP, se deben transformar en direcciones físicas.
- En el momento de *compilación* y en *tiempo de carga* las direcciones lógicas y físicas son idénticas. Para reubicar un proceso es necesario recompilar o recargarlo
- En *tiempo de ejecución* las direcciones lógicas (virtuales) y físicas son diferentes. La reubicación se puede realizar fácilmente. El mapeo entre virtuales y físicas se realiza por HW → MEMORY MANAGEMENT UNIT (MMU)

MMU

Memory Management Unit

Es un dispositivo de HW el cual es parte del procesador y que mapea direcciones *virtuales* a *físicas*. Re-programar el MMU es una operación privilegiada por lo tanto se realiza en MK.

El valor en el “registro de realoación” es sumado a cada dirección generada por el proceso usuario al momento de acceder a la memoria. *Los procesos NUNCA usan direcciones físicas*



Mecanismos de asignación de memoria	
<ul style="list-style-type: none"> ● Particiones Fijas: <ul style="list-style-type: none"> ○ Memoria se divide en particiones o regiones de tamaño fijo las cuales pueden ser del mismo tamaño o no ○ Alojan un proceso cada una, cada proceso se coloca de acuerdo a algún criterio en alguna partición (<i>First Fit, Best Fit, Worst Fit, Next Fit</i>) 	<ul style="list-style-type: none"> ● Particiones dinámicas: <ul style="list-style-type: none"> ○ Varían en tamaño y número ○ Alojan un proceso cada una ○ Cada partición se genera en forma dinámica del tamaño justo que necesita el proceso
<p>Ambos mecanismos generan el problema de FRAGMENTACIÓN, la cual se produce cuando una localidad de memoria NO puede ser utilizada por no encontrarse en forma contigua</p>	
<ul style="list-style-type: none"> ● Particiones Fijas <ul style="list-style-type: none"> ○ Existe la fragmentación interna → es la porción de la partición que queda sin utilizar 	<ul style="list-style-type: none"> ● Particiones dinámicas: <ul style="list-style-type: none"> ○ Existe la fragmentación externa → huecos que van quedando en la memoria o medida de los procesos finalizan ○ Solución: compactación, pero es costosa
Esquema de Registro Base + Límite	
<p><u>Problemas!</u></p> <ul style="list-style-type: none"> ● Necesidad de almacenar el Espacio de Direcciones de forma <i>continua</i> en la mem física ● Los 1eros SO definían particiones fijas en mem, luego evolucionaron a particiones dinámicas ● Fragmentación ● Mantener “partes” del proceso que no son necesarias ● Los esquemas de particiones fijas y dinámicas no se usan 	<p><u>SOLUCIÓN</u></p> <ul style="list-style-type: none"> ● PAGINACIÓN ● SEGMENTACIÓN

PAGINACIÓN

La memoria física es dividida lógicamente en **marcos** (pequeños trozos de igual tamaño)
Por otro lado, la memoria lógica, es decir el espacio de direcciones, es dividida en **páginas** (trozos de igual tamaño que los marcos)

El SO debe mantener una tabla de páginas por cada proceso, donde cada entrada contiene el marco en la que se coloca la página

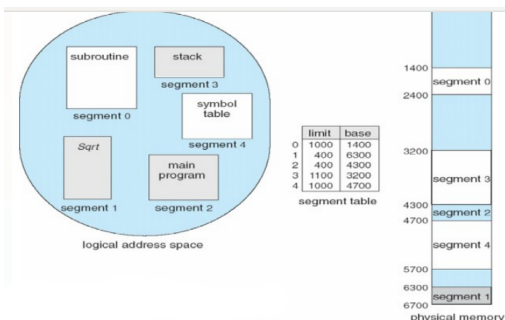
La dirección lógica se interpreta como un número de página y un desplazamiento dentro de la misma

SEGMENTACIÓN

Es un esquema el cual es similar a la visión de usuario. El programa se divide en partes/secciones.

Este programa es una colección de segmentos, los cuales son unidades lógicas como un programa principal, variables globales o locales, stack, entre otras.

La segmentación puede causar fragmentación

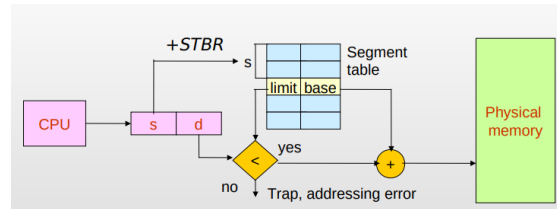


Todos los segmentos de un programa pueden no tener el mismo tamaño
Las direcciones lógicas consisten en dos partes

1. Selector de segmento
2. Desplazamiento dentro del segmento

ARQUITECTURA

- Tabla de segmentos → permite mapear la direc lógica en física. Cada entrada contiene base (DF de comienzo del segmento) y limit (longitud del segmento)
- Segment-table base register (STBR): apunta a la ubicación de la tabla de segmentos
- Segment-table length register (STLR): cantidad de segmentos de un programa



Ventajas

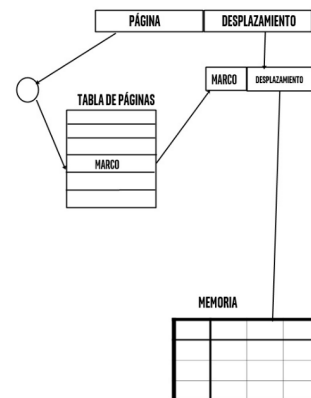
- Compartir
- Proteger

SEGMENTACIÓN PAGINADA

- La *paginación* es transparente al programador y elimina la fragmentación externa
- La *segmentación* es visible al programador, facilita modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección
- **SEGMENTACIÓN PAGINADA:** cada segmento es dividido en páginas de tamaño fijo

Se refiere a un esquema de administración de memoria que divide la memoria física en segmentos o bloques de tamaño variable, y cada segmento se asigna a una tarea o proceso. Cada segmento se divide adicionalmente en páginas de tamaño fijo.

Modelo Paginación



- Intel 386 segmentación con paginación para la administración de la memoria con un doble esquema de paginación