ANEXO SYSTEM CALLS

SYSTEM CALLS

NUMERO DE SYSCALL:

* x86 32bit
  + write 🡪 4
  + exit 🡪 1
  + INSTRUCCIÓN : int 80h
  + PARAMETROS:
    - EAX: num de syscall
    - EBX: primer param
    - ECX: primer param
    - EDX …
    - ESI
    - EDI
* x86 64bit
  + write 🡪 1
  + exit 🡪 60
  + INSTRUCCIÓN : syscall
  + PARAMETROS:
    - EAX: num de syscall
    - RDI: primer param
    - RSI: primer param
    - RDX …
    - R10
    - R8
    - R9

DEPENDIENDO DE LA ARQUITECTURA:

CAMBIA el numero de syscalls

CAMBIA la forma de pasar los param al kernel

ANEXO BLUE SCREEN OF DEATH

Se producen cuando ocurre una trap estando en modo kernel

PROCESOS 1:

Que es un proceso? es un programa en ejecución (sinónimo de tarea y job)

|  |  |
| --- | --- |
| Programa | Proceso |
| Es estatico | Es dinámica |
| No tiene | Tiene Program Counter |
| Exixte desde que se edita hasta que se borra | Su vida es desde que se solicita hasta que termina |

Un proceso incluye:

* Sección de código (texto)
* Sección de datos (var globales)
* Stack(s) (datos temporarios: parámetros, var temp y dir de retorno)

Stacks:

Un proceso cuenta con 1 o mas stacks (Generalmente modo usuario y modo kernel)

Se crean automáticamente y se ajusta su medida en run-time

Se forma de Stack-Frames, estos se pushean al llamar a la rutina y se poppean al retornar de ella

El stack frame tiene los param de la rutina(var loc) y datos necesarios para rec el stack frame anterior (Program counter y stack pointer anterior)

Atributos de un proceso:

PID identificación del proceso y PPID identificación del proceso padre

Identificación del usuario que lo inicio

Si hay grupos, grupo que lo inicio

En multiusuario, desde que terminal y quien lo ejecuto

PCB (Process Control Block)

* Est de datos asociada al proceso
* Exixte una por proceso
* Es lo primero que se crea cuando se crea un proceso y lo ultimo que se borra al terminar este
* Contiene info asociada con cada proceso:
  + PID, PPID, etc
  + Valores de registros de la CPU
  + Planificación
  + Ubicación en memoria
  + Accounting
  + E/S

Context switch

* Se produce cuando la CPU cambia un proceso por otro
* Se debe guardar el contexto del proceso que sale
* Se debe cargar el contexto del proceso que entra y se inicia la ejecución desde la instrucción siguiente a la ejecutada ultima en dicho contexto
* Es tiempo NO productivo
* El tiempo que consume depende del HW

Sobre el kernel:

* Es un conjunto de módulos de software
* Se ejecuta en el procesador como cualquier otro proceso
* Diferentes enfoques:
  + Entidad independiente
    - El kernel se ejecuta fuera de todo proceso
    - Se utilizaba en los primeros SO
    - cuando se realiza una System Call, se salva el contexto y el control pasa al kernel del SO
    - El kernel tiene su propia región de memoria
    - El kernel tiene su propio stack
    - Finalizada su act, le devuelve el control al proceso
    - IMPORTANTE:
      * El kernel NO es un proceso, Procesos solo son programas de usuario
      * Se ejecuta como una entidad independiente en un modo privilegiado
  + El kernel dentro del proceso
    - El código del kernel se encuentra dentro del espacio de direcciones de cada proceso
    - El kernel se ejecuta en el MISMO contexto que algun proceso de usuario
    - El kernel se puede ver como una colección de rutinas que el proceso utiliza
    - Dentro de un proceso se encuentra el código del programa(user) y el código de los módulos de SW del SO
    - Cada proceso tiene su propio stack (uno de usuario y otro de kernel)
    - el proceso se ejecuta en modo usuario y el kernel del SO se ejecuta en modo kernel (cambio de modo)
    - El código del kernel es compartido por todos los procesos
    - Cada interrupción es atendida en el contexto del proceso en ejecución
      * EN MODO KERNEL
      * El SO decide si se continua ejecutando e proceso

PROCESOS 2:

Estados de un proceso: (diferentes estados por lo sque pasa durante su vida)

* Nuevo (new)
  + Un usuario dispara un proceso. Un proceso se crea por otro proceso: su padre
  + En este estado se crean las estructuras asociadas y el proceso queda en la cola de procesos, normalmente en espera de ser cargado en memoria
* Listo para ejecutar (ready)
  + Luego que el scheduller de largo plazo eligió al proceso para cargarlo en memoria el proceso queda en estado listo
  + El proceso solo necesita que se le libere la CPU
  + Esta en lista de procesos listos (ready queu)
* Ejecutandose (running)
  + El scheduler de corto plazo lo eligió para asignarle CPU
  + Tendrá la CPU hasta que se termine el periodo de tiempo asignado (quantum o time slice), termine o hasta que necesite realizar alguna operación de E/S
* En espera (waiting)
  + El proceso necesita que se cumpla el evento esperado para continuar
  + El evento puede ser la terminación de una E/S solicitada, o la llegada de una señal por parte de otro proceso
  + SIGUE EN MEMORIA, pero no tiene la CPU
  + Al cumplirse el evento, pasara al estado listo
* Terminado (terminated)

Transiciones

* NEW-READY: Por elección del scheduler de largo plazo (carga en memoria)
* READY-RUNNING: Por elección del scheduler de corto plazo (asignación de CPU)
* RUNNING-WAITING: El proceso “se pone a dormir”, esperando por un evento
* WAITING-READY: Termino la espera y compite nuevamente por la CPU
* CASO ESPECIAL!! RUNNING-READY:
  + Cuando el proceso termina su quantum sin haber sido interrumpido por un evento, pasa al estado ready, para competir por CPU, NO esta esperando por algún evento
  + Se trata de un caso distinto a los anteriores, porque los procesos son expulsados de la CPU contra su voluntad
  + Esta situación se da en algoritmos APROPIATIVOS

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Planificación de procesos: Colas

* El SO utiliza las PCB como abstracciones de los procesos
* Las PCB se enlazan en colas siguiendo un orden especifico
* EJEMPLOS:
  + Cola de trabajos o procesos
    - Contiene todas las PCB de procesos en el sistema
  + Cola de procesos listos
    - PCB de procesos residentes en memoria principal esperando a ejecutarse
  + Cola de dispositivos
    - PCB de procesos esperando por algun disp. de E/S

Módulos de la planificación

* Son módulos del kernel que realizan distintas tareas asociadas a la planificación
* Se ejecutan ante determinados eventos que así lo requieren
  + Creación/Terminacion de procesos
  + Eventos de sincronización o de E/S
  + Finalizacion de lapso de tiempo
  + Etc
  + Long term Scheduler
    - Controla el grado de multiprogramación
    - Puede no existir y absorber esta tarea el de short term
  + Medium term Scheduler
    - Si es necesario reduce el grado de multiprogramación
    - Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema
    - Swap out (sacar de memoria) Swap in (Volver a memoria)
  + Short term Scheduler
    - Decide a cual de los procesos en al cola de listos se elige para que use la CPU
  + Dispacher
    - Hace cambio de modo de ejecución, despacha el proceso elegido por el Short term
  + Loader
    - Carga en memoria el proceso elegido por el long term

PROCESOS 3:

* CPU- bound
  + Se utiliza la CPU en su mayoría
* I/O bound
  + Se utiliza la E/S la mayor parte del tiempo
* La vel de la CPU es mucho mas rápida que la de los disp. E/S

Planificación:

Necesidad de determinar cual de los procesos que están listos para ejecutarse, se ejecutara a continuación en un ambiente multiprogramado

Algoritmo de planificación:

Que algoritmo se utiliza para realizar la planificación del sistema

* Algoritmos apropiativos (preemptive)
  + Situaciones que hacen que el proceso en ejecución sea expulsado de la CPU por el short term scheduler
  + Puede ser expulsado:
    - Se le termina su quantum (Round Robin)
    - Llega a la cola de listos un proceso de mayor prioridad (Prioridades Apropiativo)
    - Llega a la cola de listos un proceso con menor tiempo restante (SRTF)
  + EJEMPLOS:
    - Round Robin
    - SRTF
    - Prioridades Apropiativo
* Algoritmos no apropiativos (nonpreemtive)
  + los procesos se ejecutan hasta que el mismo abandone la CPU
  + El proceso deja el estado cuando:
    - Termina (Syscall Exit)
    - Se bloquea voluntariamente (SysCall wait, sleep, etc)
    - Solicita una operación de E/S (Syscall Read, Write, etc)
  + No hay decisiones de planificación durante las interrupciones de reloj
  + EJEMPLOS:
    - FCFC (First Come First Served)
    - SJF (Shortest Job First)
    - Prioridades No apropiativo

Categorías de los algoritmos de planificación

* Según el ambiente es posible requerir algroitmos 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
* Ej:
  + Procesos por lotes (batch)
    - No existen usuarios que esperen una respuesta en una terminal
    - Se pueden utilizar algoritmos no apropiativos
    - Metas de este tipo de algoritmos
      * Rendimiento: Maximizar el numero de trabajos por hora
      * Tiempo de retorno: Minimizar los tiempos entre el comienzo y la finalización
      * El tiempo de espera se puede ver afectado
      * Uso de la CPU: Mantener la CPU ocupada la mayor cantidad de tiempo posible
      * EJEMPLOS
        + FCFS – First Come First Served
        + SJF – Shortest Job First
  + Procesos interactivos
    - No solo interactúan con los usuarios
      * Un servidor requiere de varios procesos para dar respuesta a diferentes requerimientos
    - Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU
    - Metas 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 reproducir música, que la música deje de ser reproducida en un tiempo considerablemente corto)
      * EJEMPLOS
        + Round Robin
        + Prioridades
        + Colas Multinivel
        + SRTF – Shortest remaining time first
  + Procesos en tiempo real

Política Versus 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/admin utiliza os párametros para determinar la política
* EJEMPLOS:
  + Un algoritmo de planificación por prioridades y una system Call que permite modificar la prioridad de un proceso (man nice)
  + Un proceso pude determinar las prioridades de los procesos que el crea, según la importancia de los mismos

COSAS EXTRA:

https://catedras.linti.unlp.edu.ar/mod/quiz/review.php?attempt=75725&cmid=36306#q35

https://catedras.linti.unlp.edu.ar/mod/quiz/review.php?attempt=75870&cmid=36306

Stdin 0 (teclado)

Stdout 1 (pantalla)

Stderr 2 (pantalla)

Ls Lista los archivos en un directorio

Cat muestra el contenido de un archivo

^Z deja el proceso

Ps muestra los procesos activos de la sesión

PID identificación del proceso

PPID identificación del proceso padre

Multiprogramacion = cant de procesos en memoria

COMENTARIOS DE LA PROFE:

Quedar claro el diagrama del hardware, kernel, herramientas y Shell, QUIEN INTERACTUA CON QUIEN

En que momento se evalua si hay una interrupción? Al final de cada ciclo de ejecución

Acordarse de las trap. Acceder a una dir ilegal no es una trap, es por hardware

MMU es hardware

Una system call no genera la creación de un proceso. El mismo proceso que la llama puede acceder momentáneamente a una región privilegiada

Las PCD nunca van a memoria secundaria, siempre están en memoria principal, Es lo primero que se crea y lo ultimo que se borra

Quien crea la PCB? La System Call fork()

El loader solo carga el proceso a memoria una vez que ya esta creado

Para ver si algo es una System call, hay que ver si un proceso puede resolverlo sin salir de su espacio de direcciones.

SIEMPRE QUE UN PROCESO TENGO QUE SALIR ES UNA SYSTEM CALL

El estado del proceso esta en la PCB

Las PCB siempre están en memoria porque el kernel requiere consultarlas rápidamente

accesos no autorizados a mem se resuelven por hardware

cambio de contexto: se guarda la dirección de la tabla de paginas en la PCB

El vector de interrupciones esta en memoria

Un fork() genera un cambio de contexto al hijo

El cambio de contexto lo hace el dispacher

Orden del dispacher:

* cambio de contexto

PROCESOS 4:

Creacion de procesos: Un proceso es creado por otro proceso

Un proceso padre tiene uno o mas hijos

Se forma un árbol de procesos

Actividades en la creación de un proceso:

Crear la PCB (Todas las PCB son del mismo tamaño)

Asignar PID (Process identification) único

Asignarle memoria para regiones (Stack, Text, Datos)

Crear estructuras de datos asociadas (Fork: copiar el contexto, regiones de datos, text y stack)

El padre puede continuar ejecutándose concurrentemente con su hijos

El padre puede esperar a que el proceso hijo(o los procesos hijos) terminen para continuar la ejecución

Espacio de direcciones

El hijo es un duplicado del proceso padre(Unix)

Se crea el proceso y se le carga adentro el programa (Windows)

System Calls:

* En unix:
  + Fork() crea un nuevo proceso
  + Execve() carga un nuevo proceso en el espacio de direcciones
* En Windows:
  + CreateProcess() crea un nuevo proceso y carga el programa para ejecución

Al crearse un proceso al padre se le envía la PID del hijo para que así pueda conocer a sus hijos

¿Porque los hijos no necesitan recibir la PPID? Porque al crearse la PCB ya se almaceno la PPID del padre. ¡!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

En el ejemplo SysCall Fork

¿Por qué cuando se hacen todos estos llamados se hace el mismo PC? Porque se usan direcciones relativas respecto al inicio de donde se carga el proceso.

Terminación de procesos

* Ante un exit, se retorna el control al sistema operativo
  + El proceso padre puede esperar recibir un código de retorno (via wait). Generalmente se lo usa cuando se requiere que el padre espere a los hijos
* Proceso padre puede terminar la ejecución de sus hijos (kill)
  + La tarea asignada al hijo se termino
  + Cuando el padre termina su ejecución
    - Habitualmente no se permite a los hijos continuar, pero si se puede
    - Terminación no en cascada

Procesos Cooperativos e Independientes

* Independiente: el proceso no afecta ni puede ser afectado por la ejecución de otros procesos. No comparte ningún tipo de dato
* Cooperativo: afecta o es afectado por la ejecución de otros procesos en el sistema
  + Para compartir información
  + Para acelerar el computo (separar una tarea en subtareas)
  + Para planificar tareas de manera tal que se puedan ejecutar en paralelo

MEMORIA:

Los programas y datos deben estar en el almacenamiento principal para:

* Poderlos ejecutar
* Referenciarlos directamente

Administración de memoria

Objetivos:

* Reubicabilidad: ser capaz de mover un proceso dentro de la memoria
* Convertir direcciones lógicas a físicas al momento de leer en memoria,

Tareas:

El SO debe:

* Llevar un registro de las partes de memoria utilizadas y aquellas que no
* Asignar espacios en memoria principal a los procesos
* Liberar espacio de memoria asignada a procesos que ya terminaron
* Lograr que el programador se abstraiga de la alocacion de los programas
* 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 en común, etc
* Garantizar la performance del sistema

Se espera que el SO utilice eficientemente la memoria con el fin de alojar el mayor número de procesos

Administración de memoria

* División lógica de la memoria física para alojar múltiples procesos
  + Garantizando protección
  + Depende del mecanismo provisto por el HW
* Asignacion eficiente
  + Contener el mayor numero de procesos en memoria para garantizar el mayor uso de la CPU por los mismos

Requisitos

* Reubicacion:
  + El programador no debe ocuparse de conocer donde será alocado en la memoria RAM
  + Mientras un proceso se ejecuta, puede ser saccadp y traído a la memoria y posiblemente, colocarse en dif direcciones
  + Las ref a la memora se deben traducir según ubicación actual del proceso
* Protección:
  + Los procesos NO deben referenciar (acceder) a direcciones de mem de otros procesos
    - Salvo con permiso
  + El chequeo se debe realizar durante la ejecución:
    - NO es posible anticipar todas las referencias a memoria que un proceso puede realizar
* Compartir:
  + Permitir que varios procesos accedan a la misma porción de memoria
  + Permite un mejor uso de la RAM, evitando copias innecesarias de instrucciones

Abstracccion – rango de direcciones:

* Rango de direcciones de memora posibles que un proceso puede utilizar para direccionar datos e instrucciones
* El tamaño depende de la arquitectura del procesador
  + Es independiente de la ubicación real del proceso en la memora RAM

Direcciones:

* Lógica
  + Referencia a una localidad de memoria independientemente de la asignación de los datos en la memoria
  + Representa una dirección en el “Espacio de Direcciones del Proceso”
* Fisicas
  + Referencia una localidad en la memoria física (RAM)
    - DIRECCION ABSOLUTA
* Si se usan direcciones logicas se debe contar con algun tipo de conversión a dir fisicas

Conversion de Direcciones

* Registro Base
  + Direccion de comienzo del Espacio de Direcciones del proceso en la RAM
* Registro Limite
  + Direccion final del proceso o medida del proceso (tamaño de su espacio de dir)
* Ambos valores se fijan cuando el espacio de direcciones del proceso es cargado a memoria
* Varían entre procesos (Context Switch)

resolución de direcciones:

se usa un registro de relocacion(RR) para guardar la dirección donde esta cargado el proceso; las variables guardadas en un ejecutable quedan guardadas con direcciones relativas; al momento de recuperarla se utiliza el RR y se le suma al valor de la dirección relativa. (Resolver una dirección) de ello se encarga un MMU(Memory Management Unit) (apoyo del hardware)

*imagen del proceso:*

Dir lógicas(relativas o virtuales) vs Físicas(o absolutas)

* Si la CPU trabaja con direcciones lógicas , para poder acceder a mem principal se deben transformar en dir físicas
* Resolución en momento de compilación (Archivos .com de DOS) y en tiempo de carga
  + Dir logicas y físicas son idénticas
  + Para reubicar un proceso es necesario recompilarlo o recargarlo
* Resolución en momento de ejecución
  + Direcciones lógica y fisicas diferentes
  + Direcciones logicas = direcciones virtuales
  + La reubicación puede realizarse fácilmente
  + El mapeo entre fisicas y logicas se resuelve por hardware
    - MMU (Memory Management Unit)

MMU:

* Dispositivo de hardware que mapea direcciones virtuales a fisicas
  + Es parte del procesador
  + Re-programar el MMU es una operación privilegiada
    - Solo puede ser realizada en modo kernel
* El valor en el “registro de realocacion” es sumado a cada dirección generada por el proceso de usuario al momento de acceder a la memoria
  + Los procesos NUNCA usan direcciones fisicas

Mecanismos de asignación de memoria:

* Particiones Fijas: El primer esquema:
  + La memoria se divide en particiones o regiones de tamaño fijo (Pueden o no ser de = tamaño)
  + Alojan UN proceso cada una
  + Cada proceso se coloca de acuerdo a algun criterio (First Fit, Best fit, Worst Fit, Next Fit) en alguna particion

Si quedan espacios de memoria sin utilizar se le llama fragmentación interna

Particiones de <> tamaño: Ventaja: se puede elegir que partición usar para tener una < fragmentacion interna. Puede suceder que si la memoria esta completamente utilizada menos una part grande, y quiere entrar un proc chico se quedaría esperando por la partición chica.

Objetivo de la memoria: que se puedan ejecutar la mayor cantidad de procesos posibles.

* Particiones dinámicas:
  + Las particiones varían en tamaño y en numero
  + Alojan un proceso cada una
  + Cada partición se genera de forma dinámica del tamaño que necesita el proceso

Fragmentación:

* Esta se produce cuando una localidad de memoria no puede ser utilizada por no encontrarse en forma contigua
* Fragmentacion Interna
  + Se produce en el esquema de particiones fijas
  + Es la porción de la partición que queda sin utilizar
* Fragmentacion Externa
  + Se produce en el esquema de particiones dinámicas
  + Son huecos que van quedando de la memoria a medida que los procesos finalizan
  + Al no encontrarse en forma contigua puede darse el caso de que tengamos memoria libre para alocar un proceso, pero que no la podamosutilizar
  + Para solucionar el problema se puede acudir a la compactación, pero es muy costosa

Problemas

* El esquema de Registro Base + Limite presenta problemas
  + Necesidad de almacenar el Espacio de Direcciones de forma contigua en al memoria física
  + Los primeros SO definían particiones fijas de memoria, luego se comenzaron con las dinámicas
  + Fragmentacion
  + Mantener “partes” del proceso que no son necesarias
  + Los esquemas de particiones fijas y dinámicas no se usan hoy en día
* Solución
  + Fragmentacion
  + Segmentación

Paginacion

* Mem física dividida lógicamente en trozos de igual tamaño 🡪 Marcos
* Mem Lógica (espacio de direcciones) es dividido en trozos de igual tamaño que los marcos 🡪 Paginas
* El SO debe mantener una tabla de paginas por cada proceso, donde cada entrada contiene (entre otras) el Marco en la que se coloca cada pagina
* La dirección lógica se interpreta como
  + Un numero de pagina y un desplazamiento dentro de la misma

Soporte brindado:

Manejo optimo del recurso:

Administración de memoria entre procesos:

CPU multiplexado en el tiempo

Memoria multiplexada en el espacio

|  |  |  |  |
| --- | --- | --- | --- |
|  | Particiones | | |
|  | Fijas | | Dinámicas |
|  | = tamaño | <>tamaño |  |
| Procesos | Todo en memoria y en forma contigua | | |
| Estructuras de datos | Tablas de particiones(nro partición, dirección ini/fin, ocupada?, PID proceso) | | Fragmentos en la mem(Dir ini, dir fin/tamaño, …)  Lugares ocupados(PID, ini, fin, …) |
| fragmentación | Interna | | Externa |
| Elección de partición |  |  | Best fit: es el que me genera la menor fragmentación externa  Worst fit: es el que genera la mayor fragmentación externa  Next fit: Próximo lugar que se encuentre donde el proceso entre  First fit: primer lugar en la lista de Huecos donde entre el proceso. |
| Desventajas | Tener que tener TODO el proceso cargado y en forma contigua | | |
| Tamaño máximo del proceso | Tamaño de las particiones | Tamaño de la mayor partición | Mayor hueco disponible |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Actualmente: la memoria se encuentra dividida en marcos(frames) de algún tamaño especifico

Los procesos se dividen en paginas de = tamaño que los frames

|  |  |
| --- | --- |
|  | Paginacion Pura |
| Procesos | TODO en memoria, sin exigencia de continuidad. En la CPU se guarda la dirección de la tabla de paginas del proceso que se esta ejecutando.  Se pueden pensar también paginas de datos y de instrucciones(puede ser preferible ya que las instrucciones no se modifican usarlas en Read only) |
| Estructuras de datos | Tabla de paginas por cada proceso(Marco de cada pagina, bits de control)  PTE Page Table Entry  Esta tabla es una estructura de datos del kernel  Frames libres |
| Fragmentación | Interna (en el espacio libre de la ultima pagina) |
| Elección de partición | Se van guardando las paginas en cada marco libre |
| Desventajas | Tengo TODO el proceso en memoria |
| Tamaño máximo del proceso | Tamaño máximo de la memoria |

|  |  |
| --- | --- |
|  | Paginacion por demanda |
| Procesos | Solo la parte necesaria en memoria, sin exigencia de continuidad. |
| Estructuras de datos |  |
| Fragmentación |  |
| Elección de partición |  |
| Desventajas |  |
| Tamaño máximo del proceso |  |

Segmentación:

* El programa se divide en partes/secciones
* Un programa es una colección de segmentos, cada uno es una unidad lógica:
  + Programa principal
  + Procedimientos y funciones
  + Variables locales y globales
  + Stack
  + etc
* Puede causar fragmentacion externa
* Todos los segmentos de un programa pueden no tener el mismo tamaño
* las direcciones logicas consisten en 2 partes
  + Selector de Segmento (numero de segmento)
  + Dezp dentro del segmento
* Arquitectura
  + Tabla de segmentos
    - Perite mapear la dirección lógica en física cada entrada contiene
      * Base: dirección física del comienzo del segmento
      * Limit: Longitud del segmento
  + Segment- table base register (**STBR**): apunta a la ubicación de la tabla de segmentos (GUARDADA EN LA CPU)
  + Segment-table length register (STLR): cantidad de segmentos de un programa

|  |  |
| --- | --- |
|  | Segmentación Pura |
| Procesos | Todo el proceso en memoria, sin exigencia de continuidad. |
| Estructuras de datos | Tabla de segmentos(identificador, inicio, fin, bits de control) TLP |
| Fragmentación | Externa |
| Elección de partición | Best fit… |
| Desventajas | Cada segmento puede ser muy grande y eso no es recomendable |
| Tamaño máximo del proceso |  |

Segmentación paginada:

* La paginación
  + Transparente al programador
  + Elimina la Fragmentacion Externa
* Segmentación
  + Es visible al programador
  + Facilita la modularidad, estructuras de datos grandes y da mejor soporte a la compartición y protección
* Segmentación paginada
  + Cada segmento es dividido en paginas de tamaño fijo

|  |  |
| --- | --- |
|  | Segmentación Paginada Pura |
| Procesos | Todo el proceso en memoria, sin exigencia de continuidad. |
| Estructuras de datos | Tabla de segmentos(identificador, inicio, fin, bits de control) TLP  Tabla de paginas por CADA segmento |
| Direccion fisica | Segmento, Pagina, Desplazamiento |
| Fragmentación | Interna |
| Elección de partición |  |
| Desventajas | Todo el proceso debe estar en la memoria |
| Tamaño máximo del proceso |  |

|  |  |
| --- | --- |
|  | Segmentación Paginada Por demanda |
| Procesos | Partes necesarias del proceso en memoria, sin exigencia de continuidad. |
| Estructuras de datos | Tabla de segmentos(identificador, inicio, fin, bits de control) TLP  Tabla de paginas por CADA segmento |
| Direccion fisica | Segmento, Pagina, Desplazamiento |
| Fragmentación | Interna |
| Elección de partición |  |
| Desventajas |  |
| Tamaño máximo del proceso |  |

Si los procesos tienen muchas paginas el tener toda la tabla de paginas en memoria ocupa una parte importante de esta

Solución: paginar la tabla de paginas

Esquema: Tabla de dos niveles

Tabla de hashing o invertida: hay una sola tabla de paginas para todos los procesos

proceso | pagina | frame

Solo se mantienen las PTE de las paginas presentes en memoria física

Se organiza como tabla hash en memoria principal

La forma de organizar depende del HW

MEMORIA 2:

Memoria Virtual:

Que se necesita:

* El hardware debe soportar la paginación por demanda
* Un disp. De memoria secundaria, que de apoyo para almacenar las secciones del proceso que no están en memoria principal
* ¿Qué formato tiene una dirección virtual?

¿Qué formato tiene una dirección física? Nro de Frame, desplazamiento.

¿Qué formato tiene una dirección virtual? Pagina,

TLB Translation Lookaside Buffer: es una cache que guarda las ultimas PTE referenciadas del proceso que se esta ejecutando. Cuando el proceso termina se limpia la TLB.

Tiempo efectivo de acceso a memoria:

p = tasa de fallos

EAT = (1-p)\*memAccess + p\*(page\_fault\_overhead + [swap\_page\_out] + swap\_page\_in + restart\_overhead)

Memoria virtual paginación por demanda

Tamaño de paginas:

Pequeño

Menor fragmentación interna

Mas paginas requeridas por proceso 🡪 tablas de paginación mas grandes

Mas paginas pueden residir en memoria

Puede tener ventaja para sacar paginas de la ram, ya que es ma sprobable que una pagina ya no se use.

Grande

Mayor fragmentación interna

La memoria secundaria esta diseñada para transferir grandes bloques de datos mas eficientemente 🡪 Mas rápido mover paginas hacia la memoria principal

Usualmente se utilizan tablas pequeñas pero se mueven de a grupos para simular tablas grandes

Translation Lookaside Buffer (TLB)

* Cada referencia en el espacio virtual puede causar 2 o mas accesos a la memoria física
  + Uno o más para obtener la entrada en tabla de paginas
  + Uno para obtener los datos
* Para solucionar leste problema, una memoria cache de alta velocidad es usada para almacenar las entradas de paginas
  + TLB
* Contiene las entradas de la tabla e0paginas que fueron usadas más recientemente
* Dada una dirección virtual, el procesador examina la TLB
* Si la entrada de la tabla de paginas se encuentra en la TLB (hit), es obtenido el frame y armada la dirección física
* Si la entrada no es encontrada en la TLB (miss), el número de página es usado como índice en las tablas de páginas del proceso
* Se controla Si la pagina está en la memoria
  + Si no es un Page Fault
* La TLB es actualizada para incluir la nueva entrada
* El cambio de contexto genera la invalidación de las entradas de la TLB

Asignacion de marcos:

* Tamaño del conjunto residente: Cuantas paginas de un proceso se úeden encontrra en memoria
  + Asignacion dinámica: El numero de marcos para cada proceso varia
  + Asignacion fija: Numero fijo de marcos para cada proceso
    - Asignacion equitativa: la misma cantidad para todos los proceoss
    - Asignacion proporcional: Se asigna acorde al tamaño del proceso
      * si = size of process pi
      * S = Suma(si)
      * m = total number of frames
      * ai = frames para pi = (si/S)\*m

Reemplazo de paginas:

Reemplazo de paginas

¿Que pasa si ocurre un fallo de pagina y todos los frame están ocupados? 🡪 se debe seleccionar una pagina victima

¿Cual seria el reemplazo optimo? Que la pagina eliminada no sea referenciada en un futuro próximo

La mayoría de reemplazos predicen según el pasado

Algoritmos de reemplazo

* Optimo: es solo teórico
* FIFO: Es el + sencillo
* LRU: Requiere soporte del hardware para mantener etiquetas de tiempo
* 2da Chance: Una avance del FIFO, beneficia a las paginas mas referenciadas
* NRU: (NON Recently Used) Utiliza bits R(referenciado) y M(modificado), Favorece a las paginas que fueron usadas recientemente. ~R~M ~RM R~M RM

Tipos de reemplazo:

* Reemplazo Global:
  + El fallo de pagina de un proceso puede reemplazar la pagina de cualquier proceso
  + El SO no controla la tasa de page-faults de cada proceso
  + Puede tomar frames de otro proceso, aumentando la cantidad de frames asignados a el
  + Un proceso de alta prioridad podría tomar los frames de un proceso de baja prioridad
* Reemplazo Local:
  + El fallo de pagina de un proceso solo puede reemplazar sus propias paginas
  + No cambia la cantidad de frames asignados
  + El SO puede determinar cual es la tasa de page faults de cada proceso
  + Un proceso puede tener frames asignados que no usa, y no pueden ser usados por otros procesos

|  |  |  |
| --- | --- | --- |
|  | Reemplazo Local | Reemplazo Global |
| Fija | * Nro de frames para un proceso es fija * La pagina a ser reemplazada es elegida entre los frames asignados a ese proceso | NO es posible |
| Dinamica | * El numero de frames asignados a un proceso puede cambiar para mantener el trabajo de los procesos * La pagina a ser reemplazada se elige entre los frames asignados en ese momento al proceso | La pagina a ser reemplazada se elige entre todos los frames disponibles en memoria. Esto causa que la cantidad de paginas de los procesos varie |

Políticas en el manejo de MV:

Política de

Por demanda

Prepaginacion

Política de ubicación

First fit, best fit…

Política de reemplazos

LRU, LFU, …

Asignacion de marcos

Cuantas paginas de un proceso se pueden encontrar en memoria

Asignacion fija

Asignacion equitativa: si tengo 100 frames y 5 procesos, 20 para cada proceso.

Asignacion proporcional: Se asigna acorde al tamaño del proceso.

Asignacion dinamica

Demonio de paginación: se encarga de revisar la RAM y de agrandar o recortar los frames de cada proceso.

ENTRADA/SALIDA

Responsabilidades del SO:

* Controlar dispositivos de E/S
  + Generar comandos
  + Manejar interrupciones
  + Manejar errores
* Proporcionar una interfaz de utilización

Problemas

* Heterogeneidad de dispositivos
* Características de los dispositivos
* Velocidad
* Nuevos tipos de dispositivos
* Diferentes formas de realizar E/S

Generalidad

* Es deseable manejar todos los dispositivos de I/O de una manera uniforme/estándar
* Ocultar la mayoría de los detalles del dispositivo en las rutinas de niveles mas bajos para que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock;

Planificación

* Organización de los requerimientos a los dispositivos

Diseño de capas:

* Capa de usuario:
  + Librerias de funciones
    - Premiten acceso a Syscalls
    - Implementan servicios que no dependen del kernel
  + Programas de apoyo
    - Demonio de impresión (spooling)
* Software independiente SO
  + - Interfaz uniforme
    - Manejo de errores
    - Buffer
    - Asignación de recursos
    - Planificación
  + El Kernel mantiene la información de estado de cada dispositivos o componente
    - Archivos abiertos
    - Conexiones de red
    - Etc
  + Hay varias estructuras complejas que representan buffers, utilización de la memoria
* Controladores de dispositivos (Drivers)
  + Contienen el código dependiente del dispositivo
  + Manejan un tipo de dispositivo
  + Traducen requerimientos abstractos en los comandos para el dispositivo
    - Escribe sobre los registros del controlador
    - Acceso a la memoria mapeada
    - Encola requerimientos
  + Comúnmente las interrupciones generadas por los dospositivos son atendidas
  + Interfaz entre el SO y el Hardware
  + Forman parte del espacio de memoria del Kernel
    - En general se cargan como módulos
  + Los fabricantes del HW implementan el driver en función de una API
  + .
* Gestor de interrupciones
  + Atiende todas la interrupciones del HW
  + Deriva el driver correspondiente según interrupción
  + Resguarda información
  + Independiente del driver

Reemplazo de paginas

¿Que pasa si ocurre un fallo de pagina y todos los frame están ocupados? 🡪 se debe seleccionar una pagina victima

¿Cual seria el reemplazo optimo? Que la pagina eliminada no sea referenciada en un futuro próximo

La mayoría de reemplazos predicen según el pasado

Algoritmos de reemplazo

Optimo: es solo teórico

FIFO: Es el + sencillo

LRU: Requiere soporte del hardware para mantener etiquetas de tiempo

2da Chance: Una avance del FIFO, beneficia a las paginas mas referenciadas

NRU: (NON Recently Used) Utiliza bits R(referenciado) y M(modificado), Favorece a las paginas que fueron usadas recientemente. ~R~M ~RM R~M RM

Alcance del reemplazo

Reemplazo local:

El fallo de pagina de un proceso solo puede reemplazar sus propias paginas

No cambia la cantidad de frames asignados

El SO puede determinar cual es la tasa de page-faults de cada proceso

Un proceso puede tener frames asignados que no usa y no pueden ser usados por otros procesos

Reemplazo global:

MEMORIA 3

Hiperpaginacion (Trashing)

* Un proceso esta en trashing cuando pasa mas tiempo paginando que ejecutando procesos
* Hay una baja importante en la performance

Ciclo del trashing:

1. El kernel monitorea el uso de la CPU
2. Si hay baja utilización 🡪 aumenta el grado de multiprogramación
3. Si el algoritmo de reemplazo es global, pueden sacarse frames a otros procesos
4. Un proceso necesita mas frames. comienzan los page-faults y robo de frames entre los procesos
5. Por swapping de paginas y encolamiento en dispositivos, baja el uso de la CPU

El scheduler de CPU y el trashing

Control del trashing

* Se puede limitar el trashing usando algoritmos de reemplazao local
* Con este algoritmo, si un proceso entra en trashing no roba frames a otros procesos
* Si bien perjudica la performance del sistema, es controlable

Conclusión sobre el trashing

* Si un proceso cienta con todos los frames que necesita, no habría trashing
* Una manera de abordar esta problemática es utilizando la estrategia de Working Set

Administración de archivos 1

Que es un archivo?

Objetivos del SO en cuanto a la administración de archivos

* Cumplir con la gestión de datos
* cumplor con las solicitudes del usuario
* Minimizar/eliminar la posibilidad de perder o destruir datos
  + Garantizar la integridad del contenido de los archivos
* Dar soporte de E/S a distintos dispositivos
* Brindar un conjunto de interfaces de E/S para tratamiento de archivos

Archivos regulares:

* Texto Plano:
  + Source File
* Binarios
  + Object File
  + Executable file
* Pipe (NO regular?)

Directorios:

* Archivos que mantienen la estructura en el filesystem

Atributos de un Archivo

* Nombre
* Identificador
* Tipo
* Localización}
* Tamaño
* Protección, seguridad y moitoreo
  + Owner, Permisos, Password
  + Momento en que el usuario lo modifico, creo, accedió por ultima vez
  + ACLs (Access Control List) : Sirven para controlar los permisos de un archivo con mayor especificación sobre los grupos que creo(Ej no dar permisos de escritura a un miembro en especifico de un grupo)

Diferencia entre protección y seguridad

* protección: proteger el archivo internamente
* seguridad: Proteger ante la posibilidad de ataques externos

Directorios:

* Contiene información acerca de archivos y directorios que están dentro de el
* El directorio es un archivo
* Interviene en la resolución entre el nombre y el archivo en si mismo
* Operaciones:
  + Buscar un archivo
  + Crear un archivo
  + Borrar un archivo
  + Listar el contenido
  + Renombrar archivos

BufferCache: área de memoria donde ese guardan los bloques recientemente referenciados

…

Metas del sistema operativo

* Brindar espacio en disco a los archivos de usuario y del sistema
* Mantener un registro del espacio libre; Cantidad y ubicación del mismo dentro del disco

Conceptos:

* Sector
  + Unidad de almacenamiento utilizada en los discos rigidos
* Bloque/Cluster
  + Conjuntos de sectores consecutivos
* File System
  + Define la forma en que los datos son almacenados
* FAT (File Allocation Table)
  + Contienen información sobre en que lugar están alocados los distintos archivos

Buffer Cache

Los bloques cuando se traen a memoria se ponen en una hashqueu

cabecera de buffer (tamaño fijo)

* 5 punteros:
  + 2 punteros uno al inicio y otro al final de la hashqueu
  + 2 punteros uno al inicio y otro al final de la free list
    - relaciona aquellos buffer que no están siendo usados por ningún proceso
    - buffer libre: un buffer que no esta siendo usado por ningún proceso
    - buffer ocupado: esta siendo usado por uno o varios procesos
    - bufer de escritura retardada: son los buffer que están marcados como modificados (delayed write) puede estar libre u ocupado
  + 1 puntero al buffer

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Null |  |  |  | HQ0 | 12 |  |  |  |
|  | 12 |  |  |  | 16 | Null |  |  |  |
|  |  |  |  |  | Null | Null |  |  |  |
|  |  |  |  |  | Null | Null |  |  |  |
|  |  |  |  |  | Bloq12 | Bloq16 |  |  |  |
| Hfreelist | HQ0 | HQ1 | HQ2 | HQ3 | Hblo12 | Hblo16 |  |  |  |

buffer

todo buffer esta en una sola HQ, si esta libre también esta en la freelist

cuando un proceso libera un buffer siempre se pone al final de la freelist

El momento en que un buffer se lleva a disco es cuando esta primero en la free list, con DW y se requiere una cabecera nueva

1er escenario:

* El bloque esta en la hash queu y esta libre
* Se saca de la free list

2do escenario:

* El bloque no esta en la hash queu
* se debe buscar un bloque libre para usar la cabecera del nuevo bloque pedido
* se actualizan los punteros y se ubica en la HQ correspondiente

3er escenario:

* El bloque no esta en la HQ
* se debe buscar un bloque libre pero el primero de la freelist esta marcado como DW
* se busca el segundo de la freelist y así sucesivamente hasta encontra uno que no este marcado como DW todos los DW por los que paso se guardan en disco, y siguen ubicados en el principio de la freelist una vez que terminen de guardarse

4to escenario:

* El bloque no esta en la HQ
* La free list esta vacia
* El proceso debe esperar a que se libere (sleep) cuando despierta debe revisar nuevamente si el buffer esta en memoria

5to escenario:

* El bloque esta en una HQ pero esta siendo usado por otro proceso
* El proceso debe esperar a que se libere el buffer para poder usarlo