S.O. Clase 2

Virtualización de la CPU

Dar la impresion de que cada proceso tiene un CPU independiente para realizar este.

Que es un Proceso

Un programa en ejecución, es decir recidente en memoria.

Espacio de direcciones de un proceso

- Conjunto de direcciones en memoria accesibles por el proceso
- El tamaño del espacio depende de la arquitectura de la cpu

Estados de un proceso

Un proceso puede tener varios estados

- New : cuando es recien cargado.
- Ready : cuando es incorporado, es decir esta esperando ser ejecutado por la cpu en un thread.
- Running : Las instrucciones del proceso estan siendo ejecutadas, es decir el S.O. hace un scheduler dispatch, es decir deja de estar ready y pasa a estar runnning.
- Interrupt : cualquier I/O puede causar una interrupción, por ejemplo al abrir un archivo en c el proceso main es interrumpido.
- Waiting : El proceso esta esperando que algun evento ocurra, por ejemplo recepcion de una señal.
- Terminated : El proceso termina su ejecución

Un procesador solo puede tener un proceso en ejecución a la vez, varios procesos pueden estar en estado "listo" o "en espera"

Process Control Block (PCB)

El PCB contiene varios datos que describen el estado del proceso. - Estado del proceso. - Program counter : indica la direccion de la siguiente instruccion que sera ejecutada por el proceso - Registros de la CPU : Dependen de la arquitectura de la CPU. Cuando ocurre una interrupción, los registros y el program counter son respaldados por el S.O. para reanudad el proceso cuando la interrupción es atendida. - Información de planificación de CPU : Prioridad del proceso. - Contavilidad: Cantidad de la CPU utilizada.

Multiplexación de la CPU

- Con multiprogramación el S.O. mantiene varios proceso ejecutando en memoria principal.

- Solo un proceso esta en estado running a la vez.
- Generalmente el tiempo de CPU asignado a un proceso (quantum de tiempo) puede ser corto en la percepcion.
- Alternación entre varios procesos es lo que nos da la nocion de estar ejecutando mas de un programa a la vez.

Cambio de CPU entre procesos (Context Switch)

- Cambio de CPU entre procesos se conoce como "cambio de contexto"
- Los PCB guardan la informacion de estado de los procesos. Esta info es para los cambios de contextos.
- Los cambios de contexto se pueden gatillar como resultado de una interrupción, por ejemplo, un timer.

Planificación de Procesos

- Uno de los objetivos de la multiprogramación es "mantener la CPU ocupada" de la forma mas eficiente posible.
- El tiemp de la CPU es compartido por muchos procesos, que pueden ser origidados por uno o más usuarios.
- Los componentes del S.O. que toman las decisiones de asignación de CPU a los procesos son los planificadores de procesos.
- Un proceso migra entre las distintas colas del S.O. durante su ciclo de vida.
- El S.O. debe elegir los procesos a ejecutar en la CPU bajo ciertos criterios.
- El proceso de selección es realizado por un planificador específico.

La naturaleza de los planificadores esta en función de tiempo. - Planificador de corto plazo. - El despachador de corto plazo toma decisiones muy frecuentemente. - Planificador de medio plazo. - Encargado de decidir cual es el siguiente proceso que sera cargado a memoria fisica, administra el SWAP (Cambio). - Planificador de largo plazo : tiene sentido en un sistema de procesamiento de lotes (batch) - Procesos listos se mantiene en un pool en almacenamiento masivo. - Hablamos de minutos.

Colas de Planificación

- A medida que los procesos son creados en el sistema, entran en una cola de procesos, la cual contiene todos los procesos en el sistema.
- Los procesos que residen en la memoria principal y que están listos y esperando ejecución so mantenidos en la "cola de procesos listos"
- Un proceso nuevo entra inicialmente en la cola de los listos. Espera ahí hasta que es elegido para ejecutar(despachar).
- El S.O. mantiene también una cola de dispositivos.

¿ Que pasa cuando un proceso es despachado ?

- El proceso podría requerir I/O.
- El proceso podria crear un nuevo subproceso y esperar la terminación del subproceso.
- El proceso podria ser sacado de la CPU forzosamente por el S.O. como resultado de una interrupcion y ser puesto por el S.O. en cola de los listos.
- En los dos primeros casos, el proceso eventualmente terminara su espera y volvera a los listos.
- Un proceso se mantiene en esta rotacion de estados hasta que es terminado por el S.O.

Procesos pueden ser acotados por CPU o por I/0 - Acotado por CPU: El mayor tiempo es utilizado en operaciones de computo. - Acotado por I/0: El mayor tiempo es utilizado en operaciones de I/0

Operaciones de Procesos

- Un proceso puede crear varios procesos nuevos, mediante una llamada al S.O. durante la ejecución.
- El proceso *creador* es llamado "proceso madre". Cada proceso nuevo es un "proceso hijo". Cada proceso hijo puede tambien iniciar procesos hijos, con lo cual se conforma un "arbol de procesos".
- La mayoria de los S.S.O.O. identifican cada proceso con un identificador de proceso (process identifier, PID), el cual es por lo general, un valor entero.

Creacion de procesos.

- Cuando un proceso crea un nuevo proceso, existen dos posibilidades de ejecucion:
  - El proceso padre continua ejecutando con sus hijos.
  - El proceso padre espera a que los hijos terminen.
- Hay tambien dos posibilidades con respecto al espacio de direccionamiento de un nuevo proceso:

Fork()

Es una funcion de Unix que se usa para generar duplicados de un proceso particular. - Return 0, es proceso hijo. - Return < 0, no se creo el proceso. - Return 0 >, es proceso padre con PID del hijo.

Terminación de Procesos

- Un roceso padre puede terminar un proceso hijo por varias razones:
  - El hijo a excedido el consumo de recursos. En este caso, el proceso padre debe contar como un mecanismo para inspeccionar el status del proceso hijo.
  - La tarea asignada al hijo ya no se requiere.
  - El proceso padre esta terminando y el S.O. no permite a los procesos hijos continuar si el proceso padre termina.
- Terminación en cascada: Si el proceso padre termina (en forma normal o anormal), los procesos hijos son terminados por el S.O.

Procesos Livianos (Threads)

- Proceso: Es una abstraccion del SO utilizada para respresneta rtodo lo que se requiere para ejecutar un programa.
  - Tiene dos componentes:
    - Al menos un flujo de ejecucion secuencial principal.
    - Recursos protegidois: estado de memoria principal y estado de IO.
- Flujos de ejecucion de un proceos: puede ser uno solo o varios concurrentes.
- Nos vamos a referir como proceso liviano o thread a un flujo de ejecucion que vive dentro de un proceso.
- Un proceso puede contener uno o mas threads.
- Threads encapsulan concurrencia.
  - Son el componente "activo" de un proceso.
- Los espacios de direcciones encapsulan protección de memoria.
  - Previenen que procesos defectuosos "boten" el sistema.
  - Son el componente "pasivo" de un proceso.
- Los threads tiene stacks independientes epro comparten el heap del proceso padre.

Ventajas de utilizar Threads

- Mejorar la respuesta del sistema al usuario: Especialmente en las palicaciones interactivas con interfaz de usuario.
- Facilidad de recursos compartidos: Los procesos solo pueden compartir recursos a travéz de tecnologías de IPC como la memoria compartida y el paso de mensajes, estos mecanismos deben ser configurados por el programados.
- Economia: La creacon de procesos es mas costosa que la creacion de un thread, ademas los threadss son mas rapidos para alternar que un proceso.
- Escalabilidad; se pueden aprovechar mas las potencialidades de una arquitectura de CPU.

Modelos de Threads

- El soporte para threads puede implementarse en espacio de usuario o en espacio de kernel del SO
- Threads en espacio de usuario: user thredds, y en espacio de kernel: kernel threads.
- Los user threads no son manejados por el kernel del SO, sino por bibliotecas que ejecutan en espacio de usuario.
- Practicamente todos los sistemas operativos modernos proveen soporte para kernel threads.
- Sea la implementacion de threds a nivel de usuario, siempre los user threads tendran una relacion con los kernel threads:
  - Modelos N a 1
    - Implementa multiples user threads asociados a un kernel thread
    - La administracion es realizada por una bilbioteca de threads en espacio de usuario.
    - El proceso completo se bloquea si un thread realiza una llamada bloqueante al sistema.
  - Modelos 1 a 1
    - A cada user thread le corresponde un kernel thread.

        - Permite mayor concurrencia que el modelo muchos a uno.
    - Modelos N a N
        - El programador puede crear tantos user threads como se requiera.

Estado de Threads

- Todos los threads en un proceso comparten:
    - El contenido de la memoria (variables globales, heap).
    - El estado de IO.
- Existe informacion de estado privada para cada thread.
    - Se mantiene en un Thread Control Block (TCB)
        - Registros de CPU
        - Stack de ejecucion
        - Informacion de planificacion, prioridad, tiempo de CPU.

Ciclo de vida

Es el mismo ciclo que el ciclo de procesos.

Creacion y terminacion de threads

- Semantica de creacion y terminacionde threads es analoga a la creacion y terminacion de procesos.
- Un thread puede crear threads hijos y esperar su terminacion.
    - La operacion join se utiliza tanto en el contexto de procesos como el de thread.
- Los thread hijos pueden ser cancelados por el padre.

Bibliotecas de Threads

- Puede ser implementada a nivel user threads.
- Puede ser implementada a nivel de kernel threads.
- Pthreads es una biblioteca para C/C++ que maneja threads.