Arquitectura se enfoca principalmente en hardware, aquí se ve el procesador, administrar memoria, jerarquia memoria, cache, l1, l2, io.

El hardware es construido sabiendo que tenemos memoria, procesador, io, dispositivos, buses, sistema de itnterrupciones. Estos hardware se construyen tal que sobre el hay un sistema operativo.

Tal que sistema operativo es uun software.

Linux es 95 por ciento en c. el otro 4 en lenguaje de maquina. Ese pedacito es el que se comunica con hardware. Lo otro esta en alto nivel.

Este software es la primera capa que corre sobre el hardware.

El hardware tiene servicios. Uno de estos son las interrupciones. Sst op y hardware se comunican a traves de las señales que son interrupciones, tal que se envian señales y cuando, tal que tienen significados de comunicación.

Sst op administra recursos de hardware, tal que administra el o los procesadores, memoria de computador y dispositivos io, tal que son recursos de hardware. El sst op los administra para dar servicios a los procesos. Por sobre sst op tendremos servicios usuarios, por ejemplo la app zoom, tal que corre como proceso, lo otro puede ser un browser, compilador, todos estos piden servicios a sst op, los que son por ejemplo necesito abrir archivo, ejecutarme, memoria, tal que los procesos piden recursos, tal que se los piden a sst op. Y esto es a traves de llamados a sistema. Tal que el sst op da servicios y los procesos acceden a modulos de sistema operativo que proveen servicios. Tal que estos son llamados a sistema. System call.

Sea por ejemplo p=malloc(sizeofint) tal que procura memoria. Un proceso usuarios solicita servicios para obtener recursos. El sst op los administra. El sst op es la interfaz para solicitar, tal que hace una interrupcion para pedirle recursos a hardware.

Sst op administrador que da recuros a procesos a traves de llamados a sistema.

Se puede abrir un terminal xterm donde puedo abrir para un comando. Aquí hay varios procesos, servidor x, que es el que administra espacios de pantalla y luego tiene un administrador de ventanas, este va a tener un xterm y este va a tener un bash. Entonces sea por ejemplo que en el terminal escriba ls, no estoy escribiendole a sst op, sino que es un proceso para llamarlo, un proceso usuario. Entonces bash es interprete de comandos. El xterm es el que maneja una ventana, luego esta el administracion de ventana que seria el gnome o de la gui, y el servidor es el que administra espacios en la pantalla, entonces aquí hay varios procesos que se construyen uno sobre otro que se van pidiendo permiso de forma piramidal, el servidor x le pide servicios a sst op y sst op hace una interrupcion para hardware.

En ese minuto sst op llama a hardware. Dentro de sst op estan los drivers. Drivers y controllers son diferentes aunque en español se dicen igual. Los drivers son la parte de sst op que se comunica con controllers y controller es el dispositivo hardware.

Sea un disco, este tiene dentro un controlador asociado, este controlador es la inteligencia en hardware del dispositivo y para comunicarme se usa el driver. Tal que son diferentes, tal que driver es el modulo que se comunica con el controlador.

El termino nucleo o kernel.

Que incluye y que no incluye el kernel?

El kernel es la parte mas basica del sistema operativo. Pero el kernel no tiene componente hardware. El kernel es lo fundamental del sistema operativo, tal que dentro de kernel estan los servicios y funcionalidades basicas para que funcione la maquina. No hacemos con este mucha diferencia con el sst operativo, lo tomaremos como simil.

Todo el sistema operativo es kernel, tal que el sistema operativo incluye todos los servicios del sst op. Otros por ejemplo serian dejar fuera el sistema de archivos, pero estos son diferentes tipos de kernel. Tal que puedo dejar servicios, tal que quedan como serivicio usuario, tal que lo instalo o corro como proceso este servicio, no como kernel o sistema operativo.

El que se aparte o no es debatible y sujeto a desiciones de desarrolladores.

Hubo una vez que desarrolladores se querellaron contra windows, sea que windows venia con el proceso de browser, pero este estaba puesto como parte del sistema operativo y no podia ser borrado, tal que browser estaba en el kernel. Se llamaron a expertos. la comunidad gano.

Antes microsoft no permitia instalar otro browser porque tenia su propio que estaba dentro de kernel.

Veremos los recursos basicos de sst op, admin de memoria, de procesadores, archivo.

Kernel y sistema op sera lo mismo según estos conceptos.

Repaso de arqui:

Procesadores, memoria, io. Esto es lo mas basico de sistema computacional. El termino cpu es el mismo de procesador. Tendre la memoria principal ram, y todos los dispositivos io, tendre el disco duro como mem2, interfaz de red, monitores, monitores fisicos de io. Todos se comunican a traves de los buses, que son de hardware, donde viajan las señales. Dentro del procesador tenemos el archivo de registros, el program counter, la alu, memoria cache l1. Fuera de procesador, y dependiendo del hardware, tiene la l2, dependiendo si va dentro o fuera del chip.

Registros de proposito general, al programar en assembler, para almacentar enteros, string, stack, puntero, con los que uno programa. Hay registros que no son visibles y no pueden ser ocupados. Estos son program counter que tiene la dir de la instrucción que se va a buscar y el instruction register almacena la instrucción. Entonces tendre direcciones de memoria para almacenar datos. Otro es el program status word que es el psw.

Intel tiene nombres de registros tipicos. Como eax que acomula operandos y resutlados de calculos, ebx genera direccion base de arreglos tal que a(i)=8 almacena el head del array.

Ecx es para los loops, tal que se utiliza un registro de procesador especifico para hacer esto.

Los que si se pueden manipular son los punteros a la tope y la base del stack.

Uno de los registros mas importantes es el psw, que en intel se llama eflags, este registro es el que contiene status del proceso que se esta ejecutando actualmente. Entonces este es un registro de 32 bits, cada uno de estos me dice algo sobre lo que ocurre en procesador.

Entonces sea pr ejemplo f =g/h, tal que h es 0, va a ocurrir un error. El proceso no puede continuar, entonces el bit overflow de este registro se va a encender, que hay overflow, tal que activa o inactiva sea lo que suceda. Supongmaos una operación aritmetica que dice que valor es negativo, entonces la flag va a pasar de 0 a 1, y si es zero tambien hay un bit especifico, entonces se van a prener bits que hablan sobre que pasa con la respuesta. Tambien hay modos, tal que esta ejecutandose en modo virtual o no. Y tambien hay un bit que explica si proceso permite o no interrupciones, tal que si esta encendido, entonces no va atender interrupciones o problemas. Entonces va guardando informacion sobre el status del proceso en procesador.

Estamos hablando de registros de hardware para sst op.

La tabla de interrupciones es donde se instalan todas las interrupciones de sistema donde estan los manejadores de interrupciones. Entonces sea que sst op avisa a hardware a traves de system call interrupciones, debe atender a llamados a sesta en el registro iddr, que apunta a la tabla de interrupciones.istema especificos que estan especificados en la tabla de interrrupciones, tal que es una tabla instalada en memoria,

Sea un codigo en assembler, puedo pedir el codigo en assembler al escribir –s.

Veremos los registros que hemos hablados: manipulacion de stack, de string, registro del stack, etc. No vamos a ver el psw, el eflags o pc, porque no tenemos acceso a eso. Hay un llamado a write, que es un system call que llama a sistema operativo del proceso. Necesito un serivicio de sistema operativo. Todo lo otro lo hace proceso usuario.

Entonces entro al kernel, donde esta el servicio de imprimir en pantalla.

Estamos conectando hardware con software. Este codigo al ejecutarse se carga a memoria, se trasforma a lenguaje de maquina, se carga en memoria y se procede al proceso ciclo fetch, tal que va a memoria a buscar informacion. Fetch, decode, memor execute y write back. Entonces decodifico lo que necesito, busco los operandos, los ejecuto y lo guardo el resultado. Esto nos ayuda a entender lo que sucede en un procesador al ejecutar codigo.

El procesador lo que hace es este ciclo fetch y luego al terminar un ciclo aumenta el pc. Es necesario si esque este ciclo no termina, puedo interrumpirlo.

Cada vez que hace un ciclo, va a preguntar si hay que atender una interrupcion.

Sea divicion por 0 un error, va a ocurrir que se va a prener el bit de overflow. Esto indica al procesador que cuando se checkea va a darse cuenta del error, no va a seguir ejecutando el codigo, debe matar el proceso, entones entra a otro proceso de matar este proceso. Entonces esto va a ser un mecanismo para detener lo que ocurre en el ciclo, tal que llegan a traves de señales, io, mismo programa de ejecucion.

Prpcesador checkea y sirve la interrupcion de io- la interrupcion es una señal, llega.

Clase 3:

Interrupciones esten o no habilitadas: al llegar interrupcion las checkea o no, puede que llegue pero no las checkea y sigue con el proceso. El proceso checkea al finalizar la interrupcion, que es el servicio de la interrucion, atiende la interrupcion. Hay diferentes formas de atender la inerrupcion, que es lo que el sst op hace, que lo decide el sst op.

Interrupciones de io en intel. Esquema que grafica que dentro de placa madre. El bus conecta a los dispositivos con el cpu. Lo que representa es cuando el io quiere comunicarse con procesador, entonces se envia una señal al procesador.

El bus conecta io.

Cuando io quiere comunicarse con procesador, manda señal que es programable interrupt controller que son irq, que son a su vez interrupt request. Todos tienen un numero. Cada uno de los dispositivos tiene un irq asignado. El peak dice le voy a mandar al procesador una señal de interrupcion y le voy a deciri de donde es la interrupcion, ocurre una interrupcion y por el otro canal le mando informacion sobre la interrupcion, que irq es, y esta viaja y va al procesador. Llega al procesador y si interrupciones estan habilitadas, al ifinalizar cualquier proceso ve si hay interrupciones pendientes, tal que ve las que hay sea cualquier irq, entonces hay que ir a atenderla o servirla, tal que ahí ocurre la interaccion sistema operativo y hardware.

El inta es el aknowloadgement del procesador, sea que lee el irq, de vuelta manda el reconocimiento de que ha leido la interrupcion para que sea borrada, de que ya fue atendida.

Entonces, eso es anivel de hardware.

A nivel de software tengo un proceso en su ciclo fetch ejecutandoce. Cuando termina una ejecucion de instrucción, va revisando por interrupciones. Para nosotros excepcion y interrupcion seran lo mismo por ahora. Al descurir una interrupcion, va a haber un cambio yo salto al manejador de interrupcion que es el exception handler. este manejador pertenece a el sistema operativo.

Las interrupciones pueden llegar en cualquier momento, solo cuando termina un proceso es que checkea si hay interrupciones pendientes para servirlas.

Entonces sea que tengo un proceso y hay una interrupcion en el ciclo fetch, entones va a haber un cambio, un salto donde va a ir a manejador de interrupciones que es un modulo de sistema operativo que atiende interurpciones. Comienza el ciclo fetch de procesamiento de interrupciones. Cuando este termina, puede retornar a donde habia quedado en el proceso y puede seguir con su ejecucion. Entonces una interrupcion hace un cambio del flujo de control, inicialmente en el proceso y luuego en el handler de interrupciones, tal que habria un salto hacia el kernel, la interrupcion me hace entrar al kernel donde esta el manejador de interrupciones.

Cada interrupcion tiene un numero, hay algunos irq que estan predefinidos y otros que estan libres ys on definidos por sistema operativo.

Eso significa en algunos casos eliminar procesos, errores de procesos de memoria con segmentation fault.

Las de sistema operativo las usa para sus system call o para drivers. Algunos de estos irq los designa a dispositivos io, cada uno va a tener un irq, cada irq va a tener asociado un driver. Por ejemplo cuando compro tarjeta grafica debo bajar drivers especificos para el sst op, que deben ser asignados y cargados a memoria, para saber como usarlo cuando generan interrupciones.

Cuando el procesador ve una interrupcion, hace un jump a manejador de interrupciones especifico para la interrupcion, al finalizar puedo hacer 3 cosas: si interrupcion fue un error, el programa debe abortarse, no puedo retornar a programa que estaba ejecutando. Entonces lo que el procesador puede hacer es ir a otro proceso. Esto significa a su vez retornar ejecucion de otro proceso. Una vez que termine de procesar, yo puedo seguir procesando otras cosas. Otra opcion es retornar a instrucción actual, esto es una interrupcion recuperable. Por ejemplo al ingresar a seccion de memoria que aun no esta definido, entonces este va a ser un page fault. La atencion va a ser ir a buscar la informaciony puedo retornar al proceso el dato que estaba en memoria. Luego se sigue a la siguiente interrupcion.

Es importante enfatizar que la deteccion y transferencia de control es realizada por el procesador o hardware, y el manejo de la interrupcion es por software.

Sea que instalamos un driver, al instalarlo, el sst op lo pone en alugna parte en memoria. Ese driver es el codigo para manejar la interrucion asociada al irq del dispositivo. Entonces van a haber tablas de interrupciones.

Una de las cosas al prender un computador es instalar la tabla de interrupciones, indica las direcciones en memoria para cada interrupcion, tal que es una tabla de punteros, tal que cada unoo de estos es un driver. Yo debo decirle al hardware donde instale esta tabla, esa direccion de la tabla es el registro base de la tabla de interrupciones. Cuando el sst op se prende, instala la tabla y carga al registro del procesador asociado, la direccion en donde esta la tabla. Este registro lo vimos y es idtr que es intel en el caso. El sst op escribe esto con la tabla que tiene los punteros a codigo. Cuando llega una señal de interrupcion, como hace el control de jump? Porque tiene la tabla que puede dirigirlo. Dado el numero, entonces se a donde voy para atender el codigo al kernel. Esto es el inicio del servicio de atencion de interrupcion.

Este es el mecanismo basico para que hardware hace que sst op vuelva a ejecutarse en procesador.

Recalcar que en instante cuando se esta ejecutando proceso usuario, el sst op esta en memoria, no esta controlando nada. Cuando proceso usuario se ejecuta el sst op no esta haciendo nada. Solo puede tomar desiciones cuando se esta ejecuntando en procesador y son mutuamente exclusivos, este cambio de control lo hace el hardware, y entonces el hardware hace un jump indirecto al manejador de interrupciones, tal que el procesador ejecuta a donde hizo jump.

Se pueden dividir procesos sincronos y asincronos.

Cuando pongo alarma le doy pido servicio a sst op y este a su vez le pide permiso a hardware. Entonces hardware manda la interrupcion diciendo “es la hora que me pediste” sst op recepciona la informacion y asi se ejecuta el proceso. Esto son señales asincronas.

Las sincronas, son llamadas a sistema, utiliza una interrupcion que le llamaremos trap, que es interrupcion producida por software.

Todas las otras son producidas por mismo procesador o relojes, pero si un sistema tiene un call supongamos, y esto es un llamado a sistema operativo, esto va a utilizar el software, que seria un trap.

Intel llama a todas interrupciones excepciones asincronas y exepciones a las sincronas.

Interrupciones son señales que llegan a procesador, eso es lo unico que importa.

Para cada procesador ocurren interrupciones, tal que nombra en proc/interrupt todos los irq y las veces que ha sido activado.

Al apagarlo se resetea. Hay irq que tienen que ver con temperatura, eventos, interrupciones de io, function call interrupts que son los traps, tal que llaman a system call. Importante local timer interrupt, este es el que mas veces ha ocurrido. Interrupcion generada por reloj, ese reloj manda varias veces por segundo la informacion, como un tren de interrupciones.

Manejo de interrupciones: la unica forma de tomar el control de sistema es que interrupcion le vuelva a dar el paso.

Una operación de io ha finalizado, tal que estaba leyendo en disco. El controlador envia la señal interrupcion al procesador. Controler es el dispositivo hardware de dispositivo y el driver es

El manejoador de interrupciones. Entonces el controlador envia la señal que es issue, el procesador checkea por interrupciones: aknowloadgemente. Entonces carga en el stack el psw y el program counter. El procesador sabe donde esta el stack y el procesador lo guarda, esto lo hace el procesador no el sistema operativo. Guarda estas cosas porque los registros van a ser sobreescritos por el manejador de interrupciones,es otro contexto. Por lo que guardo el contexto en el stack. Una vez que hace esto, carga el pc con el nuevo valor con la direcion dada por el idtr, que es la direccion de memoria para el manejador de interrupciones para correr el codigo para interrupciones. El program counter comienza el ciclo fetch de la instrucción especifica del manejador de interrupciones.

Ahora, por software el sistema operativo manejador de interrupciones se van a cargar os siguientes registros para el contexto. Una vez que esto esta cargado, se guarda/salva y comienza a ejecutar este proceso. Una vez que termina la interrupcion, puede restaurar el estado del proceso que habia suspendido. Al hacer el jump ese proceso habia sido suspendido. Entonces obtengo los registros del stack, los cargo al procesador, y es como si volviera al proceso donde lo habia dejado, lo reanudo.

Muchas labores son realizadas por procesador y unica forma de que vuelva control a sst op es que procesador me de el paso cargando en program counter el manejador de interrupciones para procesar.

Esto de guardar cosas en el stack es muy importante. Como guardo als cosas en stack para interrupciones? El programa se ejecuta y ocurre una interrupcion luego de la instrucción n. tal que checkeo si ay interrupciones despues de una interrupcion. Por hardware los registros que estaban en procesador, el program counter, se hace un push al stack de con el pc y el psv. En general a los registros del procesador se hace un push al stack que maneja el sst op. El hardware y el software lo que hacen es guardar el estado del proceso, sus registros. Una vez que ocurre esto yo se donde debo saltar, cargo el nuevo contexto, con la direccion de inicio del manejador de interruptores. Tal que coloco la nueva direccion y ahora comienza el ciclo fetch con la direccion de inicio del manejador de interrupciones.

Una vez que termino, vuelvo y restauro los contenidos, tal que cargo la interrupcion n+1. Y ahí sigue el ciclo fetch de ese programa.

Hay 2 metodos de sst op. Tal que puede ser proceso o no.

Que pasa cuando estoy atendiendo una interrupcion y llega otra: en el caso puede ser que estoy en proceso y hay interrupcion, termina intstruccion asi que voy a la interrupcion, en la interrupcion recibo otra interrupcion y al terminar una instrucción de la interrupcion salto a la segunda interrupcion si es que es mas prioritaria, sino esperaria a que terminara la primera interrupcion, vuelvo al proceso y ahí voy a la interrupcion segunda. Tal que dependiendo de la prioridad y urgencia, van a ser mas perentorias de hacer, como guardar cosas si es que computador se esta apagando.

Una interrupcion no mascareable no puede ser ignorada por el procesador, tiene muchisima prioridad. Una mascareable si puede ser ignorada, puedo dejarla para despues. Las interrupciones mascareables intr pueden desabilitarse seteando en 0 el bit de eflags de interrupciones. El registro 9 de estas si ese bit esta en 1 significa que interrupciones estan permitidas. Es decir, cuando procesador termina de checkear ultima interrupcion, pregunta si esta permitido, y dependiendo de esto permite o no permite interrupciones esta es la instrucción de assembler cli y sti para 0 y 1 respectivamente. Las no mascareables no pueden ser deshabilitadas.

Trap es una interrupcion intencional, que es un llamado a sistema. Las interrupciones como son por hardware, son privilegiadas. Uno como usuario no puedo simplemente mandar las interrupciones que quiero. Las interrupciones como son por hardware permiten hacer cmabios en los privilegios de ejecucion. Cuando el hardware ve que viene un trap, puede cambiar los privilegios. Quiero cambiar los privilegios ya que voy a ingresar al kernel, a los servicios de sst op. Por ejemplo call open, tal que debe ser kernel que es privilegiado y debo crear un trap.

Fault es error potencialmente recuperable. Un abort es error no recuperable, tal que debo eliminar al proceso que lo produce. Las interrupciones vienen de multiples partes, como se puede ver, hay muchas interrupciones que se gatillan a cada rato.

Clase 4:

Objetivo de ejemplo es mostrar utilidad que tiene el uso de interrupciones para utilizacion de procesador.

U= tiempo utilizado en ejecutar instrucciones de procesos usuarios/ tiempo total

Todas estas instrucciones son de procesos usuarios y el timepo que se demoran. Cuando ingreso al kernel se agrega al tiempo total, tal que se obtiene el porsentaje de ocupacion.

Supongamos una utilizacion de 0.5 seria que la mitad del tiempo estaria en procesos usuarios y la otra mitad de procesos del kernel. En el computador la utilizacion es muy baja, porque mucho tiempo del procesador es del kernel. Mientras mas tiempo me dedique al kernel, mas baja es mi utilizacion. Como la medida del rendimiento del procesador se mejora con las interrupciones.

Sea que tenga un proceso 1, proceso2, y tengo un modulo dejio. Cualquier proceso usuario se va a ejecutar un cierto tiempo y va a hacer un call read, que es un system call, luego sigue la lectura y puede invocar un call write. El proceso 2 es un proceso que solo hace computo, es muy raro que un proceso no tenga io, pero este sera el ejemplo. Entonces en el proceso 1 pido servicios al sistema operativo, que son atendidos por el modulo de io del sistema operativo.

Sea el codigo del manejador del disco, tal que el modulo de io es el driver del disco especifico para leer. El modulo del driver cuando le piden u servicio io, lo que pretendemos es que el proceso 1 comienza jecucion, se ejecuta en procesador hasta la lectura de call read, necesito una operación io. Se hace una transferencia de control al sistema operativo que va a manejar el sistem call. Este sistem call se manda al software del driver y debe hacerse la transferencia fisica de los bits de informacion, es decir la parte hardware. Un driver entonces lo que hace es que se comunica con el controlador, setea memoria, todo lo software, y finalmente al disco le dice go, que es mandar los bits fisicamente desde el disco a la memoria del computador, esto solo lo hace hardware. Una vez que termina transferencia de bits le manda una interupcion: termine, te he mandado los bits. Entonces el driver luego checkea errores, libera memoria, etc. Entonces el modulo de driver tiene paete software, parte hardwre y parte software. Cuando termina operación io, entonces se puede retornarl al proceso usuario para continuar con la ejecucion.

Voy a asignar unidades de tiempo se demoran las cosas, todas las instrucciones de proceso 1 se demoran 10 unidades de tiempo. Supongamos que proceso 2 se demora 40 unisdades de tiempo sin io. En modulo de tiempo es 10, 30 y 10. 10 unidades demora io en comunicarse. La transferencia depende de cuantos bits quiero transferir. Voy a calcular utilizacion sin interrupciones. Supongamos que procesador no posee mecanismo de interrupciones.

El proceso 1 se comienza a ejecutar, luego viene el call read. Esto produce sistem hace un jump al espacio del kernel, comienza a ejecutar 10 unidades de tiempo, luego 30 de transferencia. No hay manera de saber que ha terminado o no de enviar informacion. El procesador se queda preguntando si el disco ha terminado iterativamente, tal que se preocupa explicitamente del fin del io, esto se le lama como pooling, ya que esta preguntando muchas veces checkeando si ha terminado, pendiente hasta que finaliza la transferencia, luego 10 unidades de tiempo y vuelve a proceso. Es importante que el procesador estubo pendiente por 30 unidades de tiempo y no pudo hacer otras tareas.

Luego 10 unidades de tiempo en p1 y se hace un call write. Nuevamente se ejecuta el modulo de io, se hace pooling y retorno al proceso 1, para finalizar el p1. Al terminar el p1, ejecuto proceso 2, que se ejecuta sin servicios de io por 40 unidades de tiempo.

En total u=70/170 que es 41%

Esto es porque sea que haya ejecutado ambos procesos, hubo un pooling donde no se aprovecho este tiempo de espera.

Entonces sea que tengo proceso p1, hace un call read, hace un trap que es system call, el modulo de io ejecuta la primera parte de software y le da instrucción a disco para transferencia fisica, el procesador no debe preocuparse, ya que le van a avisar de la vuelta. Entonces puedo ejecutar en esas 30 unidades el proceso 2. se produce la interrupcion de el moduo de io que ha terminado la ejecucion de transferencia, entonces vuelvo de el p2 a el modulo io, termino la interrupcion y puedo decidir a que proceso voy dependiendo de la prioridad. En este caso p1. Entonces vuelvo a hacer lo del modulo io al llamar call write en p1. Entonces me queda de la transferencia un tiempo que utilizo para terminar proceso p2, me quedan 20 unidades libre de tiempo, que se pierden. Luego vuelvo a hacer lo que queda de la interrupcion, y vuelvo a finalizar el proceso p1.

En total esto seria u=70/150 que es 46%

Segunda unidad del curso: proceso

A veces usamos palabras programa y procesos como similes, pero eso no es asi: prorgama es conjunto de instrucciones en lenguaje particular. Un proceso es cuando programa es compilado, linkeado y cargado en memoria.

El proceso es una entidad que vive en el sistema computacional, entidad viva, dinamica y conocida por kernel, este la administra y la conoce, es un programa en ejecucion. O es la instancia de un programa corriendo en computador o unidad de actividad caracteriada por ejecucion de secuencia de instrucciones en un estado actual y conjunto asociado de recursos de sistema.

La labor principal de sst op es basada en servir a los procesos, tal que sirve los requerimientos de los procesos.

El sistema operativo esta construido para orquestar los recursos fisicos del hardware y asignarselo a los procesos cuando hacen sistem call y los solicitan. Entonces ofrece y atiende procesos, no nos atiende a nosotros, sino que quieres interactuan con sst op son los procesos.

Un codigo en c no es un proceso.

Este programa entonces lo que le ocurre es que lo compilo, pasa por muchas etapas, hasta convertirse en un proceso. Podemos obtener etapas con –save-temps.

El programa fuente se pre procesa cpp. Cuando uno ejecuta instrucción tal que ejecuto comulador gcc que invoca a cpp, que es procesador de c, tal que transforma el codigo fuente a otro codigo escrito en c, preprocesa los · del codigo, tal que los define por ejemplo n 10, tal que dice que string n lo cambie por 10 en todos los lugares apropiados. No define, sino que es un reemplaador. Asi mismo el include, que no incluye librerias, cuando estoy usuando printf que es una funcion de librería de c, pero no esta en mi codigo. Lo que hemos hecho con el include es que incluye el archivo stdio.h en ese punto del codigo. Stdio es un archivo texto, tal que se expande todo el define en el codigo, incluye el archivo. Tiene typedef generalmente.

El resultado del preprocesador es un .i.

Luego viene la etapa de compilación tal que revisa que este bien usado, tal que viene la etapa de ensamblador, todo el codigo c se transforma a codigo assembler. El codigo assembler es .s.

Hay un llamado a callprintf, esta transformado pero aun debe buscar las funciones, tal que transforma del codigo de assembler a codigo de maquina. Este ya no se puede ver, esta en binario, pero aun no se han linkeado las funciones. Entonces el linkeador busca todas las funciones y genera un ejecutable que linkea todos los archivos .o para comprender lo que ocurre. La funcion de librería se une al codigo. Tal que sea que tengo un archivo.o y lo linkeo con math.o ya que estoy usando math.o. estoy haciendo un linkeo statico, y lo que genera es que todo queda en un solo codigo ejecutable.

Es asi que se genera el printf de la funcino. Lo que obtenemos es un programa ejecutable pero no es un un proceso.

Entonces este prograa ejecutable no es un proceso, solo cuando lo hecho a correr. Cuando lo ejecuta ocurren un monton de actividades hecha por sst op, para que este programa entre a procesador debe transformarse en proceso por sst op. La otra forma de crear procesos es con fork, tal que crea procesos.

Si transformo un programa ejecutable con pid fork,

Sea que yo haga cosas en computador, creo procesos con sistema operativo.

voy a crear un proceso con esto con el sst op, tal que crea y administra los procesos, le da los recursos, memoria, etc etc.

Para eso el sistema operaivo debe crear una estructura que identifique los procesos dentro del kernel, lo que llamaremos pcb, tal que es una estructura de datos que maneja el kernel para cada proceso, tal que lo identifica tiene el stado prioridad, datos de contexto, pc, etc.

Lo que ocurre es que el sistema operativo tiene una lista de estos pcb. Y generalmente puede ser una lista doble enlazada. Que seria una lista de procesos que se estan ejecutando, tal que tiene el pcb de cada uno, en esta estructura guardo informacion. Sea que hecha a correr un proeso creouna cajita de esta estructura y la abrego a la lista. Es el carnet de identidad del prroceso del ssitema operativo. Puedo acceder a todo lo que significa el proceso, pero no es el proceso, sino que es el identificador de el proceso. El proceso esta en memoria. Se orgniza como codigo binario en segmentos de memoria.

El programa ejecutable que se obtiene del linkeador al cargarlo en memoria, el sst op lo organiza en segmentos en un espaio de direcciones de un proceso en memoria virtual, tal que hay segmentos del codigo, datos globales, heap, stack de usuario, stack de kernel, librerias, linkeadas, etc.

El codigo en el texto es ell que yo programe y este hace uso de las librerias. Tambien estan los datos globales, la memoria dinamica estara en heap, y el stack de usuario se utilizza para paso a prarametros, llamados a funciones y variables locales. Todos los procesos tienen este espacio. Supongamos que tengo un codigo principal, y de repente invoco la funcion h de a,b. Declaro int a es 5 y int b es 8.

Entonfes hay una funcion que es void h(a,b) sera floay g

Cuando el proceso esta ejecutandose e invoco funcion, entro a la scope, a la localidad e la funcino h y se crean varias cosas, este es un jump que tubo que poner en el stack un frame nuevo, tal que hubo de meter al stack el pc program counter, para guardar el contexto de la funcion.

Clase 5:

estructura que tiene proceso que tiene estructura importante que es pcb, que es una estructura como registro que identifica a un proceso. Cada proceso tiene su pcb que es almacenado en una lista. De esta estructura salen punteros que apuntan realmente al proceso que esta en memoria. El proceso mismoto tiene una estuctura que tiene un esspacio de direccione sque tiene el texto, datos globales, heap, stack de usuario, kernel.

Supongamos que tengo un porgrama

flloat pi

Char\*q

Main(){

int i=8,

char\*p,

q=malloc(3 size of char)

p=malloc(3 size of char)

En la primera parte del codigo van las instrucciones, luego en la segunda van los datos globales qie serian pi y q, tal que se procura espacio para pi y q, las otras variables en main son locales a main. No es una variable local. Esta se procura en el stack. Cuando se ejecuta en programa, se procura espacio para i y p. cuando el codigo hace malloc para p,tal que p esta en el stack y la memoria que apunta esta en el heap, y para q que esta en global, tambien su informacion va a estar en heap.

Entre medio puedo tener muchas librerias.

Esta descripcion del codigo de todos estos segmentos se le llama los segmentos del proceso en memoria especificamente la virtual, tal que es organización logica del proceso, que no necesariamente esta cargado en memoria fisica, y si se carga, va a depender de como se cargue la memoria de esto, dependiendo de la forma de organización de memoria fisica puede ser diferente, independiente de la organizacion de la memoria virtual.

Ahora, la memoria fisica, principal del computador, el kernel se instala como parte del proceso en una sola parte de la memoria. Se instala en una parte de la memoria y ahi se queda, pero estas son accecibles al codigo, al proceso. Entonces va a haber una parte accecible a memoria virtual integrada al kernel. Entonces sea que tenga un call read, va a tener llamados a sistema a sst op. Entonces tambien estara disponible para que proceso entre a espacio de direcciones virtuales.

Entonces falta todo lo que es el kernel.

El kernel en si no se copia dentro del proceso, es un solo kernel, pero todos los procesos tienen acceso a entrar al kernel.

Contexto de ejecucion:

Toda la informacion del proceso es toda la informacion relacionada con el estado actual de ejecucion del proceso, que seria el estado actual del procesador en el momento, por ejemplo los registros del procesador. Entonces sea que el hardware guarda en psw, pc en el stack y el manejador de interrupciones guarda el restro de los registros, tal que guardan el contexto de ejecucion de un proceso, tal que guardan los contenidos de los registros del procesador como pc, sp, eflags, etc. Tla que si entra otro proceso al procesador, va a comenzar a usar los registros y cambiara el contexto. Por eso que antes de que proceso salga de procesador se debe guardar su contexto. Este contexto se guarda y se carga dependiendo si entra o sale proceso. El contexto de hardware cotiene los registros necesarios pc y psw(eflags),cuando ocurre una interrupcion el proceso se esta ejecutando y le procesador guarda el psw

Contexto de ejecucion:

Toda la informacion del proceso es toda la informacion relacionada con el estado actual de ejecucion del proceso, que seria el estado actual del procesador en el momento, por ejemplo los registros del procesador. Entonces sea que el hardware guarda en psw, pc en el stack y el manejador de interrupciones guarda el restro de los registros, tal que guardan el contexto de ejecucion de un proceso, tal que guardan los contenidos de los registros del procesador como pc, sp, eflags, etc. Tla que si entra otro proceso al procesador, va a comenzar a usar los registros y cambiara el contexto. Por eso que antes de que proceso salga de procesador se debe guardar su contexto. Este contexto se guarda y se carga dependiendo si entra o sale proceso. El contexto de hardware cotiene los registros necesarios pc y psw(eflags),cuando ocurre una interrupcion el proceso se esta ejecutando y le procesador guarda el psw y le pc en el stack, guarda el contexto de hardwe. Luego en el stack se guarda el resto del contexto del proceso. Al irse y venir se cargan los procesos.

Entonces un proceso se compone de su descriptor pcb, su espacio de direcciones, su contexto.

Los estados de el proceso.

Cuando uno hecha a correr el proceso pasa por estado nuevo. En este estado, el proceso esta siendo creado, creando el pcb, tal que se procura memoria,inicializa, todo esto lo hace en sistema operativo y lo preppara. Cuando esta listo para correr en procesador pasa en estado no corriendo que es equivalente a estado listo. Tal que este ya esta aceptado en sistema para que corra en procesador. Eventualente el proceso entra a procesador para ejecutarse y este se le llama etapa running, esta corriendo. El proceso tambien puede salir del estado de ejecucion con un time out para lvolver al estado de listo ya que se le acabo el quantum de tiempo. El quantum de tiempo por ejemplo le da una cantidad de tiempo que es limitada, y una vez que termina ese tiempo ocurre una interrupcion, el proceso es sacado del procesador y vuelve a quedar en listo. Se guarda el contexto del proceso que sale del procesador. Cuando entra el proceso, hay que cargar el contexto del proceso que entra al procesador. Entonces el proceso puede pasar mucho rato en listo, puede ser su turno, termina su q tiempo o puede finalizar todo lo que debia correr. Entonces llega a estado de salir. Entonces al salir debo eliminar la memoria y todo lo que he procurado.

Existen otros estados. Cuando esta ejecutandose por ejemplo, hace un call read. Entonces llama a sistema. Este llamado a sistema produce eventualmente que proceso salga a estado de bloqueado, esto es porque quien hace el io es el sistema operativo. El sistema operativo se va a ejecutar en dos fases: fase inicial y fase final, entre medio una transferencia fisica de bits. Durante este tiempo de transferencia de bits el proceso esta esperando que lleguen los datos, por lo tanto entra a un estado de bloqueado, esta esperando a un evento que por ejemplo que el io termine. Cuando espero no tengo derecho a entrar al procesador, ya que esta boqueado el proceso. Cada vez que un proceso hace un io, vamos a hacer que proceso salga a estado de bloqueado. Y de repente obtiene la informacion necesitada del io. Por lo que una vez obtenido el io, vuelve a estado de listo y puedo entrar a procesador. No solamente existe la posibilidad de salir del io, sino distintos eventos que me hacen que distintos eventos me llevan a bloqueado, por ejemplo seria un breakpoint. Solo puede volver al procesador cuando obtiene la infomacion que esta esperando, tal que se desbloquea y compite nuevamente para entrar a procesador.

Entonces, los procesos aceptados a sistema listo, estan en memoria principal. El que esta corriendo esta en procesador, pero su codigo en memoria principal. Los bloqeuados igual estan en memoria principal. Supongamos que llegan muchos procesos a sistema.

Entonces lo que comienza a ocurrir es que se acaba la memoria principal, si se acaba o queda poco, y llega un proceso con alta prioridad. Entonces puedo suspender procesos que estan en listo o bloqueado y llevar a memoria secundaria, tal que estan en disco, y no pueden entrar a procesador, por lo que solo aquellos en memoria principal pueden entrar a procesador, entonces estaran los listo suspendidos y bloqueado suspendido. Parecido a swap out y swap in. Swap out es que sistema operativo saca proceso de mem principal y lo lleva a memoria secundaria. Tambien los bloqueados son suspendibles. Eventualmente sistema operativo puede reactivar los suspendidos con swap in, volverlos a memoria principal. Un proceso que esta bloqeuado suspendido y su evento que espera ocurre, es mandado a listo suspendido.

Procesos suspendidos, habiamos dicho que sistema operativo puede swapear temporalmente algunos procesos. El io es extremadamente lento en comparacion a procesador, procesador puede hacer muchisimas cosas mientras el io ocurre. El tiempo de io es muy grande, entonces un io puede ser swapeable y puede asignarle memoria a procesos sin io, esto es una de las politicas.

Como programadores no tenemos control sobre estos aspectos de swap, es desicion de sistema operativo, no percibimos que los procesos pasan por estos estados. No es visible ni como usuario ni como programador.

Sea que tengo 3 procesos usuario a b c y el kernel.

El tiempo transcurre y el proceso a entra a procesador y sejecuta, esta corriendo.

Entonces tendremos 3 estados: corriendo, listo, bloqueado.

Supongamos que pasa el quantum de tiempo con el proceso a, se hace la transferencia haciael kernel que planifica otro proceso, este seria el proceso b. el procesador comienza a ejecutar proceso b, y este hace un io. Lo que ocurre es una solicitud de io entonces este entra al estado de bloqueado y se transfiere el control nuevamente al kernel que vuelve a planificar quien va a entrar y este se ejecuta en un quantum de tiempo y que termina nuevamente volviendo a kernel, planifico, le toca al proceso a por q tiempo, planifico al proceso c y se ejecuta por q unidades de tiempo. B sigue bloqueado.

Hay muchos mas eventos que hacen que salga del procesador, por ejemplo cuando viene un proceso de mayor prioridad.

entonces supongamos que cambiamos el diagrama y proceso a b c y tambien tengo al kernel.

Supongamos qu eproceso a se ejecuta por q unidades de tiempo, luego debe entrar al kernel el cual planifica al proceso b, que se ejecuta por el quantum, entro al kernel, el cual planifica al proceso c, que se ejecuta por q unidades de tiempo… y asi.

Entonces sera la utiilzacion q/q+t, tal que t es el tiempo que se demora en utilizacion del kernel, o tambien llamado overhead o sobrecarga.

Supongamos que T, independiente de valor de q, es el 20 porciento de q.

Tal que u=q/q+0.2\*q, tal que seria 83% de utilizacion si es que t es 0.2 de q

se busca que t sea muy pequeño

de tal forma que utilizacion sea muy grande. El valor de t. estamos hablando de que tan eficiente es el kernel.

Sea t =0.1q → q/1.1\*q lo que es 0.9 de utilizacion.

El tiempo para hacer sus labores entre procesos es el kernel, tal que de este depende la eficiencia.

Para que a entre a procesador hay que cargar el contexto y para sacar el proceso a hay que guardar su contexto, esto con todos los procesos.

Un diagrama de procesos para varios procesos, tal que hay muchos procesos que pueden estar en mismo estado, tal que kernel tendra una cola de pcb de procesos que estan listos. Tal que se tiene una lista de pcb que son estructuras identificadoras, el proceso como tal esta en memoria. Si hay varios procesos en listo, se van a ir encolando y tendra un algoritmo que decide que procesos entran a procesador, esto se le llama despachar tan que se carga el contexto del proceso elegido en el proceso. Planificar es el algoritmo para elegir al proximo proceso que ingresa a procesador. Despachar es cargar el contexto del proceso elegido en el procesador. No confundir.

Estas son dos actividades importantes: la plafinicacion y luego el despacho.

Cuando un proceso hace un io, entonces se bloquea. Todas las listas de pcb son listas de procesos bloqueados las que se muestran en azul de la imagen, porque estan esperando por un evento. Puede ser por ejemplo un evento de io, señal de proceso que esta esperando, de pagefault, tal que puede que necesita acceder a una variable que debe ir a buscarla a disco, tal que son cosas que bloquean y hacen esperar a proceso por un evento.

El contexto del proceso es el conjunto de los contenidos de los registros del procesador, este se debe guardar cuando proceso sale y cargar cuando proceso entra. Este es el cambio de contexto. En el procesador hay un contexto por el proceso y luego al salir va a tener otro contexto. Entonces reanudo un contexto distinto.

Cuando hay que hacer cambio de contexto: io, trap, system call, el proceso sale del procesador. Si ocurre interrupcion, el proceso hay que destruirlo y debe entrar otro contexto. El proceso se le acaba el tiempo y debe swapearse. El page fault fallo de memoria que debe ir a buscar a disco y finalmente system call de io.

Un cambio de contexto o de proceso son lo mismo, pero no es lo mismo que un cambio de modo de ejecucion de proceso. El modo de ejecucion tambien cambia en el procesador, pero no tiene que ver con el contexto. El modo es el privilegio del procesador con la que se esta ejecutando procesador y por lo tanto el nivel de privilegio del proceso, tal que hay diferentes niveles de privilegio.

Supongamos que kernel se instala en memoria, tendre 2 procesos rojo y verde. Ambos en memoria. Supongamos que se esta ejecutando proceso a, y luego llama a un read. Esto hace que entremos al kernel. Esto va a hacer un cambio de modo. Esto es porque dentro del kernel uso funciones privilegiadas. No hay un cambio de contexto en este caso. Cuando el sistema operativo termina el read y comienza la transferencia fisica, puede ir a planificar al proceso b y ejecutarse, tal que hay un cambio de modo y cambio de contexto, esto es porque vuelvo al proceso usuario y cambio de contexto a otro proceso. Tal que cambiar de proceso 1 a kernel seria solo cambio de modo porque sigo en el mismo contexto. Esta es una forma de ver el kernel.

Modos de ejecucion de sistema operativo. Que es el modo de ejecucion del procesador, tal que este es un nivel de privilegio que permite ejecutar el kernel. Tal que tendre modo usuario y modo kernel, tal que cambio el bit de eflags para decir que estoy privilegiado.

Clase 6:

cambio de contexto y cambio de modo. Cambio de contexto es cuando sale un proceso y entra otro proceso. Esto es porque procesador tiene proceso donde estan los registros, y se guardan como contexto al cambiar de proceso.

Labores a realizar por cambio de contexto: guardar el contexto del proceso que sale. Luego se actualiza el pcb del proceso, esto es porque el proceso cambia de modo running a modo listo, tal que el pcb lo tiene indicado, tambien los tiempos de ejecucion que son medidos. Cuando el proceso sale del procesador hay que guardar los tiempos y todos los detalles en el pcb. Y al salir el pcb se pone en la cola de listos. Luego se decide por otro proceso que esta en la cola de listos, esto lo hace el planificador y hay que actualizar el pcb del proceso que esta en ejecucion, se carga todo. Es posible que el proceso va a tener una estructuras de administracion de memoria, tabla de paginas y esta debe ser conocida por procesador y sst op. Se restaura el estado del proceso con el contexto de este proceso y luego estaria listo para ser ejecutado.

Antes se ejecutaba por hardware pero ahora se usa software (el cambio de contexto) se tendra la nocion que el cambio de contexto lo puede hacer el procesadoro puede ser por software que lo hace el sistema operativo. Por algun motivo linux se dio cuenta que kernel puede hacer cambio de contexto, pero esto necesita el hardware por interrupciones. La ventaja de linux para cambios de contexto por software es que el kernel es mucho mas portable. El cambio de contexto cuando es hecho por procesador hace que kernel tenga que comunicarse por hardware, pero si es logico no importa lo de hardware y procesador especifico.

El cambio de modo y el cambio de contexot: es el sst op un proceso?

Hay 2 modelos de sst op: como colección de procesos, que tiene sentido y existen. Otro tipo de sistema operativo no son procesos, que se ejecutan usando el contexto del proceso usuario, tal que no es un proceso. El sistema operativo puede o no ser un proceso. Para entender esto debemos recordar de que es un proceso: tiene un pcb como identificador, espacio virtual de direcciones donde esta el codigo, stack, heap, y ademas del espacio virtual el contexto del proceso.

Entonces depende del modelo si es proceso o no. linux y unix, ios, no son procesos. En particular linux como sistema operativo kernel, no tiene un pcb dentro del kernel, tampoco espacio virtual de direcciones ni contexto.

Cada vez que se realiza un servicio a sistema operativo se llama a una rutina de sst op, no es un proceso el sistema operativo. Por lo que todo sistem call es un llamado como rutina, es como leer codigo, un jump sobre el espacio de memoria que es un servicio, ya esta en memoria y no cambia el proceso.

Entonces cambio de modo usuario a modo privilegiado. Todo lo que esta ocurriendo es que sigue utilizando el contexto del proceso usuario.cada proceso tiene su contexto, su espacio virtualy pcb.

Si debo entrar al kernel debo cambiar mi modo.

En cambio el otro modelo basadp en proceso, todos los servicios serian una colección de procesos, cada proceso tendria su pcb, contexto y espacio virtual. El sst op tendria muchos procesos que servirian a procesos usuario, tal que tendria procesos de sistema operativo, tal que seria una invocacion a otro proceso, habria un cambio de contexto y cambio de modo.

El sistema operativo es como que fuera parte del proceso, pero no es que se replique. El sst op se carga en memoria fisica en alguna parte en direcciones especificas. Luego se cargarian los procesos. No es que dentro del proceso incluya a sst op, sino que cada proceso solicita servicios de sistema operativo, entonces tendre un jump trap que es llamado a sistema donde entro a parte donde esta sst op donde esta open, cambiando mi modo para entrar a ese espacio.

Luego para volver al proceso tengo que cambiar mi modo. Entonces tienen derecho a usar servicios pero para eso deben entrar al kernel. Cuando estamos en el punto de sistem call al hacer el jump sigue el mismo contexto de proceso. Cuando entro al kernel tambien voy a tener que guardar el contexto pero no voy a cambiar uno por otro, solo que lo guardo y sigo usando el contexto. Cuando vuelvo del sst op, restauro mis valores guardados antes de ir. Cuando entro a sst op, hay un cambio de modo pero no de contexto y al salir de sst op hay cambio de modo pero no de contexto.

Entonces sea que tenga mas procesos, que pasa si proceso 0 hace un read, entonces entro al kernel y hay una operación de io, tal que hay un momento en que no puedo hacer nada, entonces como proceso queda bloqueado, debo planificar en kernel otro proceso. Entonces me voy al proceso 1. aquí hay un cambio de modo mas un cambio de contexto. Estaba usando el modo del proceso 0, pero como cambio el proceso, entonces hago cambio de modo y de contexto para cambiar de proceso y ejecutar el que viene

el otro modelo de sistema operativo basado en proceso, cada proceso tiene su pcb, espacio virtual, contexto, sst op tiene muchos procesos que sirven y procesos de sistema operativo. Cuando chago un call read, lo que pasa es la invocacion del servicio que presta otro proceso, cambio de contexto y cambio de modo. Los procesos de sistema operativo son demodo kernel.

De tipo que sst op es proceso:

Entonces tendre el hardware, y tendre procesos que estan en mismo nivel tal que son procesos de usuario y otros procesos de sistema operativo.

Cuando un proceso requiere algo, invoca el servicio y ese serivicio lo presta un proceso. El microservicio es un proceso.

El modelo de proceso de sistema operativo tiene procesos usuario y procesos de sistema operativo. Cada uno tiene su propio contexto, espacio virtual, etc. entonces si veo la memoria fisica lo que veo es una cantidad de procesos, tal que ya no es codigo de kernel sino que son procesos separados.

Supongamos que invoco lectura desde un proceso, y hay un proceso que tiene todos los servicios de io. Entonces para llegar alla hago un cambio de modo mas un cambio de contexto.

Ya que voy a ejecutar un proceso con privilegio. Ya no es un jump, sino que es un cambio de proceso. Cuando el proceso termina, entonces hay que planificar otro proceso. Entonces voy a tener otro proceso que es el planificador. Entonces mientras que el proceso que llama a proceso io se bloquea entonces el planificador va a elegir el siguiente proceso a ejecutar. Entonces este tendra cambio de modo y de contexto para ir a el proceso siguiente.

En el modelo 1 no es un proceso o colección de procesos, en el modelo 2 todo es proceso, van a haber diferentes procesos que se dedican a serivicios especificos que pueden ser por ejemplo todos los de io.

Generalmente a esto se le llama un modelo monolitico (modelo 2) porque el kernel incluye todos los servicios o funcionalidades.

Porque del sistema operativo 1 a sistema operativo 2 hay cambio de modo? No es necesario si ambos son privilegiados, hay algunos que no necesitan privilegios, peude ser que hay algunos que tengan o no privilegio tal que el ir de uno a otro puede implicar cambio de modo. Supongamos que para todos se tiene el mismo privilegio, por lo que no habria cambio de modo.

Api: aplication program interface. Esto quiere decir que tengo un software por ejemplo orientado al objeto donde defino cuales son los servicios que los objetos sirven. Tal que tengo muchas clases que cumplen funciones para el servicio. Entonces esto define las interfaces con las que debo utilizar esa aplicación. Esto generalmente se entrega como codigo, puedo bajar de github toda una api y tengo todo el codigo orientado al objeto, a partir de esto construyo codigo ajustandome a la interfaz.

Por ejemplo supongamos una api de optimizacion y tengo una clase que optimiza.

Entonces declaro mis metodos. Estos van a ser publicos. Supongo que tengo clase optimizador y si quiero usar optimizador debo usar los metodos publicos, solo veo la signature, que es el como usar el optimizador, y esto es la api, que es la definicion de todas las clases de mi software. Tiene mas que ver con el software.

Hay mucho software para procesamiento de imagenes, y yo me documento con la api, tal que conosco la forma en como comunicarme con la api. Este software puede ser un codigo fuente tal que incluyo, uso, puede ser microservicio que pido a api. Tal que sea que pongo huella digital supongamos, lo que ocurre es que se manda solicitud a servicio a una empresa que valida huellas. Esta clinica no tiene acceso a software de empresa, pero hace uso del servicio y la api, la api no solo sirve como se extiende el servicio. El microservicio es un proceso que presta un servicio pequeño.

Si uso librería: Yo linkeo codigo con codigo ejecutable, no tengo derecho a modificarlo, ya que esta en librería, esta librería puede tener implementaciones, y lo visible es la api, que es lo publico, es la declaracion ede como usar el software, yo puedo incluirlo, heredarlo o obtenerlo como librería que es ejecutable, por lo que no podria editarlo o heredarlo, solo puedo usarlo a traves de la api.

Linux no es un proceso.

Hemos hablado de cambio de modo de ejecucion, hay 2 modos: modo usuario que no es privilegiado y no accede a kernel y modo kernel que puede acceder a todo de forma privilegiado. El cambio de modo de ejecucion es valido para ambos modos de sst op proceso y no proceso.

En sistema operativo donde kernel se ejecuta en contexto usuario, es necesario realizar un cambio de modo de ejecucion.

Cuando ocurren las interrupciones, entonces acordarse que program counter y psw se guardan porque voy a entrar al kernel. Se guarda tambien el modo en el que estaba anteriormente, se cambia de modo usuario de modo kernel, tal que el el manejador de interrupciones puede ejecutar interrupciones privilegiadas.

Para terminar hoy veremos el modelo de procesos de unix. El anterior es modelo generico que puede ser utilizado o no. pero linux tiene uno parecido.

Proceso es creado, tal que fork es el que crea procesos, el proceso esta en modo creando, una vez que ha sido creado todo y hay suficiente memoria para ser aceptado, pasa a estado de listo en memoria. Pero si no hay suficiente memoria pasa a estado listo swappeado suspendido. Entonces estando en el estado swappeado se libera memoria, de memoria primaria a secundaria y viseversa. Al ser elegido proceso para entrar a procesador, entra a modo kernel y luego a modo usuario. Unix no es un proceso, es un sistema operativo que se ejecuta en el contexto de los procesos usuario. Unix es como linux, no es un proceso. El kernel se ejecuta en el contexto de proceso del usuario. Estando en modo usuario, pido un servicio tal que le mismo proceso hace un interrupt entra al modo privilegiado dentro del mismo proceso,tal que tendria el running usuario y el running modo kernel del mismo rpceso. Si proceso termina pasa a estado de zombie que es cuando se esta destruyendo el proceso. Al hacer io, entra a kernel running y pasa a bloqueado que es asleep in memory.

Cuando no queda mas memoria seria listo suspendido y bloqueado suspendido.

La diferencia esta en que explicitamente divide el running en usuario y kernel. Hay un estado nuevo que es preempted. Esto significa desapropiado. Eso significa que sea que un proceso se esta ejecutando en running y llega proceso de alta prioridad, inmediatamente va a listo y debe entrar a procesador, lo que hace el sst op es que lo saca, lo desapropia, lo saca al estado desapropiado. Lo saca, lo desapropia y asi puede entrar el de alta prioridad. Al terminar, puedo elegir nuevo proceso de cola de listas pero tiene prioridad el desapropiado por sobre los que estan listos.

Clase 7:

modelo de estado de procesos en linux. Se parece al modelo generico ya visto. Cada sistema operativo modela estados de proceso como le paresca, cada uno tiene supropio modelo de ejecuicion de procesos, es organización de transiciones que procesos tienen cuando se ejecutan dentro de sst. En el modelo estado de unix la unica forma de entrar a procesador es estando en el estado listo para correr en memoria, tal que organiza transiciones y permisos, cuando son apropiadas y cuando no. cuando tienen permitido cuando no para que entren a diferentes partes.

Si vemos elmodelo de estado de linux es mas sensillo, tiene menos estados. Ya no esta estado nuevo, sino que es start que pasa directamente a running, tal que es cualquier proceso listo en la cola de espera para ingresar a procesador. Interruptible es el estado de los bloqueados. Y linux engloba a todos los procesos bloqueados, no hace diferencia entre los suspendidos y los no suspendidos, sea de donde este en la memoria. Los procesos bloqueados estan esperando por un evento, tal que cuando ocurre va a ocurrir interrupcion que los saca de ese estado, tal que linux lo llama interrumpible para devolverlo a listo. Entonces sea que este en running puede ir a task running en procesador cuando esta esperando en interurmpible vuelve a listos. Cuando un proceso es detenido por otro proceso es el task stopped. En el task interrumpible hicieron una solicitud por ejemplo de io o sistem call o o pidiero un servicio tal que estan esperando que termine. Pero el task stop fueron detenidos por otro proceso..estaban ejecutandose pero fueron detenidos por algun otro proceso, no estan pidiendo otro servicio pero si hubo uno que lo detubo, esto puede ocurrir entre padre hijo para detener al hijo.

Tal que por ejemplo pongo breakpoints, tal que cuando llega ahi se detiene. El ide hecha a correr a proceso hijo y le manda señal en el punto de ejecucion, el proceso es detenido por proceso padre. Y la unica forma de que vuelva es que el proceso que lo detubo lo devuelva a task running ready, no puede ir directo a task running procesador.

Task zombie es una tarea o proceso que esta terminando. Cuando proceso

task zombie es equivalente al end. Todo lo que haya dentro del proceso ya terminado no es relevante, todo eso debe ser eliminado de memoria. Unavez que finaliza elimina, solo depende que si los haya guardado en el padre no no.

un task uninterruptible esun proceso bloqueado que espera condicion de hardware. Este aparece y son dificiles de matar, tal que no es interruptible con un kill. Al parecer esperan señal que no es de proceso ni de sistema operativo, es una señal de hardware y no puede ser despertado por ningun otro proceso.

Hay codigos asociados a estado de proceso de linux. Esta la r de running que es aplicable para ambas running, s es sleeping que es el bloqueado. D es el sleeping ininterrumpible, t es stopped y z es el zombie. Linux usa la palabra task para todo tal que serian equivalentes a proceso para linux. Algunos piensan que task esuna hebra, pero task es un concepto mas abstracto que proceso y hebra.

Puedo abstraer proceso y hebra como tarea. Cualquier actividad que se planifica en procesador. Y esto seria unn task. Como creo procesos es con fork. Fork es un servicio de sistema operativo para crear proceso nuevo, la signature es pid t fork(void) y retornaun pid t, que es un tipo definido en types,h

dentro de types se define un pidt. Tal que no alega, ya que incluye el codigo que lo define.

Fork no recibe argumentos y retorna un entero, tal que crea un proceso hijo que es la copia exacta del proceso padre, lo unico diferente es el pid del proceso. Fork crea un proceso y si lo creo bien retorna un entero, y ese entero va a ser el pid del hijo. Tal que es el identificador del hijo que lo retorna en el proceso padre. Si es -1 entonces es error. Lo interesante es queal crear proceso hijo, entonces en el proceso tambien se esta ejecutando y es el valor 0. el fork es llamado 1 vez pero es devuelto 2 veces, fue llamado en padre y retorna en padre y en hijo.

Sea un proceso que tiene un main y se esta ejecutando hasta que invoca a p que le asigna un fork. En ese minuto lo que ocurre es un trap. Porque es un llamado a servicio de sistema operativo tal que hay un cambio de modo pero no de contexto. El sistema operativo crea un proceos hijo dando este servicio. El pid del hijo seria supongamos 215 que se lo devuelve al padre. Y el pid del padre seria 84. el fork crea un proceso hijo identico al padre. Todo el codigo que sigue se copia, todas las funciones del proceso padre tambien, es una clonacion identica. Lo unico que cambia es el pid.

Decimos que el fork fue exitoso porque retorna al pid del proceso hijo que es 215 y abajo retorno 0, siempre al hijo retorna 0 si fue exitoso. Si fue -1 solo va a haber retorno en el padre. Por lo tanto va a retornar -1 y va a saber que no lo creo. Lo que se devuelve del fork en el padre es el pid del hijo.

Un proceso es codigo, datos globales, stack, librerias, heap. Ademas de esto existe un pcb y un estado, que es el conjunto de todos los contenidos del procesador, el contexto.

El fork lo que hace es copiar exactamente todo al hijo, copia el codigo. Copia los datos. Copia el heap. Copia las librerias. Copia el stack, copia todo. Sin embar el el proceso tiene un pcb propio, ya que es otro proceso copiado. Ademas tiene su propio estado contexto, esto es porque el proceso hijo necesita un estado inicial para comenzar su ejecucion. El proceso hijo no comienza desde el principio, sino que comienza desde el retorno del fork. Ahi comienza realmente el fork, esto es porque el sistema operativo copia el estado del padre al hijo, que es el program counter, entonces voy a comenzar justo despues del fork con el hijo, tal que el fork retorna en el padre el pid del hijo y en el hijo retorna 0.

veremos un codigo de fork y pid.

Sea un programa para crear un proceso hijo con fork.

Se declara una variable del tipo pidt y intento crear el hijo con fork, intento hacer un pid. Si es 0 es el hijo, si no es 0 es el padre. Esta es la forma de ejecutar el proceso hijo del padre en el computador, porque ejecutan el mismo codigo pero puedo diferenciarlo y separarlo con el if. La unica forma de que el padre sepa quien es el hijo es con lo que retorna el fork. De otra manera no se puede, cada proceso ademas puede obtener su propio pid y el pid de su padre. El pid del padre del padre se obtiene con ppid, y este es constante ya que es la bash. Los pid de procesos en ejecucion de programas van cambiando.

Bash interprete de comandos, ess la que crea y forkea realmente el proceso padre a hijo y lo ejecuta.

Que es lo que reporta el comando ps, son varios datos: el pid. Lo que esta usando, su tamaño, rsss es el resident size, lo que realmente de eso esta cargado en memoria principal. Siempre esta cargado mucho menos de lo que el proceso es. El estado puede ser s de sleepy bloqueado, otro estado puede ser running, en el minuto que lance el comando en bash se vera como running el propio proeso que lo ejecuta, pero ya ha terminado ya que se muestra en pantalla.

No existen 2 procesos con mismo pid, luego del fork no se sabe si padre o hijo se ejecutan primero. Según la clase pasada, el kernel puede crear hijos? El padre puede crear procesos estando o no en modo kernel. El innit es un proceso creado por el kernel del sistema operativo, el kernel como tal no es un proceso al menos en linux, y a partir del innit se crean todos los demas, eso es visible con pstree que es el como se crearon todos los procesos. Todo proceso es creado por otro proceso menos el innit.

La multiplicacion que se ve en el pstree es porque crea un proceso multihebreado del mismo proceso.

Clase 8:

el fork y como copiarlo, cada uno es independiente del otro. Los procesos compiten por el uso del procesador. No puedo asegurar que primero se ejecute el padre y luego el hijo, hay que asumir que proceso padre e hijo compiten, no esta definido. Si ejecuto hartas veces eso puedo variar, depende de la planificacion:

sea que tengo dos fork, tal que padre hace fork, creo un hijo. A padre se le retorna 20 y a hijo 0. luego el codigo se encuentra con un nuevo fork, por lo que ambos procesos crean procesos hijos. Por lo que el padre devuelve un nuevo hijo, el hijo1 retorna el pid del hijo y el hijo del hijo devuelve 0.

el padre tiene un pc, y cuando esta haciendo el fork el pc comeinza desde despues del fork, retornando ademas el fork, pero la siguiente ejecucion es la siguiente instrucción.

Todo lo que tenia el proceso padre en la creacion se le da al hijo.sea que el padre tiene un i=0, y no sumo en el padre pero si en el hijo, el hijo que tiene va a tener un i=1, que se hereda al siguiente hijo que el hijo crea.

Figarse que si tengo una variable int i=0, el hijo comienza con i=0, pero puedo reasignarlem y va a ser diferente de lo que tenga el padre.

Todos los procesos del sistema estan en el arbol de procesos, el primero que se crea es el innit.

En gerarquia de padre hijo, son iguales.

Pero si quiero crear un proceso diferente este va a ser el exec. Hay diferentes formas de usarlo pero nos enfocaremos en el execde

al crear un fork como hago que el hijo sea distinto al padre? Invoco a exccve tal que ejecuta un nuevo programa en el programa que ejectua la invocacion. Este execve indica el nombre de un nuevo programa ejecutable. Este proceso ejecuta un nuevo programa en el proceso mismo. En extricto rigor no se crea un proceso nuevo, sino que sobreescribe el espacio virtual de direcciones con algo nuevo.

Supongamos que tengo el proceso padre que tiene un espacio virtual de direciones, tal que tengo el codigo, los datos globales, el heap, las librerias, el stack.

Este ejecuta un fork en alguna parte de su codigo. Se crea un nuevo proceso con la copia ideantica de lo que tenia el padre. En el codigo hay una invocacion a un exec, y tambien estaria en el del padre.

Pero el hijo despues de ser forkeado, hace exec y se le da el nombre del programa, y este esta en memoria secundaria, donde puede encontrar el programa. Cuando encuentra el exec, lo carga en el espacio virtual de direcciones, tal que es eliminado el espacio virtual anterior y se crea uno nuevo con el nuevo codigo que viene, con nuevos datos globales, etc. pero el pid del proceso no cambia, sigue siendo el mismo pid. Entonces el exev lo que hace es efectivamente eliminar el espacio de direcciones antiguo y ejectua el nuevo.

Muchas veces vamos a querer hacer un exev del fork. Sin embargo antiguamente si yo queria construir un proceso paralelo, el fork hace sentido, ya que es un programa paralelo, tal que se replica el codigo que se le pueden dar algunas diferencias para paralelismo. Pero si quiero que el proceso hijo sea totalmente diferente al padre se usa el exev.

Cada sst op puede añadir cosas, por ejemplo linux añade clown. Tal que es unavariacion de fork para poder manipular mas el fork, es mas versatil. Fork es rigido.

Entonces en execve se busca el nombre, no es un programa fuente esta ejecutado y linkeado. El primer argumento es un string del nombre, lo siguiente es el argumento que se le pasa. Tal que le puedo pasar los parametros en el arreglo como los string que acompañan la linea de comando., por ejemplo ./lab1 ~~i arch.txt seria arv[0]= /lab1, argv[1]= “~~i” arg[2]= archivo.txt

y el tercero es para elegir el ambiente, tal que puede ser por default. Cuando exec falla retorna un entero, y si tubo éxito, nunca retorna. Esto es porque yo ya cambie el codigo en el proceso hijo y el exec ya no existe nada de lo que estaba ocurriendo o que se esperaba que ocurriera del anterior programa.

Execve es el llamado a sistema, el que nos presta el servicio. Puedo llamarlo de diferentes formas, estas son variaciones de exec, que finalmente invocan a exec, tal que son como front ends, tienen diferentes argumentos, pero finalmente lo que hacen es llamar aexecve.

Exevl es una lista, v es vector yo le estoy pasando un vector con los argumentos, una l es que yo listo los argumentos, tal que seria una lista de string de lineas de comando. Entonces l significa que listo los argumento, y los v es que paso un vector de strings.

Otro mas es p, que quiere decir es usa el path definido en el abiente para encontrar el ejecutable, tal que no paso medioambiente en mis parametros, tal que el prorgama ejecutable buscalo en el medioambiente del padre, tal que el medioambiente es heredado del padre.

fflush(stdout) se u¡limpian todos los buffers para que todo salga por el stdout, para que todos los caracteres impresos salgan por la terminal. Tal que le digo despues el execve(camino que quiero ejecutar, argumento 0 que es el mismo nombre del prorgama que es el argv0, null) tal que si falla imrpimeque no puse pudo hacer. Entonces la bash es lo mismo que hago yo aquí. Que es que creo al proceso hijo que por un momento es una bash completa y en el hijo se ejecuta el programa excalc en este caso. Tal que el proceso padre esta esperando que el hijo termine con un wait.

Cuando termino el excalc, la funcion del padre no retorna porque no existe un hijo a quien esperar.

Tengo un proceso que llamaremos padre, que eventualmente hace un fork que crea un hijo. Este hijo va a decir soy el hijo, entonces este va a hacer un exec, el padre va a ir al else. El padre va a hacer otra cosa como crear mas hijos, una vez ue esto ocurre el hijo es distinto al padre pero tiene el mismo pid. Entonces el padre va a hacer un waitpid por todos los hijos que creo.

La bash tiene un loop, en terminos abstractos es que hace un read comand, parse s y lo que hace es fork y if pid es igual a 0, entonces hace el exec del comando.

Lo que hace el padre es un wait al hijo. Y cuando termina el proceso, el proceso padre vuelve al loop, parsea, read command y hace un exec. Si hacemos el exec al padre, ya no puede crear mas hijos.

Esto es importante en el concepto de servidor: sea que tengo un proceso que esta escuchando comandos o requerimientos de la red, cada vez que llega uno, crea un hijo y el padre sigue ejecutandose, tal que el padre siempre esta leyendo por io y le manda al hijo lo que obtiene para que el hijo lo sirva, tal que el padre esta en un loop escuchadno y al hijo lo hace servir el requerimiento, y el apdre puede o no esperar porel hijo, tal que queda el padre liberado de estar escuchando por la red, tal que siempre esta escuchando.

Servidor entonces seria un proceso que siemrpe esa escuchando y al obtener requerimiento forkea proceso para que el hijo sirva el requerimiento. Lo otro que necesitamos ver es como se van a comunicar dos procesos.

Lo que yo quiero es eventualmente que dado dos procesos p1, p2. Tal que son procesos independientes, como le puedo mandar informacion de p1 a p2. Este mecanismo es el mecanismo de los pipes, tal que es un camino de comunicación, es un tubo. El sistema operativo lo maneja. Tal que el pipe como lo implemento? Con el kernel como un descriptor.

Si uno crea el pipe, tal que incluyo el unistd.h y el llamado a la funciones int ppe(int fd[2]) esta funcion lo que hace es crear un pipe, que es un descriptor de archivo. Estos descriptores estan en fd2, si tubo éxito el pipe, entonces pd0 va a estar el descriptor de lectura, fd1 va a estar el descriptor de escritura.

Esto significa que el proceso va a poder escribir por fd1 y puede leer por fd0. Tal que el pipe se crea en un proceso. Dado un proceso, si este proceso llama a pipe, con un arreglo de 2, lo que ocurre es que kernel crea el pipe tal que el proceso puede escribir pro fd1 y puede leer por fd0. Es decir que se puede comunicar con si mismo, tal que hay un camino de comunicación unidireccional. Por lo tanto el proceso se puede mandar informacion a si mismo. Por ejemplo write fd1 y alguna informacion buffr de un largo, con toda la informacion que esta en el buffer va a ir por el pipe, por cualquier tipo de informacion como enteros, array. Y read va a ir a fd0, temp y el tamaño. Entonces la informacion va a estar guardada en write hasta que la leo, tal que puedo no leer todo, pero a la medida que leo, voy sacando las cosas de pipe.

El objetivo es que como dos procesos pueden mandarse informacion por un pipe.

Que es un descriptor? Sea un arreglo de enteros, entonces fd0 va a ser un entero y fd1 va a ser un entero, talque estoy escribiendo numeros, tal que son numeros a una tabla de los descroptores que estoy manejando. Un descriptor de archivo se declara como int f=open(“archivo”, “r) tal que abro un archivo en modo r que es lectura, que es un identificador. Estos identificadores como f que identifica al open en read de el archivo es una tabla de descriptores. Esta tabla se indexa con los numeros, tal que va a estar el descriptor que me llevan al archivo, tal que f seria el numero de la tabla de descriptores de archivo. Entonces sea que este read(f, buff, size) tal que leo un descriptor de archivo, me va a retornar el numero. El pipe tambien es un descriptor. El pipe en realidad se materializa con dos descriptores fd1 y fd2. Todos los descriptores de archivo van a estar en la tabla y retorna entero. El pipe tambien es un descriptor, que tiene dos descriptores uno para escritura y uno para lectura. Una rchiovo normal tiene algo en disco, pero pipe no se materializa en memoria secundaria, se maneja a nivel de kernel en su propia memoria, no tiene asociado bloques de disco como los archivos ya mostrados.

El write y el read leen por ejemplo, sea una tabla de descriptores, supongamos que se abrio el archivo f open de archivo para lectura, sea en la tabla de descriptores de f va a salir a un puntero a una cosa llamada vnode, que es muy parecido a fat. Del vnode me voy a la memoria secundaria donde estan los bloques de informacion del archivo que quiero leer. Por ejemplo el archivo que quiero leer. Entonces un read(f, buffer, size) estoy leyendo caracteres de size en el buffer, tal que f es el descriptor que lee, los busca en el disco y los deja en el buffer. Si no hay nada escrito en el pipe, tal que en el proceso 2 hace un read y esta vacio, la lectura se bloquea hasta que llegan todos los caracteres que estoy pidiendo.

El proceso p1 manda la informacion que quiere, por lo que si yo voy a leer al pipe, yo se lo que quiero leer del pipe. Ahora, entre ambos procesos se debe establecer un protocolo, supongamos que p1 le manda una estructura a proceso 2, entonces va a tener que mandarle informacion para que pueda obtenerlo. Hay que escribir los protocolos sobre este pipe.

Clase 9:

como ejecuto prorgrama que es distinto con el fork y exec, que cambia completamente espacio virtual con nuevo programa ejecutable con nuevo pid.

Como comunico procesos: el pipe es comunicar procesos via kernel, a traves de descriptores de archivo. Retorna entonces fd0 y fd1 que son dos descriptores, que sirven apra indexact, tal que fd1 es el escritor y fd0 es el lector. Si creo un pipe tengo la posibildiad e enviarme informacion a mi mismo.

El proceso lo primero que hace es llamar a pipe, y puede dar error si es que da -1. sino imprimo los numeros de los descriptores, luego el proceso envia mensaje por el write. El mismo proceso, lo lee por fd0. Lo que saca del pipe lo coloca en una variable, tal que yo me envio mensaje y lo envio luego a stdout que es de la forma write(1, mensaje, size) tal que 1 es el descriptor de escribir a stdout.

Se asigno a fd0 y fd1 3 y 4 respectivamente. El 1 esta reservado para el standaroutput el 0 es standar input y 2 es el standart error.

Sobre esos numeros se comienza a asignar numeros nuevos al crear nuevos pipes, voy creando nuevos descriptores.

Lo que queremos es hacer que los procesos se comuniquen entre ellos s traves de un pipe. Pipe no tiene asociado archivo o disco, todo se maneja a nivel de kernel en memoria principal. Como lo hago para comunicarme?

Protocolo: un proceso crea un pipe

leugo mismo proceso crea un hijo

luego el padre cierra el lado de escritura, tal que cierra el fd1

el hijo cierra el lado de lecetura tal que cierra el fd0

----

tengo un proceso y ese proceso genera un pipe

luego el proceso hace un hijo, tal que el hijo conoce el pipe, hereda descriptores de entrada y salida.

Entoces padre e hijo tendran fd1 y fd0, tal que el pipe tiene fd1 y fd0 comod descriptor. El hijo cierra la lectura, el padre cierra la escritura. Entonces el padre va a poder enviarle informacion al padre ya que el hijo tiene abierta la escritura y el padre tiene abierta la lectura.

Esto es un protocolo que es facilmente confundible, que ambos por ejemplo esten escribiendo por el pipe. La informacion va a quedar mesclada y va a dar error. No puede ocurrir que padre e hijo quieran leer a mismo tiempo el pipe. Entonces tendremos un camino unidireccional de hijo a padre. Tambien puedo hacerlo al reves.

Lo importante es que tengamos la nocion de que al hacer el fork al ser identico al padre conoce a los numeros de los descriptores, tal que son unicos para todos, el numero asignado fd1 y fd0 son unicos en el sistema entonces permite ser usado para comunicación.

Como lo hago para comunicación bidireccionalmente?

Entonces sea que tengo un proceso padre, tengo un proceso hijo, y tengo dos pipes.

Entonces tendria pipe1 y pipe2. Entonces tendremos como pipes fd1 y fd2. Tal que como padre tendre fd1-1 por el pipe fd1 para el hijo fd1-0 y ademas tendre el hijo fd2-1 hacia el pipe fd2 y luego hacia el padre fd2-0 para que lo lea.

Entonces padre pipes fd1 y fd2, luego padre crea hijo, tal que hereda ambos pipes. El padre cierra fd1-0 y fd2-1 y el hijo cierra fd1-1 y fd2-0. Asi queda como en el ejemplo. Luego de eso los procesos pueden hacer write y read de padre a hijo y de hijo a padre dependiendo del pipe.

Ejemplo a nivel de sistema operativo. Supongamos que tengo ls-l me dicta todos los archivos que tengo.

Dup y dup2:

llamados a sistema para duplicar descriptores. En el sentido de que le asignan un alias a un descriptor existente. Asi que cada uno tiene un nombre de descriptor, es un unico numero, va a ser el mismo descriptor. Son heredables de padre ahijo. El dup y dup2 entonces, sea #include unistd.h, int dup(int fd), int dup2 (int fd, int fdnew) dado un descriptor abierto en el sistema, dup me va a dar un alias del descriptor, por lo que todo lo que lea o escriba por el dup, sera tambien del fd. Y en dup2 lo que hago es que le doy un alias y un nuevo numero, que sera del nuevo alias. Cuando dup2 tiene éxito va a ser el mismo que le he solicitado. Por ejemplo puedo tener un dup2(fd[0], 8) !=8) tal que si no me da la igualdad va a dar error, tal que le doy un alias a mi descriptor y debe ser igual al alias que le di. Le puedo pedir que me duplique descriptor que esta siendo usado por otro y me da error.

Entonces esto lo veremos con los pipes. En dup me da el mismo descriptor que ya esta dado y en dup2 le doy y me devuelve el alias idealmente.

Sea el proceso a que tiene tabla de descritores que usa. 0,1,2 son reservados que son stdin, stdout, sterror. Entonces voy a tener un archiovo y mi descriptor 5 va a estar apuntando a una tabla de descritproes del sistema, donde va a obtener todos los descriptores de todos los procesos, entonces va a apuntar a un descritor, tal que hay informacion como por ejemplo flags, permisos, owner, del tipo de archiov, fechas, etc (todo sobre archivi) entonces va a haber un puntero a vnode que es el fat y este vnode tambien tiene distintos campos que uno de estos es puntero a memoria secundaria. El proceso va a tener unos descriptores abiertos y elijo el que quiero, el cual vnode va a organizar archiov que apunta a la memoria con la informacion.

Otro proceso b tambien tine su propia tabla de descriptes que esta usando, 0,1,2 esta ocupados. Estos estan reservados. Y supongamos que abro archivo y tiene su descriptor que apunta desde 8. desde el vnode de descriptor apunto a misma informacion en memoria. Como un proceeso accede a informacion de un archivo. Dentro tiene tabla de descritores y el kernel va a tener tabla de descriptores que va a saber donde esta la memoria. Los pipes son lo mismo pero sin la informacion de disco, sino que va a estar en memoria principal.

Entonces un dup en un proceso va a sinificar que tengo un procesos a y un proceso b. ambos tienen tabla de descriptores, y es posinle entonces que la tabla de descriptores del kernel tenga en la tabla de proceso a el 4 que apunta a descriptor de kernel. Y proceso b tiene 5 que apunta a otro lugar de kernel. Lo que sucede aquí es que el 5 es un dup del 4, tal que desde el kernel van a apuntar al mismo vnode. Entonces cuando descriptores son distintos cada uno tiene sus vnodes, pero en pipes si yo duplico descriptor, van a apuntar a mismo vnode.

Para simplificar vamos a concentrarnos en parte kernel.

Entonces en proceso a va a tener su tabla de descritores y proceso a crea un pipe, voy a crear dos descriptores que van a ser fd0 y fd1, que es lo que va a materializar al pipe creado por proceso. Puedo entregarle dos numeros que seran en este caso 3 y 4 respectivamente.

Que pasa si crea un hijo luego de crear el pipe. Como vimos, se va a crear el proceso b que es la copia identica al padre con la misma tabla de descriptores. Entonces el 3 apunta a a fd0 y 4 apunta a fd1. Ambos apuntan descriptores apuntan a mmismo vnode, que se maneja en memoria principal.

Es por esto que dos procesos pueden comunicarse con pipe. Entonces asi se pueden comunicar los procesos.

F de fopen son librerias, todos ellos invocan a open, a close, a read, write, seek. Estos son las funciones finales que se ejecutan que son system call. Fopen va a llamar a open por ejemplo.

Por lo tanto estas funciones no acceden directamente a estructuras ya que terminan llamando a sistema.

Todo lo que se puede hacer con fopen por ejemplo, se puede hacer con open.

Si continuamos con el ejemplo, el proceso a crea un pipe y luego un dup fd0, hay varios descriptores, tengo 3 y 4, generan pipe, tal que tengo fd0 y fd1, estos apuntan al mismo vnode. Y ahora hace un dup de fd0. Puedo leer la misma informacion por fd0 que por fd1, entonces proceso crea un pipe, crea un hijo, y duplica descriptores tengo el proceso a con su tabla de descriptores, y crea un pipe por ejemplo 3 y 4, tal que 3 apunra a fd0 y 4 apunta a fd1. Entonces se hace fork y crea copia identica al padre, por lo tanto tendre 3 y 4 que apuntan a fd0 y fd1, estos apuntan a vnode. Supongamos que hace la siguiente dupolicacion: el padre hace dup2 fd1, stdout, fileno Y el hijo hace un dup2 fd0,stdin,fileno.

Esto es una macro que significa el numero de descriptor standart que es stdout y stdin, tal que todo eso vale 1 o 0 gracias a los include podemos hacer eso. Todo lo que lea por stdin se va a leer por el pipe, entonces en el padre el fd1 se duplica con el stdout, tal que es el alias. En el otro lado el stdin se duplica con fd0.

Entonces este orden es importante.

Crear pipes, crear hijo, hacer duplicados. Lo que ocurre es que todo lo que el padre escribe por el stdout se va a ir automaticamente por el pipe, tal que redirecciono su salida standart al pipe. Si el proceso b lee del standarin, va a tener que leerlo por el pipe, tal que redirecciona. Tal que hace un comando de pipes.

Clase hebras 10:

Al momento hemos trabajado con el concepto de proceso, podemos pensar que vamos a introducir un elemento nuevo al sistema. Las hebras no son tan distintas a los procesos, siempre hemos estado trabajando con hebras, no tener la idea de que las hebras son totalmente distintas de los procesos. Es un concepto nuevo que se extrae del concepto de proceso. Es decir que es un concepto que el mismo proceso incluye particularmente. Al hablar de proceso vemos los aspectos de programa de ejecucion o entidad que puede planificarse en :

procesador (bloqueo, espera, run, ejecuta) donde va avanzando en la ejecucion.

Lo otro es que es un area virtual de procesos, espacio virtual de direcciones, donde estan las diferentes partes de un proceso (datos, stack…) tal que es un algo que materializa el proceso en algun area de memoria.

Por otro lado esta la capacidad de planificacion del procesador.

Que parte de esta area virtual entra a la planificacion? Nada, sino que lo que entra es el contexto: conjunto de los contenidos de regristros de procesadores en el procesador, tal que es una foto del procesador al salir, tal que es el contexto donde podemos distinguir contexto hardware y de ejecucion software.

La capacidad de contexto permite este avance. El fetch es permitido por el program counter, tengo un stack pointer, instrucciones, etc. todo esto hace posible de que el proesador ejecute estas lineas de codigo y aceda a los datos. La primera caracteristica contexto es la que define realmente a la hebra del proceso, tal que esta caracteristica no es nueva, pero es que la materializamos con conceptos adicionales.

La segunda es los recursos del proceso. Principales recuros: memoria principal, memoria secundaria.

Desde ese punto de vista siempre todo proceso ha tenido una hebra, al menos una hebra, tal que es la capacidad de que proceso ingrese a procesador y se ejecute.

Hace años atras ese dieron cuenta que este concepto puede ser explotadopara que proceso tenga multiples instancias de ejecucion.

Sean los elementos de un proceso, donde tengo codigo, stack, heap, conjunto de registros del procesador, tal que lo ultimo es el contexto del proceso, que es el corazon de lo que entenderemos como una hebra. Sea que un procesador quiere tener multiples hebras, entonces seran multiples contextos de ejecucion. Ademas se necesita un stack para cada proceso de ejecucion. Sea un proceso como visto anteriormente, el proceso tenia un stack unico donde habia stack usuario y kernel, tal que ahora por cada hebra se tendrà un stack de usuario por cada hebra de ejecucion del proceso, tal que hace posible que el contexto, sea que existan varias hebras con diferentes contextos, entonces que no se molesten entre sus variables locales.

Supongamos que tengo un codigo, y llamamos a una funcion f con parametros a b, la funcion f int a float b; int z … tal que se ejecuta una hebra en su contexto y luego otra hebra con otro contexto. Tal que puede luego ambas hebras tener diferente program counter, tal que ambas necesitan al contexto.

diferente.

Y cada uno hace modificaciones a una variable que puede llamarse igual, pero no se sobreescriben entre estas ya que no es global, sino que local: diferente contexto.

Si hebra roja sale y hace fork, entonces usara stack de kernel y si hebra verde quiere usar kernel, tabien tendrà su propio stack de kernel, tal que pueden hacer invocaciones de llamado a sistema. Es el mismo proceso que estan ejecutando, pero ahora permito que hayan varias ejecuciones del mismo proceso y permito que sean entre ellas independientes.

Entonces como siempre estamos dentro de un proceso, una hebra esta dentro de un proceso y utiliza los recursos de los procesos. Las hebras solo se diferenncian entre ellas sobre el contexto y el stack . Pero todo el espacio virtual de direcciones es el mismo (text, data, heap) tal que usan lo mismo entre las hebras. Pero gracias a que ellas tienen un stack y contexto distinto, tienen un flujo de control independiente, tal que pueden hacer sus propios ciclos fetch, comparten los recursos del procesos que son: codigo, datos globales, heap, descriptores.

Si un proceso termina, entonces todas las hebras pertenecientes a ese proceso, tambien terminan.

Si el proceso sale/muere, entonces como las hebras exiten dentro del proceso, entonces deja de existir. Este es un concepto antiguo, ya no es muy utilizado.

Ahora las hebras se planifican de forma independiente de las hebras, tal que el proceso tiene multiples instancias de proceso, tal que ya no se planifica proceso, sino que se planifican las hebras independientemente como si fueran procesos.

Cuales son los beneficios de multiples hebras de ejecucion? Principal: la creacion de un proceso que tiene que replicarse a si mismo es muy caro cuando se hace fork. Ya que el fork, al crear copias exactas de si mismo siendo que es muy pesado este proceso, entonces fork tiene la gran desventaja que copia y crea copias exactas, tal que es muy caro, hay qu eprocurar memoria, nuevo id en sistema… pero si tengo un servidor que fuera.

Si yo en vez tengo un servidor que fuera muy quebreado, lo que creare sobre un proceso es que creo 4 hebras, tal que tendrè 4 hebras. Cuanto cuesta un contexto? Cuanto pesa?

Si pensamos por ejemplo sobre un stack, voy a tener unas pocas variables, tal que voy haciendo push y pop sobre la informacion del stack, el contexto de un proceso van a ser los registros del procesador, por lo tanto estos son pocos kilos a diferencia de 250, tal que el proceso va a poder.

Beneficio de las hebras: la creacion de un proceso duplicado fork tal que sea un servidor que crea multiples procesos hijos exactos a el, tal que el proceso pesa 250 mega bites, entonces si tengo 4 hijos voy a tener 1gb + 250 mega bites del padre. Entonces fork tiene la gran desventaja que copia y crea una copia exacta del rpoceso padre, eso tambien es costoso en tiempo, hay que procurar memoria, hay que poner nuevo id del sistema,

si yo tubiera un servidor que fuera multi hebreado entonces tendrè varios contextos de hebra sobre el mismo proceso, tal que tengo supongamos 4 herbas, el contexto para cada hebra? Por ejemplo que necesito un stack donde va avanzando la ejecucion con push y pop, tal que voy a tener pocas variables, tal que cada contexto es el conjunto de registros del procesador, tal que cada contexto pesa pocos kbites

a diferencia de 250 gm, tal que entonces las hebras van a compartir el mismo codigo, tal que siga escuchando. Entonces tengo instancias de ejecucion, este es uno de los grandes beneficios de multihebreado.

Ahora se usa multihebra, no multiproceso en las aplicaciones. Juegos, servidores, base de datos, browser, todo es multihebreado.

El concepto ha superado a multiples procesos, pero esto tiene algunas desventajas.

Cuando tenemos multiples procesadores, entonces las hebras pueden usar multiples procesadores, tal que tengo computacion paralela.

Pero este no es un beneficio tan real tal que sea que tenga una arquitectura multiproceso, entonces cada proceso va a procesarse en uno de los procesadores, tal que deben comunicarse con algun metodo especial como pipes y mensajes, pero las hebras se comunican a traves de memoria compartida, tal que comparten los datos globales del proceso.

Esto significa que sea el codigo, datos globales, entonces sea un arreglo en los datos globales arreglo a, si la hebra negra accede a alguno de estos elementos y lo lee, si hay otra hebra que quiere leer los elementos tambien puede, tal que son un mismo proceso en mismo sistema, tal que el area de memoria compartida de datos globales es la misma para ambas hebras, esto se puede transformar en una ventaja ya que no se necesita invocar al kernel o servicios delkernel para comunicarse, por ejemplo si dos procesos quieren comunicarse pueden hacerlo por pipper, pero las hebras simplemente pueden ir a los datos globales sin kernel. Pero supongamos que hebra roja quiere escribir un dato y una hebra verde esta leyendo los mismos datos, esto es una condicion de carrera, tal que dos hebras leen y escribden datos al mismo tiempo esto es un peligro de concurrencia, tal que permite a las hebras comunicarse sin la necesidad de invocar al kernel tal que es muy rapida y eficiente, pero es una desventaja ya que como se comparten los datos pueden llegar a na condicion corrupta, tal que se producen errores en los datos, tal que quedan inconsistentes.

El heap es compartido de una manera especial. Supongamos que tengo main() y dentro de este hay un char \*p, viene una hebra roja y a p le asigna malloc(100) luego hebra verde hace malloc(100). Hay un problema o no hay un problema?

Lo que salva la situacion es saber que main es una funcion como toda otra funcion, por lo tanto p es una variable automatica, que se maneja en el stack de las hebras, tal que p es local a la funcion main. Entonces puedo tener diferentes funciones, pero son invocadas tal que se ejecutan en el proceso, pero tambien usan el stack de usuario, tal que el p es distinto, tal que va a estar en los stack, tal que en el stack de la hebra roja esta p, en el stack de la hebra verde esta en p. pero el heap es el mismo, tal que p rojo va a apuntar a 100 bites tal que en memoria comienza desde 0 a 99 y p verde va a apuntar a 100 bites arriba de p rojo tal que p+100, tal que se van mapeando las direcciones dinamicas, tal que es compartido pero es local a la hebra, entonces no se sobreescribe.

El area de heap es donde se mapea la direccion dinamica, aunque es compartido, cada p es local de la hebra, no se sobreescribe.

Pero si en c escribimos p+101, tal que la hebra roja puede accedel al heap de la hebra verde ya que es compartida, no es protegida entre ellas, es compartida, puedo arreglarmelas para dejar al escoba.

Otro beneficio de la shebras es que un proceso se estructure con background y foreground. Una app puede tener varias hebras que estan bloqueadas o dormidas, yy que se activan para acer labores de limpieza cada cierto tiempo, tal que un proceso que esta procesadndo archivo y generando archivos de log y debe borrarlos, la actividad de hacer archivos de logs no para, esta leyendo, estudiando, transacciones y monitoreo de red y esta haciendo muchos log. Otras hebras cada 3 dias se activan y borran los logs. Esto permite que el monitoreo de red nunca se detenga, porque si la hebra se detiene para borrar logs, deja de hacer la actividad de leer. Lo mismo con interfaces, tal que hago edicion de fotos, entonces yo puedo estar hechando a correr una actividad con complejos procesos, pero en este sgundo puedo seguir haciendo otras cosas como una nueva imagen, tal que la interfaz es atendida por una hebra y si invoco una funcion de procesamiento, esta es atendida por otra hebra, tal que sigo atendiendo al usuario sin bloquear la antividad anterior.

Sea un proceso monohebreado de 1 proceso, tal que hace un rpc (remote procedure call y remote metod invocation) solicitud de servicio que no esta en maquina, esta en otro lugar, tal que compro por ejemplo un bono y hay que poner la huella, tal que lo envia con servidor que presta servicio por ejemplo registro civil tal que pregunta y espera respuesta el procesador, tal que esta esperando respuesta bloqueado porque es una operacion de io, tal que es una escritura por io y ahi queda una lectura hasta que llegue de vuelta el mensaje y se activa y sigo ejecutando running, luego hago otra solicitud y se hace la espera y luego sigo ejecucion cuando llega. La gracia de tener proceso multihebreado, tal que invoca un proceso de io, tal que se bloquea, la hebra se bloquea pero el proceso puede inmediatamente rcear otra hebra independiente de la que quedo bloqueada para que emita un segundo requerimiento y al llegar respuesta la hebra 1 sigue ejecucion y la hebra 2 queda en ready asumiendo que es un solo procesador, tal que solo 1 proceso dentro de 1 procesador intercaladamente, tal que la hebra roja al bloquearse le da paso a la hebra azul.

Entonces sea que 1 hebra se bloquea, en monohebreado, todo se bloquea. Por lo tanto es mucho mas eficiente con multiples hebras que se van intercalando si una se bloquea.

Esta caracteristica que ocurre al intercalarse cuando otra hebra se bloquea pertenece a un tipod ehebra particular llamada hebra a nivel de kernel. Esto va cambiando todos los años, actualmente todos los procesos son multiples hebreados tal que casi no existen hebras a nivel de usuario.

Una hebra a nivel de usuario tal que es simulada, tal que sea un proeso pero las hebras no son entidades que existan para el kernel, el kernel solo conoce una hebra que es monohebreada, y gracias a una libreria de simulacion hace como que hay multiples hebras, tal que hace diferentes stacks, contextos para cada una de ellas, y la hebra de la libreria va a ir alternando las multiples hebras, y avanzando en la ejecucion de las hebras de usuario. Entonces las hebras de usuario estan simuladas, que no son conocidas por el kernel. El kernel solo conoce la hebra de la libreria y la va planificando tal que la hebra de libreria es una hebra de kernel, tal que la ha idenficiado como tal, la conoce, y las hebras de usuario son desconocidas.

Que sucede si una de las hebras de usuario hace un io? Lo que ocurre es que la hebra de kernel hace io, tal que queda bloqueada, y no puede seguir simulando el resto de las hebras, tal que ninguna puede seguir su ejecucion, cuando una hebra se bloqueapor un servicio, entonces todas las hebras se bloquean, ya que en realidad la que se bloquea es la de kernel. Sea que tngo otros procesadores, por ejemplo 3, no puedo darle una hebra a cada procesador, porque solo existe la hebra kernel, ya que las otras esstan simuladas, tal que no es explotable el paralelismo.

Las hebras a nivel de kernel son las hebras que existen ahora, tal que es una hebra separada conocida por el kernel tal que tienen su thread id. (antes teniamos program id, process id) cada una es planificada a un quantum. Tal que sea que una tread id hace un io, esta se bloquea, pero la otra hebra sigue, tal que pueden explotar el paralelismo en diferentes procesadores si es que hay varios, tal que sigue bloqueada la hebra que hizo io.

Las hebras pueden entrar al espacio del kernel, esto implica que kernel es multihebreado, tal que el kernel como codigo debe estar preparado para que entren varias hebras a su espacio.

Entonces sea kernel, y tengo un proceso con varias hebras, tal que hebra azul invoca read, lo que ocurre es que va a kernel con un cambio de modo usuario a privilegiado, tal que ahora la hebra esta ejecutandose en el kernel.

Sea otra hebra roja que entra al kernel para hacer pipe, tal que hay cambio de modo. En el kernel hay varias hebras ejecutandose, tal que es diferente hebra modo usuario a hebra nivel de usuario, tal que hebras pueden tener diferentes modos, modo privilegiado o usuario, tal que no las hace que sean hebras de usuario, son cosas diferentes. Entonces necesitamos un kernel especial que permita multiples hebras, tal que es un kernel simetric multiprocesser.

La caracteristica de que son las hebras que entral al kernel.

El simetric multiprocesser

es la caracteristica de 1 sistema operativo, de su kernel para poder permitir la ejecucion de multiples hebras de su codigo, tal que el kernel permita multiples hebras en su espacio. Hace años, los kernel estaban clasificados, tal que eran monohebreados o smp. Entonces habia que elegir entre estos dos tipos de kernel.

Ahora el kernel son todos smp.

Sistema multiprocesador:

Entonces lo que sucede es que sea un tipico sistema computacional con multiples processador, el io, el mem.

Pueden compartir por ejemplo un l2 (Cache) tal que l1 es el cache para cada procesador.

No hacemos diferencia entre muchos chips con 1 core o chips con multiples cores. Para el sistema operativo es una unidad de sistema de procesamiento, tal que es de memoria compartida, tal que tengo una sola unidad de memoria.

A esta maquina hay que instalarle un sistema operativo, tal que le puedo instalar un smp o un no smp.

Un smp es aquel que el kernel puede ejecutarse en cualquier de las unidades de procesamiento. Tal que peudo tener en cada procesador el kernel.

Tal que son independientes las hebras, dos hebras ejecutando en el espacio de kernel yotra en proceso usuario. Entonces estas hebras por ejemplo podrian planificar, tal que planifican por ejemplo hebras nuevas. Entonces el kernel puede ejecutarse en cualquiera de los procesos.

Pero tambie puede ejecutarse solo en un procesador p0, este es el estilo maestro esclavo, tal que es el unico permitido para que planifique y el usuario puede usar solo procesador 1 y 2. entonces sea que p2 entra a kernel, debe esperar a que p0 se desocupe ya que solo puede ingresar a kernel en procesador p0, esto lo restringe bastante.

Clase 11:

hemos estado revisando el concepto de hebra, las hebras no son algo distinto a los procesos, no son un concepto de ejecucion distinto, sino que es simplemente la caracteristica propia de los procesos de ejecucion, tal que lo que entra a los procesadores es el contexto de ejecucion, para crear una hebra lo que basta es tener otro contexto, stack, tal que comparten proceso de memoria, tal que comparten codigo, pero al estar dentro de un mismo proceso entonces comparten varias cosas como direcciones virtuales, codigo global ,heap, librerias, etc. entre ellas no hay conflicto de ejecucion mientras no escribamos a memoria global. Cuando se comunican se puede hacer en variables globales, tal que puede avisar con flags, pero esto da problemas de sincronizacion y concurrencia. Entonces veremos como crear estas hebras en linux.

Sea posix, tal que es un standar de hebras en lenguajes c, no es una libreria, sino que es una especificacion muy abstracta o completa, pero no dice como ser implementada. En cambio libreria es implementacion de standar. En estricto rigor hay multiples librerias de posix, pero solo 1 stardart. Por ejemplo solaris puede implementar standart, pero deben satisfacerlo regardless.

Como usar esta api de posix en linux. Estas api provee 2 conjuntos principales de funciones: administracion de hebras, sincronizacion de hebras. Ahora se vera solo admin.

Creacion, destruccion, detaching, joining.

Las hebras poxixs, pifrex, no es lo mismo para api para programacion paralela. Multiples hebras no es paralelismo, pero existen muchas ventajas de multihebras para monoprocesador, pero si tengo smp, las hebras pueden aprovechar paralelismo. Pero no son lo mismo, ya que no esta orientado a darle al programador funciones que faciliten su labor para construir aplicacion paralela, esya diseñada para que el programador cree, destruya, sincronize hebras. Estas labores son necesarias para programacion paralelas, pero tambien esta open mp que si es un standar para programacion paralela en memoria compartida. Y esta basado en hebras, no necesariamente pifred. Openimp da un monton de soluciones y herramientas para programar en paralelo. Todo lo que hago en openmp lo puedo usar en pthread (pifred, escuche mal) pero en pthread thengo que programar y hacer muchas cosas, es decir que con estas herramientas como open mp me provee las funciones para facilitar la programacion paralela.

Entonces como construyo hebra en programa en c. primero, debemos hacer el include pthread.h que incluye definiciones. El tipo pthread-t esta definido en pthread.h tal que es un tipo conocido y definido, tal que no alega. Entonces creo una hebra con phreadcreate. Revisaremos sus argumentos. Primero es el identificador de la hebra, el thread id. Este thread id es thread , en prinicpio la especificacion, el standar, no me dice que tipo pthread es.

Lo que dice es que si uso esta variable, o mejor dicho si identifico una hebra basta con usar esa variable, pero no que pthread va a ser implementado de una forma especifica, tal que todas las funciones que actuen sobre el identificador actue como esperamos. Es decri que estas estructuras son opacas, no nos interesa como esta definido pthread, lo que si se es que la variable thread es el identificador unico de la hebra en todo el proceso y es una variable de salida, por eso es un puntero. Luego vienen los atributos de creacion, uno puede crear una hebra con ciertas propiedades, o caracteristicas, en esta estructura uno puede setear estas propiedades, generalmente con estas propiedades por defecto bastan y simplemente pasamos un null para que se tomen las propiedades por defectos. El siguiente argumento es la funcion que la hebra va a ejecutar. Entonces en este minuto la especificacion va a ser que solo se va a poder crear una hebra a traves de una funcion. Toda hebra nace dentro de una funcion especificada en este argumento. Esta funcion debe tener 2 caracteristicas principales: 1 argumento de tipo void\* y debe retornar nada. La funcion que inicializa mi hebra no puede recibir un puntero a void y no retona nada, eso es parte de la especificacion. Ultimo argumento son los argumentos que va a recibir el void\* tal que se los paso. En caso de exito el pthreadcreate retorna 0, sino retornara cualquier cosa menos 0.

muchas de estas estructuras son opacas. En el sentido que no nos interesa como estan definidas. Muchas veces usamos valores por defecto nulo y otras veces funciones que el standar provee para acceder a cosas de la estructura como getters y setters. Si quiero cambiar un atributo de esta estructura por ejemplo, puedo usar pthread attr setname y si quiero leer la estructura getname, donde el name es el nombre del atributo.

Estos son algunos de los atributos relevantes para la creacion de la hebra. Recordar que una hebra tiene propio contexto y stack, por defecto el sistema le va a crear su propio stack que no es accequible al programador, pero puedo darle los parametros del stack y sus direcciones de memoria, tal que debo procurar memoria para el stack y debo pasarle los atributos a pthread atributes, si le apso nulom entonces el sistema vera donde pone stack y donde lo maneja, pero si yo lo creo entonces puedo ver lo que hay dentro del stack, ya que al darselo es una direccion de memoria que tengo en el proceso, tal que tengo acceso al contenido del stack de la hebra. Otro atributo importante es la politica de planificacion de la hebra. Una vez que ya tengo hebras, no es que se este planificando el proceso y aparte de las hebras, sino que sean todas hebras, entonces las hebras para que entren a procesador deben competir para entrar a procesador, tal que voy a crear politicas de estplanificacion prioridad para las hebras, como fifo, round robin, etc.

principalmente dicen que cada hebra sea planificada para cierto algoritmo, si le doy nulo, el defecto es sched other. Esta es la planificacion de default para la mayoria de los procesos. Estas son politicas, y le puedo pasar parametros para estas politicas. Por ejemplo para round robin le puedo pasar el quantum especifico de tiempo. Otro parametro es el estado de desacoplamiento de la hebra. Por defecto las hebras se crean joinable. O attachadas a la hebra padre. Si yo quiero crear la hebra desattachada seria un parametro especifico para entregarle a el creador de hebras.

Estos atributos son estructuras opacas, pero lo que hacemos es usar los getters y setter para cambiar valores. Muy importante es inicializar los atributos por defecto, aunque quiera cambiarlos despues, es buena costumbre siempre inicializar los atributos de la hebra, aunque sea por defecto,

entonces este es el concepto: lo que hago es lo siguiente (lenguaje c) sea el programa principal, la hebra principal, tal que es la hebra de la funcion main, es la unica que se ejecuta en el main. En main va a haber solo 1 hebra siempre. Sea que creo una hebra pthreadcreate (id de la hebra, los atributos de creacion que ahora sera null, la funcion de creacion de la hebra que sera task, argumento que le voy a pasar a task por ahora es null) la hebra de main continua la ejecucion. Entonces sea otra funcion que es task, necesariamente es void task\*(void\* arg) entonces tengo int j y codigo. Cuando ocurre el pthread, entonces la hebra creada comienza a ejecutarse, se ejecuta en la funcion task y recibe argumento nulo, tal que son argumentos de creacion null. Ahora la hebra principal podria volver a crear otra hebra a \*task, nuevamtnte hago pthread create(id2 de hebra, nulo, task, void\*i) tal que se crea una nueva hebra con nuevo stack y contexo, tal que no habrian problemas con las 2 hebras que estan en misma funcion, tal que la nueva hebra recibe como argumento en la funcion task el argumento I.

Una vez que las hebras han sido creadas en la funcion, pueden estas a su vez crear otras hebras por ejemplo en task hay un pthreadcreate() y asi sucesivamente. Una vez que la hebra esta creada, puedo asignarle supongamos dentro de la hebra creada en task hay una funcion que cambia j, pero no es posible que una hebraque haga una hebra diferente para esta funcion, ya que deberia crear una funcion que fuera void\* que llame a las funciones que yo necesito en el siguiente argumento.

Cuando termina la hebra? La hebra roja termina cuando sale de la funcion o termina la hebra. La hebra roja no es que vaya a ejecutar loque dice que ejecute como por ejemplo no entran a la funcion, sino que crean una hebra para que vaya a la funcion, las hebras padres no se meten a hebras, sino que crean una hebra hijo para que haga la funcion, pero al terminar esta funcion esta hebra muere, por ejemplo pueden terminar de las siguientes maneras: termina funcion, exit de pthread, pthreadcancel, si es que el proceso que contiene todas las hebras termina, entonces todas las hebras se daran por terminadas.

Sea un programa principal, la funcion donde van a nacer las hebras, pthread id, pthread attr, I, status,5 ysi escribo pthreadattrinit(art) es inicializado null, si creo 5 hebras en un loop, luego de crearla, hay un pthread join tal que la hebra se queda esperando a la hebra que creò hasta que la hebra que termine, luego de esperarla vuelvo a crear otra hebra y asi iterativamente, esto de esperar a la hebra mientras se cree tiene que ver con lo desatachado, todas las hebras se crean por defecto joinable atachadas. Esto siginifica que le padre puede esperar por ella y le puede hacer un join.

Joinable atachado: todas las hebras sonc readas atachadas al padre, tal que la hebra main padre puede esperar y atachar sus hebras con un pthread join. Esto implica que la hebra padre se va a bloquear en pthread join, tal que tenemos la hebra principal que crea otras hebras en el main y estas comienzan a trabajar de forma independiente que le padre, y estas 4 compiten por el procesador. No existen una prioridad entre las hebras padres e hijas, pero si es creada atachada y el padre espera por sus hijas, entonces el padre se bloquea y las hijas trabajan, cuando la hebra invoca un ptread exit, voy a join que es el valor de retorno de la hebra hija. Entonces sea que espero por la hebra hija me debe pasar el valor de retorno. Este es una doble indireccion a una indireccion, es bien sucio, cuando se ocupa es demasiado puntero y no es tan facil, recordar que la hebra padre se bloquea, la hebra hija al hacer exit le entrega un valor de status. Si la hebra hija es desatachada o not joinable, la hebra padre no puede esperar por la hebra hija. En el sentido que el pthread join no va a tener ningun efecto. O lo mismo decir que el pthread join no bloquea a la hebra padre, es decir que retorna inmediatamente, no puede recibir ninguna informacion de la hebra hija, este valor status no va a recibir nada por ahi, ya que no es posible de crear, no se bloquea, no sabe cuando termina la hebra de la hija. Entonces depende de lo que quiero saber o hacer. Entonces para esto es pthread attr setdetachstate( …, detached)

entonces tenemos un programa tal que defino el id para la hebra, estructura para los atributos, leo argumentos, inicializo la hebra por defecto tal que es acoplada, y la creo: Puntero a id, puntero a atributo, funcion void runner, argumento 1 de la linea de comando, el uso de este programa, el numero va como string como void asterisco (re sucio) y luego entero, lo que deberia entregar un nul.

Esto secompila con g++ -npthread y la libreria que quiero linkear.

Entonces supongamos le pasamos argumento 3, lo que pasa es lo siguiente. Primero hebra hija imprime y imprime luego la suma. La suma dentro es 6, tal que el ejercicio es que la hebra hija entregue la informacion a hebra padre. Entonces la hebra hija deberia entregarle desde el exit al join al status, rdyo no sucede ya que no hace lo que esperamos. Esto no es tan obvio (al profe no le funciono xd) tal que el padre puede desatachar a la hebra hija, tal que sea un loop infinito, ya que el padre espera por la hebra hija y no hay hebra hija, porque va a estar esperando para siempre. Este proceso thread1 es hijo de la bash, ademas, tiene 1 hebra hija, tal que cada vez que tenemos entre parentecis llaves seria la cantidad de procesos que tiene es decir hebras. Y asi casi todos los procesoss son multihebreados con n hijos. Tal que la hebra hija ahora estaria consumiendo el procesador, la vamos a desatachar ahora, se vuelve a compilar, se ejecuta, y vemos que, el padre queda esperando. La hebra padre esta haciendo un pthread exit, tal que obliga a la hebra del padre boquearse esperando a que terminen sus hijos, pero el hijo esta en loop infonito. S es que en vez de eso uso exit, altiro termina, ya que padre no esta esperando a hebra hija, entonces elimina al hijo en terminar, tal que elimina el loop infinito.

Este ejemplo es importante ya que ocurre que quiero crear muchaas hebras y para esto necesito una arreglo de id para las hebras, cada vez que creo una hebra la guardo en la posicion del arreglo correspondiente. Le paso mis atributos, todos ejecutan ranner, i es parametro y se castea a thread number, tal que debe imprimirlo, no confundir id con la creacion de id numericos del indice del loop, son totalmente diferentes, ya que el primero sirve para trabajarlo, el otro es un id correlativo.

Siempre en c es un problema el casteo, siempre da warnings, puedo eliminarlos a mi conveniencia con flags. Estoy tratando de usar el id entregado por el sistema para imprimirlo como un entero, esto no es correcto ya que es una estructura opaca que puede o no ser convertible, en estricto rigor uso el correlativo para que pueda identificarlo, pero guardarlo aquí como el verdadero.

Se crean 2 hebras, el loop es que el padre crea la hebra pero no necesariamente aparecen las cosas en orden , porque las hebras se planifican según un algoritmo de planificacion lo que no garantizan un orden que yo quiero, puede que esto sea desordenado cuando tenga multiples hebras independiente de cuando yo las cree. Que pasa si le paso el puntero a i? Caos. Lo quito, todo funciona bien. Mucho cuidado con cada paso.

--Clase 12 y 15 fueron planificacion--

-clase 13 y 14 fueron ejercicios--

Clase 16:

hemos visto fundamentalmente las labores de sistema operativo que se encarga de manejar procesos y hebras, tal que son los usuarios principales del sistema operativo, son los que necesitan memoria, io, mandar señales a otros procesos, tiempo en el procesador. Sistema operativo esta orientado a asignarle estos recursos. No esta orientado a satisfacer a usuario humano, para esto es las aplicaciones. Las aplicaciones le piden recursos a sst operativo. Esto es conciderando que cada proceso y hebra es independiente a los otros. Concurrencia y sincronizacion es cuando dos tareas llamese proceso o hebra comparten recursos en forma concurrente. Para nosotros una tarea es un proceso o una hebra. Cuando 2 o mas tareas trabajan en conjunto, entonces deberian compartir algun recurso como un archivo, memoria, datos, eso hace que las cosas cambien. Al compartir recursos hace que existan problemas cuando el proceso trabaja en forma aislada, no debe preocuparse de nada. Pero si tengo 2 hebras que trabajan conjuntamente, debemos forzar mecanismos de sincronizacion son mecanismos a nivel de programacion, tal que se debe programar de tal forma que las actividades de estos procesos esten sincronizadas. Si no ocurre esto, entonces seva a producir una corrupcion de recursos compartidos. Por ejemplo que el dato que estan compartiendo queda en un estado inconsistente.

Esa corrupcion es un riesgo. Existen riesgos, puede ser que no ocurra nada, pero nos interesa eliminar todos los riesgos, no podemos dejar a lo aleatorio que suceda o no sucedan estos riesgos. Para esto existen varios ejemplos de trabajos en conjunto. A esto le llamamos en forma concurriente. La concurrencia es relevante porque es un requisito fundamental para el capitulo de hebras, ya que cuando al ocurrir aislada no es concurrente ya que no esta compartiendo recursos con nadie. La concurrencia no es lo mismo que paralelismo, pero se parecen mucho. Debemos tener claridad para diferenciarlas.

Por ejemplo, para ejemplificar, la concurrencia se diferencia de paralelismo ya que concurrencia es cuando 2 o mas tareas se ejecutan intercaladamente en un mismo procesador entonces supongamos que tengo 2 hebras rojo y azul. Sea que hebra roja se ejecuta en cierto periodo quantum de tiempo, luego hay un cambio de contexto y comienza la ejecucion de la hebra azul por un quantum de tiempo. Reanudo hebra roja que continua con la ejecucion.

Eso es la ejecucion concurrente, ya que van al mismo tiempo en el procesador, pero van en proporciones de quantum de timepo.

Paralelismo es cuando 2 o mas hebras o tareas se ejecutan al mismo tiempo en procesadores distintos.

Tal que en concurrencia es el mismo procesador con diferentes hebras. En cambio en paralelismo son dos procesadores simultaneamente se procesan ambas hebras.

A su vez entonces en cada procesador del paralelismo puede haber concurrencia, tal que pueden estar en concurrencia 2 hebras en cada procesador.

Esa es la diferencia entre concurrencia y paralelismo. Para paralelismo necesitamos varios procesadores en cambi en concurrencia necesitamos solo 1 procesador.

El hecho que existan 2 procesadores, se necesita que sea un programa multihebreado, tal que segmentar el proceso rojo en diferentes partes para que trabajen paralelamente los segmentos. Aunque tenga miles de procesadores, mi hebra seguira en un solo procesador.

Los problemas de sincronizacion no solo ocurren en sistemas multiprocesadores paralelos isno que tambien en monoprocesadores. Esto es importante ya que podemos quedar con la idea que los problemas ocurren solo en paralelismo, pero la verdad es que puedo tener muchos problemas con concurrencia. Esto tambien va a ocurrir si tengo multiprocesadores, pero no es una condicion esencial de tener multiples procesadores para que tenga esto sproblemas.

Esto es porque en monoprocesadores falta un monoprocesamiento para tener concurrencia, esto es una caracteristica de una funcion intercalada en un procesador.

Cuale sosn los beneficios de concurrencia, esos son los mismos beneficios de multivareada. Al tener multiples procesadores, voy a aumentar mi rendimiento. Pero esto requiere concurrencia, tal que sea una aplicación multihebreada, tal que el aumento de rendimiento de la aplicación es solo si fue construida en multiples hebras, sino nova a haber aumento de rendimeinto. Otra caracteristica que no tiene que ver con paralelismo, es sobre la interactividad de las aplicaciones. Esto es en un monoprocesador. La interactividad de las aplicaciones ocurre cuando estoy en un brownser y en una pestaña busco un sitio. Si este sitio se demora, abro otra pestaña con otro sitio, y lego puedo seguir asi, tal que cada busqueda de cada pestaña esta siendo atendido por una pestaña distinta. Antiguamente no se podia hacer eso, ya que un proceso al hacer io se bloquea, por lo que cuando busco un sitio es que hago un io y queda bloqueada mientras busco la respuesta y si mi app no tiene el poder de hacer mas hebras, entonces quedara bloqueada y no podre crear otra pestaña. Otro ejemplo son los ides, por ejemplo visual code, code blocks, atom, etc. estas aplicaciones de forma automatica graban en disco lo que estoy haciendo, esto lo hacen una hebra que no veo, tal que el ide esta atendiendo por ejemplo edicion de texto, otra el menu, otra que esta en el background que no veo como actua que esta cada ciertos milisegundos guardando lo que hago, tal que si se va la luz, va a estar guardado todo.

Todo esto que esta aquí es para monoprocesadores de lo que estoy hablando.

Porque lo que van haciendo es que un proceso nunca acapara completamente, tal que es una variacion de un round robin, por lo tanto un proceso nunca acapara completamente un procesador, siempre hay cambios de contexto. Tal que hay muchos contextos concurrentes. Todo esto en un solo procesador.

Recordar que los quantum son del orden de 500 milisegundos, tal que parece como que todo esta siendo llevado a cabo a la vez, tal que es todo lo que hace antes del cambio de contexto y el overhead es del orden de microsegundos para cambiar de contexto. Cuanto puedo haber hecho en 5 segundos? Millones. Tal que parecen como paralelas.

Por lo tanto existe tambien paralelismo en los computadores. La mayoria de las aplicaciones son multihebra. Todas las aplicaciones que se construyen, todos son multihebra, porque todos los procesadores de hardware son multicore, por lo que no tendria sentido que las app fueran monohebrea. Todos son hebras de kernel. Las hebras de usuario quedaron en el pasado, no existen actualmente.

No pensar que hebras de kernel son hebras del sistima operativo.

La hebra de usuario es la hebra que se simula como si fuera una hebra distinta, cuando los sistemas operativos no permitian multihebra entonces las librerias simulaban multiples hebras, por lo que la aplicación era monohebreada que simulaba multiples hebras.

Esta app ejecutaba un poco de cada hebra, es decir simulaba como si existieran varias hebras pero era una sola. Una hebra de usuario no puede utilizar los multiples procesadores, no hay paralelismo cuando tengo hebras de usuario, no existen. Solo las librerias que las simulan. En cambio hebras de kernel son verdaderas que son planificadas por el sistema operativo tal que tienen su propio id en sistema operativo, tal que en linux no procesa procesos o hebras, sino que todos son tareas y son planificados como tal, tal que una hebra de kernel es una hebra que puede hacer uso de los multiples procesadores, todos los laboratorios seran multihebra con multinucleo.

Entonces sea que tenga mis 8 cores, tal que todos tienen actividad. Zoom es una aplicación multihebreada y estoy usando mis 8 nucleos. Ustedes cuando hechamos a correr algo multihebreado activan todos los nucleos.

Hoy vamos a ver el siguiente problema de cuando los datos quedan en un estado inconsisitente. Sea que tengo una lista que se llama list y es una estructura concurrente en el sentido que hay 2 hebras que estan insertando datos en la lista,

supongamos que en este minuto tengo 2 elementos. Entonces este seria list. Supongamos que tengo el elemento 9 y 2, entonces tengo 2 hebras que insertan elementos a esta lista. La hebra roja inserta 5 y la hebra verde inserta 8. tal que cuando hacen la insercion? No lo se, mis dos h ebras lo hacen concurrentemente por lo que no se cuando los van a insertar.

Supongamos que la hebra roja entra primero a hacer el insert de 5, tal que hace un malloc, tal que p apunta a un malloc que he creado para meter 5. lo mismo hace verde, tal que hace un p que apunta a malloc que ha creado para meter 8 y hace p next tal que asigna a la lista. Luego hay un cambio de contexto, vuelve a la hebra roja y la hebra roja p next le asigno a list. Luego este puntero lo muevo a la hebra roja y sale la hebra roja tal que a list le asingo p tal que el puntero queda en el insert de verde. Tal que ambas han efectivamente insertado, pero quedo un desorden. Tal que ahora tengo una lista 8, 9, 2. pero puede haber quedado tambien 5,9,2. las unicas listas validas deberian ser 5,8,9,2 o 8,5,9,2. ambas son validasy no importa cual de las 2, lo concurrente soporta cualquier orden, ya que no importa que el 5 o el 8 esten en otro orden, pero ya no es valido que pierda el 5 o el 8. tal que este es un estado corrupto, ya que puedo seguir recorriendola y puedo no darme cuenta que perdi uno de los inserts.

Esto que ocurrio aquí es una condicion de carrera, tal que el resultado de operación depende del orden de ejecucion de 2 o mas hebras.

A veces el orden de ejecucion produce resultados correctos o consistentes, tal que pueden haber resultados correctos. O consistentes. Pero tambien puede estar el riesgo que este incorrecto. Por lo que una condicion de carrera presenta errores posibles potenciales de concurrencia. Una condicion de carrera siempre es un riesgo. De que mi programa funcione o se caiga, o produsca datos incorrectos. Nosotros queremos eliminar todos los riesgos.

Como elimino los riesgos? Como quito las condiciones de carrera? Debemos identificar los riesgos y eliminarlos en la sincronizacion.

Nuestro objetivo es identficar y eliminar los riesgos uno a uno.

Conceptos:

seccion critica: trozo de codigo ejecutado por multiples hebras donde accedo a datos compartidos y al menos una de las hebras escribe sobre los datos. Tal que es un pedazo de mi programa y en ese pedazo leo y escribo datos. Tal que debe haber al menos una escritura de datos. Esto quiere decir que si ninguna hebra modifica los datos, si ninguna hace un write, entonces no hay seccion critica. Tal que si solo hacen read de dato compartido no hay seccion critica, si nadie escribe en a ya que la lectura no modifica los datos. Pero puede ser que en otra parte del codigo haya algun otro codigo con seccion critica que hace un write sobre ese mismo dato. Entonces ahi tendriamos que hemos identificado una seccion critica que seria el pedazo del codigo que inside en el otro pedazo de codigo, ambos. Pero si nadie escribe en ese dato no hay seccion critica.

Si no hay recursos compartidos no hay seccion critica, tal que al no haber recursos compartidos como listas, etc, no hay recursos compartidos.

Si el programa es monohebra no hay seccion critica. No confundir con monoprocesador o monoprocesamiento. Monoprocesamiento se refiere a un procesador, en cambio monohebra se refiere a solo una hebra ejecutandoce. Puedo tener monoprocesamiento con multiples hebras, por lo que si podrian haber secciones criticas. Las instrucciones p next = list y list = p, es donde acceden a recurso compartido tal que tenemos 2 hebras, se modifica la lista con un write, tal que es una seccion critica, porque estoy modificando y escribiendo, tal que al menos en uno de sus accesos es write y el otro es la consecuencia.

Porque p no es seccion critica? Tal que es la creacion del puntero malloc(). Donde se procura p? Donde p se aloja. Esta en el stack de la herba, tal que traen sus propiios stacks, tal que p se procura de forma local en los stacks de cada hebra. Una cosa es p y otra cosa es a lo que apunta.

Clase 17:

capitulo de concurrencia y sincronizacion elemento muy importante para evitar errores en programas multitarea o multihebra. Principal error es que datos qudan inconsistentes aun que no den error. Soluciones para sincronizar y esto es identificar en el codigo la seccion critica tal que es un trozo de codigo ejecutado por multiples hebras que acceden a datos compartidos y al menos una de ellas escribe sobre esos datos. Segundo elemento es forzar la exclusion mutua, tal que sea en la seccion critica debemos poner requerimiento de exlusion mutua, tal que sea que una hebra se ejecuta en seccion critica ninguna otra puede estar en una seccion critica. Entonces sea un codigo identificado que tiene una seccion critica, entonces la exclusion mutua si es que pasa una hebra, no puede pasar otra mientras esta esa hebra en la seccion critica, no pueden otras hebras ejecutar esas lineas de codigo, tal que la dejo afuera. Esto es abstracto, como lo hago es deteniendo a las otras hebras en ese punto para que no continuen, leugo la hebra al salir, entonces la otra hebra puede entrar. Sea otra tercera hebra que quiere ingresar a la seccion critica, la segunda hebra va a estar bloqueandola de pasar. Entonces es mutuamente exclusiva. Tal que esto pasa en un programa multihebreado.

La seccion critica esta definida por el codigoq ue accede a datos compartidos, entonces yo estoy leyendo o escribiendo datos. Puede ser que en trozos de codigo que acceden a mis.o codigo pero son diferentes codigos que referencian a estos. Entonces sea un stack que tiene varios elementos y hay varias hebras que estan trabajando sobre el stack, entonces en una seccion accedo a top y en otra accedo a topp tambienm tal que una es pop y otra es push, entonces yo estoy consultando en ambas funciones sobre el top que es la posicion compartida del stack. Tal que ambas son secciones criticas, por lo tanto son mutuamente exlusivas porque acceden al mismo dato compartido entonces si hay una hebra en el pop no va a poder otra hebra entrar al push y viseversa y asi mismo ninguna otra hebra va a poder entrar a estas secciones criticas mientras exista una hebra en la seccion critica ya mencionada.

Entonces sea un modelo de concurrencia para inventar la solucion, por ahora estaremos modelando el problema con el concepto de la seccion critica, exclusion mutua y recurrencia. Entonces este sera el tipico codigo de concurrencia con el que trabajaremos. Tal que tengamos un proceso que tiene un loop infinito donde dentro de este loop identifico una seccion critica cs() y una vez que la identifico no me importa lo que hay dentro de ella, ya que al ser seccion critica solo debo forzar exclusion mutua, tal que creo un mecanismo de entrada y de salida a la seccion critica. Entones sea que antes de todo y luego de todo esto tenga codigo no critico, voy a tener un loop infinito. Entonces debemos crear mecanismos para la excepcion mutua de la seccion critica tal que garantize que el requerimiento de exlcusion mutua.

Lo que esta dentro de la seccion critica no es importante, solo que debe ser finito, debe terminar, no deben haber tampoco errores o programas de programacion. Nuestra labor es solo identificar los mecanismos de entrada y salida.

Entonces, vamos a inventar soluciones al enter y al exit para garantizar la exclusion mutua.

cs=sc en ingles.

La solucion que inventemos debe satisfacer 4 requerimientos requisitos.

Primero es que satisfaga exclusion mutua, tal que cuando hebra esta en seccion critica, ninguna otra hebra lo esta, tal que la ejecucion de las operaciones de las secciones criticas no son concurrentes. Las herbas en si, si son concurrentes, pero las operaciones dentro de la seccion critica no son concurrentes, tal que solo puede ser una herba a la vez.

Que no se produzca deadlock, tal que todas las herbas quedan ejecutando algo infinitamente o estan bloqueadas o en busy waiting. Entonces el deadlock ocurre porque la solucion hace que todas quedan esperando.

Ejemplificado, el deadlock es cuando por ejemplo la hebra 0 espera que la hebra 1 haga algo, y la hebra 1 espera a hebra 2, y la 3 espera a 2 y la 3 espera a 0, tal que estan en una espera circular.

Esto significca que todas estan esperando o bloqueadas.

Entonces en nuestra solucion de la solucion critica tenemos un enter critical section y luego la seccion critica, y luego el exit. En el deadlock se produce que todas se quedan en el enter, tal que la primera hebra espera a la segunda, la segunda espera a la tercera y la tercera espera a la primera. “no, despues de usted”. Tal que esperan para siempre, “abrazo mortal”.en el sentido que quedan nuncas y nunca mas continuan, debo matar el programa porque es una espera indefinida, tal que es un error de sincronizacion, esto lo produce el mecanismo de entrada. La solucion debe ser sin deadlock, el deadlodk se produce para todas las hebras.

el otro requerimiento es que no debe prodcir inanicion, tal que una hebra nunca logra entrar a la seccion critica, tal que esta puede afectar a una hebra de todas las herbas.

Entonces sea que tengo el enter, por algun motivo no se deja iniciar, luego llega la verde y la verde entra, pero la roja no pudo, y luego esta sale y llega otra hebra mas que pasa por la seccion critica y luego llega la hebra negra y lo mismo, pero sigue la hebra roja esperando poder entrar, tal que entra en inanicion, tal que se olvidan de la hebra roja, tal que el sistema entra en inanicion es un problema de error de sincronizacion y de concurrencia. Debemos permitir que todas entren en algun momento. La hebra roja nunca logra meterce a la seccion critica.

El ultimo requermiento de una solucion perfecta es el progreso: sea una hebra que se ejecuta el enter, entonces es una señal de que va a ingresar a la seccion critica. Si la seccionc critica esta libre, entonces

debe entrar a la seccion critica.

Progreso: sea el enter, la seccion critica y el exit , supongamos una hebra azul, roja y verde, tal que ninguna ha entrado a seccion critica, la hebra roja ingresa al enter, significa para nosotros que quiero ingresar a la seccion critica tal que no hay nadie ahi entonces deberian darle paso, pero si no le dan paso y que debe esperar a hebra verde, y ingresa a seccion critica, pasa por todo, sale y luego la roja debe pasar…. Esto es no progreso. Esto es porque la hebra verde no habia llegado cuando la seccion critica esta vacia, la hebra roja debe ser permitida la entrada. Entonces cuando entra al enter y la seccion critica esta libre, debe entrar a la seccion critica. Estos son todos los requerimientos para solucion pperfecta. Algunas veces no se satisfacen todas, pero la obligatoria es la 1. ya que esta garantiza que los datos esten correctos, que no hayan errores, tal que las hebras puedan ejecutarse, generalmente me podre dar cuenta de los demas requerimientos, pero lo dificile s detectar la inconsistencia y datos erroneos.

Enfatizar que un codigo o hebra puede tener mas de una seccion critica, dentro de un codigo, puedo tener multiples secciones criticas identificadas, tal que la seccion critica accede a datos. Entonces sean los datos accedidos 1 para sc1, sc2 datos 2, y sc3 a datos1.

Entonces si hay una hebra roja en seccion critica 1 no puedo acceder a seccion critica 3, tal que si hay una hebra en sc2, no puede entrar otra a la misma seccion critica. Pero si pueden aquellas que no son exclusivas como sc2 y sc3.

Entonces la excusividad de la ejecucion esta dada por los datos que estoy modificando. Debe quedar claro que no debo hacer suposiciones de velocidad o asumir orden, no puedo pensar que una hebra va a ser ejecutada primero que otra, mi orden no es el mismo que el del computador. Para demostrar solucion correcta no debo hacer ninguna suposicion sobre el orden, para demostrar algo incorrecto si puedo suponer los errores tentativos para que suceda algo erroneo, esto significa que es lo pruebo como uno de los casos de incorrectitud, pero no puedo suponer nada para aquello correcto.

Atomicidad: es otro concepto importante, operación atomica, tal que no se divide en partes, esto significa que la operación se ejecuta completamente o no se ejecuta, tal que se ejecuta sin ser interrumpida. Por ejemplo un estornudo es atomico, tal que es sin divisiones. Lo importante es que la atomicidad no es lo mismo que la exclusion mutua, pero esta no implica atomicidad. Exclusion mutua no implica atomicidad. Esto es porque en la seccion critica por ejemplo pueden ocurrir cambios de contexto. Supongamos un enter, sc, exit. Es posible que la seccion critica sea muy grande y tenga muchas lineas de codigo, tal que una hebra roja ingresa ya que la seccion critica esta libre, tal que se le acaba el quantum de tiempo en la mitad de la seccion critica, tal que hay un cambio de contexto y llega hasta ahi nomas. Entonces puede seguir la siguiente hebra por la misma cantidad de quantum de tiempo, tal que cuando la hebra roja quiere volver no puede porque esta siendo ocupada. Luego la hebra roja sigue ejecutandose pero puede volver a detenerse de nuevo tal que no alcanza a ejecutarse en el quantum de tiempo. La ejecucion en la seccion critica puede ser en partes, no es necesariamente atomica, hay cambios de contexto en la hebra y tambien de las otras hebras fuera de la seccion critica, cosa que aprovechan avanzar si tienen la oportunidad. La atomicidad en cambio, es algo que no es interrumpido. Pero si sc fuera atomico, esto significa que cuando la hebra roja ingresa a la sc entonces no va a ser interrumpida, esto seria atomico. Entonces la atomicidad implica exclusion mutua. En cambio la exlcusion mutua no implica atomicidad.

Existen tipos de soluciones con los que podremos solucionar la sc.

Por hardware, tal que son instrucciones nativas de procesador tal que me apoyo de instrucciones de procesador.

Usar lenguaje de altonivel algoritmo, software. Como c.

otro es usar instrucciones de servicio de sistema operativo que garantiza las que son mutuas.

Lo otro es usar una propiedad del lenguaje que estoy utilizando, tal que no es lo mismo que software, ya que en software implemento a mano mi algoritmo, pero hay compiladores que ya tienen estructuras sintacticas que ya implementan exclusion mutua.

El termino lock a veces se usa en forma generica para entrada y salida de seccion critica. Lock.acquire() es enter y lock. Release(). Tal que adquiero una llave magica para ingresar a la seccion critica, entonces estas operaciones son para poder entrar y nadie mas puede obtener esta llave mientras el release libere la llave para que otro lo ocupe.

La atomicidad es peligrosa ya que podemos pensar que funciones sensillas son atomicas, esto es erroneo. Lo verdadero es que atomicidad implica exclusion mutua por lo tanto no deberia proveer el enter y el exit de mi seccion critica, tal que deberia ser tan sensillo que no deberia haber problema, ya que se hace o no se hace la instrucción. Entonces en lenguaje de alto nivel podemos pensar erroneamente que hay instrucciones atomicas por la sensilles de ellas. Tal que se crea una ilusion de atomicidad por ser lenguaje de alto nivel pero al ser compilado y traducido a lenguaje de maquina lo podemos separar en muchas instrucciones. Entonces lo que realmente ocurre en una suma por ejemplo es que cargo el valor de memoria i y lo pongo en acomulador, luego incremento acomulador y lo que esta en acomulador lo vuelvo a i con un store. Todo esto para suma.

Supongamos que i parte con 5. sea inc(i) tal que incrementar, y otra hebra que hace lo mismo, tal que hebra naranja y hebra verde. Si i tiene 5 y ambas hebras son ejecutadas deberia quedar en 7. pero en la traza nos podemos dar cuenta del siguiente error: esta traza consiste en mostrar las lineas de ejecucion que ejecutan cada una de las herbas enrtonces ambas hebras por ejemplo veremos lo que les sucede. Tal que hebra a, b. valores i y acc.

Vamos a provar que puede haber error.

Entonces ejecuto ambos loads, tal que i parte con valor 5 inicializado en memoria, luego se hace un load i, acc, tal que ahora acc tiene valor 5, luego inc(acomulador) tal que el acomulador ahora tiene valor 6. luego sucede un cambio de contexto y se ejecuta b. tal que hace lo mismo, tal que se carga en i el 5 y en acomulador tambien, y luego se hace un inc(acomulador) tal que ahora tenemos un 6 en el acomulador. Entonces se hace un storage acc,i tal que ahora i vale 6. entonces de nuevo ocurre en hebra a el storage acc, i, tal que se carga el valor 6 en el acomulador, tal que el acomulador pertenece al contexto de esa hebra o proceso, cuando hay cambio de contexto hay que cambiar todos los registros del procesador, tal que cuando hebra a sale, esta se restaura en lo que habia entonces se hace un storage de acc que es 6 en i, tal que i sigue siendo 6. entonces esto no es atomico.

Para nosotros desde ahora en adelante todas las operaciones de alto nivel van a ser no atomicas, sin embargo vamos a asumir que las asignaciones son atomicas. Por ejemplo i=0 es atomico, tal que cualquier asignacion es atomica. Pero cualquiera que sea i--, i++, i=i+1 no es atomico, pero si lo que asigno simple es atomico. No debemos podenerle una seccion critica a lo atomico.

Clase 18:

elementos fundamentales para una solucion para unproblema de solucion mutua en sistemas concurrentes.vamos a tener

4 tipos de soluciones: hardware, software,sst op, compiler de lenguaje.

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Hoy veremos solucion de hardware:

la mas sensilla es deshabilitar las interrupciones. Recordar que una hebra se ejecuta hasta que se bloquea por algun evento o es interrumpida como por expiracion de quantum. Entonce spara garantizar exclusion mutua, la hebra deshabilita las interrupciones antes de entrar en seccion critica. Tal que una hebra es interrumpida siempre y cuando las interrumpciones estan habilitadas, si las deshabilito entonces la que se logro ejecutar va a ser la unica que puede ser ejecutarse. Entonces sea que tenga otra hebra cuando una hebra deshabilita las interrupciones, entonces las otras hebras no pueden entrar al procesador, la unica manera en que puedens eguir es que la hebra sea interrumpida tal que cambio el ciclo normal de interrupciones, tal que no habria forma de sacar a la hebra del procesador, tal que garantizo la exclusion mutua, tal que cuando la hebra termina vuelve a habilitar als interrupciones y luego pueden continuar las siguientes hebras, tal que una hebra nuevamente vuelve a deshabilitar las interrupciones tal que no habria nada de cambios de contexto tal que solo esa hebra tiene la habilidad de cambiar de contexto e interrumpir. Esto significa que la seccion critica se ejecuta en forma atomica si es que deshabilito las interrupciones. Porque se va a ejecutar de principio a fin sin ser interrumpida, no como en otros casos de cuando se les acaba el quantum y ocurre interrupcion para que entre otra hebra. Para deshabilitar la interrupcion en assembler es cli y para habilitar es sti. Clear interruption y set interruption, tal que estas son las de intel para habilitar y deshabilitar interrupciones, pero esto compromete el context switch, tal que cualquier labor de emergencia de alta prioridad que debe haber un proceso atendido de alta prioridad no va a poder hacerse. Esto es peligroso. Es peligroso que el proceso no puede ser interrumpido para que sst op pueda tomar el control. Esto no funciona para multiprocesador, tal que sea que tengo multinucleo tal que tengo un procesador o nucleo 0, 1, 2…. si la hebra estaba ejecutandose solo va a ser en 0, no en los otros procesadores. Entonces sea que deshabilite las interrupciones para uno de los nucleos, entonces otra hebra va a buscar otro procesador para hacer su cometido en la seccion critica, lo que es una violacion al criterio uno que es el mas importante. Funciona para cualquier numero de hebras tal que no importa cuantas hebras hayan, tal que las detiene a todas, pero no esta libre de inanicion. Esto es porque una hebra que deshabilita las interrupciones y el planificador de infinitamente permiso a otras hebras y no a una hebra x.tal que le de el paso a todas menos a una y el cambio de contexto siempre prefiera otra hebra y no la hebra x. Entonces lo que esta ocurriendo es que aquí se le rehabilita entonces se vuelve a planificar, lo que significa que puede que la hebra x puede quedar en inanicion. Nosotros no tenemos el privilegio de superusuario para habilitar y deshabilitar como usuario, solo puede el modo kernel hacer esto. Hay otras que no necesitan privilegios como por ejemplo:

Otra solucion de hardware es una que mantengo las interrupciones habilitadas. Pero lo que podemos hacer es usar otras instrucciones de hardware que no necesitan privilegios para implementar la sc. Sea una instrucción de assembler test set que su algoritmo funciona de la siguiente manera: dada una variable dada, tal que tenga valor 0, le cambio a 1 y retorno verdadero, sino es 0 entonceds retorna falso.

Esta funcion esta implementada en asembler y dijimos que todas las instrucciones de maquina son atomicas. Tal que sea la suma no es atomic aporque es de alto nivel, emcambio todas las assembler son atomicas. Las de alto nivel son compiladas en instrucciones de maquina. Entonces sea que es atomica testset atomica, entonces esta cambia y retorna bool. El cliente es otro programa principal main que va a correr muchas hebras de manera concurrente, tal que n hebras que ejecutan la funcion t. entonces hay n hebras de la funcion t. voy a tener una variable inicilizada en 0 que es bolt=0, tal qu eimplemento la seccion critica de esta manera, tal que en la funcion t accedo a variables compartidas, lo que accedo no me interesa,pero si me interesan las hebras, entonces sea que que implemento el enter como un while testset(bolt) y exit como bolt=0, entonces hago un loop dentro de la funcion, cosa que para cada hebra cuando quieren entrar al espacio critico deben checkear que pueden o no pueden entrar al espacio critico con testset(bolt) y luego de salir del espacio critico permiten a otra hebra entrar reseteandolo a 0 el bolt, tal que tengo mecanismos de entrada y mecanismos de salida.

Supongamos que viene una hebra roja y ejecuta el test set bolt, tal que bolt esta inicializado como 0, entonces anoto que bolt vale 0 inicialmente, entonces al hacer testset bolt, entonces cambio el bolt a 1 y retorno verdadero, lo niego, y entonces como esta en un while puede quedarse ahi si es que es verdadero lo que obtengo, pero al ser falso, tal que permito a la hebra roja que entre a la seccion critica. Este es un while sin cuerpo, de forma tal que pregunta por test set bolt hasta que se hace falsa, tal que cuando test set da verdaero, entonces al incresar cambio bolt a 1, sale del while y entra a sc. Luego si otra hebra quiere entrar mientras hebra roja esta dentro, entonces si ahora bolt es 0, entonces retorna falso, lo niego a verdadero por lo tanto la hebra que habia entrado vuelve a preguntar iterativamente mientras no se libere la otra hebra, seteando nuevamente al bolt = 0, permitiendo a la hebra que habia llegado que pueda salir del while para entrar al sc. Tal que esa hebra estaba bloqueada. Si viene otra hebra tal que entra a testet bolt y retorna falso, entonces me queda verdadero el while y sigo preguntando tal que queda en lo que llamamos spin lock o busy waiting. Entonces sea que la hebra roja termina, asigno bolt = 0, tal que estas asignaciones son atomicas, cambia su valor continua. Entonces ahora le toca al azul: pregunta, obtiene verdareo, retorna verdader, lo nuega y sale del while para entrar a la seccion.

Entonces un spin lock, es lo mismo que un busywaiting. Que es espera ocupada. Esto es tipico porque siempre pasa algo en este estilo. Un while con condicion que puede o no tener cuerpo, pero generalmente lo que ocurre es que vuelva recurrentemente sin cuerpo y si logra salir del while es que entra a seccion critica, tal que el loop depende de la condicion. Sea quetengo una hebra que quiere ingresar a la funcion critica, tal que la hebra llega, se le da una signacion y un quantum de tiempo, tal que parte del quantum lo ocupa en algo no critico y de repente ejecuta en este punto el spinlock, tal que quiere ingresar a la seccion critica, tal que queda en el spin lock, para ejemplificar que la hebra con el resto del quantum de tiempo queda haciendo de forma inutil la pregunta, la unica forma que cambien el valor del while es que depende a bolt, la unica manera que cambie el bolt es que otra hebra intervenga, pero esto no va a ocurrir mientras este dentro de procesador, tal que malgasta procesador mientra esta esperando y pregunta a cada rato preguntando si puede entrar.

Cuales son las ventajas de las instrucciones de maquina? No deshabilitan o comprometen la capacidad de atender procesos, no compromete la capacidad de atender procesos de alta prioridad, lo otro es que es aplicable a cualquier numero de hebras, tambien es simple y facil pero las desventajas es el spinlock, no es necesariamente tan calo pero si es complejo si la seccion critica es muy larga, tal que si es enorme, entonces esto es mucho para una seccion critica, tal que es un desperdicio tal que tendre muchas hebras bloqueadas en el busywaiting, tal que desperdician ciclos de procesador . Pero si la sc es pequeña entre 2 o 3 instrucciones esto no va a ser tan caro. Una hebra puede tener la suerte de darse una ejecucion y comienza su spinlock, porque ejecuta el enter pero puede ser que termine el spinlock y comienze la seccion critica. Puede ser que sea una seccion critica pequeña puede queel quantum la tome por completo.

Es posible la inanicion por la misma razon, todas las instrucciones de maquina tienen las desventaja de sufrir una posible inanicion ya que cuando vuelven al procesador pueden gastar todo el quantum en el spinlock, y cuando planifican otra hebra esta puede llegar a la sc y asi otras mas, tal que a la hebra en spinlock vuelven a cerrarle la puerta.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Soluciones por software son construidas algoritmicamente en lenguajes de alto nivel.

Por ejemplo en lenguaje c para 2 hebras. Entonces supongamos una sc identificada, tal que hay un enter y un exit diferentes para cada funcion, tal que turno es compartida inicializada en 0 y ejecuto 2 hebras de forma concurrente, tal que comparten la sc.suponamos hebra verde comienza, pregunta si turno es diferente de 0, entonces esto es falso, entra al while, entra a sc. Ahora la hebra azul entra y dice que como turno es 0, se queda esperando en el ciclo while que es un spinlock y estara esperando hasta que se le acaba el quantum de tiempo. Luego vuelve a la hebra que lo que hace es cambiar turno de 1 a 0. entonces ahora la hebra azul se da cuenta que su condicion es falsa por lo que puede ingresar a la seccion. La hebra verde se repite, y como es diferente a 0, se queda esperando. Cuando la hebra azul salga va a cambiar turno a 0, entonces en la hebra verde va a poder salir del loop y entrara a la seccion. Si me toca a mi entro yo, sino entonces el otro esta enla seccon. Esto garantiza que la hebra 1 o 2 esten en la seccion critica, tal que turno tiene valor 0 o 1, por lo que es uno o el otrom tal que satisface requerimietno de exlucision mutua.

Pero no satisface requermiento de progreso. Porque no? Cual es la condicion de progreso? Si esta ligre la seccion critica, alguien debe entrar. Tal que sea que yna hebra ya entro a seccion critica, sigue ejecutando su propio codigo. Pero, que valor tiene turno si hebra azul quiere ingresar? Turno tiene valor 1 al salir de la seccion critica de verde, por lo que la hebra azul va a poder ingresar a la seccion critica porque la condicion permite que salga a la seccion critica. Lo termina, turno es 0, y que pasa si quiero volver a entrar? Va a estar con turno 0 por lo que va a quedarse en el ciclo while esperando algo que no va a ocurrir, por lo que queda en spinlock. Esta libre la secion critica pero no puede, ya que depende de que la hebra verde se haga denuevo y pueda setear turno a 1 nuevamente, por lo que no satisface el progreso. Entonces solo es posible que funcionen las hebras de forma que estarian extrictamente alternadas, de la forma hebra azul, luego hebra verde, luego hebra azul y nuevamente la hebra verde y asi sucesivamente de forma secuencial. Esto no es progreso y tampoco es concurrente. Lo que quremos es que independiente del orden de ejecucion esto lograra ser independiente. No hay caracteristica de concurrencia sino que es una alternacion extricta de ejecucion de forma secuencia. Esto es solo aplicable a 2 procesos, tal que solo pueden ser 2 hebras, como lo hariamos para n hebras?

La idea es evidenciar las dificultades.

Leer las otras soluciones en el libro-

una idea muy buena presenta muchos problemas

solucion de peterson

funcinoa para 2 hebras y satisface todas los requerimientos de una exclusion mutua,

se usa un arreglo para definir la necesidad flag para indicar el deseo de ingresar a la seccion critica y turno es a quien le toca.

Flakg se inicializa en falso

mi enter es el setting de flag, de turno y de checkeo de while estas son instrucciones que se ejecutan a alto nivel ya que son 3 instrucciones para el enter. Las que se ejecutan de forma atomica son las assignaciones flag y turno pero no el checkeo del whhile. Entnonces sea que esta inicializado en falso y sea la hebra verde tal que digo que flag 0 y que el turno lo seteo a 1, entonces la hebra verde entra al while mientras otro quiera entrar y turno es el de la hebra verde, entonces me mantengo esperado,entonces toda la condicion es falsa . Ahora si viene una hebra roja tal que quiere entrar entonces le doy el turno a 0. emirntras o tra quiera ingresar y se cumple el turno recientemente seteado a 0, entonces se cumple la condicion para salir del ciclo, y asigna a su flag 0 tal qe señaliza que ha salido y sigue con su codigo no critico. La hebra roja encuentra que la flag es falsa, por lo que sale del spinlock y puede ingresar a sc. La hebra verde vuelve al while y quiere ingresar de nuevo mientras la herba roja esta en sc. Entonces mientras la otra hebra quiera ingresar, entonces la hebra verde se queda en el spinlock, solo cuando flag 1 sea falso dado por la hebra roja que sale, entonces la hebre verde sale del spinlock hy puede entrar a la sc.

Satisface las 4 cosas. Y como es que satisfaceria el progreso? Sea la hebra verde que sale de la seccion critica y esta en codigo no critico. Viene la hebra roja, le doy el turno y le da a flag true y turno true, pero me doy cuenta que la flag esta aun seteada en 0 true para la hebra verde

como demostramos que satisface exclusion mutua?

Esto se hace una demostracion por contradiccion. Esto significa que asumo que ambas hebras ingresan a la seccion critica.

Clase 19:

Demostración de Peterson, no debe ser un ejemplo.veremos que la demostración utiliza el codigo, basado en que el algoritmo en especifico funciona o no.

Iremos intercalando slides, la demostración utiliza el codigo, debe estar basado en que el algoritmo en especifico funciona.

Demostración por contradicción:

Esto significa que si la solución falla, ambas tareas o hebras entran a la seccion critica, entonces si decimos que viola la exclusión mutua entonces entra a la seccion critica y otra también. Entonces, esto significa que la hebra roja esta en la seccion critica, tal que la condición debe ser falsa. Se niega por toda la formula y se trabajan (por tablet, dibujo)

Entonces me queda que en la hebra roja si flag de 1 o turno igual a 0 son verdaderos entonces verdadero y para hebra verde que flag 0 es falso o turno es 1 entonces seria verdadero.

Si es que ambas hebras están en seccion critica, ambos flags serian verdaderos tal que aun no han cambiado sus flags a falso, tal que seria falso lo que se obtuvo por la formula, tal que lo que me quedaría para que la formula fuera verdadera es que dependería de turno, tal que para hebra roja debería ser 0 y para hebra verde es 1. Estamos diciendo que el turno es 0 y el turno es 1 que es contradicción. Lo cual no puede ser.tal que se llega a un absurdo, cuando entran a la seccion critica tal que no puede ser que entren a la seccion critica, tal que si pasara llegaría al absurdo.

2 puntos: no he puesto nada de cambio de contexto ni velocidades, esto es formal.

Otro punto es que se utiliza para demostrar el codigo tal que no hago una demostración de funcionamiento sino que veo cómo está el codigo escrito. No hay que hacer suposiciones, hay que hacer suposiciones en base solo al codigo, no a su comportamiento.

Hemos dicho que ambos entran a la seccion critica y no puedo usar el caso de que una esta en seccion critica y la otra no. Violar la seccion critica es asumir que ambas están, lo que hemos obtenido como absurdo, tal que ambas pasan el spin lock y logran pasar a sc, tal que solo pueden entrar cuando el while es verdadero. Si la hebra verde estubiera en algún otro lado no puedo suponerlo, pero como están en las ecciones críticas puedo dar por dado que si o si que deberían ser verdaderos para estar en la instancia. Esta es para solo 2 procesos.

Algoritmo de la panaderia, no se trata pero lo veremos simplemente. Esto es para múltiples procesos, la dificultad sería mucha para n casos, tal que debería satisfacer todas las condiciones, tal que seria perfecta pero muy limitada. Cuando hebras llegan a seccion critica, se le da un ticket único. Entonces al llegar, todo los tickets son distintos y un ticket va a ser mayor que otro. Entonces solo el enter va a ser complejo por su naturaleza. Entonces sea qué hay una suma no atómica, después de e la hebra tome el ticket debe hacer un loop doble, tal que espera a todas las hebras y después de tomar su ticket va por un loop que espera a que su ticket sea el menor. Tal que el algoritmo simula una espera. Este algoritmo es muy complicado y no se utiliza. Lo que si se utiliza son los semáforos del sistema operativo.

///////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Para n procesos o hebra,s lo que se utiliza son los servicios de sistema operativo.

Servicio principal de sistema operativo es: para implementar exclusion mutua tiene semáforos y locks.

Entonces es un llamado al sistema, tal que es como un malloc, pipe, fork, etc. tal que pide servicio a sistema operativo para llegar a seccion critica.

Un semáforo es una variable entera que se accede solo por 2 operaciones: wait y signal. S es variable entera. Mis llamados a sistema son wait y signal. Wait decremento el semáforo tal que si el valor es negativo la tarea se bloquea, sino sigue su ejecución, cuando una hebra va a llegar. Sc, va a invocar wait al semáforo, tal que es aquel que con la acceso a seccion critica. Tal que wait(s), sc, signal(s). Tal que s toma valor. Entonces al querer entrar a sc, llamo a wait, tal que decremento el semáforo. El semáforo es aquel que controla el acceso a la seccion critica. Si el valor resultante es negativo se va a bloquear dentro del wait, de otra forma va a entrar a sc, tal que cuando sale de la seccion critica hace un signal que incrementa valor de s. Si valor resultante es menor o igual a 0, entonces se despierta una tarea que fue bloqueada por wait, todas las que llegan. Pueden haber muchas bloqueadas por wait tal que todas decrementan al semáforo.todas las que lleguen decreental el semáforo, y si es negativo no van a poder entrar a semáforo.al hacer signal incremento valor de semaforo. Si es menor o igual a 0 van a haber hebras esperando en wait, entonces si incremento en signal, puedo entrar a sc.

Cómo está escrito wait y signal a alto nivel.

Sea un semaforo tal que tiene una variable entera y una cola de procesos que es donde se bloquean las tareas (una queue) pueden haber muchos semáforos. Cuando una hebra invoca el wait, decremento el semáforo y lo pongo en la cola del semaforo si es que es menor a 0. Si es que es positivo entonces pude hacer el proceso. En signal cuando incremento y esta en negativo aún, puedo despertar a otra para que pueda removerse de la cola del semaforo. Entonces esta hebra bloqueada en semaforo ahora esta en cola de listos, tal que puede esperar en su turno para ejecutarse. Entonces la hebra qué pasó por sc saca otra hebra del estado bloqueado, tal que saca la hebra y la despierta. Este signal permite que una hebra salga del wait y pase a la cola de listos para ejecutarse y pueda ingresar a seccion critica. Recalcar que el sistema operativo garantiza que estas operaciones se ejecutan como si fueran atómicas. Wait y signal son complejas y hacen hartas cosas pero sat op garantiza que se van a ejecutar como si fueran atómicas, tal que ingresan las hebras a wait y signal como si fueran atómicas. Entonces ninguna hebra va a ingresar a wait o signal mientras este otra en wait o signal, no es posible overlapping.

Semáforos contadores (lo anterior) porque s puede tomar cualquier valor entero, tal que decrementar dice cuantas van a entrar y incrementar cuantas salen. A pasar por seccion ritica y las que salen cuantas han pasado. En general semáforos toman valores mayores que 1.

En cambio semáforos binarios solo toma valores 0 o 1, tal que abierto o cerrado. Tal que es como un lock. Entonces si incremento y estoy en 1 me quedo en 1, y si decremento y estoy en 0 me quedo en 0.

Los semáforos binarios tienen la misma capacidad que los semáforos contadores, en el sentido que sea una solución implementada en semaforo contador, también sera implementada en semaforo binario con una adición algorítmica tal que creo algoritmos adicionales para implementarlo. Esto es para una solución de exclusion mutua para sc semaforo.

El wait es equivalente a enter cs critical section. Y el signal es equivalente al exit.

Entonces este algoritmo inicializa s en 1, y ejecuta en paralelo muchas hebras que hacen uso del mismo codigo. Entonces llega una hebra a la función, llega a wait, decremento 0 y como es 0 entra a la seccion critica. Si es que llega otra hebra será decrementado a -1 entonces se va a guardar en queue, luego llega otra hebra y s es -2 y se guarda en el queue. La primera hebra sale, ejecuta el signal, que incrementa el semaforo a -1 entonces al meneos una hebra esta aun en el wait, se desbloquea la hebra verde, y se la deja en el estado ready, sea que termina el tiempo de la primera hebra, cambio de contexto y entonces la hebra verde retorna del wait y a la seccion critica, hace el signal y esto significa que queda en 0, tal que es menor o igual que 0 tal que todabia me queda al etos unna hebra entonces entra la hebra esta ready, cambio de contexto, y logra entrar a sc. Tal que no puedo tener una hebra en el signal y en el wait , se ejecutan de forma exclusiva y atómica. El valor del semaforo es importante para veré cuantos pueden ingresar a la seccion critica.

Sea 2 inicializado en 2, la hebra roja lo deja en s = 1, luego hebra azul y también puede ingresar tal que s= n, tal que cuantas hebras pueden entrar a la seccion critica, también estos semáforos sirven de esta forma para otras cosas como elegir una cantidad de hebras. Cuando tengo yo semáforos contadores existen 2 posibles aplicaciones: implementar exclusion mutua (estricto n = 1) comunicación (n) tal que esta comunicación no tiene que ver con secciones criticas.

Semáforos entonces pueden usarse de 2 maneras, exclusion mutua y comunicación

Problema típico:

Sea productor consumidor es un algoritmo genérico tal que procesos producen tienes , otros información, otros consumen o usan la información. Pensemos en que tengo muchos procesos o agentes productores que depositan en un area de memoria compartida información, tal que tengo un agente consumidor que extrae información del area compartida. Entonces hay agentes productores y hay un agente consumidor, puedo tener mas consumidores,pero en esta ocasión tendré solo uno. Vamos a implementar un codigo usando hebras con el modelo productor consumidor. Basta con que tengamos un solo consumidor para hacer una asimetria entre los agentes que ponen elementos en el buffer y los que sacan los del buffer. (Agentes). Basta con que sea un buffer de memoria compartida y los productores van colocando ítems de a uno y el consumidor saca los elementos del buffer. El buffer compartido es acceso exclusivo, debo imp0lementar accedo con exclusion mutua. Es decir qué hay un solo productor produciendo un item en el buffer o solo hay un consumidor. Solo un agente debe estar haciendo su labor dentro del buffer. Lo que hace algoritmicamente es mostrado por el profesor.si el buffer es pequeño necesito hacer una indexación circular del buffer.supongamos que el buffer es un arreglo de estructura de algún tipo de item como el buffer es finito se va llenando y vacíe ando dependiendo de los productores y consumidores.

Sea un buffer de largo 10 tal que insertan en el 5 tal que in busca cual es el espacio libre siguiente, y out cual es el item que debo sacar.

Entonces se que 1,2,3,4 estan ocupados y 5 es donde puedo meter con x y out es el primero 1. entonces saco del buffer que seria 1 y mi out se mueve a 2. luego al meterle al buffer entonces se va moviendo el in a 6, 7,8,9,0 tal que es 0 porque in+1%10 es 0, entonces me puedo mover en un arreglo infinitamente de 10 elementos. El productor produce un elemento y paara nosotros no es importante lo que es, lo relevante es el acceso al buffer compartido. El productor debe checkear si hay elementos en el buffer, tak que se pregunta en el spinlock while(in+1%n ==out) mietras el buffer esta lleno no puedo seguir metiendo componentes en el buffer, tal que si logro salir del while lo meto al buffer circular. Esto para productor.

Para consumidor, si buffer vacio, entonces no voy a poder sacar elementos del buffer porque no hay. Pero de ser que si halla, salgo del while y puedo obtener uno de los componentes y lo va a sacar.

Esta es la idea del problema.

Cuales son las cosas que se comparten?ambos estan escribiendo y leyendo in y out. Entonces en ese sentido los productores escriben el buuffer y el consumidor tambien lo escribe, no es que solo lo lea, sino que sca un elemento del buffer. Tal que modifica algo que lee producer.

Esto esta implementado con spinlocl y yo queiro implementarlo con semaforo.

Sea un semaforo para exclusion mutua tal que s=1, y otro semaforo para comunicación(2 semaforos)

Estos permiten controlar las condiciones de vacio y lleno,tal que va a ser posible controlarlas, el otro semaforo es para exclusion mutua.

O hay un consumidor o u productor en seccion critica. El semaforo que inicializa el tamaño completo del buffer va a poder pasar el productor a producir mientras no sea negativo, talq ue halla espacio en el buffer. Y si es negativo, entonce sesta lleno el buffer. Tal que controla. El semaforo dice pasa mientras halla espacio vacio para producir, para rellenar. Y para consumir debe haber al menos un elemento en el buffer, tal que va a estar detenida si es que el wait lo deja en –1. Supongamos que hebra roha va por productor, va la hebra azul que consume. En wait queda bloqueada la hebra azul porque el bufer esta vacio. Luego la hebra verde produce elemento en el buffer y incrementa full a 0. esto es mandar una señal para permitir que hebra azul que estaba bloqueada en full pueda continuar. Me sirve para que se comuniquen las condiciones del problema. sea en el caso que el buffer esta completamente lleno, la hebra verde se bloquea porque ya no puedo rellenar mas, llega la azul, la libera, incrementa empty, tal que la hebra verde ahora puede ingresar a la seccion critica. Entonces la hebra le comunica una condicion y la otra tambien. Empty y full van tomando diferentes valores.

Clase 20:

Clase pasada vimos aplicación de concurrencia con productores y consumidores, como os semaforos pueden ser utilizados para dos funciones principales distintas: exclusion mutua y llevar control sobre la aplicación. Por ejemplo señales de buffer vacio y lleno. Al hacer un signal cambia valor de variable y despierta otro proceso. En el caso de los semaforos de comunicación los interpretamos como comunicación, pero en extricto rigor no lo es.

Aplicación parecida donde no existe un registo seccion critica que acceda a registros compartidos, este es un problema de filosofos comensales. Sean 5 filosofos, cada uno tiene un plato de comida en frente, para comer este plato necesita 2 tenedores. Entonces este filosofo piensa y despues quiere comer. Comer usando 2 tenedores y despues vuelve a pensar. Si filosofo esta comiendo, usa 2 tenedores, tal que sus comensales aledaños no podrian ya que le falta el otro tenedor. En extricto rigor no existen datos compartidos pero si una sincronizacion de acceso a los tenedores. Una solucion seria de la forma tal que hay 2 funciones, pensar y comer, no me importa lo que haya dentro, y tampoco sabemos si hay sc, no nos interesa. Lo que nos interesa es que filosofo pueda sincronizarse para el acceso a los tenedores. Entonces los tenedores van a ser semaforos contadores, tal que comienzan con valor 1 (esta o no esta tenedor, si hago eat y esta disponible va a quedar en 0 tal que no esta, y si vuelvo a hacer eat y no esta en tenedor va a quedar en –1 tal que debo esperar a que sea 0 ya que volveria a estar disponible): esta disponible el tenedor. Entonces un filosofo va a estar pensando y luego quiere comer, debe tomar 2 tenedores. Entonces sean filosofos 0,1,2,3,4. Sean los tenedores 0,1,2,3,4. El filosofo 4 piensa y ahora quiere comer, entonces logra obtener el tenedor 4 y el tenedor siguiente que es 0, entonces puede tomarlo. Entra a comer. Luego llega filosofo 1 y ve si tenedor si esta disponible el tenedor 1 y el siguiente que es el tenedor 2, ya que estan disponibles, puede comer. Luego llega el filosofo 0, el semaforo tenedor 0 esta ocupado por lo que lo queda esperando. Luego llega el filosofo 3, su tenedor 3 esta disponible pero no lo esta el siguiente tenedor que es 4, por lo que queda esperando. Supongamos que lo primero que hace es signal de 0, entonces libera el siguiente tenedor y luego el propio, tal que desbloquea el tenedor 0 y libera el tenedor 4, despertando al proceso 3 y proceso 0. entonces el proceso 0 toma el tenedor 0 y no puede obtener el tenedor 1 por lo que se bloquea. Luego el proceso 3 puede tomar, ademas del tenedor 3 que ya tenia, el tenedor 4 que anteriormente estaba ocupado.

En esta solucion a lo mas hay 2 filosofos comiendo, no pueden haber 3 filosofos comiendo.

Se produce deadlock. Este es uno de los requerimientos para la solucion perfecta, por lo que no nos sirve. El acceso a comer debe ser sincronizado, nos permite efectivamente que sea sincronico pero es posible el deadlock, tal que todos los semaforos quedan en espera sin poder comer nunca.

El filosofo esta obligado a comer con ambos tenedores.

Se produce deadlock de la siguiente manera:

(hay diferentes formas pero con una basta) supongamos que cada filosofo toma su tenedor en cambios de contexto intercalados.

Ahora supongamos que cada filosofo quiere tomar el tenedor siguiente, todos quedan bloqueados, tal que todos quedan esperando y nadie come.

Todos toman el tenedor a su iz y quedan bloqueados.

Esta solucion es simple pero produce un error, como soluciono? Evita que todos los filosofos tomen el tenedor a su izquierda, restringir a maximo 4 filosofos que quieren comer, de los 5 que son, para que no se produzca deadlock. Entonces voy a tener un semaforo que vale 4, tal que me controla el acceso a elegir, tal que sea un filosofo que quiere comer, solo lo podrà hacer cuando este semaforo no sea negativo, tal que el quinto filosofo que quiere comer, va a quedar en –1 tal que va a quedar en este while sin tomar el tenedor. Entonces los demas van a poder seguir intentando comer sin que el quinto filosofo haya tomado su tenedor, tal que el filosofo 4 queda detenido, si no puede, entonces ahora vuelven a checkear los filosofos si pueden comer, y el filosofo 3 se va a dar cuenta que efectivamente puede tomar el tenedor 4, lo que significa que puede tomar el enedor y comer, tal que no se produciria deadlock. Si filosofo 3 deja de comer, entonces 2 va a poder comer sin problemas, y luego 1, y luego 0. entonces la idea es que es un semaforo que pueden pasar n filosofos hasta que se bloquee la espera.

Los semaforos en realidad producen hartos problemas, ya que el programador debe saber usarlos apropiadamente e inicializarlos sin que ocurran problemas de sc. Entonces sea un semaforo n inicializado en 0 y otro en 1, tengo un consumidor y un productor tal que me equivoque y tengo un signal y luego un wait y luego al reves, tal que ahora entrarian 2 consumidores y no solo 1.

Clase 21:

Semaforos en si no son no probeen la solucion como tal, sino que son una herramienta de sst op con objetivo de que el programador pueda comunicar una condicion o idea entre procesos o sobre exclusion mutua, pero es posible equivocarse en inicializar, error de posicion, tal que seria problemas que programador debe solucionar.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Monitores.

Comenzamos a ver diferentes tipos de soluciones, primero vimos de hardware, luego vimos de lenguaje de alto nivel por ejemplo soluciones de peterson en c, luego servicios de sistema operativo con semaforos, y cuarto en lenguajes de programacion, que son estructuras de lenguaje de programacion.

Estos son, los monitores.

Que es un modulo de software que garantiza acceso exclusivo a datos compartivos, tal que es una caracteristica dada por el lenguaje de programacion. El monitor es un modulo al cual accedo con exclusion mutua, tal que el lenguaje me la da, tal que no debo yo programarla en especifico.

Un monitor es como un objeto que puedo crear y destruir, tal que hay datos privados, distintos procedimientos publicos o privados, codigo de inicializacion que es el constructor y variables de condicion. la gracia de esto es que tiene una puerta unica que garantiza exclusion mutua, tal que si una hebra esta dentro de monitor ejecutando funcion k, otra funcion no va a poder porque ya hay una hebra adentro, tal que bloquea las hebras y hay una cola de aquellas que esperan que termine la hebra adentro. Cuando sale, se permite nuevamente otra hebra que entre, y asi las hebras van accediendo al monitor en fifo, tal que first in first out, solo puede entrar uno para garantizar exclusion mutua, tal que si entra a seccion critica, no van a haber problemas. Cosas adicionales: variables de condicion que estan en lista de cola. Estas variables es un mecanismo nuevo de sincronizacion. Implicitamente en la entrada hay un semaforo implicito, no tengo porque crearlo, tal que esta on y of binariamente mutex, tal que la puerta se maneja con un semaforo, lock, mutex, etc. El monitor es un objeto que tiene funciones, datos, inicializacion y variables de condicion.

Entonces pensemos que la hebra que habia entrado y salido quiere volver a entrar. La hebra que le sigue, encuentra una condicion que le prohibe continuar con su ejecucion. Por ejemplo, sea que la hebra es un consumidor y encuentra el buffer vacio, tal que no hay y debe bloquearse, envez de salir y entrar de nuevo, se bloquea dentro del monitor en una variable de condicion tal que es un lugar donde la variable se bloquea. Entra la siguiente hebra que tambien es consumidor, esta vacio, asi que se pone a la cola de la variable de condicion. Cuando las hebras se bloquean dentro de variable de condicion en monitor, entonces se abre la exclusion mutua. Cuando una hebra entra es porque esta abierto y luego al entrar se cierra, pero al bloquearse esta puerta se abre. Entonces entra una hebra productora, produce item. Despierta a uno de los dormidos o bloqueados en esta variable de condicion. Entonces la hebra productora no solo crea item sino que tambien deja en listo a la hebra consumidora dormida.

Solo voy invocar estos procedimientos, entonces sea esto en una hebra ejecutandose en main, sea que el monitor se llame M, tal que M.procuderk() entonces la hebra entra al monitor y entra al procedimiento de monitor. Puede haber otra hebra que quiera entrar a mismo monitor, pero no puede por exclusion mutua.

Entonces, las variables de condicion, puedo tener todas las que yo necesite, son definidas por el programador que construye el monitor. Las variables de condicion se pueden acceder con solo 2 funciones: wait y signal. Estos wait y signal no son de los mutex, tienen el mismo nombre pero son bien diferentes. Uno es variable de condicion y otro es mutex. Utilizaremos wait(c) y signal(c) tal que son variables de condicion y utilizaremos sen(wait) y sem(signal) para semaforos. No puedo usar variables de condicion como semaforos.

Un wait de condicion bloquea la ejecucion del proceso hebra que llama o invoca a este wait, y siempre bloquea. A diferencia de un semaforo que le doy un wait que decrementa y ve si se bloquea o no. No hay un valor numerico, no es un valor de veracidad, es solo un lugar donde la hebra se bloquea al invocar wait. En ese sentido la variable de condicion siempre esta en rojo. Cuando una hebra invoca un signal entonces se resume la ejecucion de esta hebra, tal que una hebra hace signal para desbloquear otra hebra. Se parecen a semaforos pero no son iguales y cumplen funcion distinta.

Como puedo hacer una solucion para productor consumidor con monitores. La idea es obtener un objeto donde dentro esten los datos compartidos pero estos son accesados de forma exclusiva y los productores y consumidores van a proceder a consumir y producir de forma exclusiva. Antes de definir el monitor, como vamos a usar el monitor?

Sea que tengo una creacion de productor y consumidor, y la hebra productora va a producir un item y esta produccion la agrega al monitor. Entonces estaria solicitando entrar al monitor de la forma M.append(). lo mismo con el consumidor, tal que trata de sacar un elemento del buffer invocando un M.take al monitor. Una vez que la saca la consume, esto realmente no nos interesa. Gracias a que esto es un monitor, solo va a haber una hebra consumiendo o produciendo dentro del monitor.

Definiremos nuestro monitor.

Entonces definimos nuestro objeto monitor m, tal que el lenguaje me provee el objeto monitor, tal que es una estructura sintactica. Defino mis datos privados, defino mi buffer de n caracteres de datos compartidos, este se maneja de manera circular tal que array circular. Tal que debo indicar el in y el out del arreglo. Y tambien existira una variable que cuenta cuanto hay en el buffer y variables de condicion que controlan el buffer lleno y el buffer vacio.

Tengo mi constructor inicializador, lista de inicializacion, tal que inicializo in=0, out=0, count=0. Y tengo 2 funciones publicas que me permiten acceder al monitor, las cuales son append y take.

Entonces defino ambas funciones del monitor m, tal que append, que es la funcion que invoca el productor. Pregunto si esta lleno el buffer, entonces no podre seguir creando y queda en un wait not full. Supongamos que hay espacio, entonces coloco mi elemento en posicion i y incremento el valor de i para el siguiente que pueda añadir, y hace un signal. Este llegaria a el wait not empty.

El consumidor invoca a m.take, tal que saco un elemento del buffer. Supongamos que esta vacio el buffer, entonces se bloquea antes del take. Si viniera un productor, entones independiente de que este bloqueada o no la hebra que esta pidiendo un objeto, si es que creo un item, voy a ir si o si al wait del consumidor para despertar o avisar que si hay un item, tal que despertaria si es que estaba vacio y si no lo estaba no hay efecto. Entonces saco un elemento de un buffer, decremento el i y aviso que si o si no estaria lleno el buffer donde los productores, de manera que despierto a alguno que este bloqueado si es que hay alguno que este esperando o no tiene efecto si no hay problemas. Entonces signal en variables de condicion no tiene efectos colaterales si no existe hebra o hebras bloqueadas en la variable de condicion.

Eso no es asi en los semaforos, porque siempre hay efectos del signal en decrecimiento o crecimiento. El signal vs siempre es avisar.

Variables de condicion en pthread mutex (en c++ no hay monitores).

En java hay synchronized, inmediatamente el objeto va a tener exclusion mutua, es lo mas parecido a monitores en lenguaje de programacion.

En la mayoria de los lenguajes se crean nativamente las hebras, pero hay que hacer manualmente la sincronizacion.

Filosofos con monitor no es muy util ya no ayuda a entender monitores. Recalcar que el monitor se puede considerar como un elemento para comunicar asincronamente a varios agentes, tal que son tareas, hebras, etc.

Tal que los objetos, una hebra puede depositar informacion asincronamente y otra hebra que saca informacion del objeto y la deposita en otro objeto, y a su vez esta informacion depositada es guardada en otro objeto. Tal que estas acciones no tienen un orden predeterminado de como acceder a estos datos, por lo tanto son asincronas, tal que entre ellas no se ponen de acuerdo cuando ingresar, sacar y poner datos. Puede pasar al reves entonces la ultima hebra se bloquea y espera a la hebra segunda hasta que llega con la informacion y desperta para seguir con la accion que quiere hacer, tal que es una forma herramienta para compartir informacion.

Sincronizacion con hebras posix: exclusion mutua con las hebras, las hebras nos proveen mutex y variables de condicion. En lenguaje c no hay monitores, por lo tanto lo que hay son mutex que son parecido a semaforos binarios y variables de condicion que son parecidas a monitor.

Estos mecanismos son para hebras de un mismo proceso, es decir que voy a poder sincronizar estas hebras con estas herramientas, pero si es otro proceso, tendra que ser con variable compartida. Entonces un mutex es un semaforo binario, mismo servicio, tal que mutex esta abierto o cerrado. Estas son mutex de pthread, tal que pthread mutex lock es el que me provee wait y pthread mutex unlock es el signal, lo mismo de lock y unlock.

Entonces viene una hebra, hay un pthread mutex, pthreadmutexlock(mutex variable) tal que sigue la seccionc critica, y para salir pthreadmutexunlock(mutex variable). Esto es lo mismo que wait y signal, tal que llega al lock y esta cerrado entonces quedan bloqueada, hasta que sale la hebra que estaba ejecutandose, tal que la hebra que estaba primero en la fila de espera va a entrar a la cola de listos del procesador, retorna para la seccion critica, sale, y la siguiente de la espera se va a la cola de listos para entrar a la seccion critica.

Entonces pthread mutex es inicializado con pthread mutex initializer, si no se inicializa es impredesible y posiblemente erroneo.

Esto funciona para muchas hebras, todas deben compartir el mismo mutex y se podran sincronizar de esta forma.

Pthtread tambien provee un trylock, tal que intenta ingresar, pero si esta cerrado no se va a bloquear. Si es diferente a ebusy va a entrar, y de otra forma, no se va a bloquear. Las variables de condicion son wait, signal, funcionan muy parecido a los wait y signal de monitores. La funcion wait recibe la variable de condicion, un mutex. La hebra que invoca con wait simepre se bloquea en la variable de condicion y signal recibe solo variable de condicion y despierta a la primera hebra de la lista y si no, tampoco tendra un efecto colateral. Broadcast lo que hace es despertar a todas las variables que esten bloqueadas en la variable condicion, tal que le mando un signal a todos. Todas estas funciones son atomicas. Sea que tengo n hebras bloqueadas en una variable de condicion en un arreglo y hago un pthread signal y llamo a c{i}… esto no es lo mismo que el broadcast (que es atomico), porque no es atomico.

Una variable de condicion tambien puede ser inicializada con pthread initializer, sino el uso de ella va a quedar indefinido y puede hacer errores.

Para finalizar veremos productor consumidor usando pthread hebras.

Definiremos una estructura objeto buffer compartido, no es un monitor, sino un buffer compartido y le dare acceso exclusivo a este buffer exclusivo, tal que hay un buffer, un head, tail que es simil a in y out, tendre contadores de numero de elementos ocupados y vacios tal que full y empty y tendre un mutex exclusivo y tendre dos variables de condicion tal que condicion de buffer lleno tal que se bloquea productor y condicion de buffer vacio que bloquea consumidor.

Sea el cliente main, defino la estructura, las hebras, inicializo el mutex, variables de condicion y con el buffer init inicializo el head tall full empty.

Creo 2 hebras, una productora y una consumidora y les paso el buffer compartido. Entonces la estructura esta siendo compartida por la productora y la consumidora. La hebra padre espera a que termine todo esto. La productora intenta ingresar y meter un elemento a arreglo pero si ya esta lleno, se bloquea en not full dentro del monitor. Cuando venga un consumidor va a sacar un elemento del buffer y envia un signal a not full haya o no haya un productor bloqueado. Uso el mutex para exclusion mutua, para eso me sirve. Para el productor mientras el buffer este lleno, yo me bloqueo, no puedo seguir poniendo elementos en el buffer, debo bloquearme. Supongamos que el buffer no esta lleno, voy a poner elementos en el buffer y mando señal a consumidor que esta not empty, que pudo haberse bloqueado de haber estado vacio, y la hebra productora libera la exclusion mutua. El consumidor hace lo mismo. Primero exclusion mutua con mutex compartido y luego dice mientra,s buffer esta vacio, yo me quedo bloqueado y como no esta vacio, envio signal a not full a productor que puede que haya estado bloqueado tal que es un signal que va de vuelta al not full, donde productor puede estar bloqueado. El objetivo de que el mutex este donde esta, le tengo que avisar cual mutex abre, tal que le digo yo me voy a quedar bloqueado y el mutex esta cerrado producere deadlock, por lo tanto debo indicar junto con la variable de condicion a que mutex se abra, tal que existira una hebra que siga, y los signals seguiran su curso desbloqueando cosas. La hebra que recibe el signal no pasa directamente, sino que vuelve a preguntar. Asi se debe hacer en el laboratorio: no es lo mismo que decir if buffer full entonces pthread cond wait, pareciera que bastara con if pero la solucion debe ser un while, tal que despues de ser despertada con el signal, pero vuelvo a preguntar la condicion, tal que si es falsa efectivamente salgo de la condicion, ya que puede que este la posibilidad que se haya vuelto verdadera nuevamente por algun motivo. En monitor bastaria con hacer un if ya que garantiza exclusividad, pero como este no la garantiza, estoy obligado a poner variable de condicion y while para que otra hebra ingrese a la seccion efectivamente.

Clase 22:

Administracion de memoria.

Este capitulo es introductorio sobre forma de administración de sistemas de memoria computacional, hemos visto admin de procesos y ahora admin de memoria.

Conceptos utiles: concepto de reubicacion, tal que puede cambiar de ubicación de memoria principal. Basta con pensar que memoria es espacion unidireccional donde se aloca un proceso, tal que proceso al ejecutarse debe estar en alguna parte de memoria. Proceso puede ser suspendido y mandado a memoria secundaria, para que vuelva pueden meterlo en otro lugar, a esto se le llama reubicacion. Para esto es necesario que proceso recida en memoria virtual. Concepto memoria virtual y real. Esto se logra gracias a memoria virtual. Cuando proceso se procesa en procesador, las referencias deben ser a memoria logica, no puedo referenciar memoria fisica ya que si se mueve, esta ha cambiado y ya no es la misma. Pero si referencio virtualmente independiente de fisica, puede reubicarse, por lo que hay que traducir referencias a memoria virtual a direcciones de memoria fisica. Esta es la estructura de procesos que conocemos tipico, tal que le llamamos espacio virtual de direcciones. Porque el proceso aca, todas sus direcciones y referencias son referencias virtuales, no fisicas. Lo veremos con detalle. Lo otro que es importante es que procesos se protejan el uno de otros. Sea que un proceso se esta ejecutando, se debe evitar que proceso 2 tiene prohibido que salte a referencia virtual de proceso 1, tal que las referencia a proceso sean exclusivo a menos que sea explicitamente compartida, tal que sea utilizado por proceso 1 y 2. cualquier entrada ilegal produce error, de aquí se obtienen muchos problemas por segmentation fault, tal que accedo a memoria que no deberia acceder. Sea que escriba shartp\*p y se me olvida procurar memoria, y le quiero asignal c, y no le hice a malloc, tiene una direccion cualquiera, tal que p tiene direccion que no corresponde a proceso tal que produce segmentation fault, este es un mecanismo de defensa.

La organización local es de como el proceso se estructura con su codigo por ejemplo datos globales, stack, heap. Esto es organización local. Los programas estan organizados como modulos, esta organización se utiliza tal cual en el capitulo de segmentacion tal que el proceso esta segmentado en heap, datos, codigo. Vamos a ver que en memoria virtual se cargan tal cual.

En cuanto organización fisica, sabemos que memoria se organiza en niveles, esto lo hemos visto en gerarquia de memoria, tal que tiene niveles: registros de procesador, luego cache con varios niveles, memoria principal, memoria secundaria, que sera el disco, los flashdrives son memorias terciarias tal que son removibles. El sistema de administracion de memoria va a colocar el proceso en memoria. La gracia del sistema operativo va a ser que el acceso a datos y codigo sea muy eficiente. La mayor parte del tiempo y del proceso, se va a encontrare en memoria principal. Pero no necesariamente es asi, pedazos del proceso pueden encontrarse en memoria secundaria. Es muy probable que parte de este proceso este en memoria secundaria, otra parte en memoria principal, otros en cache, etc etc. Esta es una labor importante de sst op para que sea eficiente.

Esquemas de particionamiento.

Veremos como memoria fisica se particiona. Es un sistema de memoria antiguo cuando no existia memoria virtual, pero antes de estudiarlo, lo veremos de forma simple sin involucrar memoria virtual para entender concepto. Sea que solo exista en este caso memoria fisica, el proceso se debe cargar a memoria fisica completamente. Esa memoria fisica podemos particionarlas de dos formas:

Particionamiento fijo: de igual tamaño. Sea que tenga una memoria de 64 mb, y lo divido en 8 porciones iguales de 8 mb, tal que se puede cargar en cualquier posicion siempre y cuanto sea menor a ese tamaño y este disponible a los procesos, se carga completamente. Sea que sea un proceso de 6 mb, llevo las cuentas de lo que llevo cargado y lo que esta libre, tal que genero dentro de la particion utilizada una fragmentación interna de 2 megabytes ya que no los utiliza. Puede ser que le proceso crezca y solo puede crecer 2 mb. Sea un proceso que sea de 1 mb, puedo cargarlo en cualquier fragmentacion, sea que la cargo a uno de 8 mb, tendre 7 mb libres en fragmentacion. No puedo cargar uno que sea mayor a 8 mb, no puede ingresar a sistema. Sea que tengo todos los modulos ocupados y supongamos que la suma de todas las fragmentaciones es mas de 8 mb, si viene otro proceso y todas las particiones originales estan todas ocupadas, no puedo cargarlo aun que tenga la memoria fragmentada que esta en cada proceso de sobra, no puedo ubicarlo.

Una posibilidad de solucionar esto, es utilizar lo que es overlay, tal que el proceso es mas grande que particion, tal que voy particionando yo como programador y voy particionando a medida que voy ocupandolo para que se fuera cargando de a poco. En los primeros sistemas de dos de ibm existia una sola particion de sistema. La memoria que podia direccionar el sistema era de 64 kilobytes aunque la memoria fisica fuera mas. Pero de eso solo direccionaba 64 kb, eso era solo una particion y el proceso debia caber en esto, pero en realidad era menos porque debia utilizar tambien un tamaño el sistema operativo, pero si yo queria correr algo mas grande que lo que habia, entonces lo que hacian era poner marcas y dividirlo en overlays, tal que primero se cargaba el overlay 1 y cuando necesitaba cosas de overlay 2, sacaba el overlay 1 y metia el overlay 2, yo anunciaba en el codigo las particiones, a veces debia particionarlo en muchos overlays, tal que hacia go to e iba eligiendo los overlays que queria utilizar para procesar cierta cosa que queria hacer.

La otra posibilidad es particionamiento fijo de diferentes tamaños, tal que tengo particiones de 2, 4, 6,… tal que procesos mas grandes que no cabian en el anterior, caben aquí... pero y si es impar por ejemplo? No nos permite solucionar el problema fundamental que independiente de las particiones, estas siguen siendo fijas y esto es problemático, tal que el problema se carga en memoria fisica siempre, no hay virtual y no hay reubicacion.

Cuando las particiones son de diferente tamaño puedo definir que particiones de diferente tamaño tal que sea que una particion que sea mayor a 8 y menor a 16, es la de 12 y esa esta en un lugar especifico, eso aminora la fragmentacion interna. Porque cada proceso ocupa una particion que se ajusta mas a su tamaño verdadero lo que produce menos fragmentacion interna de no ocupar particiones. Sea que estan ocupados los mas pequeños que van primero pero viene un proceso nuevo pequeño, ese deberia esperar hasta que pueda ingresar a la particion que mas se ajusta o lo otro es elegir cualquiera aun sea mucho mas grande que la necesitada. Si importa la ubicación del proceso, pero debe ser flexible. En el anterior fijo uniforme no importa donde pongo proceso, pero ocmo en la increcental cambia, comienza a omportar. El particionamiento fijo de la memoria no es usado en sst op modernos, pero lo que si existe es paginacion en el contexto de memoria virtual que tambien es un particionamiento de memoria fisica en contexto de memoria virtual... en general es para sistemas antiguos.

Particionamiento dinamico es que no la particiono, sino que la voy particionando a medida que viene un proceso, particiono y sigo con lo que me queda, hasta que no puedo añadir mas. Si viene un proceso que no puedo ingresar, debe esperar a procesos que terminen para liberar memoria. En el otro, si el proceso es mayor al segmento, nunca iba a poder, pero en esta caso simplemente va a tener que esperar.

Lo que va generando esto es fragmentacion externa, que es conjunto de pedacitos de memoria que va quedando dispersos y no son utiles tal que tendria 16 por ejemplos megabites dispersos, no puedo cargarlos. Esto es fragmentacion externa. Va generando pedacitos de memoria inutilizable. La unica forma seria compactando la memoria principal, tal que deberia mover los procesos para obtener un espacio completo que si me sirva. Pero compactacion es un proceso de io que es caro. Porque generalmente los procesos para compactar la memoria lo que sucede es que los procesos los llevo a disco, se sacan todos a disco, libero toda la memoria y los reordeno. Esto es caro en tiempo.

En este caso, supongamos que la memoria esta de la forma (horrenda) y llega un proceso que pesa 3 mb, donde puedo cargarlo? En diferentes lugares. Pero hay que tomar una decisión... no es lo mismo cargarlo en uno que es de 4, 6, 20. Donde me conviene? En el de 4, porque generaria menor fragmentacion externa. Este es el algoritmo best fit, si yo digo carguemoslo en el de 20 es el mas grande donde quepa el proceso, este seria el worst fit. Entonces hay diferentes algoritmos, y cada uno tiene implicancias para el resto de procesos para uso de memoria. La inuticion dice carguemoslo en el de 4, y este produce una fragmentacion de 1, y este a la larga seria mas inutil que la fragmentacion de 17 que queda con el mas grande. El best fit usa la particion mas pequeña habible, pero produce peor rendimiento, es mas lento, porque es de orden n, esto es porque debo tener orden, debo recorrer para saber cual es el mas pequeño donde cabe el proceso. Y produce una fragmentacion externa peor. El worst fit se parece al best fit, tiene mal rendimiento. Encambio el first fit es el que busca el primero que sirve, el primero que encuentro ahí lo cargo, tal que altiro lo cargo al primero que sirve, es simple, rapido, y generalmente produce mejores resultados. El next fit utiliza la memoria como si fuera circular, busca el proximo desde donde quede la ultima vez. El first fit comienza desde el inicio de la memoria, encambio el next fit comienza desde donde quedo en su ultimo acceso. Tienen rendimeintos similares.

Sea la memoria fisica, entonces los procesos, que son areas achuradas son los cargados y lo blanco son procesos disponnibles. Sea el ultimo proceso que se encargo la memoria, desde aquí comienza el next fit. Y el best y el worst recorren todo. Sea un bloque que llega de 16 mb, va probando si cae o no cae, esto puede ser varias veces hasta lograrlo entonces ahí lo carga. Los otros comienzan desde el inicio. Eso si importa si es que es un algoritmo hecho tal que voy usando la memoria de al lado.

El metodo buddy se utilizo en sst op antiguo y se utiliza para procurar bloques de memoria en potencias de 2. tal que comienzo con una memoria disponible que es 1gb, 2gb, 4,8... es en potencia de 2. esta se va particionando recursivamente en la mitad de cada una de las particiones de memoria hasta que encuentra un bloque donde si pueda ser cargado, casi como biseccion.

Como fuciona este algoritmo? La idea es la siguiente: yo voy a hacer esta divicion recursiva hasta que el tamaño de mi proceso es mayor a la mitad de la particion y menor o igual a la particion donde estoy, tal que es mayor que la mitad del bloque que estoy visitando.

Entonces sea el primer bloque que visito y el largo del bloque es aprox el 60% del bloque. Es ese mayor que la mitad? si, y es menor que el bloque completo. Si es asi, entonces el proceso se carga en este espacio de memoria. Sea que fuera mas pequeño el bloque, menos que la mitad, entonces la pregunta es, es mayor que la mitad? No. Entonces vuelvo a particionar, vuelvo a preguntar en este segmento. Es mayor este bloque que la mitad del proceso? Si. Y ahí termina.

Entonces este bloque quedo particionado en 2 bloques, el bloque siempre se divide en la mitad aunque el proceso solo ocupe una parte, esto produce fragmentacion. Sea 1 mb. Llega una solicitud de 100 kb. Es 100 mayor que 512? Entonces particiono 512. es 100 mayor que 128? No, particiono. Es 100 mayor que 64? Si, por lo tanto el bloque utiliza todo este segmento de 128.

Le quedan otras particiones que son de 128, 256 y 512. Ahora existe no un todo, sino de estos trozos. Se pueden ver como nodos binarios. Sean 240 que llega ahora, busca donde cabe. Es 240 mayor que la mitad? Entonces puedo utilizar el de 256.

Se va produciendo esta divicion recursiva que va fragmentando.

Sea que se libera el proceso, los hermanos nodos se fucionan, “buddy”. Tal que se van uniendo de nuevo con su partner nodo, no pueden ser de otro nodo de otro arbol. Esto produce fragmentacion interna. Es fijo o dinamico? Es un sistema relativamente hibrido, hay un algortimo a particionar siempre igual, sin embargo, el arbol que se genera a partir de esto, puede ser diferente a medida que van llegando los procesos. Si vienen en otro orden, me van a generar otro arbol, por lo que las particiones van a ser distintas, no es estrictamente fijo, cambia, tiene su dinamismo, pero no lo es completamente.

Direcciones logicas y fisicas (siguiente clase).

Estos procesos sin memoria virtual, el proceso puede crecer dentro de su particion fija, mas alla no puede, por lo que el particionamiento fisico y bruto sin usar memoria virtual ya no se utiliza, era un limitante, siempre el proceso iba a tener como limite el mayor tamaño que podia recibirlo. Sea una memoria de 1 gb, que pasa con un proceso de 1,5 bg? Que pasa si crece el proceso a medida que funciona o se ejecuta? No puede, la gracia de memoria virtual no restringe al proceso, siempre va a ingresar a memoria.

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Administración de memoria capitulo 7

Materia cierra aspecto que aun no es entendible de la memoria. Comenzaremos a ver memoria. Luego la entrega de pep1, recorreccion. Pep2 es concurrencia. Repasar ejercicios.

Administración de memoria es importante porque me doy cuenta de que aun no diferenciamos entre memoria virtual y física. Cap 7 y 8 es memoria física. Entenderemos cada uno y sus diferencias. Sistemas operativos rabajan con sst de memoria virtual. Si proceso esta en memoria, se swappea, lo hemos hablado muy a grandes rasgos, esto es mas especifico. Espacio virtual de direcciones como se ve en virtual? Como se ve en físico?

Conceptos básicos:

Capacidad de reubicación de un proceso en memoria física: e este minuto haremos énfasis de memoria física y virtual pero eventualmente veremos solo la diferencia. En memoria están cargados los procesos, no podemos decir datos, ya que lois datos pertenecen a espacio de memoria en procesos. Lo que existe es el segmento de datos globales, de heap… etc. En memoria solo hay procesos. Y el kernel que se carga cuando uno bootea. Lo demás puros procesos. Estos procesos cargados en memoria física al ubicarlos se permite que salgan a memoria secundaria que haga swapp out para secundaria y swap in. Sea que tengo en memoria física el kernel, y tengo un proceso. El proceso va a ocupar dirección de inicio hasta la de fin. Swap out permite que se vaya a memoria secundaria. Y también permite que vuelva. Esto es solo posible si se utiliza memoria virtual. Entender que no todos los sistemas de admin de memoria permiten esto. Los que no lo permiten son los que no trabajan con memoria virtual. Entonces requisito de administración es requisito de memoria virtual. Esto es relevante. Dentro del código están todos los segmentos, tal que en uno de ellos procesos hay un call compute. Esta función compute esta en una dirección especifica. Es una función que yo he creado y el call hace que la dirección cambie a donde esta la función. Todas estas sono direcciones físicas. Tal que cuando yo hago un call copute lo hare a una dirección virtual que se traduce a dirección física, tal que el call implica una dirección virtual que se traduce a una dirección física. Entonces este call si yo tengo toda esta traducción, tal que el código es independiente de mi función, tal que sea que donde sea que este el call, siempre va a hacer le mimo call, dan a donde la misma dirección virtual, pero será distinta en memoria física.

Todas las direcciones tienen dirección virtual. La memoria, el compute me produce una dirección y este es donde esta la dirección virtual de donede esta la memoria virtual, no tiene que ver en la memoria física. Tal que da lo mismo donde este físicamente, se llama a la misma función virtual.

Ambos call compute tienen direcciones físicas distintas, solo necesito traducirlos a virtual que me traduce y busca donde esta físicamente. No se cambia la dirección virtual, de forma que me permite reubicar la memoria física. Los procesos utilizar su espacio de direcciones virtuales como si fuera toda la memoria virtual, tal que pueden haber espacios no utilizados. Otra característica importante es que me provee protección.

Sea que tengo mi memoria física, y tengo mi proceso 1 y proceso 2, entonces, esto serán memorias físicas. Si tengo una dirección supongamos 000ff34 y en el proceso p1 tengo un go t, un proceos no puede acceder a las direcciones físicas de otro proceso, es violación de segmento. No puedo acceder a otro proceso en memoria. Porque no es necesario hacer esto en virtual? Es solo una memoria física que comparten los procesos. Hay un método d protección de procesos, nombre para cambiar las direcciones cuando cambia las direcciones virtualmente tal que se hace una randomizacion de donde comeinzan los procesos, tal que el bottom cambiaba randommente. Así entonces se evitaba que se atacara con algún programa peligroso el stack, ahora todos los segmentos tienen una dirección de inicio que se randomiza, tal que cada vez ue hago correr mi código comienza de forma diferente.

El checkeo tiene que hacerse en tiempo de ejecución, tal que deben estar en memoria física, solo se puede checkear en tiempo de ejecución donde se hace una interrupción. En cada ejecuciond e cada instrucción se debe checkear qque sea valido el acceso o no.

También debo permitir que dos procesos compartan memoria, entonces si uno es proceso 1 y otor proceso 2, supongamos que hay una dirección virtual, que pasa si estos dos rocesos quieren compartir memoria, tal que compartir un sector de memoria, lo que ocurre es que supongamos que se hace un go to, escribiendo a una dirección virtual, tal que la dirección virtual se traduce a física. Para que ambos procesos en memoria sean validos, y apuntan a la misma dirección virtual, sea que tengo los dos procesos con espacio virtual diferente, entonces genero una memoria compartida entre ellos procesos, tal que es la misma, tal que tienen su propio stack, heap, código, datos, pero sistema peprmite que dentro de espacio virtual hay espacios de memoria compartida. Entonces en memoria física van a apuntar a misma dirección virtual de memoria con el permiso correspondiente, tal que va a ahberun área de memoria virtual que se matea a memoria física. Tal que va a pasar a una etapa de traducción y me va a pasar a la memoria física correspondiente, tal que debo permitir que compaprtan memoria en memoria virtual y restringir en la memoria física.

Tal que la memoria virtual va a apuntar a la memoria física compartida y a su vez los procesos que son compartidos están mapeados en partes diferentes y comparten memoria que apunta a la misma parte.

Kernel es uno solo que se carga en memoria física y todos los procesos entran al kernel, el kernel esta mapeado en todos los procesos. Si necesito compartir memoria entre procesos van a apuntar a misma parte.

Hebras comparten la memoria de los datos globales dentro de un proceso. Estamos hablando de cosas diferentes.

Todos los procesos pueden tener una dirección virtual que manda a memoria física distinta, un proceos accede a memorias físicas que no tenga, si la traducción le da a un sector invalido, que esta fuera del rngo, después de la traducción seria error. Pero cuando ees memoria compartida entonces seria distinto, va a dar a un rango distnto de mis pasos como es memoria compartida se va a permitir.

Característica: la memoria sea organizada en módulos locales comopor ejemplo que hemos organizado espacio virtual de direcciones organizado en segmentos, entonces la memoria física también. Cuando tengo memoria física voy a poder tener la posibilidad de organizar en módulos. Entonces puedo cargar en una partre un código de un orpceso especidico y enn otra parte los datos globales, el stack en otra parte especifica, etc etc. Esta es una caracteriitica de memori vvirtual, entonces puedo cargarlos como segmentos, quizás incluso no seguidos, pero para eso voy a tener que mapearlos en donde se van a encontrar. No tengo que cargar todos los segmentos si no los necesitos. Si no he hecho ningún malloc o new dinámico, el secmento heap va a ser vacio, por lo tanto no va a existir en memoria física, solo si lo mapeo va a existir, localizando localmente la memorai física, tal que mapeo de virtual a física. Una cosa es lo que direcciono y otra es donde estoy situado. El código esta situado en el espacio de direcciones físicos. Ahí vive, pero esta direccionado desde espacio virtual, siempre va a direccionar las direcciones que le corresponde las virtuales.

Hay otra forma.

///////////////////////////////////////////////////////////////////////////////////////////////////////////

Clase 23:

Capitulo 7 sobre administración de memoria

Los sistemas antiguos no tenían direccionamiento virtual, hemos visto memoria física, no existía suporte sobre memoria virtual. Esta viene a facilitar administración de la memoria, hacerla mucho mas poderosa. Debemos entender las memorias físicas y lógicas. Los sistemas de admin virtual están basados en 2 tipos de dirección:

física que es absoluta, dirección de la ram, es una dirección de hardware de la ram. Si uno tiene una memoria física de 2gb, significa que existen 2\*2 elevado a 30 direcciones. Cada dirección representa un byte. En un bite puede estar solo 1 elemento.

.En cambio la dirección lógica es una referencia independiente de ubicación real en memoria física. Es un símbolo, por ejemplo como un rut. Nuestro nombre es como la dirección lógica, tal que se pueden repetir, pero se les asigna una sola dir física. Esto seria la diferencia en dirección física o lógica.

Una dirección relativa es una forma de expresar dirección lógica, es un ejemplo de dirección lógica. Que podría coincidir con la dir física. Sea la memoria física, y la memoria física tiene 16 bytes. Tal que tendríamos los bites 0000, 0001,…

Si allgun elemento se coloca en esta dirección, esta será su memoria física físicamente. Además, existe un espacio virtual de direcciones, espacio del proceso, tal que tiene este espacioin para guardar el código, el stack, los datos, heap. Esto también tiene una dirección. Pero cada proceso tiene su propio espacio de direcciones tal que dentro del espacio virtual esta mapeado como 0000, 0001… y así.

Las direcciones del espacio virtual se le llaman direcciones virtuales, son aparte de dirección física, y si se cargara utilizaría direcciones físicas. Supongamos que en el código hay una instrucción que dice “a=a+1”, tal que a es un símbolo que se traduce a una dirección y los datos están en otro segmento, tal que se almacena el valor de a en los datos. Y si quiero sumarle, hago un fetc, sumarle 1 en el procesador y hacer un write en a. esto es lo que hace el compilador que lo manda al espacio virtual y busca a en direcciones virtuales.

Si el proceso se cargara a memoria, la referencia de a =a+1, supongamos que se carga en memoria física a=a+1, lógicamente a esta en espacios direccionales. Tal que al mencionar a “a”, es que tengo la dirección virtual, sigue siendo en el espacio virtual, pero también esta en mem física en 1011. Entonces la función hay que traducirla y esto nos va a dar la dirección que esta en memoria física desde virtual. El proceso referencia espacio virtual de sus direcciones.

El pc a que apunta? Que es lo mismo de contador de programa.pregunta, pc apunta a virtual, stack pointer es a virtual, todas las direcciones del proceso se refieren a espacio virtual y cuando sse cargan a memoria física siguen direccionando a virtual. Tal que sea un dato que se carga en físico, el sistema debe traducir de virtual a física. Hay que traducirla de tal manera que direccione a física.

Esta es la clave de memoria virtual.

En ningún momento hablamos de memoria secundaria, sino de memoria hardware ram y virtual. En estricto rigor la dirección virtual no existe, y cuando una memoria virtual se carga en físico, sigue direccionando a virtual, hay que traducirlo.

Entonces tenemos lógica, física y relativa.

La dirección física como la virtual podemos referenciarla como algo linear 00001, 00002…

Y la relativa es un offset desde un lugar dado, tal que de x lugar a y lugar puedo contar relativamente la distancia entre los puntos.

Entonces la traducción de la dirección relativa, le llamaremos dirección virtual. Quien envia dirección virtual? Viene del procesador. El procesador al ejecutar en ciclo fetch, ejecuta el código del proceso, al ejecutarlo emite direcciones a variables, program counter, y estas son virtuales.

Por cada referencia a memoria, esta dirección se manda a un sistema de administración o traducción de memoria. Este es un sistema simplificado de traducción de direcciones de virtual a físico. Entonces lo veremos simplificado. En este caso será que será en el caso relativo a el proceso tal que el inicio es 0 a el final de la memoria virtual del proceso.

Dado un registro base, tal que seria la dirección física de inicio, tal que la dirección base es donde comienza el código, tal que iría supongamos en 0100 de mem física, tal que el proceso de espacio virtual se carga desde 0100, tal que este valor es el registro de donde comienza el proceso. Dada la dirección física le sumo el offset que es el desplazamiento para obtener la dirección virtual como desplazamiento y comparo si no he exedido el limite de mi proceso como por ejemplo estar en 1111. Uno de los criterios es protección de acceso para que no sea ilegal, por lo que lo checkea, si falta espacio va a dar un segmentation fault. Dada mi dirección virtual relativa le doy el offset y puedo saber donde esta virtualmente. Si sumo estas dos cosas me va a dar mi ubicación en memoria física absoluta, es decir, donde se encuentra en memoria física. 7

Existen otros métodos de traducción, estos sistemas dependen de la partición.

Partición de memoria: paginación y segmentación.

Paginación divide en bloques pequeños y de igual tamaño a memoria física llamado marcos, divide los procesos de memoria lógica o virtual en bloques del mismo tamaño llamados paginas.

Entonces diremos que memoria física lo vamos a separar en marcos y memoria virtual en paginas.

Todo esto se refiere a paginación.

Supongamos que tengo una memoria física, y esta dividida en 16 bytes.

Lo que diremos es que esto se va a dividir por ejemplo en 4 bytes por cada marco. Tal que tengo el marco 1,2,3,4. Esta división es efectiva en el sentido que cuando necesite memoria física, voy a ocupar todo el marco o no lo ocupo. Procuro memoria en la medida que hay marcos disponibles. Si la memoria se divide en segmentos, los marcos son muy pequeños, de orden de kylobities, así no tiene que cargarse completamente la memoria virtual en mem física.

Cuando dibujo memoria virtual de un proceso, es solo de uno. Y si dibujo mem física es de todo el sistema de memoria física. Espacio virtual de direcciones de un proceso, no me refiero al del sistema porque no existe. el espacio virtual es propio de cada proceso.

El espacio virtual esta hecho por código, heap, etc. Tal que supongamos que sea muy largo el espacio virtual. El sistema de paginación va a decir que vamos a separar memoria virtual en paginas del mismo tamaño. Tal que entonces voy a tener paginas del proceso tal que 1,2,3,4,5,6…

Tal que estoy cortando la memoria física y la memoria virtual. Entonces si quiero cargar la memoria virtual a memoria física, entonces voy a mover una pagina i a un marco j. en pagina i voy a tener supongamos un poco del código, datos globales y heap. En otra pagina puedo tener memoria y librería. No importa lo que quepa en el molde, lo cargo en un marco.

El pc es virtual. El stack pointer es virtual. Todas las dir del rpceos se refieren a espacio virtual

Ejemplo de memoria principal:

El proceso a es un proceso que al paginarlo se compone de 4 paginas. El proceso b al paginarlo se crea en 3 paginas y así sucesivamente. Entonces tendré espacio virutal de proceso a y b.

Y tendré el espacio físico del sistema.

La gracia de esto es que la pagina 0 del proceso a puede cargarse en cualquier marco de la memoria principal. La pagina 1 de proceso a se carga por separado en cualquier marco independiente de que proceso o pagina es.

Así el sistema carga cada pagina en un marco. Si entra 4 marcos de c, salen 3 marcos de b y entran 4 marcos de d. la pagina 0 se carga en 4, luego 5,6 y luego se carga en la pagina 11. Si un marco esta disponible entonces puedo cargarlo en ese marco, independiente de cual marco es o el orden.

La pagina virtual se carga en un marco físico.

Supongamos que tengo espacio virtual de un proceso. Mi código es largo. Tengo que dividirlo en paginas pequeñas, por lo que tendría múltiples paginas de código.

La ultima pagina del proceso, queda con el final del código pero ya guardo todo, por lo que esto produce fragmentación interna. El molde de corte el proceso no cae justo, por lo que la ultima pagina del proceso puede tener una cierta cantidad de fragmentación.

Como es que se produce la traducción usando paginación?

En el ejemplo anterior vimos que proceso a se carga en los marcos. El sistema va a tener una tabla de paginas como directorio para cada proceso, tal que correlaciona a el marco con la pagina. La tabla de pagina mapea las relaciones. Cada proceso tiene su tabla de pagina. Tal que relaciona al numero virtual de la pagina y el marco en el que fue cargado. Esto no es un mapeo de dirección virtual a física directa, sino que es uno de pagina virtual a marco. Esto se utiliza para traducir una dirección.

El sistema operativo cuando carga proceso a memoria entre mucha de las cosas que hace es que acepta el proceso a sistema, que es crear estructura de datos para administrar proceso en memoria, una de estas es la tabla de paginas, tal que cuando lo cargo de virtual a física, voy actualizando la tabla de paginas. Cada proceso tiene la propia.

Direccionamiento con paginación:

Como es que traduzco dirección virtual a dirección física. El sistema me da mi dirección virtual, lo divido en 2 campos, uno que es el numero de paginas y el otro es el offset de la pagina, como se cuantos bits ocupar para cada cosa? Extraigo el numero de pagina que en este caso es 1, accedo a la tabla de paginas de proceso, y encuentro el numero de marco que es el 6. Entonces tomo el bit de marco y copio el offset del desplazamiento. Tal que el desplazamiento con pagina es igual a desplazamiento con marco, tal que es una relación.

Ejemplo: dirección física de 16 bytes, direccionando en bytes, tal que los tipos de datos que tienen 4 bytes, la dirección es del primer byte, y el sistema extrae los 4 siguientes.

Entonces voy a tener mi memoria virtual. Supongamos que tengo el contenido de ab,c,1,2,3,4,x,y,z,5,6,,7,0,8,9.

Supongamos que las paginas son de 4 bytes y tengo mi memoria física mas pequeña. Con 8 direcciones, esto es posible gracias al direccionamiento virtual. Significa que caen solo 2 paginas en la memoria física.

Entonces sea que la dirección virtual esta enumerada en bytes tal que 000, 001, 0010…

Las paginas son de 4 bytes, por lo que tendré espacio para 4 paginas.

Y en memoria física tengo espacio para 2 marcos.

Entonces cargo pagina 1 en marco 0, y cargo pagina 3 en marco 1.

Mi tabla de pagina que me mantiene el mapeo va a decir:

Entonces tendría la dirección 000, 001, 010, 011, 100, 101, 110, 111

Cada dirección representa dirección de un byte. A lo que apunta es donde recide un byte.

La tabla de pagina entonces estaría enumerando las paginas y dentro de esta lista va a tener el marco en donde esta situado:

Pagina marco

1. -
2. 0
3. –
4. 1

Supongamos que quiero acceder a la parte de la pagina que esta en 1110 virtual, entonces la divido en 1 byte y 3 bytes en el offset dentro de la pagina. Tal que primero calculo el offset o desplazamiento, que en este caso para poder mapear cada byte en la pagina o marco son desde 00 a 11. Lo que queda es para el numero de paginas, es decir, cuantas veces puedo representar mis paginas. Tal que 11-10. Tal que es la cuarta pagina, y en esta cuarta pagina el tercer byte.

Entonces me voy a la lista pagina-marco y veo que mi pagina 11 se correlaciona con el marco 01.

Reemplazo 11-10 por 01-10.

Tal que me redirecciona a la parte donde el valor que en este caso es 8 esta cargado en memoria física.

Si llego a sacar una pagina del marco de memoria debo actualizarlo, lo mismo con poner una pagina en otro marco.

Tal que direcciono la relación.

Clase 24:

Forma en que se particiona memoria virtual y fisica, tal que una forma es paginacion (memoria fisica y virtual se dividen en partes de igual tamaño, trozos) y marcos.

Segmentacion se segmenta la memoria de la misma forma de que proceso esta segmentado. El proceso tiene un espacio virtual de direcciones, el mismo que hemos dibujado hartas veces que tiene codigo, datos globales, heap, stack. Entonces esta es la memoria virtual, y esta segmentada. Siempre lo ha estado. La memoria fisica la vamos a segmentar o mejor dicho, parte disponible (en estricto rigor ya tiene una parte para sst op) ahora la memoria no se particiona en paginas, sino que sea que yo cargo el codigo, lo cargo completo y este lo cargo en su completitud. Lo mismo con los datos globales. Si quiero cargar el stack lo mismo, y esto lo meto como un paquete cada uno en la memoria fisica. Ojo que hay que tener presente que hay uno de los espacios como el del stack que va creciendo, por lo que hay que tenerlo presente a la hora de meterlo en la memoria fisica.

Entonces van quedando espacios disponibles, estos pueden ser utulizados por varios procesos, tal que se segmenta de acuerda a mismos segmentos de memoria virtual. Entonces puedo tener otro proceso distinto con su memoria virtual, tal que un proceso supongamos tiene harto codigo, y otro pro ejemplo tienen hartos datos globales, pero esto tambien puede dar segmentacion externa, tal que mientras voy ocupando los espacios puede ser o no justo. Es muy parecido a particionamiento dinamico. La diferencia con esto ultimo, esque el proceso por ejemplo no tiene cargado el heap, ni el stack, tal que de memoria virtual a memoria fisica, en memoria fisica esta el heap y stack pero en mem fisica no esta. Porque en memoria virtual no es necesario que todo el proceso se cargue en la memoria, en cambio al particionamiento dinamico metia todo el proceso de una vez, de forma contigua y completa. En cambio en segmentacion se segmenta el proceso en diferentes lugares. Necesito guardar las direcciones de mis segmentos, por lo que necesitare una tabla de segmentos.

Esta tabla de segmentos va a mapearme el numero de segmentos a la direccion donde comienza el segmento.

Entonces la tabla de segmentos me va a mapear de numero de segmento a dirección inicial física.

La tabla se segmentos entonces seria de la forma:

Para cada segmento voy a tener una entrada como en el caso de la paginación. En la tabla de segmentos hay harta información, la mas importante es la dirección base que da el numero binario que nos dice donde esta el código, tal que me indica la dirección del inicio del segmento.

“aquí comienza el segmento” en paginación en cambio, tenia el numero de marco con el offset para construir la dirección.

En cambio en segmentación se me da la dirección física completa para ir al lugar especifico, es absoluta. Que me indica la dirección física de donde comienza el segmento. Entonces va a ser un numero de segmento que me va a mapear la dirección.

Sea que quiero ir a una parte del código de m virtual, tal que se tiene un offset. La dirección seria el inicio del código mas el offset. Tal que voy desde el código a la tabla de direcciones y la tabla de direcciones me manda, entonces le sumo el offset o desplazamiento que tenia desde la dirección virtual. Entonces sea que estoy en una parte del segmento, esta parte va a tener un desplazamiento desde el inicio del segmento, que es el offset. Luego voy a donde esta la tabla de segmentos que me envia a donde esta el inicio de la memoria física, y en la memoria física le agrego el desplazamiento offset pertinente para llegar a donde esta el código especifico que seria el mismo de memoria virtual.

Esta es la forma de direccionar segmentos.

Segmentación particiona la memoria en elementos lógicos, tal que particiona memoria física y virtual. Pero los segmentos tienen un largo variable, el del código a el del heap puede ser muy diferente, no sufren fragmentación interna porque son cargados completamente cada segmento. No es necesario cargar el proceso completo, sino que el segmento completo. Los sistemas van a definir tamaño maximo de segmento. Tal que puedo determinar el numero de bits para direccionar el segmento.

Lal igual que el caso anterior, voy a tener una tabla donde direcciono el numero de segmento y otro con el offset del segmento. Usando el caso anterior. Sea un campo para direccinoar el numero de segmento y otro campo para el offset del segmento.

Entonces sea el numero de segmento, obtengo la direccion base del segmento.

Sea que tengo un computador de 1 gb, tendre 30 bits para direccionar el gb, tal que direcciono a la dir fisica. En paginacion yo concatenaba el nro de marco con el nro de offset. Pero en segmentacion debo sumar ambas direcciones en un total binario, como se para el caso de segmentacion el tamaño del offset y del marco? Va variando por el tamaño de cada segmento sobre su programa principal, datos, funciones, etc

Detallaremos mas claramente lo que es memoria virtual.

Una de las caracteristicias es que un proceso puede usar segmentacion o paginacion, lo uno o lo otro. No necesariamente contiguo en memoria. Proceso no tiene que ser completamente cargado en memoria fisica, puede ejecutarse con solo un pedazo y a medida que lo necesita, cargo pagina o segmento que me permite ejecutar funcino.

El stt op trae a memoria principal pocos trozos de programa, es decir segmentos o paginas.

A este conjunto de paginas o segmentos se le va a llamar conjunto residente o resident size.

Ese conjunto puede ir cambiando con e tiempo, ya que pueden entrar o sailir segmentos o paginas, tal que varian con el tiempo dependiendo de como se ejecuta el proceso.

Cuando un proceso intenta acceder a una direccion que no se encuentra en conjunto residente, se produce un page fault. Entonces sst op emite la interrupcion asociada, que produce que se haga un servicio a sst op que es cargar la pagina que no se envontro. Hay que traerla. Se interrumpe, pongo proceso en estado bloqueado y emite requerimietno para el trozo necesitado. El sst op emite el req de io que es elctura a pagina a disco y mientras se hace el io, se puede ejecutar otro proceso.

Cuando la pagina esta en memoria principal, entonces se actualiza la tabla de pagina y el proceso vuelve a la cola de listos y el proceso vuelve a acceder a la dir de cola de listos, tal que antes no se encontraba generando page fault, pero ahora la va a encontrar.

El proceso se esta ejecutando y supongamos que a[i] le asigno 8. en ese momento, esto se traduce en una direccion virtual.dentro del codigo es una direccion virtual, lo quiero buscar a[i] pero no lo encuentro, esta la paginacion pero no esta en marco. Tal que debo traer con un requerimiento de io a[i], tal que a[i] pertenece a un segmento, y este debe ser cargado en disco. Todas las paginas y segmentos estan en disco, y se actualizan en disco sea que hallan cambios.

Entonces proceso queda bloqueado y no vuelve a listos hasta que se le avise que fue añadido. Luego el proceso vuelve a ejecutar la instrucción, encuentra la informacion que necesitaba y sigue. A esto se le llama miss de tabla de pagina, tal que direccion virtual no puede ser traducida a fisica ya que no esta cargada, produciendo page fault.

Como no tienen porque estar completamente cargados en memoria, permite que mas procesos puedan recidir en memoria. Esto ayuda para mejorar la utilizacion del procesador.ya que si hay hartos procesos en el sst, es probable que procesador siempre tenga la oportunidad de planificar procesos mientras otros procesos hacen io, disminuyendo espacios donde procesador no hace nada y esta esperando.

La mejora al procesador tiene un limite , ya que se puede producir “trashing” en memoria virtual. Es ventajoso esto hasta cierto punto, luego de eso se produce trashing.

Un proceso puede ser asi mas grande que la memoria disponible, tal que memoria virtual es mas grande que memoria fisica, permitiendo que funcione. Y memoria fisica asi mismo uede alojar no solo a esa memoria virtual sino que a varias, tal que comparten memoria fisica, ya no existe esta restriccion. Es posible entonces tener pedazos de memoria virtual en memoria virtual, tal que mem fisica tiene derecho a toda la memoria virtual.

La desventaja es que si es mucha la diferencia, se pueden ocurrir muchos page fault, tal que muchos fallos de pagina. Aunque el sistema puede ejecutar, va a decaer. Se necesita igualmente memoria fisica. Solo que no es limitada por ella.

Cada proceso tiene un espacio virtual.

Sea que tengo un computador con 32 bits, si tengo el registro que direcciona la memoria es de 32 bits. Esto significa que voy a tener 2 elevado a 32 bytes, que es 2 elevado a 2 y 2 elevado a 30. Un sistema computacional de 32 bits va a tener una memoria virtual de 4 gb. Esta memoria virtual de 4 gb, puede tenerla cualquier memoria virtual. Entonces cada memoria virutal a va tener derecho a 4 gb. Entonces esta es una forma de ver la memoria (mas que almacenar al memoria) tiene que ver con la vision de los procesos que tienen sobre la memoria, los procesos creen que corren en 4 gb, pero solo estan viviendo a traves de segmentos.

Un sistema de 48 bits. Esto es 2 elevado 48 bytes. Esto es 2 elevado 8 por 2 elevado 40, lo que seria 256 tbytes.

2 elevado a 10 = 1kbyte

2 elevado a 20 = 1 mbyte

2 elevado a 30 = 1 gbyte

2 elevado a 40 = 1 tbyte

2 elevado a 50 = 1 pbyte

2 elevado a 69 = 1 hbyte.

En estricto rigor pc usan 64 bits, pero realmente utilizado son 48 bits. Esto es utilizado actualmente. El hardware esta preparado para 64 bits, tal que tienen derehco a que procesos tengan 256 tbytes.

Dentro de memoria virtual esta sst op, tal que kernel utiliza mem virtual. Luego en ese proceso va a estar el codigo, stack, etc.

Tal que irian las direcciones de la forma 0x00000(kernel) a 0xffffffffe

Eso es la clave en administracion en memoria virtual.

Porque funciona este metodo? Tal que el proceso no este recidente completamente en memoria y se ejecute eficientemente? Es posible por el principio de localidad: la mayor parte del tiempo, lo que se carga en la memoria es lo que se usa repetidamente, tal que hay una localidad espacial y una localidad temporal. Tal que si yo accedo a una direccion de memoria de un dato,e s muy probable que pueda acceder a la direccion vecina de adelante o de atrás. Los arreglos al procesarlos funcionan asi, tal que los procesos secuencialmente. A esto le llamaremos que es una vecindad, son direcciones vecinas, el principio de localidad dicen que se acceden en tiempos parecidos.

La localidad temporal tiene que ver con el codigo. Sea que tengo mi codigo, es muy probable que el ciclo fetch vaya obteniendo siempre el program counter +1, tal que es una vecindad o linealidad de direcciones. Tal que si traigo todo un contenido de codigo de un proceso, esto es local, ya que voy a ir accediendo a la siguiente direccion de forma secuencial en tiempos parecidos. El principio de localidad es un principio que tiene que ver con lo que yo programo. Si yo programo mal es probable que el principio no los cumpla.

Si no es secuencial no se cumple la propiedad.

Se debe explotar los principios de localidad. Tal que puedo traer pedazos de procesos vecinos, tal que no debo preocuparme por otros trozos de codigo.

Entonces sea que cargo un pedazo de codigo, todo perfecto en la pagina ya que es secuencial, pero ya al terminar con esas instrucciones va a ocurrir un pagefault. Y luego de que obtiene el trozo, vuelve a funcionar secuencialmente hasta que termine e trozo.

Puedo decir que memoria virtual es ineficiente porque como proceso no esta cargado en memoria hay que traer los procesos, pero esto es salvable por el principio de localidad.

Conjunto residente tenga buena localidad.

Supongamos que inicialmente proceso falle hasta que obtiene las paginas que si necesita, el conunto recidente correcto. Luego ocurre un pagefault y puedo traer pagina o paginas, tal que no es solo la pagina que hahce e pagefault, sino tambien otras aledañas.

El direccionamiento de memoria virtual con paginacion es a traves de la tabla de paginas. La tabla de paginas tiene una entrada para cada pagina del proceso. Esa entrada no es donde e sta la pagina, sino que es el numero de marco de la mem fisica de donde esta en la maquina. Cada proceso va a tener el nro de pagina, el offset. (virtual adress)

Y la tabla de pagina de entrada va a tener el numero de pagina, p, m y controlador.

El bit p de presencia o validez

Y bit m de modificacion o suciedad.

Se encienden p cuando esta cargada, al no estarlo pagefault. El bit de modifcacion cambia de 0 a 1 cuando se modifiica. Sea que le cambio el numero a a[i], tal que la pagina cambia de 0 a 1, hay que guardarla a disco.en memoria secundaria va a haber una version actualizada del proceso completo, estan todas las paginas, tal que se van reguardando cuando voy modificando.

Dir virtual a dir fisica. Usando tabla de pagina. Divido mi dir virtual en 2 campos, tomo el nro de pagina, accedo a la tabla, obtengo el marco, concateno el offset que ya me dan, y voy a memoria fisica.

Tabla se enuentra en memoria fisica.

Este numero de papgina en realidad es un offset a partir de el puntero de la tabla de pagina. Este puntero es un registro que me dice inicio y final de la direccion de memoria fisica de la tabla de pagina. El nro de pagina se utiliza como offset. Cuando se carga el proceso en procesador y se hace cambio de contexto, tambien sst op le va a decir a procesador: “este proceso tiene su tabla de papgina cargado en x memoria fisica” y esto es lo que se carga es el valor en el registro del procesador, tal que la suma es dentro del procesador. Esto ocurre dentro del procesador, no lo hace el sst op,

El procesador se ejecuta, emite una direccion y todo lo que hace es del procesador, no esta involucrado el sst op, solo se involucra cuando ocurre pagefault, tal que sst op toma control de la situacion.

Cuando la direccion se accede a algo que no esta aun en memoria fisica, va a ocurrir pagefault, tal que es un fallo recuperable. En cambioi un segmentation fault es irrecuperable, tal que se elimina proceso.

Clase 25:

Mecanismo de traduccion de virtual a fisica a traves de tabla de pagina. La tabla de pagina es una tabla por proceso, va a tener un nro de pagina virtual a nro de marco, que se usa para componer la direccion fisica. La tabla de pagina no contiene las paginas del proceso como tal, sino que estan o en memoria fisica o memoria secundaria. En memoria secundaria se encuentra todo el proceso.

La tabla de pagina esta en memoria fisica y se encuentra bajo esquema de paginacion. Si estamos viendo un sst de admin de memoria virtual donde memoria esta paginada, entonces todo esta paginado. Tabla de paginación tambien se encuentra paginada.

La tabla de pagina, al estar paginada, entonces podemos hacer el calculo de cuanto mide la tabla de paginacion en memoria.

Supongamos que el sistema del computador es de 32 bits.

Las paginas son de 4 kbytes.

Si nuestro sst computacional es de 32 bits, y las paginas son de 4 kbytes, entonces seria 2 elevado a 2 por 2 elevado a 10. Lo que implica 12 bits para el offset, el resto 20 bits es para el numero de pagina.

Todo esta dado por el tamaño de la pagina que es 4 kb, tal que necesitare 12 bits para direccionar cualqueir offset dentro de una pagina, tal que seria 2 elevado a 12 direcciones.

La tabla de pagina estaria paginada en paginas de 4 kb, cada entrada esta compuesta de bits para numero de marco, control, p y m. el largo de la entrada de pagina entonces, sera que cada entrada de la tabla de pagina es de 4 bytes, eso significa que el ancho de la tabla es de 4 bytes, y el largo 2 elevado 20, ya que tengo 20 bits para direccionar entradas, entonces la tabla va a pesar 2 elevado 20 x 2 elevado 2 que es 4 mbytes.

Todo lo que esta en memoria esta dividido en paginas de 4 kb, y la tabla pesa 4 mbytes.

Eentonces cuantas paginas usa la tabla de paginas?

2 elevado 22 /2 elevado 12, tal que seria el tamaño total de la tabla de pagina dividido por el tamaño de cada pagina. Esto nos da el numero total de paginas que es 2 elevado 10.

Hemos calculado que esta tabla de pagina en memoria se divide en 2 elevado 10 paginas, voy a tener 2 elevado 10 paginas.

Una tabla de pagina es bastante pesada, ya que encesitare 4 mbytes para guardar tablas que mapeen mis paginas.

Una forma de disminuirlo es hacer las paginas mas pesadas.

Supongamos que las paginas son de 8 kb, el sistema es de 32 bits. Lo vamos a dividirlos 8 kb en 2 elevado a 3 x 2 elevado a 10. Entonces offset es 13 bits y nro de paginas es 19 bits.

Entonces mi ancho va a ser 4 bytes y mis entradas el largo va a ser 2 elevado. Esto implica que la tabla de papgina pesa 2 elevado 19 x 2 elevado 2 y esto seria 2 x 2 elevado 20. Tal que es 2 mbytes, tal que duplicamos el tamaño de la pagina, entonces ahora necesitariamos la mitad de paginas.

Esto significa que necesitaria la siguiente cantidad de paginas: 2 elevado 21/2 elevadlo 13 y esto nos da 2 elevado a 8, que es 256 kb.

El sst puede controlar el tamaño controlando el tamaño de las paginas idealmente las pag. Generalmente las paginas son de 4kbytes, 8 kbytes, 16kbytes. Las paginas no son del orden de megabytes, son relativamente pequeñas. Una vez que sistema elige el tamaño, ese va a ser el que va a utilizar siempre.

Independiente de el tamaño, en general, las paginas pesan mucho, aprox 4, 8 mb.

Cada direccion virtual es traducida a direccion fisica a traves de tabla que esta en memoria principal, esto es una lectura a memoria principal desde virtual a fisica y luego vuelvo a ir a memoria principal para buscarla, tal que son 2 accesos.

Una forma de disminuir el tamaño requerido por tablas de paginas es utilizando tabla invertiida de pagina, tal que es una tabla para todos los procesos en vez de una tabla por cada proceso.

Esta tabla invertida es compartida por todos los procesos.

Tiene un mapeo distinto al que tenia el de cada proceso, mapea de forma distinta para un mismo objetivo.

Entonces tendremos en la tabla invertida lo siguiente: el numero de pagina virtual, tal que yo ya se como se divide la pagina con su offset, tal que la pagina es un argumento a una funcion de hash, entonces me manda a la pagina especifica. La funcion de hashing lo remapea de lo que tiene la pagina hacia lo que es el marco. Por lo tanto, lo que hago es ir a la tabla donde me manda por el marco, y es ahí donde se almacena el numero de pagina, el id del proceso que esta entrando a esta pagina. Si esta entrada esta libre, entonces la pagina se almacena en el marco, y de aquí saco el numero de marco que ya me lo han dado y se manda a la direccion fisica que s ecompone por marco y offset. El offset se mantiene.

Puede ocurrir que el marco ya este ocupado la tabla de pagina con otro marco, entonces lo que va a hacer es buscar secuencialmente otra parte donde este libre. Ok, lo que hago es que la pagina del proceso especifico se va a almacenar en el marco especifico, tal que ese va a ser el que se va a utilizar para crear mi memoria fisica. En una tabla invertida dd epagina lo que ocurre son coliciones, tal que dos virtuales se mapearon a 1 fisica, para solucionar es que busca entrada disponible de marco, por lo que en este sentido la tabla es invertida. Siempre es virtual esta almacenado en fisico especifico, tal que necesitare tambien guardar el id de la pagina del procesoo. Este es mucho mas ancho. Cada cosa va a tener 4 bytes aproximadamente, supongamos que son 16 bytes, como ancho de pagina. En cambio el anterior es 8 bytes aprox.

Cuanto pesaria esta tabla de pagina inversa? El largo seria el numero de marcos que hay en memoria principal.

Supongamos un sistema de 32 bits con paginas de 4 kb, y 1 gb de memoria principal.

Entonces para 32 bits serian 12 para el offset y 20 para las paginas.

Supongamos que cada entrada de la tabla de pagina o que el ancho de la tabla invertida de pagina es de 16 bytes. Entonces, el largo de la tabla invertida, cuantos marcos hay de memoria principal?

1 gb es lo mismo que 2 elevado 30 y eso lo divido a el tamaño de la pagina, tal que 2 elevado 30/ 2 elevado 12, tal que tendre 2 elevado 18 marcos. Tal que seran 2 elevado 18 entradas. La tabla invertida de paginas pesa 2 elevado 18 x 2 elevado 4. esto me da 2 elevado 22 y esto es 4 mb.

Entonces son 4 mb que son para todo el sistema, no es para cada proceso. Entonces no necesito tanta memoria para mapear esto. La desventaja son las coliciones. Cuando ocurre una colicion (sea la dir fisica, donde voy a ir a buscar mi direccion que tengo por el marco y el offset).

Dada una virtual adress que me mapea en la tabla inversa a el frame adress.

Sea que el virtual adress quiere buscar un marco que ya usado, entonces me direcciona con un puntero a otro acceso que si esta libre, esto es otro acceso a memoria fisica. Entonces a lo minimo tendria que acceder 3 veces a memoria fisica.

Generalmente lo ideal es 2 accesos, pero con esta forma de tabla inversa puedo tener muchas coliciones que cada una significan un acceso a memoria.

Entonces el tiempo de acceso es importante, entre mas, mas me demoro. Pero lo bueno es que uso mucha menos memoria.

Concepto de tlb translation lookaside buffer.

Es unn sistema de hardware que esta dentro de procesador, memoria asociativa, mantiene pedazos de memoria de pagina. Quiero evitar 2 accesos a memoria, como puedo evitarlo? haciendo que parte de la tabla de pagina llevarlas desde memoria al procesador. Este es un lookaside buffer, que es hardware especial dentro de procesador que contiene entradas de la tabla de pagina, el tlb es chico. Solo estan una de estas tablas de paginas aquí, que son las que mas estoy usando.

Sea que viene una direccion virtual, en vez de ir a la tabla de pagina, lo que hago es checkear el tlb que esta en el procesador, ver si el mapeo esta en la tlb. Si esta en la tlb, se habla de un tlb hip. Saco de esta tabla el marco en las entradasde lookaside, lo compongo con el marco inmediatamente. Tal que el checkeo de tlb es paralelo, en orden 1 se sabe si esta o no esta.

Cuando es un hipp puedo obtener el nro de marco,y obtengo direccion fisica, de otro modo es un tlb miss, tal que voy despues a la tabla de paginas. Si la encuentro, entonces saco el nro de marco, construyo mi direccion fisica. Pero tambien puede ocurrir que la pagina no este cargada en memoria principal, esto es pagefault. Cuando voy a la tabla de apgina y veo que el bit de presencia esta en 0, tla que la pagina no esta aun en memoria principal, entonces debo cargar la pagina que esta en memoria secundaria y traerla en memoria principal.

Este proceso de page fault es un proceso de io que bloquea el proceso hasta que la pagina no este cargada.

Entonces nuevamente, dada una direccion virtual, checkeo la tlb, este checkeo es dentro de procesador, por lo tanto no es un acceso a memoria, es un checkeo dentro de procesador por lo que es muy rapido. Si no lo esta entonces voy a tabla de pagina para acceso a memoria y si no lo esta entonces voy a memoria secundaria con un io.

El tiempo deacceso depende de donde encuentre la direcccion. El tiempo de acceso promedio de generacion de la direccion fisica, va a depender de si fue un heap de tlb, de memoria principal o memoria secundaria.

Entonces el Ttlb va a ser el tiempo de acceso a la tlb microsegundos

Luego Tmem va a ser tiempo de acceso a memoria principal milisegundos

Luego Tsec es el tiempo de cargar la pagina desde disco a memoria principal

Tal que cada uno es mucho mas pequeño que el siguiente.

Htlb: probabilidad de encontrar el numero de marco en la tlb

Htp: lo mismo en tabla de pagina tp

Entonces el tiempo promedio de generacion de la direccion fisicia seria: HTLB\*TTLB+(1-HTLB) \* TTLB + HTP \*TMEM+(1-HTP)(TMEM+TSEC)

Una vez que esta tabla esta cargado, el priceso espera en la cola de listos par volver a hacer ne lo que fallo, incluyendo el tiempo de bloqueado, los io, etc. Todo se suma.

Entonces es el tiempo promedio en el que este en la tlb y el tiempo promedio que no esta en la ltb.

-----------------------------

Todos toman el tenedor tal que produce deadlock

Para solucionarlo debo evitar que entren los 4 filósofos tal que restringió. Hago un nuevo semaforo que es room, lo que hace es evitar que entren los 4. Uno de estos no va a poder participar, tal que no van a tomar ningún tenedor. Entonces el 2 puede tomar el 3, tal que libera el deadlock, liberando el flujo, evita que entren los 5 filósofos a competir.

Room debería ser n-1 tal que filósofos n.

Vimos la solución por hardware y software, sistema operativo con mutex. Ahora con lenguaje. Los semáforos tienen ciertos problemas, tal que depende del programador, tal que no puedo garantizar la exclusion mutua. Solo si lo hago bien para el caso.

Ejemplo: en el consumidor tengo un wait y un signal, tal que es enter y exit. Pero si se me queda por ejemplo un wait de mas, tal que decremento, incremento y decremento, tal que si hago un signal antes de un wait, entonces no haría nada, no hay cambio, por lo tanto siempre va a poder pasar. También puedo inicializar lo mal el n, tal que es erróneo, entonces la locks son preferidos solo cuando estoy seguro que lo he programado bien.

Los semáforos no solo permiten exclusion mutua sino también coordinación entre hebras, tal que tengo semáforos de exclusion mutua y semáforos de condiciones, tal que puedo condicionarlo a que solo entren 3 hebras, tal que el monitor siempre garantiza exclusion mutua. Un lenguaje de programación tiene la capacidad de crear un un objeto que sintácticamente lo provee el lenguaje de programacion. Pienso el monitor como un objeto y tengo varias hebras y me garantiza a exclusion mutua que solo una hebra va a poder entrar a la exclusión mutua.

Quien puede sacar y poner cosas en esta caja es solo si se entra y es exclusiva. Las variables locales se acceden solo en el monitor, provee encapsulation, tal que solo puedo acceder a ella con métodos en getters y setters. Se dice que uno ingresa a monitor cuando invoco los métodos públicos, tal que puedo llamarlos desde afuera. Solo una va a estar adentro. Entonces voy a tener una cola de hebra que ha invocado un método del monitor. Solo cuando hebra sale la exclusion mutua se va a abrir para que pueda ingresar otra al monitor, no hay porque usar semáforos.

Entonces sea un monitor los datos locales son los datos compartidos, tal que se encapsulan dentro del monitor para que su acceso sea exclusivo. Los datos globales al monitor, tal que son locales del monitor, tal que de afuera no puedo acceder, solo dentro del monitor.

Entonces tendre los métodos del monitor que pueden ser públicos o privados y tendre los privados que son los constructores.

Cuando la hebra dentro del monitor sale, permite a la hebra del inicio de la cola ingresar. No es atómico, se bloquea dentro del monitor en la cola de sus variables.

Las variables de condición condition variables, estas son específicos del objeto monitor como struct. Además tenemos una cola para cada condición de variable, cada condición tiene una cola de hebras que se bloquean en el monitor.

Supongamos el productor consumidor. Buscamos la solución. El buffer compartido lo meto dentro del monitor, tal que son métodos. Sea un buffer vacío en el monitor. Viene una hebra que entra y es una hebra consumidora. La hebra va a entrar a este método y pregunta si el buffer está vacío, y como es cierto no puede consumir: se bloquea. Wait() sobre la variable de condición. Tal que la hebra va y se bloquea en la cola de condición. Queda bloqueada dentro del monitor, tal que me quedo bloqueado invocando wait(). Lo mismo ocurre si buffer esta lleno y entra una hebra productora, sea una hebra que quiere producir y buffer esta lleno, no hay espacio, debe bloquearse, tal que hace wait() tal que se va a bloquear a la cola de variable de condición. La variable de condición es un lugar donde la hebra espera. El wait siempre bloquea a la hebra, no hay valores de positivo, negativo o condiciones, no hay un valor de posibilidad, solo es un binario de siempre bloqueo si lo invoco. Cuando la hebra se bloquea dentro del monitor debe permitir que la que esa esperando.

Tengo la hebra roja que es consumidora, encuentra el buffer vacío y se bloquea. Afuera del monitor hay otras hebras bloqueadas. Al bloquear adentro, despierto a las que están afuera del monitor. Esto garantiza exclusión mutua, tal que abre la exclusion mutua para que entre la hebra y enseguida se cierra al entrar la hebra. Ingresa hebra azul tal que es productora y produce elemento en buffer, tal que ya no está vacío. La hebra productora manda un signal al bloqueo de las consumidoras con el fin de avisar a posibles bloqueadas en el monitor que son consumidoras. Este signal es que la que esta bloqueada en la cola puede volver y vuelve justo despues de su wait. Esto es solo si la hebra azul sale del monitor para que la hebra roja sea la exclusiva en todo el monitor.

Todo esto lo hace la variable de condición, no hay porque controlarlo.

Lo mismo ocurre cuando productor encuentra el buffer lleno, tal que al consumir, es despertada por el consumidor que hace un signal y luego se va para que despierte luego del wait que había llegado del productor para que cree un objeto.

Cada vez que hebra entre y no pueda hacer su cometido, se bloquea en monitor.

Va a ingresar una hebra tal que nunca se bloquee, va a entrar, va a salir normalmente sea el caso que no haya cola y no haya ninguna variable de condición que no se cumpla.

Única forma de manejar las variables de condición son con wait y signal, estos no tienen que ver con los wait y signal de semáforos.

En un monitor no voy a usar semáforos. El monitor sirve para soluciones particulares.

El monitor es un mecanismo de comunicación asíncrona entre multiples tareas.

Cuando quiero que hebras se comuniquen de forma asíncrona tal que se comunican información tal que no se cuando y en que momento lo harán, es limitado, puedo implementarlo con semáforos y mutex, y pétreas, peor es mas complicado.

Sea el productor consumidor, como se van a comportar los clientes? Tal que son los que utilizan el monitor. Sea el monitor, con mi cajita de exclusion mutua y tengo varios agentes que se comunican información de forma sincronía. Como va a ser la implementación. El productor va a estar en un loop infinito, produce elemento, ingresa elemento, y luego lo mando al monitor. El append es para ingresar al monitor. Lo mismo con consumidor, tal que ingresa a monitor para sacar elemento del monitor y luego lo consume (no importa como exactamente).

No hay semáforos, tal que m append y m take son exclusion mutua.

Entonces tengo un buffer con caracteres, mi arreglo circular, in y out y count, tal que numero de elementos en buffer y 2 variables de condición not full y not empty para vacío y lleno respoectivamente. Necesito 2 variables para que se bloquee. Estas son las 2 condiciones.

El constructor crea m, tal que inicializa las variables. Tengo 2 métodos públicos que son append y take.

Sea que el buffer esta lleno, entonces me bloqueo en variable not full tal que me bloqueo hasta que no este lleno.

Ingreso elemento en buffer circularmente y incremento,checkenando si esta no vacío. Si productor crea elemento en buffer , manda enseguida un signal a take aviasandole que no está vacío, independiente de la cantidad de objetos que pueda tener, no tiene un efecto contrario o coolateral si es que no hay una variable condicionsl.

Como lo hago en acceso concurrente.

Como implemento mutex con las hebras posix. Como sincronizo? Voy a ocupar mutex binario path read, que me provee variables de condición. No hay monitores, porque monitor es una características de c o c++ esto no esta asociado a lenguaje lo que veremos en posix. La función para bloquear mutex es path read mutex lock que es nuestro wait. El lock es igual que el wait, tal que si. El mutex esta bloqueado entonces etc etc igual que el wait. Y lo mismo de unlock para signal. Entonces como funcionaria seria declarar el pthread mutex lock, lo inicializo, pthread mutex initial, le paso un puntero. Luego viene pthread mutex lock y entonces viene la seccion critica y luego pthread mutex unlock.

Entonces el in it es super important porque parte abierto, debo inicializar el mutex que al principio esta indefinido, debo inicializado tal que debo tratarlo como un abierto-cerrado.

Entonces sea que encuentra el mutex abierto entro y lo cierro. Otra hebra lo encuentra cerrado, queda bloqueado. Cuando la hebra que entro sale, hace unlock con un signal entonces puede pasar la siguiente hebra lockeada, luego que sale puede entrar la subsiguiente que estaba esperando y así susecivamente.

Las variables de condición en pthread trabajan muy facilmente parecido a monitor, no son valores sino que es una especie de objeto que trabaja con wait y signals, tal que bloquean la hebra que lo lama siempre y despiertan hebras si es qué hay una hebra dormida, puede que no tengan sentido.

El signal siempre esta asociado a variable de condición, en cambio el wait esta asociado a variable de condición y al mutex. Tal que el wait duerme a la variable en especifico, bloqueando hebra especifica y abre mutex del monitor, bloquea la hebra y libera el mutex tomado, tal que la única forma que lo úeda usar es que este usando mutex. Si no usara mutex seria un error usar variable de condición, es necesario el mutex para hacer esto.

Productor ingresa y encuentra buffer lleno. El mutex log es para ingresar a la sc. Me bloqueo en el wait. Y si se bloquea y no libera el mutex que acaba de bloquear, entonces todo se va a quedar bloqueado. La hebra debe bloquearse para que el mutex pueda abrirse para que pueda seguir con la seccion critica, tal que puedo ingresar. Tal que puede ingresar la consumidora, entonces puedo ingresar, saco un elemento del buffer y le aviso al productor que ya no esta lleno, tal que es donde esta bloqueado productor para que este se despierte, esto es gracias a que mutex esta abierto. Seria un error e usar variable de condición y no usar mutex.

--------------------------------------------------------

Fragmentacion interna con particionamiento fijo en memoria fisica

Otra forma es que todos los procesos puedan cargarse en cualquier particion disponible, no importando el tamaño. Otra posibilidad es buscar cual es la particion suficientemente grande que pueda cargar

Numero de particiones es fijo en el caso de particiones físico. Nivel de programación es el numero de procesos que pueden estar ejecutándose en el procesador, a esto se le llama nivel de multiprogramación. Siempre puede existir una partición mas grande que no puedan ejecutarse. Estos son la base para que pueda crearse la memoria virtual

Sea que la partición mida 2 mb, van a estar esperando en la cola para en algún momento que sea su turno, tal que va a esperar al espacio exacto que se libere. En cambio en otra forma, va a buscar el primer espacio que quepa, independiente de que sea mucho mas grande o justo.

La mayoría de nosotros pensamos que el priceso se carga en memoria, tal que el proceso entero esta cargado en memoria. Esta noción ya no va a ser cierta. Particionamiento dinámico es que toda la memoria esta disponible, no hay particiones, solo lo que necesito. La memoria de 64 mb comienza completamente disponible, tal que el primer proceso llega y se carga desde el principio y me queda una articion de 36 mb disponibles. Luego llega otro proceso, se carga, y se particiona la memoria disponible. Luego se va el proceso 2, y se libera donde estaba el proceso, quedando particionado el espacio libre y el espacio que estaba libre desde el principio como separados. Entonces van a haber segmentos ocupados y liberados, se actualiza la lista. Lo que ocurre es que se van produciendo en la medida que salen y llegan procesos, tal que quedan pequeños espacios de memoria en la que ningún proceso cabe, esto es fragmentación externa. El proceso solo utiliza lo que necesita, pero la fragmentación ocurre en la memoria en pedacitos que no son posibles rellenar. Entonces todos estos poquitos van sumando.

Tengo un requerimiento, tengo la memoria, pero esta memoria esta fragmentada.

Solución: compactación: yo puedo poner todos los procesos juntitos, correrlos, y dejar un espacio libre nuevamente. Compactar es un proceso caro. Todos los procesos van a memoria secundaria y luego se vuelven a cargar, por lo que es caro en tiempo y recursos como de disco.

La fragmentación es cuando la memoria que esta disponible no me sirve.

El acceso del proceso no depende de memoria aactualmente. Pero antes estaba restringido a la cantidad que se asigna en memoria al proceso.

Supongamos un proceso de 3 megabites. Entonces este va a fragmentar aun mas a mi memoria. El espacio que me sobra no lo voy a poder ocupar independiente de donde, porque lo voy a poder compactar.

Pero para dinámico ya no da lo mismo, debo crear algoritmos para meter procesos de forma mas eficiente o según criterios que necesite.

Los algoritmos mas conocidos son: best fit, que utiliza la fragmentación mas pequeña donde quepa proceso, me va a dejar muy poca memoria fragmentada, ocupo el que mejor se ajusta. Luego esta el wort fit, que usa el bloque de memoria mas grande y que pueda alojar al proceso, la talla mas grande, me va a dejar fragmentaciones grandes. El first fit dice el primero que encuentre que le quepa, entonces va a ordenar una lista de memorias y no le importa el tamaño, solo que pueda ser ocupado. El next fit busca desde la ultima posición de la ultima vez que posiciono el anterior proceso. Entonces va a tener un orden supongamos en una iteración.

Best y worst fit eligen por tamaño, requieren revisar todos los tamaños, son de orden n o(n). en el mejor y peor caso son de orden n, tal que va a recorrer todos para poder elegir.

El first fit de orden n, puede tener en el mejor caso orden 1 o(1). El next fit también puede ser en el mejor caso de orden 1 o(1).

A final del día best y worst son los peores, ya que son ineficientes. Encambio el first y next son eficientes en tiempo.

Entonces tendré la memoria física fragmentada, tal que mi ultimo proceso era de 16 mb. Por lo tanto el first fit va a comenzar del inicio de la memoria y next fit va a comenzar desde donde se metió el anterior proceso. Ambos consideran la memoria circular, tal que si se acaba la memoria, va a recorrerlos todos de nuevo desde el inicio.

El best fit va a elegir el tercero que encuentra en el caso. Next fit va a comenzar desde el anterior, por lo que era el segundo en este caso. El worst y el best se demoran en buscr todos y elegir el mejor.

Esto es memoria física, se carga todo el proceso tal que se carga o no se carga, ya que es un todo. Es un particionamiento físico y dinámico, nunca se va a poder cargar un proceso mas grande que toda la memoria disponible. Entonces si es que pesa mas, no va a poder ser ejecutado.

Memoria virtual logra resolver este problema.

Sistema operativo debe tener una estructura de datos, tal que el primer trozo que escoge.

Memoria: tipos de memoria, memoria física es la que esta dentro de memoria principal, dirección de hardware y el espacio de este espacio físico depende de cuanta memroai tenga mi computador.

Memoria virtual: dirección independiente de memoria física, es lógica, no es una dirección de memrai física, sino que referencia a espacio virtual de direcciones del proceso, que es diferente a la dirección de memoria física.

Los procesos direccionan otra memoria, hay que hacer traducción de virtual a memoria física.

Cada uno de estudiantes están en sala en sus filas y columnas como matriz, tenemos dirección física de matriz, entonces si en memoria virtual llamo a un algo, independiente de donde este va a existir, es relativa algo.

Sea mi espacio de direcciones que puede ser cualquiera, sea un inicio, que es relativa al inicio por lo tanto 0, y luego 1,2,3,4… inicio +i (0+n) tal que es relativo al inicio.

Cuando direcciono al paso del espacio virtual de direcciones, todos partel del 0000, tal que parten del inicio, y terminarían en ffff, tal que el proceso es relativo a su inicio.

Al programar va a llamar a dirección simbólica por ejemplo, en este caso f. entonces compilador manda a dirección lógica y luego a dirección física cuando esta se ejecuta.

La idea de traducción es la relación relativa o virtual, tal que la dirección viene del procesador en el ciclo fetch va emitiendo program counter, tal que son direcciones al espacio virtual del proceso, y debe ser traducido a una memoria física para que este con memoria.. todo es una referencia a una dirección virtual, y esto se hace el hardware, se traduce de lógico a físico. Se necesitaran tablas que me digan donde esta proceso en memoria.

Procesador a va a necesitar memoria, pordemos decir que esta dentro de procesador pero que necesita tablas.

Paginación

Si yo tengo a f, este va a estar dentro de una dirección, tal que va a comenzar la funcino virtual, tal que va a tener un call jump y esto lo hace el compilador, va a ser reemplazado por un numero.

Como lo voy a cargar a memoria

La paginación se divide la memoria física y la memoria virtual.

Anteriormente solo he particionado la memoria, ahora voy a particionar la memoria física y la memoria virtual. Entonces sea un proceso lo voy a separar en paginas, en un orden de 4 o 8 kb.

Entonces, cada pagina de memoria virtual, se carga de forma independiente a emoria física de las otras.

Entonces voy a tener la memoria física main, sea que tengo un proceso a que tiene un cierto espacio virtual de direcciones.

Este espacio de direcciones va a ser 0,1,2,3,4… n-1. Entonces voy a separarlo en en paginas de 4 u 8 kb. Entonces tendré pagina 1,2,3,4… estas paginas van a poder guardarse en cualquier marco de memoria física, marcos del mismo tamaño de las paginas.

Una pagina puede ir a cualquier parte que un marco este disponible, pero no necesariamente es siempre ordenado, puede ser randomizado o sin orden. Da lo mismo donde este. El acceso al marco es por hardware, la velocidad de acceso es constante. Lo mas importante de todo es que puedo cargar solo las paginas necesarias y no cargar otras que aun no necesite. Ahora puedo tener procesos grandes y ejecutarlos en memoria física pequeña, no tengo porque cargar todo el proceso a memoria.

Da lo mismo que no este continuo, tenga otro orden, etc. Ya que voy a tener una tabla que me indique las paginas donde estén para cada proceso.

Sea una tabla de paginas a , entonces voy a tener mis paginas 1,2,3,4 y los marcos en los que esta guardados será en este caso 1,2,3,4.

Para el siguiente proceso b sea paginas 1,2,3,4 y entonces mis marcos para cada una van a ser para el caso 4,5,11,12, tal que se me indica donde están en memoria física.

Algunos procesos pueden simplemente no estar en memoria física, así como también que estén parcialmente en memoria física.

Entonces como se mapea una dirección virtual a una dirección física? Usando paginación.

Haremos un proceso para simplificar, tal que un proceso que pesa 16 bytes.

Entonces tenemos un proceso, con 16 direcciones, cada uno es un byte. Cuantos bits necesito para direccionar 16 direcciones? En este caso 4 bits para escribir cada dirección, tal que 0000,0001,0010,0011…

Supongamos que las paginas son de 4 bytes. Por lo tanto tendremos la pagina 0,1,2,3.

Dibujaremos una memoria física que tiene 8 bytes, tal que tendría los marcos 0,1. Esto es muy utilizado.

Mi tabla de paginación tendría 4 entradas: 0,1,2,3.

Supongamos que pagina 2 se carga en marco 0, tal que se carga en cada byte z,b,y,x, entonces pongo en la tabla que pagina 2 se relacióna con marco 0.

Voy a dividir mi dirección completa en dos campos.

Serian 4 paginas y mi offset de bites dentro de la pagina son 4.

Entonces numero de paginas y desplazamiento dentro de la pagina.

Entonces sea que es mi comienzo de la pagina 2 es 1000, tal que 10 es el conteo de la pagina (pagina 2) y 00 es el desplazamiento dentro de la pagina (0) tal que estoy al inicio de mi pagina.

Si busco 10-01, entonces seria pagina 2, offset 1.

Quiero ir a mi dirección física, que me manda a 00-01, el offset se mantiene ya que es una pagina completa la cargada es decir offset 1. Obtengo el numero de marco que es 00, entonces seria mi marco 0.

Así encuentro mi bit desde virtual a físico.

1 kb= 2 elevado a 10, tal que necesito 10 bits (repasar measures)

8 gb es 2 elevado a 3 x 2 elevado a 30 bytes

Todo direccionamiento es al byte.

Rpasar direccionamiento y converciones.

Dividiremos distinto la paginación, en 2 bytes. Tal que 0,1,2,3,4,5,6,7. Mis direcciones no cambian, solo el particionamiento. Mi tabla necesitara mas entradas, tal que 0,1,2,3,4,5,6,7,al que también dividiré mis marcos de forma que 0,1,2,3.

Entonces mi b tendré en paginación será 100-1. Tal que 100 es 4 (pagina 4) saco el offset bits que es 1, por lo que me desplazo 1 bit. Copio el offset para el marco y busco donde lo guarde en el marco que es 11, tal que seria el tercer marco.

-------------------------------------

Paginas del proceso que se usan no son todas, no necesito tener todas las tablas de entradas de paginas en la pagina. Puedo paginar las paginas. La tabla de pagina se puede paginar en pedazos a su vez.

En terminos generales, la tabla puede ser de muchos niveles hasta que se llega al nivel superior de raiz.

Sea que tenga las paginas del proceso, tal que pertenecen a su vez a un nivel superior. Tal que son la tabla de pagina, puede ser que tenga multinivel hacia arriba.

Antes teniamos la direccion virtual, que se dividia en un numero de paginas y en un offset para un solo nivel. Cuando son dos niveles voy a tener unos bits para la tabla raiz, tal que tendria una tabla de segundo nivel. Tomo unos pocos bits para indexar mi tabla raiz, voy a sacar un nuevo marco donde voy a tabular la pagina que busco, tal que va a ser el offset de la tabla de pagina, tal que puedo accedera la memoria principal.

Un ejemplo concreto,

Otra forma de enfrentar el problema de una pagina muy grande, se puede usar una tabla invertida de paginas, tal que en vez de mapear de una pagina a un marco, mapea de marco a pagina.

Lo bueno es que es solo una tabla invertida para todo el sistema. Los procesos, todos comparten la mismamtala invertida de pagina, tal que se me dice en la invertida que cuales paginas es estan en que marcos. Todos los procesos tienen derecho a la pagina 4, pero no el marco, el marco es unico, por lo que es mas facil de mapear.

Sea que tengo almacenado ell numero de pagina (invertido) tal que voy a tener en la tabla numeros de marco, tal que mapeo numero de marco a numero de pagina.

Por ejemplo la pagina supongamos 200, en el marco 2, esta cargada la pagina 200... pero de quien? De que proceso? Ahí va el pid, que indica el proceso especifico de donde sacar la pagina, tambien obtendre informacion adicional, como tipo.

Entonces tendre el bit de offset y nro de pagina, tal que tendre la memoria paginada, entonces el numero de bits para el offset itenen que ver con el tamaño de la pagina. Indexa directamente donde quiero ir.

Si quiero ir a la pagina virtual 200, tal que se divide en la pagina 2.

Puede venir multiples procesos de paginacion 200

La tabla de pagina es siempre del mismo tamaño. El anchoes mucho mas ancho que la de una tabla de pagina.

Hardware que esta dentro de proesador, tiw. Almacena entrada de tabla de agina. Cuando un proceso comienza a correr, sst op va a saber cual le toca ahora.

es una meomoria

Hardware que esta dentro de proesador, tiw. Almacena entrada de tabla de agina. Cuando un proceso comienza a correr, sst op va a saber cual le toca ahora.

es una meomoria

Tiempo secuundario es desde page fault hasta que proceso vuelve a procesador y pagina ya esta cargada, puede ser de orden de milisegundos.

Una vez que se ha cargado la pagina, hace un miss, cuando hay un page fault, se actualiza

Todo esto es para general¿r la direccion fisica, luego de esta generación

Sea la direccion virtual.

Si no esta llena la memoria ed marcco, puede mapear pa ágina.

Una vez que g

----------------------------------------------

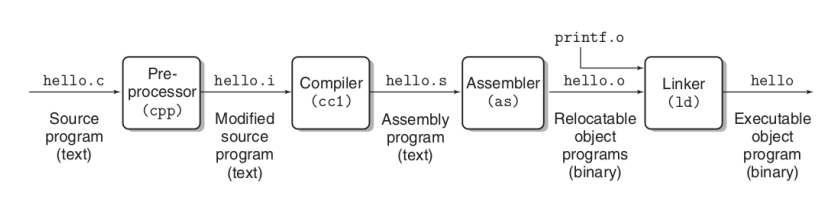
Resumen por

Programa: conjunto de instrucciones en un lenguaje en particular.

Proceso: un programa es compilado, linkeado y cargado en memoria. En ejecucion de una secuencia de instrucciones. Tiene estado actual y conjunto de recursos de sistema.

Sistema operativo: labor control de procesos

Secuencia de programa a proceso:



Preprocesador: expande includes, reemplaza macros y define. Resultado: .i

Compilador: codigo c a assembler.

Ensamblador: assembler a maquina. Resultado: “objeto reubicable”

Linkeador: resuelve referencias (modulos .o). Resultado: codigo ejecutable (no es proceso)

Formas de Crear procesos:

Usuario ejecuta comando

Invocacion al sistema desde proceso

Abrir app

So bootea procesos

Requerimientos para proceso:

Memoria procurada

Registro/estructura que tenga info sobre proceso

Bloque de control del proceso: estructura de datos que el kernel maneja sobre proceso. Tiene:

Pid (id de proceso)

Estado

Prioridad

Contexto

Espacio de direcciones de un proceso en memoria: (como se estructura proceso)

Tiene:

Codigo

Datos globales

Heap

Stack usuario

Stack kernel

\*cada proceso tiene su propio stack de usuario y propio stack de kernel

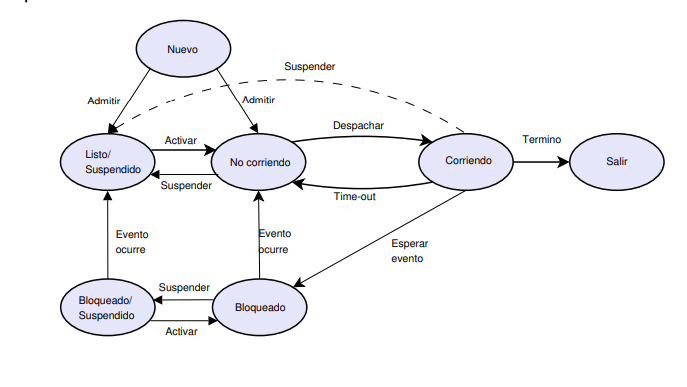
Boot-> kernel se instala en mem virtual del proceso->permite invocar servicios so

contexto de ejecuci´on: es toda la informaci´on relacionada con el estado actual de ejecuci´on para cada proceso. Es el estado actual del procesador, permite suspender y reanudar. Incluye registros.

contexto de hardware es una porci´on peque˜na del contexto de ejecuci´on y contiene los registros que se deben cargar antes de reanudar la ejecuci´on del proceso

Proceso: 3 componentes: descriptor (pcb), espacio de direcciones, contexto.

Modelo de estados de proceso:



Todo lo io es muy lento

Swapping (llevar procesos a disco de bloqueado a suspendido para liberar memoria) es op io.

Los procesos estan en cola de espera para entrar a procesador

Cambio de contexto: ocurre cada vez que el procesador comienza o reanuda la ejecuci´on de un proceso distinto (al actual).

Cuando ocurre cambio de contexto:

Trap: interrupci´on asociada a un error de la ejecuci´on de la instrucci´on actual

Interrupci´on: evento externo a la ejecuci´on del proceso

Interrupci´on del reloj por t´ermino de tiempo

Fallo de memoria. La direcci´on de memoria no se encuentra actualmente en memoria principal.

Llamado al sistema (system call): por ejemplo instrucci´on de I/O

Entalls:

Guardar contexto y registros

Actualizar pcb

Mover pcb a cola de listos

Seleccionar otro proceso

Actualizar pcb

Actualizar estructuras de admin de memoria

Restaurar el estado del procesador con el contexto del procesador

Tipos de sistemas operativos:

Contexto de proceso: al solicitar servicio de so no hay cambio de contexto.

Modo usuario-> modo kernel.

Basada en procesos: tareas de so son colección de procesos.

cada proceso so tiene su propio contexto

Especializacion: un procesador se ocupa de cierta tarea

Modos de ejecucion:

Modo usuario : no privilegiado, proceso no puede ejecutar instrucciones kernel

Modo kernel : el proceso puede ejecutar cualquier tipo de instrucciones y tiene el control absoluto del sistema

Procesadir checkea por interrupciones durante ciclo fetch

Si hay alguna interrupci´on pendiente, el procesador realiza lo siguiente:

Setea el PC con la direcci´on inicial del manejador de interrupciones

Cambia el modo del procesador de modo usuario a modo kernel, de manera que el manejador de interrupciones pueda ejecutar instrucciones privilegiadas.

Fork(): llamado a sistema para crear nuevo proceso

Crea hijo copia exacta de padre

En padre devuelve id de hijo

En hijo devuelve 0

Si es error –1

Getppid: obtiene id de su padre

Gitpid: obtiene el id propio

Medioambiente: elementos que permiten ejecucion de proceso. El como, con que, condiciones, limites.

Int main( int argc, char \*argv);

Argc: numero de argumentos

\*argv[]: argumentos listados

Exec():

Execve(nombre programa, argumentos, medioambiente, null que no deberia retornar): reescribe en proceso nuevo programa

--------------------------------------------------------

Condici´on de carrera Una condici´on de carrera (CC) es cuando el resultado de una operaci´on depende del orden de ejecuci´on de dos o m´as hebras

Secci´on cr´ıtica (SC) Una SC es un trozo de c´odigo ejecutado por m´ultiples hebras, en el cual se accede a datos compartidos y, al menos una de las hebras escribe sobre los datos

Exclusi´on mutua (EM) EM es un requerimiento sobre una SC, que dice que s´olo una hebra puede estar ejecutando dicha SC, se define por los datos que se acceden en ella, no por el c´odigo

enterCS() representa las acciones que la hebra debe realizar para poder entrar a su SC en forma segura

exitSC() representa las acciones que una hebra debe realizar para salir la SC y dejarla habilitada a otras hebras

Requerimientos para una soluci´on correcta de EM

1 Exclusi´on m´utua: Cuando Ti est´a en su SC, ninguna otra hebra Tj est´a en su correspondiente SC.

2 Sin deadlock: Deadlock es cuando todas las hebras quedan ejecutando enterCS() indefinidamente (ya sea bloqueadas o en busy-waiting), y por lo tanto, ninguna entra.

3 Sin inanici´on: Una hebra entra en inanici´on si nunca logra entrar a su SC.

4 Progreso: Si una hebra Ti ejecuta enterCS(), y ninguna otra Tj est´a en la SC, se debe perimitir a Ti entrar a la SC.

enterCS() y exitCS() pueden ser implementadas de varias formas:

1 Por hardware: Usar instrucciones nativas del procesador

2 Por software: Usar algoritmos e implementarlo en lenguaje de alto nivel

3 Por SO: Usar servicios del SO

4 Por estructuras nativas de un lenguaje de programaci´on

at´omica si (instrucción de asambler):

no puede dividirse en partes

se ejecuta completamente o no se ejecuta

se ejecuta sin ser interrumpida

Soluciones: \* desventajas: busy waiting innecesario, innanicion

Soluciones por hardware:

Deshabilitaci´on de interrupciones :

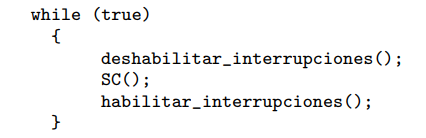
la hebra deshabilita las interrupciones justo antes de entrar en su SC

Habilidad del procesador para hacer context-switch queda limitada

No sirve para multiprocesadores

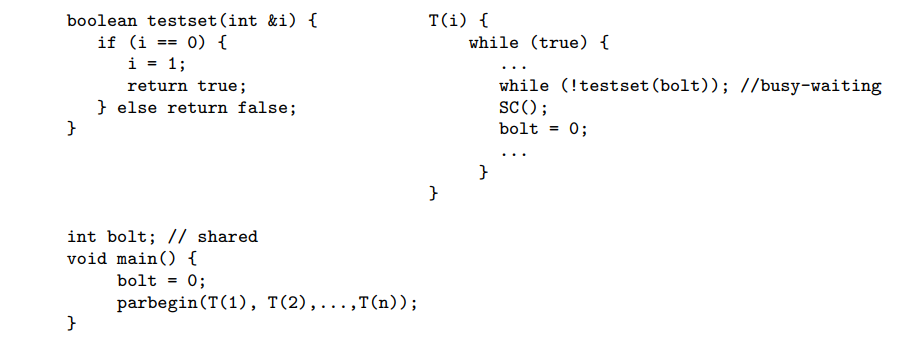
Funciona para cualquier n´umero de hebras

No est´a libre de inanici´on

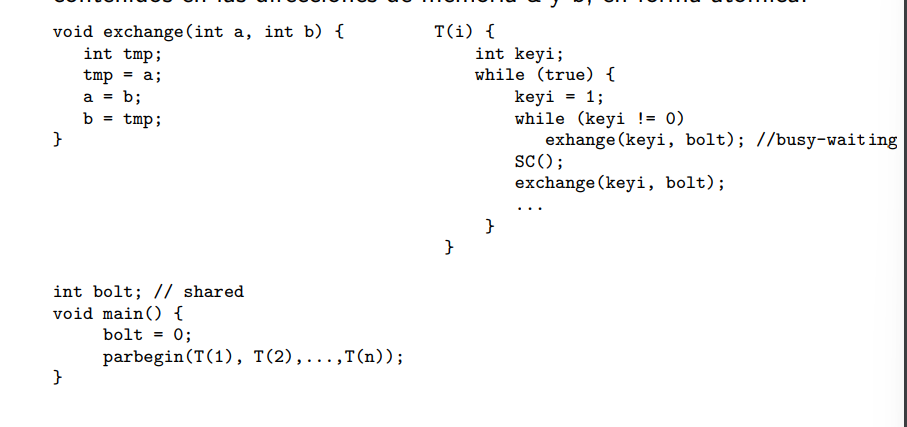


Instrucciones atomicas: instrucciones especiales de m´aquina que realizan en forma at´omica m´as de una operaci´on.

2.- Ej: testset.



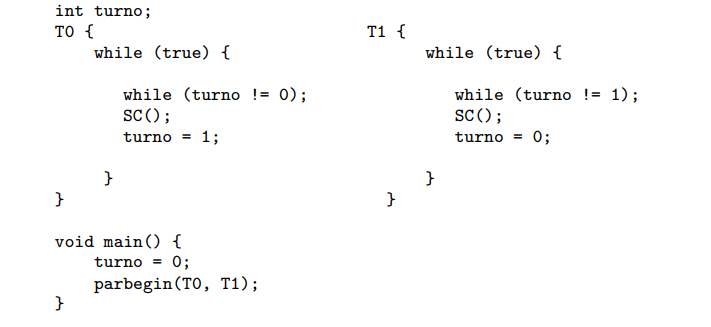
3.- ej: exchange(a,b), swap(a,b)



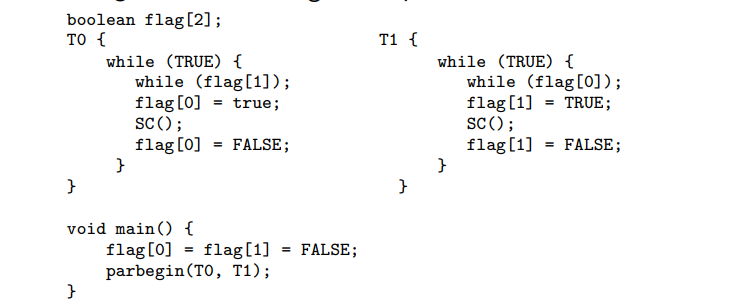
Solucion por software:

Hebras comparten turno:

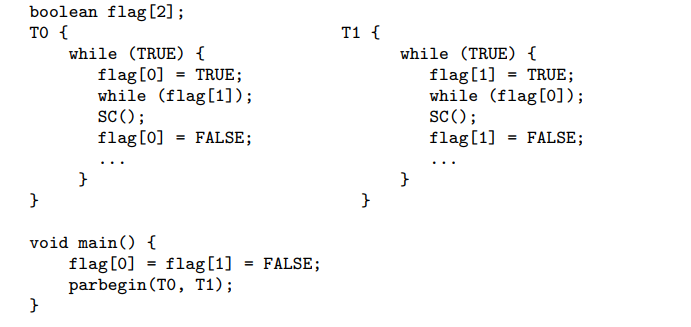
Exclusion mutua, 2 procesos estrictos, alternancia estricta, no progreso, pid de la tarea puede entrar a sc.



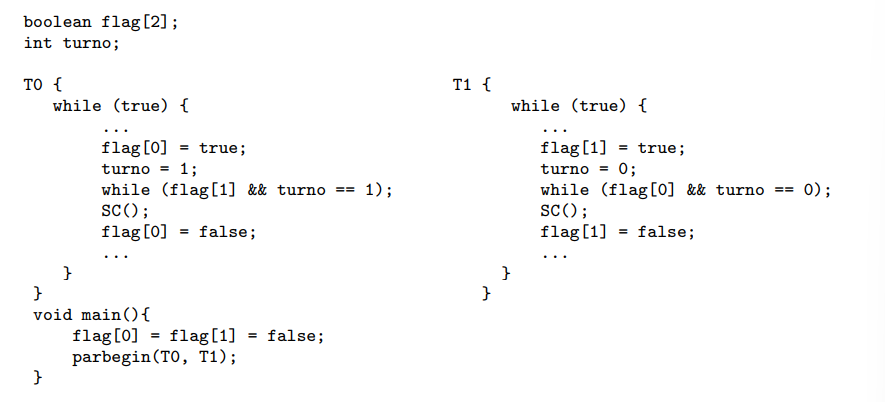
Mejorado2: se almacena el pid que esta en sc, no exclusion mutua



Mejorado3: exclusion mutua, probabilidad deadlock

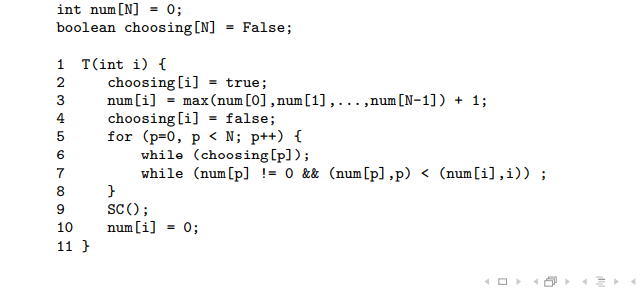


Peterson: satisface requerimientos, usa busywaiting, solo para 2 tareas

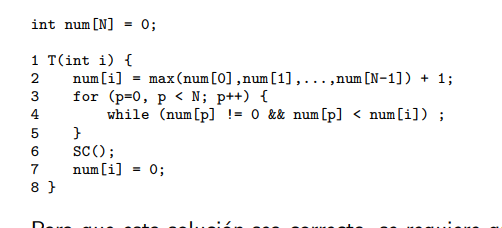


Algoritmmo panaderia: hebra ti tiene numi que es su ticket, solo con su ticket puede escribir, al entrar a sc se le checkea su numi, al ejecutarse numi se setea a 0.

Original



Simplificado (atomico)



Semaforo: a variable especial usada para que dos o m´as procesos se se˜nalicen m´utuamente

Operaciones:

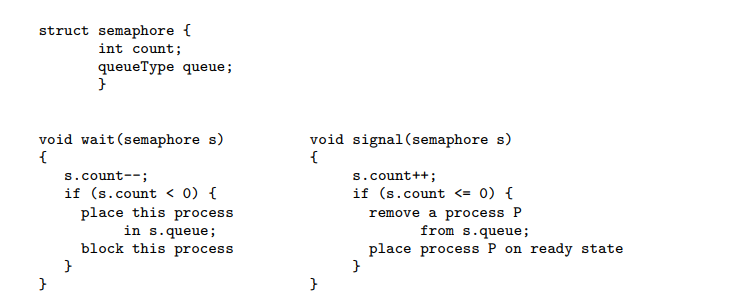
Wait(s): a variable especial usada para que dos o m´as procesos se se˜nalicen m´utuamente

Signal(s): incrementa el sem´aforo; si el valor resultante es menor o igual que cero, entonces se despierta un proceso que fue bloqueado

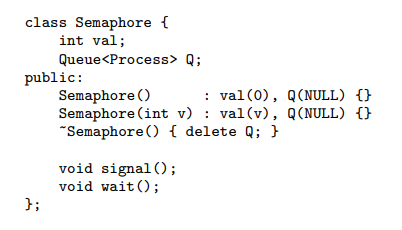
Tipos de semaforo:

sem´aforos contadores: a aquellos semaforos que pueden tomar valores mayor o menor que cero.

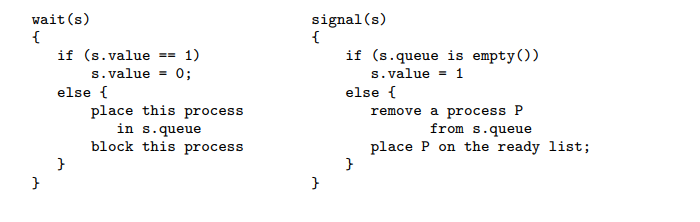
Pseudo:



Clase:

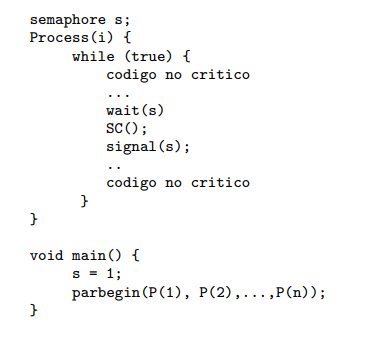


Semaforo binario/mutex/lock: valor 0 o 1



Wait=acquire=lock , release=unlock=signal

Exclusion mutua semaforo:



Productor- consumidor: en un buffer se almacenan datos compartidos

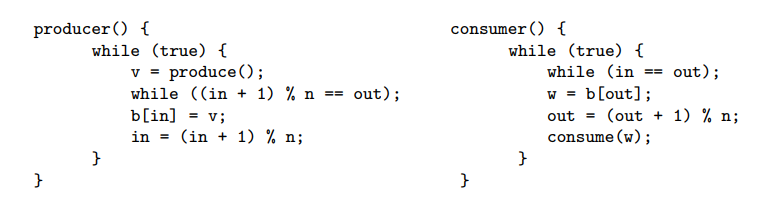
Productor: genera

Consumidor: consume 1 a la vez

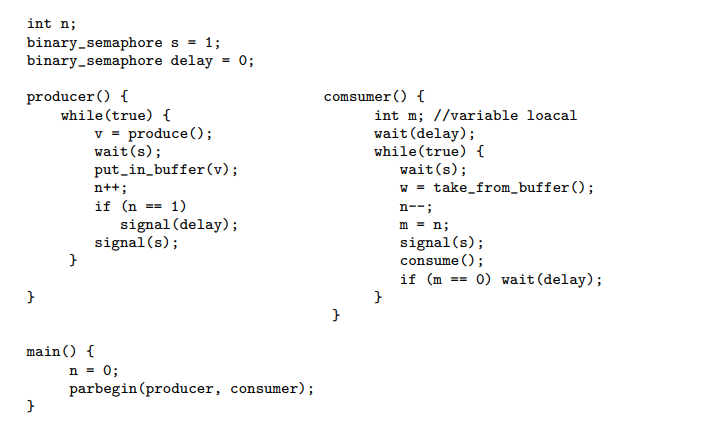
Infinito:



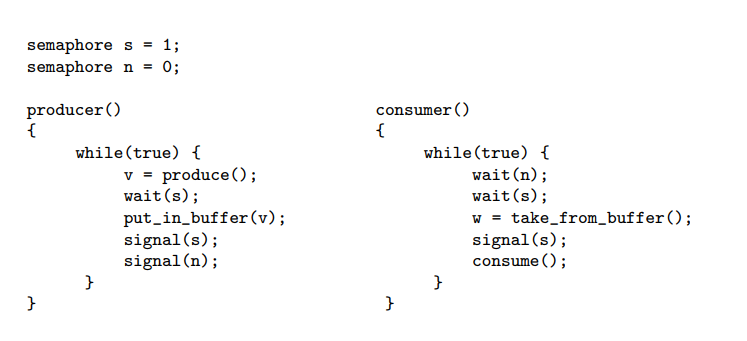
Finito: (array circular)



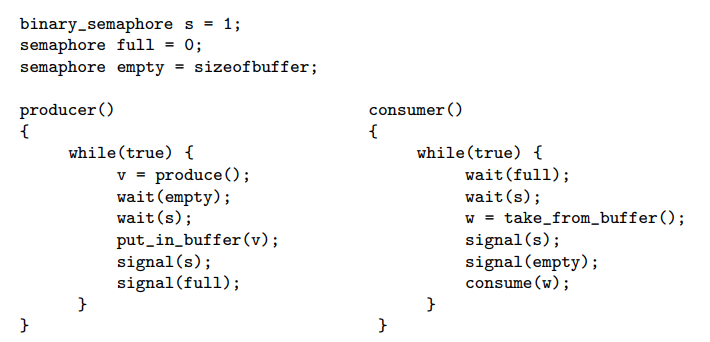
Implementado con semaforo binario:



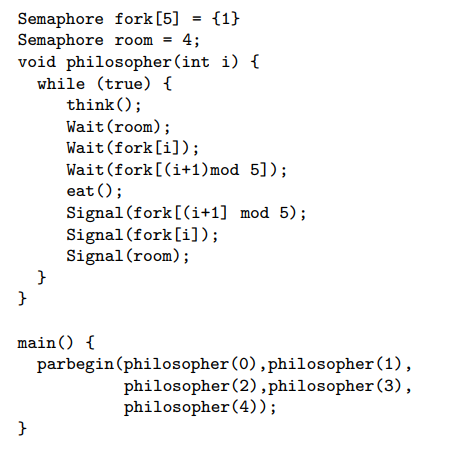
Productores y consumidores, buffer infinito, semaforo contador:



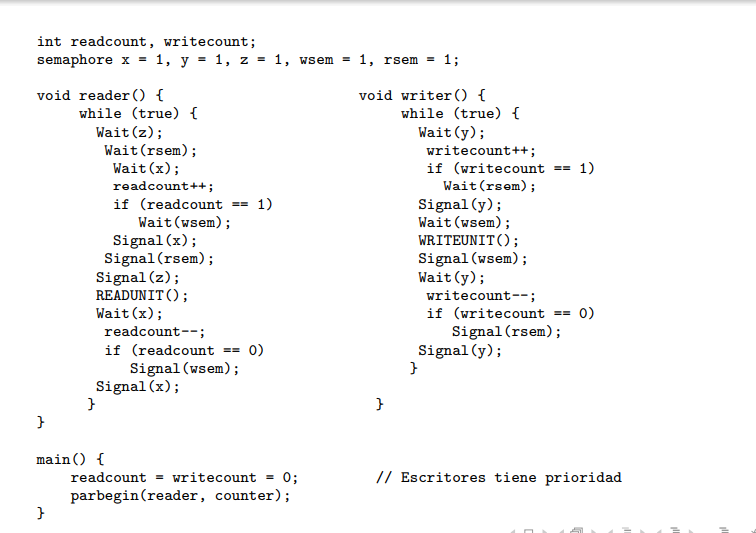
Productores/consumidores, buffer finito, semaforo contador:



Filosofos comensales:



Lectores/escritores:



Monitores:

Variables locales se acceden dentro de monitor por funciones monitor

Para ingresar a monitor proceso invoca funciones de monitor

Solo 1 proceso puede estar en monitor

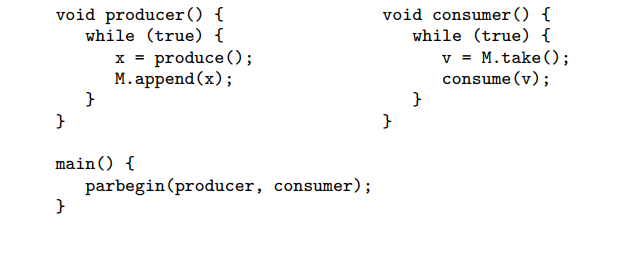
Variables de condicion:Proceso puede suspenderse dentro de monitor esperando condicion

Wait(c): bloquea proceso

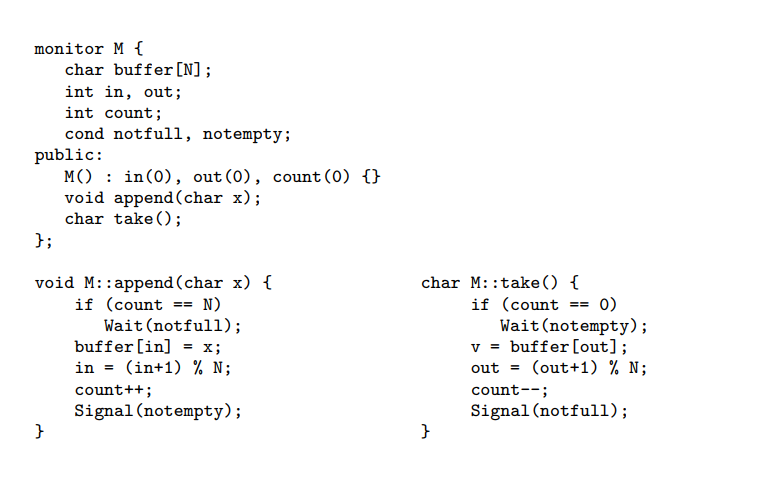
Signal(c): resume proceso

\* no tienen valor asociado

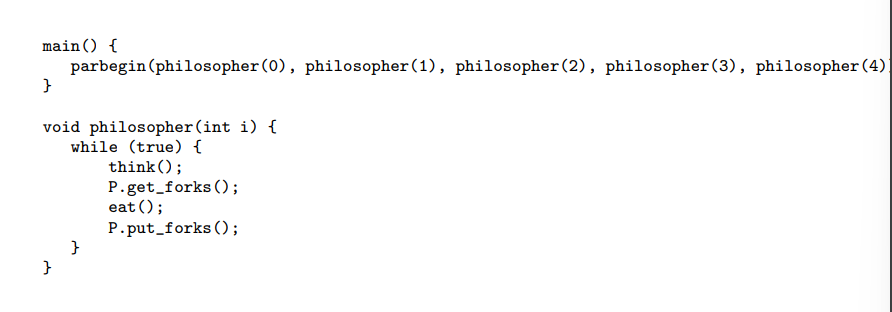
Productor consumidor de monitor



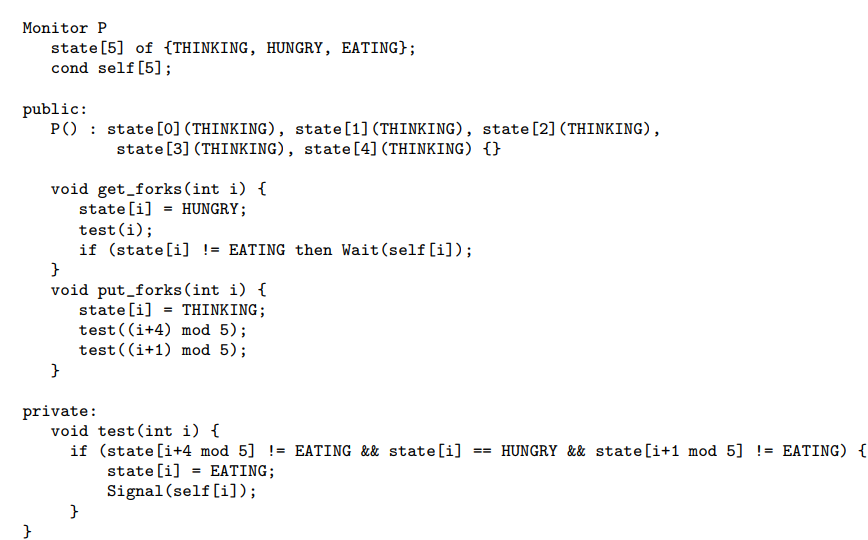
implementado



Filosofos con monitor



Filosofos con monitor



Paso de mensajes: comunicacion entre procesos

1 send(destination, message)

2 receive(source, message)

Formas:

Blocking send, blocking receive Tanto el enviador como el receptor se bloquean hasta que el mensaje llegue al receptor

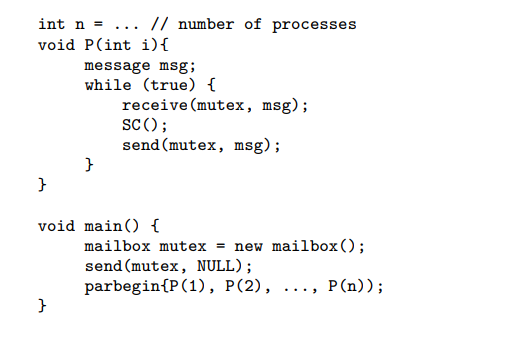
Nonblocking send, blocking receive El enviador contin´ua su ejecuci´on despu´es de enviarla El receptor se bloquea hasta que llegue

Nonblocking send, nonblocking receive Ninguno se bloquea

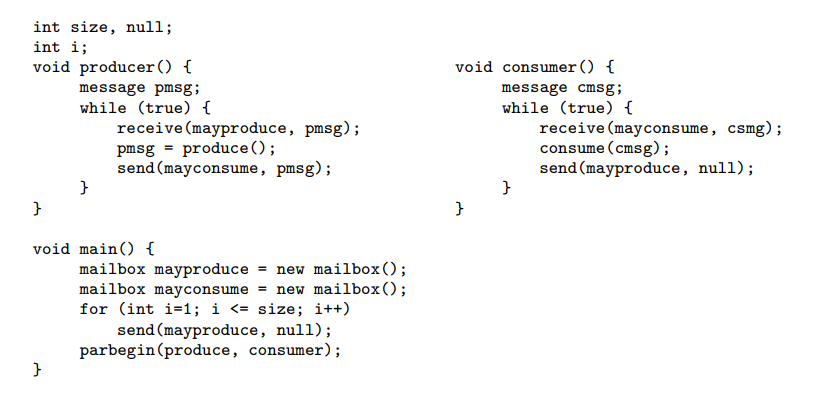
Direccionamiento directo: send incluye destino, reicive solo recibe informacion (no importa de donde)

Direccionamiento indirecto: mensajes se guardan en una estructura compartida mailbox

Recibir y enviar mensaje



Productor consumidor con mailbox

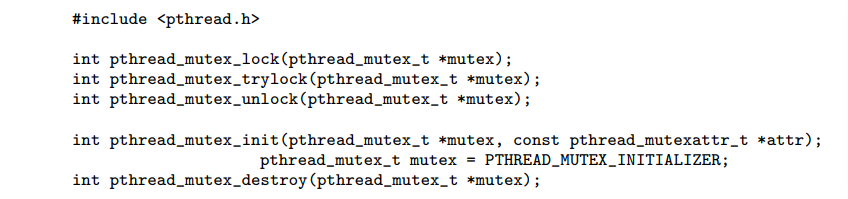


Hebras posix (funciones):

Mutex: semaforo binario, exclusion mutua a n hebras, acceso exclusivo a sc

Variables de condicion: asociado a mutex, condiciones de bloqueo y desbloqueo

\*solo para hebras de mismo proceso



Funcinoa como una llave. Un proceso lock toma la llave para entrar al proceso, y si otro proceso quiere entrar se bloquea ya que la llave o lock esta tomada. Luego, en unlock se despierta un proceso y toma la llave.

Barreras: puntos de sincronizacion donde se bloquea un grupo de hebras

Inicializar semaforos:

Con nombre: procesos no relacionados

Sem\_t \*sem\_open(const char \*name, int oflag(para crearlo), mode\_t mode(modo), value(valor de semaforo)

Sin nombre: mismos procesos y relacionados

Int sem\_init(sem \*sem (puntero a semaforo), int pshared, int value (valor de semaforo));

Int sem\_post(sem\_t \*sem) incrementa

Int sem\_wait(sem\_t \*sem) decrementa