FTF

Tema 4: Software Tolerante a Fallos

1.- Introducción

2.- Prevención de fallos software

3.- Software Tolerante a Fallos

- 3.1 Pruebas de consistencia o robustez
- 3.2 Redundancia temporal
- 3.3 Manejo de excepciones

4.- Estrategias software de recuperación del sistema

- 4.1 Recuperación hacia atrás utilizando puntos de recuperación
- 4.2 Recuperación hacia delante: N-Versiones de programas

5.- Tolerancia a fallos en el Sistema Operativo

FTF

1. Introducción

- Una de las principales causas que deterioran la garantía de funcionamiento de los sistemas informáticos es la falta de fiabilidad en el software.
- Clasificación de los fallos con respecto al momento de producción
  - Hw: En la vida operacional del sistema
  - Sw en la especificación, el diseño y en la implementación de los programas
    - Es más complejo (muchos estados, situaciones no consideradas, modelos teóricos incorrectos, implementaciones defectuosas)
    - No existe una forma de validación exacta.
    - Si el error no se descubre en la validación, puede descubrirlo el usuario.
    - En la corrección se pueden generar nuevos errores.
- Aunque el software no se degrada con el tiempo, **nunca es perfecto**, ya que varían:
  - Datos de entrada
  - Entorno de ejecución
  - Requerimientos del usuario
- Por tanto, el funcionamiento correcto pasado y actual no garantiza que no se vayan a producir fallos en el futuro. Se puede concluir que el número de fallos producidos en el sw está en relación con el tiempo de ejecución del mismo.
- Las consecuencias de los fallos sw pueden ser:
  - leves: Pérdida de un avión por un error en una reserva de billete
  - serias: 20M de personas sin teléfono en el verano del 91
  - graves: Sistemas de control en tiempo real (pérdidas económicas o de vidas humanas)

FTF

- Diferentes alternativas del software tolerante a fallos

  - Tratamiento de los fallos producidos en el propio diseño o en la implementación de los programas y aplicaciones (test inicial)
    - Se utiliza la robustez, la redundancia temporal o la diversidad en el software para detectar y recuperar los errores que tienen su origen en el software.

  - Tratamiento de las averías de los elementos hardware (detección concurrente)
    - Es el software que implementa los mecanismos de detección de errores hardware y se encarga de la recuperación y reconfiguración del sistema.
    - Utiliza la redundancia hardware para solventar el error.

  - Detección y recuperación de los errores en los sistemas distribuidos
    - Detecta los fallos de temporización en los sistemas distribuidos.
    - Es una mezcla de los dos anteriores:
      Protocolos para la comunicación entre procesos replicados
      Sincronización entre procesos
      Manejo de la consistencia de los datos replicados
      Manejo de la recuperación por vuelta atrás

FTF

2. Prevención de fallos software

- Desde los 60's ya existían técnicas de ingeniería del sw para desarrollar sw libre de errores.
- Actualmente, a pesar de las herramientas CASE y de los procedimientos de comprobación, los sistemas fallan porque no se pueden escribir las especificaciones de una manera formal.
  - El diseño, la codificación y la prueba se hace de acuerdo con el estilo, experiencia y juicio del programador
- Los costes derivados del sw son en un 60 % derivados de los procemientos de mantenimiento, debido a la larga vida del mismo y a las actualizaciones
- La complejidad del sw se combate utilizando técnicas de:
  - Modularidad: Se realizan interfaces precisas para poder comprobar sus entradas
  - Programación orientada a objetos: Se encapsulan los datos y las funciones que los pueden utilizar
  - Programación basada en capacidades: Se restringe el uso de los datos y de las funciones encapsuladas en un objeto en función del módulo que las utiliza
  - Métodos formales o prueba de correctitud:
    Estos mecanismos eliminan errores de diseño
    Intentan demostrar la correctitud del código reestructurándolo como si fuera un problema matemático.
    Ejemplo: Sistemas síncronos

3. Software tolerante a fallos

FTF

3.1 Pruebas de consistencia o robustez

- Implementan mecanismos software de detección de errores en el hardware
- Se utiliza el conocimiento que se tiene a priori sobre las características de la información para verificar la correctitud de dicha información.
- El OBJETIVO es detectar los fallos cuanto antes
- Ejemplos:
  - Detección de códigos de operación inválidos
  - Detección de la caída en las prestaciones del sistema por debajo de un determinado nivel
  - Construcción de programas robustos: El sw opera a pesar de la introducción de entradas inválidas (fuera de rango o con formato diferente). Se comprueban:
    Los datos de entrada: Mediante ECC o comprobaciones de tipo
    Las secuencias de control: Limitando el número de iteraciones de un bucle
    Las salidas.
    Si los datos son incorrectos: Se repite la entrada, se utiliza el valor anterior o se utiliza un valor predefinido

3.2 Redundancia temporal

- Se realizan reintentos hasta llegar al estado de excepción para evitar la propagación de los errores

FTF

3.3 Manejo de excepciones

- El manejo de excepciones supone la interrupción de la ejecución normal del programa, debida a la detección de una condición anómala prefijada de antemano, y el salto a una sección de código predeterminada (manejador de la excepción).
- Se utilizan para implementar la robustez y la redundancia temporal.
- Ada implementa el manejo de excepciones como parte de su definición.
- Aspectos de diseño
  - Hay que identificar los niveles apropiados para manejar la excepción dentro del sw (nivel de programa, de tarea, de módulo)
  - Se necesita conocer y clasificar las posibles causas de la excepción para cada uno de los niveles software definidos (entradas inválidas, fallo del dispositivo, fichero no encontrado, fallo en la temporización, sobrepasamiento aritmético ...)
  - Detectar su manifestación dentro del sistema que se ejecuta
  - Conocer la forma de mitigar sus efectos para cada uno de los niveles definidos: enmascarar el fallo, realizar una vuelta atrás o detener el nodo
- Aspectos de implementación
  - Se deben identificar las excepciones que estarán activas dentro de cada nivel
  - Hay que cuidar de que cada excepción tenga asignado un manejador apropiado

FTF

- Clasificación:
  - *Excepciones de la interfaz*: Cuando se presenta una petición a un componente o servidor que no está conforme con la interfaz implementada (instrucciones ilegales, sobrepasamiento aritmético, violación de protección).

    Estas excepciones no siempre indican un error, depende del tratamiento que se le quiera dar a la excepción.
  - *Excepciones de fallos*: Se activan cuando un componente detecta que no puede suministrar el servicio especificado. El manejador asociado debe por tanto implementar las medidas apropiadas para mitigar ese tipo de fallo.

- Manejo de excepciones software

  Debe existir un método por el cual un programa asocia una excepción a un manejador

  Se denomina el contexto de manejadores al conjunto actual de manejadores habilitados por un programa en un instante de tiempo

- Propagación de excepciones

  Todas las excepciones que no se traten en el módulo no se deben propagar a los demás, sino que se deben convertir en una ex. de fallo.

```
Begin F
Recibir petición de servicio (leerbloque, i) desde P
BEGIN-CONTEXTO-EXC
    if (i = No. Bloque Ilegal) entonces signal EXC INTERF
    n:= LocalizaEnDiscoBloque(i)
    Respuesta := LeeDeDisco(n)
    si (Error en Respuesta) entonces raise BLK CORRP

MANEJADORES-DE-EXCEPCIÓN
    SI (BLK CORRP) Respuesta := LeeDiscoBackup(n)
    SI (DIVISION-POR-CERO) signal EXC FALLO
END-CONTEXTO-EXC
Retorna(Respuesta) a P
End F
```


FTF

4. Estrategias sw de recuperación del sistema

- Una vez detectado el error (por mecanismos hardware, excepciones, tests, temporizadores, sumas de prueba o redundancia) se deben solucionar los efectos.
- Debido a que la mayoría de los fallos son transitorios, simplemente con volver a ejecutar la computación se puede recuperar el sistema.
  - Se debe utilizar un mecanismo que penalice lo más mínimo al sistema
  - La recuperación es compleja cuando los procesos se comunican entre si
- Se utiliza diversidad en el software para implementar rutinas de recuperación de las funciones críticas para tener un servicio continuado.
- Tipos de redundancia:
  - **Espacial** (estática) :La selección de un resultado aceptable no afecta a subsiguientes ejecuciones
    - N-Versiones de programas:
    - Programación N Autocomprobante
  - **Temporal** (dinámica): La selección del programa original o de otro alternativo se hace en la ejecución basándose en el test de aceptación
    - Bloques de recuperación

FTF

4.1 Recuperación hacia atrás utilizando puntos de recuperación

- En el momento en que se detecta el error, se solucionan sus efectos mediante la reejecución, de forma que se penalice lo menos posible a las prestaciones.
- Las técnicas de recuperación son diferentes para los sistemas distribuidos que para los sistemas con memoria compartida.
- Multiprocesadores:
  - Reinicialización global: Todos los procesadores reejecutan sus tareas desde el principio
  - Continuar la ejecución desde la detección del error y utilizar otros mecanismos para asegurar la ejecución correcta. (*forward recovery*)
  - Vuelta atrás a un estado anterior

    Resulta difícil en los multiprocesadores ya que se pueden tener copias diferentes y erróneas de las mismas variables

    Se debe utilizar un sistema de almacenamiento estable para los puntos de recuperación, que contendrán los registros del procesador y los bloques de cache y las páginas modificadas.

FTF

Puntos de recuperación basados en caches

- Se definen:
  - Datos del pdr: El estado del último punto de recuperación
  - Datos activos: Estado de los datos accedidos después del pdr
- La generación de un nuevo pdr se simplifica teniendo los datos activos únicamente en la cache. De esta manera para **establecer un pdr** basta con copiar los registros del procesador y los bloques modificados de la cache a memoria. (Esta escritura se realiza de forma atómica)
- Los pdr se generan cada vez que existe una escritura en memoria. Se hace necesario usar una cache con escritura retardada (*write-back*), para que la vuelta atrás no tenga que deshacer las escrituras en memoria realizadas desde el último pdr. En esta **vuelta atrás** se elimina el estado activo:
  - Se cargan los registros
  - Se invalidan las líneas modificadas de la cache
- Se debe llegar a un compromiso en el tamaño de la cache, ya que cuanto mayor sea, menor será la frecuencia de establecimiento de los pdr, pero se tardará mucho más tiempo al haber muchas más líneas modificadas.

Puntos de recuperación virtuales

FTF

- IDEA: Si la frecuencia de establecimiento de los pdr es alta, se puede incluir el estado de la memoria en el pdr y utilizar el disco para almacenarlo.
- PROBLEMA: El tamaño del pdr es elevado por lo que se tendrán que utilizar pdr incrementales.
- Esquema de funcionamiento:
  - Cada página virtual se duplica en dos páginas físicas cuando ésta se modifica
  - En cada página virtual se tiene:
    - Un contador v que sirve para identificar si la página se ha modificado después del último pdr (página activa)
    - Un bit l que indica cuál es la copia activa
    - Un bit s de sucio
  - Un contador V que se incrementa cada vez que se establece un pdr
- Establecimiento del pdr
  - Se incrementa V
  - A medida que se van referenciando las páginas, si v < V se copia la página activa en la de repuesto y ésta pasa a ser la activa. Si v = V no se hace nada
- Vuelta atrás (eliminación de las páginas activas)
  - En todas las páginas con v = V se decrementa v y se activa la copia de repuesto

Recuperación en procesadores comunicados

FTF

- La vuelta atrás debe considerar las comunicaciones y sus efectos desde el último pdr
- Multiprocesadores con memoria compartida
  - Es difícil ya que pueden existir dependencias de datos entre los procesos que se ejecutan en diferentes procesadores. Cuando se hace la vuelta atrás de un procesador es necesario realizarla también en otro y así sucesivamente (propagación de la vuelta atrás).
  - SOLUCIÓN: Evitar las dependencias mediante un pdr global

1) Los procesadores activan una línea de 'establecer pdr' para que todos almacenen su pdr.
   La frecuencia de establecimiento de los pdr depende del número de procesadores
   Mejora: No establecer el pdr si no hay interacciones (fallo en lectura en bloque modificado, acierto en escritura en bloque compartido, fallo en escritura en bloque compartido).

2) Establecer un pdr cuando se vuelca un bloque sucio de la cache en memoria y cuando un procesador lee un bloque modificado después del último pdr (para evitar la propagación)

Recuperación en sistemas distribuidos

FTF

- Los procesadores se comunican mediante paso de mensajes y además se supondrá que cuando un procesador tiene una avería, los demás quedan enterados en un tiempo finito
- En la recuperación por vuelta atrás se deben de evitar las inconsistencias:
  - Un procesador sabe que ha enviado un mensaje a otro procesador que no tiene constancia de ello ya que ha realizado una vuelta atrás.
  - Un procesador que envía un mensaje y a continuación falla, no tiene constancia de que lo ha enviado ya que ha realizado una vuelta atrás. En contraposición el otro procesador sí que tiene registrado que ha recibido el mensaje.
- Se debe evitar producir el efecto dominó: Para realizar una recuperación hacia atrás basada en pdr's, debido a las dependencias, si un procesador realiza una vuelta atrás, todos los que han interaccionado con él también deben de realizarla. Se forma una cadena en donde los procesadores tienen que actualizar sucesivamente sus pdr llegando en el caso extremo al primer pdr y por tanto al comienzo del programa.
- Se define *línea de recuperación* al conjunto de pdr que se recuperan para llegar a un estado consistente en el sistema.
- Técnicas:
  - **Pesimistas**: En cada comunicación se establece un pdr (malgasta tiempo y memoria)
  - **Optimistas**: Cada procesador establece su pdr de forma independiente. Además se apuntan las dependencias en los mensajes para que exista un registro de los pasos a realizar si se quiere volver a un estado consistente (*logging techniques*)

Bloques de recuperación

FTF

- Es la aproximación sw a la redundancia dinámica. Se basa en la existencia de puntos de recuperación y de comprobación
  - Ventaja: No necesito utilizar hardware duplicado
- En cada momento se ejecuta únicamente una función (la copia activa) y existen una o varias de repuesto. Compuesto por:
  - Un bloque primario que ejecuta sw crítico
  - Un test de aceptación que comprueba la salida de la rutina después de cada ejecución
  - Un punto de recuperación
  - Uno o varios bloques alternativos que realizan la misma función que el primario y que los invoca el test de aceptación al detectar el fallo
- El test de aceptación puede ser:
  - De temporización para las rutinas en tiempo real
  - Del formato de las llamadas y de sus parámetros
  - De la validez de los datos de entrada (se puede prevenir este error y disponer de una fuente de datos alternativa)
  - La validez de los datos de salida
- Técnica de Recuperación directa: Elimino los fallos que puedan ocurrir
  - Se realiza un test para detectar errores específicos
  - Como se saben las consecuencias se recupera el sistema
  - Se utilizan excepciones (división por cero)

FTF

- Técnicas de recuperación por vuelta atrás: Elimina errores imprevisibles o de naturaleza desconocida
  - Se utilizan puntos de recuperación para almacenar el estado del sistema y del proceso
  - Si se llega a un punto en donde el proceso está mal, se vuelve atrás y se ejecuta otra versión del programa
  - Las versiones se pueden ejecutar en máquinas diferentes

```
BR: ensure AT
        by B1
        else by B2
        ....
        else by Bn
        else fail
```

- Se guarda el punto de recuperación
- Se ejecuta B1
- Se pasa el test de aceptación TA
- Si falla se recupera el PR
- Se ejecuta B2
- etc ...

- El impacto en las prestaciones es pequeño
  - –Si la cobertura del test es perfecta y los fallos son independientes detectan N-1 fallos
  - –No necesita redundancia en el hw
  - –Si todo va bien el impacto es mínimo (sólo la ejecución del test de aceptación)
  - –Se necesita espacio de almacenamiento para el test y para las nuevas versiones
  - –Es lo mismo que *cold standby sparing*

Bloques de recuperación distribuidos

FTF

- Tratan de integrar la tolerancia a fallos hw y sw en una única estructura
- Consiste en la utilización de una rutina primaria y varias alternativas que están duplicadas y que residen en varios nodos interconectados a través de una red. Además también existe un test de aceptación
- Funcionamiento:
  - En condiciones normales, el nodo primario ejecuta el bloque primario y el de backup ejecuta el bloque alternativo de forma concurrente.
  - Si ambos aprueban el test de aceptación, el primario notificará el éxito al secundario y los dos actualizarán sus copias de datos locales. La salida del conjunto se corresponderá con la del nodo primario.
  - En caso de fallo en el nodo primario, éste lo notificará al secundario que pasará a ser el primario. Como la rutina ya se ha ejecutado, los resultados estarán disponibles inmediatamente.
  - En caso de fallo total del primario, la expiración de un temporizador en el nodo secundario lo activará. (mezcla de algoritmos y mecanismos tanto hw como sw)
- Mejoras:
  - Se puede dar vuelta atrás y volver a ejecutar el test con el bloque alternativo, para que en el caso de terminación con éxito, hacer de nodo de backup y poder recuperar un segundo fallo.
  - Ejecutar versiones alternas en los nodos para evitar fallos de diseño

4.2 Recuperación hacia delante: N-Versiones de programas

FTF

- Una tarea es ejecutada en varias versiones (diseñadas y codificadas por personas diferentes a partir de las mismas especificaciones), y el resultado se acepta si un número determinado de programas están de acuerdo en función de unos límites especificados.
- Se ejecutan varias réplicas al mismo tiempo (puede ser en distintas máquinas) con las mismas entradas y las mismas condiciones iniciales.
- Un algoritmo de voto (eficiente y efectivo) evalúa las salidas de las versiones y determina la decisión a tomar en caso de fallo
  - Reejecución
  - Pasar a un estado seguro y después reintentar la operación
  - Confiar en una versión especial más fiable o que ejecuta un programa de diagnosis que indica que está libre de errores.
- Existe un programa especial (driver) encargado del voto, de la sincronización y de la comunicación de los resultados a otros programas
  - Coordina la ejecución concurrente de las N versiones
  - Implementa el protocolo de acuerdo entre los resultados (voto o comparación)
  - Necesita
    C-Vectors: Vectores donde se almacenan las variables a comparar y los flags de estado
    CS-Indicators: Acciones que se realizan en los puntos de comparación
- Ventajas
  - Este método detecta fallos sw (los procesos replicados sólo fallos hw)
  - No precisa diagnosis
  - Puede enmascarar fallos

FTF

- Inconvenientes
  - Todos parten de las mismas especificaciones
  - Al realizar la votación es difícil saber si los resultados son correctos o no
    Hay que diferenciar entre errores de precisión o de inexactitud en los algoritmos
    Votación adaptativa: Se asignan pesos a los resultados
    Votación no adaptativa: Hay un rango de discrepancia
  - Se pierde tiempo en la sincronización y en la votación
- Especificaciones
  - Función a implementar
  - Formatos de los datos
  - Establecimiento de los puntos de comparación (*CC - Points*)
    Si son poco frecuentes se minimiza la influencia en las prestaciones y permitirá gran independencia en el diseño. Sin embargo, la votación no será fácil porqué habrán grandes discrepancias en los valores y se requerirán grandes esperas para la sincronización
    Si se realizan muy frecuentemente, los programas necesitarán utilizar estructuras similares, reduciendo el grado de independencia y se producirá una sobrecarga que influirá en las prestaciones
  - Algoritmo de comparación
  - Decisiones a tomar en función de los resultados de la comparación

FTF

- **Programación N Autocomprobante**
  - Se escriben N versiones diferentes y cada una tiene un conjunto de test de aceptación. Estos tests se basan en consistencias o capacidades.
  - La lógica de selección elige los resultados de cualquiera de los programas que pasan el test de aceptación.
  - Es lo mismo que *hot standby sparing*
  - Reconfiguración rápida
  - Si la cobertura del test es perfecta y los fallos son independientes detectan N-1 fallos
- **Procesos replicados**
  - Ejecución de procesos idénticos sobre diferentes procesadores
  - No se ejecutan todos a la vez (la copia está de repuesto y sólo se usa cuando falla)

FTF

N-versiones versus Bloques de Recuperación

- En ambos, la tolerancia a fallos dependerá de la habilidad de los programadores en suministrar múltiples versiones funcionalmente equivalentes pero que no contengan los mismos fallos de diseño
  - N-Versiones: Cada versión puede definir sus propias estructuras de datos privadas que se mantienen durante sucesivas ejecuciones. Ello permite utilizar diferentes algoritmos para su manipulación y por tanto la implementación de funciones diferentes se simplifica, aunque sigue dependiendo de la habilidad del programador
  - B. de Recuperación: La diversidad se logra de una forma más explícita
- Si la sobrecarga de ejecutar múltiples versiones no importa, es mucho más eficiente la detección de errores por comparación que la del test de aceptación, sin embargo:
  - La inexactitud en la votación es un problema
  - No se puede votar si existen diferentes soluciones a un mismo problema
  - La degradación funcional es difícil a la hora de votar
- Las estrategias de tolerancia a fallos de las N-Versiones son simples (estimación de daños, recuperación y tratamiento del fallo). Sin embargo:
  - Se necesitan muchos recursos hardware para ejecutarlas en paralelo
  - Siempre habrá una versión más lenta que retrasará las demás
  - En los b. de recuperación, siempre decide el tiempo de ejecución el bloque primario

FTF

- La recuperación hacia atrás es cara cuando existen procesos que interactúan entre sí o no se puede realizar si los objetos no son recuperables (ejm: Lanzamiento de un cohete).
- Las N-Versiones son difíciles de implementar teniendo en cuenta que la versión no puede afectar al sistema (sólo influirá en la votación). Sin embargo el bloque de recuperación no impone ninguna restricción en a ejecución, aunque sí que debe implementar los mecanismos para la recuperación hacia atrás.
- Como las diferentes versiones pueden almacenar registros de los resultados de ejecuciones anteriores, existe el peligro de que los pequeños errores se vayan acumulando y deriven en un resultado incorrecto. En este caso será necesaria la implementación de algún medio de recuperación para no perder garantía de funcionamiento.

  En los B. de recuperación los errores de los módulos se recuperan y no afectan a futuras ejecuciones

FTF

5 - Tolerancia a fallos en el Sistema Operativo

- El S. Operativo debe implementarse de forma que suministre una tolerancia a fallos transparente al usuario
- En las aplicaciones de gestión es importante mantener
  - La integridad de los datos
  - La integridad de la estructura de la Base de Datos
- Se utilizan transacciones atómicas
- Comprobación de las capacidades
  - Se realizan para verificar que el sistema posee la capacidad de cómputo esperada
  - El S.O. realiza tests de memoria o de otros recursos (ALU, enlaces de comunicación)
  - El S.O. asigna una serie de parámetros a los procesos
    Recursos a los que puede acceder
    Operaciones a realizar
    Procesos con los que puede comunicarse
  - Estos parámetros se almacenan en tablas que el S.O. consulta para detectar errores