Processus concurrents et parallélisme

Thème 9 - Fiabilité

Gabriel Girard

18 octobre 2022

Thème 9 - Fiabilité

Thème 9 - Fiabilité

Fiabilité

- Fiabilité : degré avec lequel un système logiciel (dans notre cas) rencontre ses spécifications du point de vue du service à l'usager, même en présence de conditions non prévues et hostiles
- Fiabilité est un concept relatif qui dépend des besoins

Correct ou valide

- Correct : un système est correct (valide) lorsque les résultats produits sont ceux désirés en fonction des données (valides) en entrée
- On montre qu'un programme est correct par tests ou preuves
- C'est une condition insuffisante à la fiabilité

Erreur

- Erreur : toute déviation du système de son comportement spécifié
- C'est une événement détectable

Faute

- Faute : c'est la cause d'une erreur
- Une faute est un état
- Une faute peut ne pas causer d'erreur

Dommage

- Dommage : corruption causée par une erreur
- Une façon de rendre un système fiable est de limiter les dommages

Approches pour obtenir un système fiable

- Éviter les fautes
- Détecter les erreurs
- Traiter les erreurs
- Effectuer la reprise

Thème 9 - Fiabilité

Éviter les fautes matérielles

- Éviter les fautes matérielles avec des composants plus fiables
  - Mémoire ECC
  - Répétition d'opérations
  - Checksum
  - Codes correcteurs
  - Majority polling
  - Etc.

Éviter les fautes logicielles

1. Approches de gestion, de conception et d'implantation
   - méthodes de gestion, méthodes de programmation, outils logiciels, méthodes formelles
2. La preuve de programme (ou pseudo-preuve)
   - model checking (Spim, Vérisoft, ...), etc
3. L'application systématiques de tests
   - méthodes boîtes noires, méthodes boîtes blanches, tests unitaires, tests d'intégration, validation, vérification, ...

Éviter les fautes des usagers

? ? ?

Thème 9 - Fiabilité

Détecter les erreurs

- Détection par l'environnement matériel
  détection et masquage
- Détection par l'application (système d'exploitation)
  - il existe différentes techniques (redondance, encodage, vérification de cohérence, tests d'acceptation, vérification temporelle, vérification par inversion, ...)
  - robustesse requise au niveau des interfaces
  - mécanismes de protection

Thème 9 - Fiabilité

Traitement des fautes

- Implique la localisation et la correction de la faute
- Localisation :
  - Il faut détecter les erreurs avant qu'elles ne soient rendues obscures par des dommages conséquents
  - On peut ignorer la faute
  - La détection dépend notre connaissance de l'application

Correction

- Correction :
  - matériel $\rightarrow$ remplacement de la pièce
  - logiciel $\rightarrow$ remplacement de lignes de code (patch, service pack, ...)

Thème 9 - Fiabilité

Reprise

- Permet de reprendre un traitement fautif suite à une tentative de correction
- Il faut d'abord identifier les erreurs, estimer les dommages et les réparer.
- Deux types de reprise : avant et arrière

Reprise arrière

- Méthode la plus générale
- Répare les dommages en retournant les processus affectés vers un état correct qui existait avant l'erreur
- Une fois ce retour en arrière fait on tente de continuer les opérations normales
- Requiert la présence de points de sauvegarde

Reprise arrière

- Retour en arrière $\rightarrow$ perte de données
- La perte est l'intervalle entre les points de sauvegarde
- Réduction de la perte : trace (audit trail)
- Problèmes : coûteux en temps et en espace

Reprise arrière

- Alternative : blocs de reprise
- Un bloc de reprise est une section de programme :

```
ensure «test d'acceptation» by     «alternative 1»
                            elseby «alternative 2»
                            elseby «alternative 3»
                             ...
                            elseby «alternative n»
```

Reprise arrière

- Alternatives $\rightarrow$ logiciels de rechange de conception différente
- L'échec est exceptionnel : le remplacement n'est pas permanent
- Récupération d'état : cache de reprise et recul (interaction)
- Que faire si les processus sont en communication ?

Reprise arrière

- Méthode simple car la reprise ne dépend pas de l'erreur
- L'évaluation des dommages ne tient pas compte de la nature de la faute
- Reprise arrière générale pour toutes les erreurs est réalisable

Reprise avant

- Complexe car l'identification de la faute et la reprise sont intimement liées
- Reprise avant est conçu comme une partie intégrale du système
- Malgré tout elle peut être «simple» et efficace
- Types d'erreurs traitables : erreurs dans les composants (prise en main des exceptions), erreurs dans les systèmes en interaction (compensation) et erreurs dans les algorithmes (programmation n-versions, blocs de reprise)

Erreurs dans les composants

- Traitées par la prise en main des exceptions ou l'auto-stabilisation
- Prise en main des exceptions :
  - requiert la prédiction des fautes possibles et exige de savoir comment elles se manifestent en dehors du composant
  - consiste à inclure, lors de la spécification du composant, un certain nombre de comportements indésirables en plus ou parmi des comportements normaux qu'il doit avoir

Erreurs dans les composants

Exemple :

On peut incorporer dans le matériel la reprise avant en utilisant des codes correcteurs d'erreurs pour traiter des informations fautives en mémoire ou lors d'une transmission. Un code correcteur donné sera cependant utilisable seulement pour corriger une classe particulière et limitée d'erreurs. Ainsi, certains codes sont conçus pour corriger des erreurs impliquant au plus n bits successifs.

Erreurs dans les composants

- Il est préférable de distinguer traitement normal et exception
- Cela est fait dans plusieurs langages («on condition», «try,throw,catch», «on exception», «stae, spie», «signal», ...)

Erreurs dans les systèmes en interaction

- Traitée par la compensation
- Fournit les moyens pour essayer de traiter avec la situation où une erreur est détectée pendant que le système est en communication avec un environnement qui ne peut être retourné en arrière
- Acte par lequel un système fournit de l'information supplémentaire dans le but de corriger les effets d'informations qu'il a déjà expédiées à un autre système

Exemple

Soit une base de données servant au contrôle de l'approvisionnement. Cette base de données est mise à jour par un message d'entrée indiquant la sortie de quelques éléments. Si, plus tard, on découvre que ce message était incorrect, il doit être possible de compenser pour le mauvais message par un autre message indiquant d'enregistrer l'acquisition des éléments de remplacement. Un simple compensation comme celle-ci est probablement insuffisante si, par exemple, le système de contrôle de l'approvisionnement, comme résultat de la mauvaise information, a recalculé les niveaux optimums des inventaires et a produit des ordres d'achats pour le remplacement des éléments impliqués.

Erreurs dans les systèmes en interaction

- La compensation utilise des techniques similaires à la prise en main des exceptions
- Il n'existe aucune solution générale
- La correction manuelle est possiblement nécessaire

Thème 9 - Fiabilité

Traitement des fautes à plusieurs niveaux

- Les techniques vues peuvent s'utiliser dans une architecture par couches
- Chaque niveau est responsable de la détection et la reprise de façon à ce qu'il apparaisse sans faute pour les niveaux supérieurs
- On étend le concept de masquage fait par le matériel
- Si le masquage est impossible, l'erreur est rapportée au niveau supérieur de façon méthodique
- Le niveau supérieur tente à son tour une reprise et un masquage
- Éventuellement l'erreur est rapportée à l'usager si elle ne peut être masquée par aucun niveau

Exemples

Soit un processus usager qui souhaite ouvrir un fichier. Le processus effectue un appel au système de fichiers par l'intermédiaire d'une routine («`open`*») qui à son tour appelle le système d'entrée/sortie grâce à la procédure «*`doio`*» qui permet de lire le répertoire de l'usager. L'opération d'entrée/sortie est ensuite initialisée par le gestionnaire du disque.*

Exemples

S'il se produit une erreur de parité pendant l'opération. Un premier niveau de masquage peut être fourni par le matériel du contrôleur du disque qui est probablement conçu pour détecter de telles erreurs et pour relire le bloc correspondant. Si, après plusieurs essais, l'erreur persiste, elle est rapportée au pilote du disque grâce à un registre d'état qui est positionné par le contrôleur à la fin du transfert. Le pilote du disque peut tenter de masquer l'erreur en réinitialisant l'opération au complet. Si cela ne réussit pas, l'erreur doit être rapportée à la routine d'ouverture (≪`open`*≫) du système de fichiers. Au niveau du système de fichiers, la reprise est possible si une autre copie du répertoire existe quelque part. On peut alors lire cette copie et effectuer l'ouverture du fichier si aucune autre erreur ne se produit. Si cela échoue ou s'il n'y a pas de copies de sécurité pour le répertoire, alors l'erreur, ne pouvant plus être masquée, est rapportée au processus usager.*

Exemples

Une autre erreur qui peut se produire pendant l'ouverture est la corruption d'un pointeur dans les listes des demandes du disque. Cela peut être causée par un mauvais fonctionnement du matériel ou une faute de programmation. Si la liste est doublement chaînée et que la routine de manipulation en tient compte, alors toute perte peut être évitée. Le dommage au pointeur sera alors réparé par les routines de manipulation masquant ainsi l'erreur au pilote et au système de fichiers.

Exemples

Il est intéressant de noter ce qui peut se produire si la liste est simplement chaînée. La demande sera perdue et les valeurs des compteurs de la liste seront incohérentes avec le nombre de demandes. Deux choses peuvent alors se produire :

- *le pilote assume que le succès de l'opération d'attente implique l'existence d'une demande dans la liste. Il retirera de la liste une demande non-existante, interprétera le résultat comme une opération d'entrée/sortie valide et provoquera d'autres erreurs imprévisibles.*
- *le pilote vérifie la liste des demandes, constate qu'elle ne contient aucune demande valide et rapporte ce fait à la procédure «*`doio`*». Dans ce cas, l'erreur sera masquée à l'usager car la demande peut être régénérée.*

Traitement des fautes à plusieurs niveaux

- Certaines erreurs ne doivent pas être masquées

Par exemple, soit un débordement arithmétique détecté par le processeur et rapporté au noyau via le pilote d'interruption. La routine d'interruption du noyau peut rapidement identifier le coupable comme étant le processus courant pour le processeur concerné. Le processus est alors éliminé avec un message approprié. Un autre exemple est la violation de la mémoire et de la protection résultant de l'exécution du programme. Cette erreur entraîne aussi l'annulation du processus courant. Dans ce cas, on doit cependant noter les violations de la protection car elles peuvent être des tentatives pour percer le système.

Thème 9 - Fiabilité

Bibliothèques

- Apache Ignite
- Failsafe
- Hystrix (Netflix)
- JRugged
- Resilience4j

ESS No.1A

- ESS (Electronic Switching Systems)
- Temps de panne maximum : 2 heures sur 40 années
- Utilise la redondance matérielle

ESS No.1A

- Détection des erreurs : timing checks, coding checks, internal checks et vérification de copies, programmes de surveillance (audit) et programmes de diagnostics
- Traitement des fautes : programmes de traitement et de reprise
- Contrôle des dommages : programmes de surveillance et de reprise
- Reprise : reprise avant, compensation et redondance pour réparer les dommages

THE

- Conçu en 1968 par Dijkstra et une équipe de 5 personnes
- Conception soigneuse par couches
- Preuve de bon fonctionnement et tests à chaque niveau (ont testé tous les états)
- Fiabilité : on force l'utilisation d'un langage de haut niveau (Algol)
- Peu de mécanismes de reprise

Guardian

- Tandem
- Tout était redondant (2 UCTs, 2 bus, 2 disques, 2 contrôleurs par disque, ...)
- Plusieurs alarmes en cas d'erreurs
- Guardian : conception soigneuse et mécanismes de traitement des erreurs
- Conception par couches et communication par messages (isole les processus et évite la propagation des dommages)

Guardian

- Traitement des erreurs
  - Processus de rechange
  - Échange périodique de messages
  - Vérification de structure
  - Vérification des opérations d'E/S
  - Mécanismes de protection

OS/MVS

- Système de production sur Mainframe → fiabilité
- Mécanismes pour la fiabilité :
  - Protection (clé et mémoire virtuelle)
  - Traces
  - Reprise : application de tests et reprise — routines de reprise (RTM et RMS), spie et espie

Unix

- Il n'est pas conçu pour être fiable
- Conception par couches
- Routines de reprise
- Signal
- Fournit le moyen de développer des logiciels plus fiables (éditeurs, makefile, RCCS, CVS, LEX, Yacc, script, ...)
- Système de fichiers aven journal (solaris, linux, ...)

Unix

```
#include <iostream>
#include <signal.h>
typedef void Sigfunc(int);
Sigfunc *signal(int,Sigfunc *);
//
//  Routine qui traite les signaux de type CRTL-c
//
void controlc(int no)
{
   cout << "Il y a eu un crtl-c, je ne termine pas " << endl;
   nb_crtl_c++;
   // je veux encore capturer les crtl-c si ce n'est pas le 5e
   if (nb_crtl_c < 4) signal(SIGINT,controlc) ;
}
```

Unix

```
//
//  Routine qui traite les signaux de type CRTL-z
//

void suspendre(int no)
{
    cout << "Non-- je ne veux pas etre suspendu" << endl;
    // je veux encore capturer les crtl-z
    signal(SIGTSTP,suspendre) ;
}
```

Unix

```
//
// Programme principal -- ne fait rien
//
int main()
{
  int b=0;
  // on veut empecher l'arret du processus par un crtl-z
  signal(SIGTSTP,suspendre) ;
  // on veut empecher la destruction du processus par on crtl-c
  signal(SIGINT,controlc) ;

  // on boucle a l'infini
  while(1) ;
}
```

Windows

- Il n'est pas conçu pour être fiable
- Outils de développement
- Système de fichiers journalisé (NTFS)
- ...