

Chapitre 1

Principes

1.1. Introduction

Ce chapitre¹ a pour objectif de présenter l'ensemble des techniques permettant de sécuriser le fonctionnement d'une architecture matérielle. Nous parlerons d'architecture matérielle car la sécurisation peut se baser sur une ou plusieurs unités de calcul. Nous laissons volontairement les aspects « logiciel » de côté.

1.2. Présentation des notions de base : fautes, erreurs et défaillances

1.2.1. *Entrave à la sûreté de fonctionnement*

Comme cela est indiqué dans [LAP 92], la sûreté de fonctionnement d'un système complexe peut être mise à mal par trois types d'évènements : les défaillances, les fautes et les erreurs. Les éléments du système sont soumis à des défaillances, ces défaillances peuvent amener le système dans des situations d'accidents potentiels.

Comme indiqué dans la norme IEC 61508 [IEC 00], la défaillance (parfois appelée panne) est une cessation de l'aptitude d'une unité fonctionnelle à accomplir une fonction requise. Comme l'accomplissement d'une fonction requise exclut

Chapitre rédigé par Jean-Louis BOULANGER.

1. Ce chapitre est basé sur du matériel pédagogique réalisé en commun avec M. Walter SCHÖN, professeur au sein de l'université de technologie de Compiègne que je ne pourrais jamais assez remercier.

nécessairement certains comportements et que certaines fonctions peuvent être spécifiées en termes de comportement à éviter, l'occurrence d'un comportement à éviter est une défaillance.

De la définition précédente, il faut déduire la nécessité de définir les notions de comportement normal (sûr) et de comportement anormal (non sûr) avec une frontière nette entre les deux.

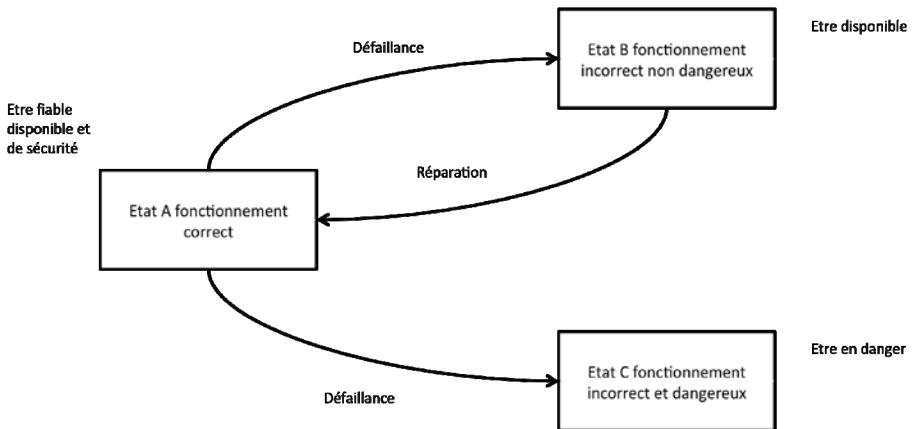


Figure 1.1. Evolution de l'état d'un système

La figure 1.1 présente une représentation des états d'un système (correct, incorrect) et les transitions possibles entre ces états. Les états du système peuvent être classés en trois familles :

- les états corrects : il n'y a pas de situation dangereuse ;
- les états incorrects sûrs : une défaillance a été détectée et le système est dans un état sûr ;
- les états incorrects : il s'agit d'une situation dangereuse non maîtrisée : il existe des accidents potentiels accessibles.

Quand le système atteint un état de repli, il peut y avoir un arrêt complet ou partiel du service. Les états de repli peuvent permettre un retour à l'état correct suite à une action de réparation.

Les défaillances peuvent être aléatoires ou systématiques. La défaillance aléatoire survient de manière non prévisible et se trouve être le résultat d'un ensemble de dégradations qui touchent les aspects matériels du système. En

général, la défaillance aléatoire peut être quantifiée du fait de sa nature (usure, vieillissement, etc.).

La défaillance systématique est reliée de façon déterministe à une cause. La cause de la défaillance ne peut être éliminée que par une reprise du processus de réalisation (conception, fabrication, documentation) ou une reprise des procédures. Etant donné son caractère, la défaillance systématique n'est pas quantifiable.

La défaillance est une manifestation externe observable d'une erreur (la norme IEC 61508 [IEC 98] parle d'*anomalie*).

DÉFINITION 1.1 (ERREUR).– *L'erreur est une conséquence interne d'un défaut lors de la mise en œuvre du produit (une variable ou un état du programme erroné).*

Malgré toutes les précautions qui sont prises lors de la réalisation d'un composant, celui-ci peut être soumis à des défauts de conception, à des défauts de vérification, à des défauts d'utilisation, à des défauts de maintien en condition opérationnelle, etc.

DÉFINITION 1.2 (DÉFAUT).– *Un défaut est une non-conformité introduite dans le produit (par exemple un code erroné).*

A partir du défaut, il est possible d'introduire la notion de faute. La faute étant la cause de l'erreur (exemples : court-circuit, perturbation électromagnétique, faute de conception).

DÉFINITION 1.3 (FAUTE).– *La faute est un processus (humain ou non) de génération d'un défaut.*

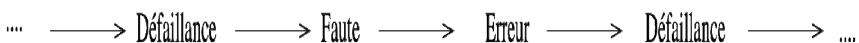


Figure 1.2. *Chaîne fondamentale*

A titre de bilan, il faut donc rappeler que la confiance dans la sûreté de fonctionnement d'un système peut être compromise par l'apparition des entraves que sont les fautes, les erreurs et les défaillances. La figure 1.2 présente la chaîne fondamentale qui fait le lien entre les entraves. L'apparition d'une défaillance peut faire apparaître une faute qui à son tour est l'occasion de faire apparaître une (des) erreur(s) ; cette (ces) nouvelle(s) erreur(s) pouvant avoir pour conséquence l'apparition d'une nouvelle défaillance.

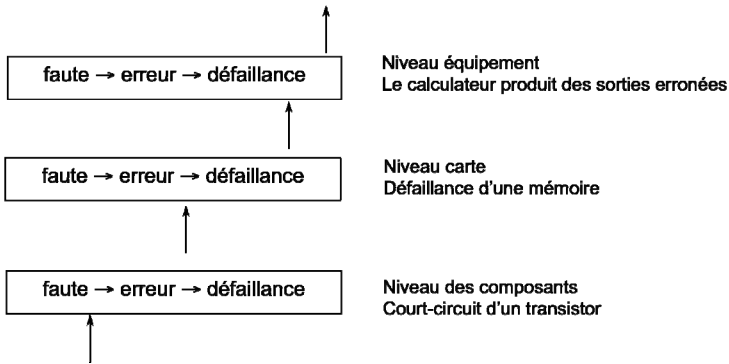


Figure 1.3. Propagation dans un système

Le lien entre les entraves doit être vu au travers de l'ensemble du système comme le montre l'exemple de la figure 1.3.

La figure 1.4 propose un exemple de mise en œuvre des défaillances. Comme indiqué précédemment, une défaillance est détectée au travers de la divergence du comportement du système par rapport à ce qui a été spécifié. Cette défaillance se produit aux limites du système par le fait qu'un ensemble d'erreurs, qui sont elles internes au système, a des conséquences sur l'élaboration des sorties. Dans notre cas, la source des erreurs est une faute dans l'exécutable embarqué. Ces défauts peuvent être de deux natures : soit ce sont des fautes introduites par le programmeur (BUG) soit ce sont des défauts introduits par les outils (de génération de l'exécutable, moyens de téléchargement, etc.) ou par des défaillances du matériel (défaillance mémoire, court-circuit sur un composant, perturbation extérieure (CEM par exemple), etc.).

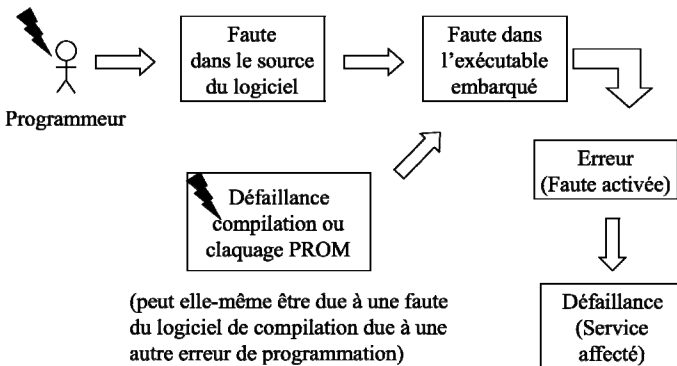


Figure 1.4. Exemple de propagation

Il est à noter que les fautes peuvent ainsi être introduites lors de la conception (défaut dans le logiciel, sous-dimensionnement du système, etc.), lors de la production (génération de l'exécutable, fabrication du matériel, etc.), lors de l'installation, lors de l'utilisation et/ou lors de la maintenance. Le diagramme de la figure 1.4 peut ainsi être décliné pour les différentes situations. La figure 1.5 présente l'impact d'une erreur humaine.

A ce point de la discussion, il est alors intéressant de noter qu'il existe deux familles de défaillances, les défaillances systématiques et les défaillances aléatoires. Les défaillances aléatoires sont dues aux processus de production, au vieillissement, à l'usure, aux dégradations, aux phénomènes extérieurs, etc. Les défaillances systématiques sont reproductibles, en effet elles ont pour source des défauts de conception. Il est à noter qu'une défaillance aléatoire peut avoir pour source un défaut de conception (sous-estimation de l'effet de la température sur le processeur). Comme nous le verrons dans la suite, il existe plusieurs techniques (diversité, redondance, etc.) permettant de détecter et/ou de maîtriser les défaillances aléatoires. Pour les défaillances systématiques, leur maîtrise est plus délicate car elle repose sur de la qualité (pratique préétablie et systématique) et des activités de vérification et de validation.

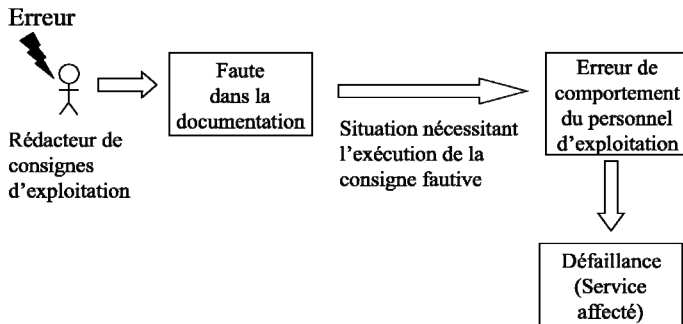


Figure 1.5. Impact d'une erreur humaine

1.2.2. Etudes de démonstration de la sécurité

La section précédente a permis de rappeler certains concepts de base (faute, erreur et défaillance), mais la recherche systématique des défaillances et de l'analyse de leurs effets sur le système est réalisée au travers d'activités telles que les analyses préliminaires de risque (APR), les analyses des modes de défaillance et de leurs effets (AMDE), des arbres de fautes, etc.

Ces analyses liées à la sûreté de fonctionnement sont maintenant classiques (voir [VIL 88] par exemple) et imposées par les normes. L'ensemble de ces études permet de réaliser une démonstration de la sécurité qui sera formalisée au travers d'un dossier de sécurité. La norme générique IEC 61508 [IEC 98], applicable aux systèmes à base d'électronique et d'électronique programmable, couvre ce point et propose une approche générale.

1.2.3. *Bilan*

Lors de la conception d'une architecture informatique, il faut tenir compte de trois types de défaillance qui sont :

- les défaillances aléatoires des composants matériels ;
- les défaillances systématiques de conception : tant au niveau matériel que logiciel ;
- les « erreurs » de spécification au niveau du système.

1.3. Architecture sûre et/ou disponible

Au niveau d'une architecture matérielle, la principale défaillance est liée à l'émission d'une sortie erronée. Il y a deux possibilités :

- l'émission d'une sortie permissive à tort, qui engendre un problème de sécurité (par exemple mise au vert d'un feu autorisant le passage d'un véhicule à tort) ;
- l'émission d'une sortie restrictive à tort, qui engendre un problème de disponibilité (exemple : les trains à l'arrêt).

En fonction de l'impact de l'absence de sortie, il est possible de définir deux familles de système :

- les systèmes intègres : il ne doit pas y avoir de sorties erronées (mauvaises données ou données correctes à un instant incorrect, etc.). Les systèmes intègres sont des systèmes où le processus est irréversible (par exemple : les transactions bancaires). Pour ce genre de système, il est préférable de s'arrêter plutôt que de mal fonctionner. Le système est dit *fail-silent* (*fail-safe*, *fail-stop*) ;
- les systèmes persistants : la non-sortie d'une donnée correcte ne doit pas se produire. Les systèmes persistants sont des systèmes ne disposant pas d'état de repli, ce qui implique que l'absence de donnée entraîne la perte du contrôle. Pour ce type de système, il est préférable d'avoir quelques mauvaises données plutôt que pas du tout. Le système est dit *fail-operate*.

Un système intègre devient un système sûr si l'on dispose d'un état de repli qui peut être atteint de façon passive. A titre d'exemple, dans le domaine ferroviaire, la

moindre défaillance vient à couper l'alimentation d'énergie. Et sans énergie le freinage du train n'est plus désactivé. Le train atteint donc de façon passive l'état de sécurité : « train à l'arrêt ».

1.4. Réinitialisation d'une unité de traitement

La section 1.3 a été l'occasion dans le cadre de la discussion sur la persistance et l'intégrité d'introduire la problématique de la nécessité de disposer ou non d'un état de repli.

En cas d'un équipement intègre, le passage dans l'état de repli est définitif. Dans le cadre de défaut fugitif, l'indisponibilité induite peut être inacceptable du point de vue du client (à titre d'exemple, on peut citer la perte de fonction ABS au sein d'une automobile). C'est pourquoi, il est tentant de passer par une étape intermédiaire qui est la tentative de réinitialisation (*reset*) de tout ou partie (une unité de traitement parmi n) de l'équipement.

L'utilisation du *reset* doit être contrôlée, plusieurs problèmes peuvent apparaître :

- le *reset* d'une unité de traitement sur défaillance, peut entraîner lors de son démarrage le *reset* de l'unité demandeuse au travers d'une divergence des contextes, on peut ainsi entrer dans une boucle de *reset* non contrôlée. Il faut alors être en mesure de garantir que les sorties seront dans un état restrictif durant ces états intermédiaires ;

- le temps de *reset* peut être très inférieur au temps de détection de l'erreur, et malgré les demandes de *reset*, le système produit des sorties alors qu'il y a une erreur. La détection d'une *boucle de reset* est réalisée au travers de la mise en place d'un *compteur de reset*. Ce compteur de *reset* devant être contrôlée. Il faut de plus démontrer que le *reset* à un effet vis-à-vis des défaillances que l'on cherche à couvrir ;

- etc.

Concernant la réinitialisation d'un équipement, il faut donc être attentif et démontrer qu'il y a une efficacité de la mesure et aucun risque de masquage d'une situation erronée.

1.5. Présentation des techniques de sécurisation

La sécurisation d'une architecture matérielle peut se faire au travers de cinq grandes techniques :

- la détection d’erreurs (paragraphe 1.5.1) ;
- la mise en place d’une diversité (paragraphe 1.5.2) ;
- la mise en place d’une redondance (paragraphe 1.5.3) ;
- la mise en place d’une reprise (paragraphe 1.5.4).

Dans le cadre de cette section, nous allons présenter ces différentes techniques et discuter leurs mises en œuvre.

1.5.1. Détection d’erreurs

1.5.1.1. Concepts

Comme le montre la figure 1.6, cette technique vise à compléter l’architecture matérielle avec un élément permettant la détection d’erreurs ; en cas de détection d’erreur, différentes solutions peuvent être envisagées comme le redémarrage ou la coupure des sorties.

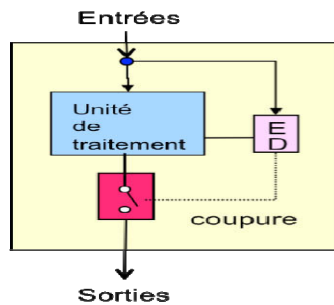


Figure 1.6. Principe de la détection d’erreurs

L’architecture de la figure 1.6 est une architecture intégrée ; en cas de détection d’erreurs, il y a coupure des sorties.

La mise en œuvre de la détection d’erreurs repose sur trois techniques :

- la détection de la cohérence temporelle : elle se fait par la mise en place d’un « chien de garde » (*watchdog*) qui va pouvoir détecter la dérive temporelle de l’application, les boucles infinies ou le non-respect d’une échéance temporelle. Le « chien de garde » peut être un élément matériel ou logiciel ;
- la détection de défaut matériel : elle se fait par la mise en place d’autotests. Ces autotests permettent de tester plus ou moins complètement un élément matériel (ALU, mémoire, voteur, etc.). Ces autotests peuvent être déroulés totalement ou

partiellement à l'initialisation, à la fin de la mission, à chaque cycle et/ou à certaines périodes. La difficulté principale de cette technique réside dans la pertinence des tests (couverture et exhaustivité) selon le moment de leur exécution ;

– la détection de défaut d'exécution : elle est mise en place au travers de la vérification de différentes cohérences dans le comportement de l'application. Il est possible d'avoir une analyse de la cohérence des entrées (corrélation d'entrée, redondance d'entrée, etc.), de la cohérence des sorties (une sortie ne peut pas évoluer n'importe comment d'un cycle à l'autre), de la cohérence du comportement (la gestion de « drapeau » au sein du code permet de vérifier le chemin d'exécution), la correction du chemin d'exécution (le calcul « hors-ligne » de tous les chemins et la comparaison « en ligne » du chemin d'exécution).

1.5.1.2. Chien de garde

Simple et peu coûteux, le chien de garde ou *watchdog* est utilisé pour détecter les erreurs (fréquentes et pas toujours tolérables) conduisant à l'inactivité (« plantage ») d'une unité centrale. Plus généralement, elle permet de détecter une dérive temporelle.

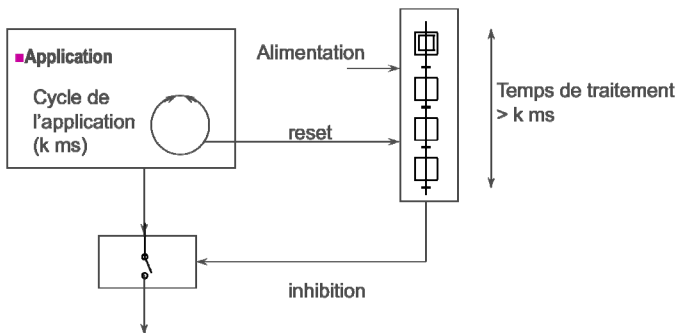


Figure 1.7. Chien de garde – Watchdog

La dérive temporelle d'une architecture peut être induite par différents types de défaillances : défaillance de l'horloge ou des *timers*, défaillance de l'application logicielle (boucle infinie, blocage, dépassement du cycle de traitement, etc.), défaillance aux niveaux des ressources, etc.

Dans le cadre de la figure 1.7, le chien de garde est un dispositif matériel, rafraîchi (ou réarmé) périodiquement par le processeur (par exemple en début de chaque cycle applicatif) et qui dispose de la possibilité de couper les sorties. Il est à noter que le chien de garde peut être aussi une application logicielle.