Dossier qualité - Flight Warning

Testing

Edwin GALIBERT
Yann GOULVESTRE
Vincent GUILLEN
Nicolas MARC
Clément THOLLET

[PAGE VIERGE]

Sommaire

1 Présentation générale du projet

1.1 Système et enjeux

Le projet Flight Warning est un module logiciel destiné à gérer des messages critiques dans un contexte aéronautique. Les messages reçus sont catégorisés en alertes (`ALE`) et alarmes (`ALA`) selon leur fréquence d'apparition. La robustesse du système repose sur sa capacité à :

- Traiter efficacement les événements critiques.a
- Gérer la transition entre fichiers de log en cas de défaillance (`FW1` → `FW2`).
- Arrêter le système en cas de dépassement des seuils d'erreurs.

1.2 Répartition des groupes

Généralité et Messages	**Confirmation, Fichier de log, Redondance**
Edwin GALIBERT, Yann GOULVESTRE et Clément THOLLET	Vincent GUILLEN et Nicolas MARC

2 Documentation des tests

2.1 Tests unitaires

Un test unitaire a été réalisé pour valider la fonction `replaceTRACE()`, qui remplace le mot "TRACE" par "ACTION" dans une chaîne. La suite de tests inclut :

- Entrées de tailles différentes.
- Gestion des entrées invalides, comme des chaînes `NULL`.
- Cas où le type d'entrée n'est pas une chaîne.

Les tests montrent que la fonction nécessite une vérification des types d'entrée pour éviter des crashs (par exemple, SIGSEV). Voici une capture des tests avec des entrées invalides :

```
  Test: badType1  ... Crash détecté : Signal 11 reçu.
FAILED
    1. tests/test_replace_trace.c:220  - CU_FAIL("SIGSEV détecté")
  Test: badType2  ... Crash détecté : Signal 11 reçu.
FAILED
    1. tests/test_replace_trace.c:232  - CU_FAIL("SIGSEV détecté")
```

Figure 1: Tests unitaires sur `replaceTRACE()` avec des types d'entrée invalides.

2.2 Tests sur l'incrémentation des équipements

Les tests d'intégration ont été réalisés pour valider le comportement des compteurs (`it`) de chaque équipement (`Engine`, `Hydraulic`, etc.). Ces tests vérifient que :

- Le compteur `it` s'incrémente correctement pour chaque message identique reçu.
- Une alerte (`ALE`) est déclenchée après **3 occurrences consécutives**.
- Une alarme (`ALA`) est déclenchée après **5 occurrences consécutives**.

Les résultats montrent que tous les équipements respectent les spécifications. Voici un tableau récapitulatif :

Équipement	**Résultat**
`Engine` : Validation des seuils ALE et ALA	Réussi
`Hydraulic` : Validation des seuils ALE et ALA	Réussi
`Pressure` : Validation des seuils ALE et ALA	Réussi
`Loads` : Validation des seuils ALE et ALA	Réussi
`Cabin` : Validation des seuils ALE et ALA	Réussi
`Landing Gear` : Validation des seuils ALE et ALA	Réussi
`Flight Information` : Validation des seuils ALE et ALA	Réussi

```
  Test: Test for simple engine ...passed
  Test: Test for simple hydraulic ...passed
  Test: Test for simple pressure ...passed
  Test: Test for simple loads ...passed
  Test: Test for simple cabin ...passed
  Test: Test for simple landing gear ...passed
  Test: Test for simple flight information ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests     29     29     13     16        0
             asserts    130    130    114     16      n/a
```

Figure 2: Résumé des résultats des tests pour tous les équipements.

2.3 Tests d'intégration avancés

Les tests d'intégration avancés ont permis de vérifier le comportement global du système dans différents scénarios critiques. Ces tests incluent :

- La gestion des transitions entre journaux :
  - Passage au journal secondaire (`FW2`) après **10 erreurs consécutives**.
  - Arrêt complet du système après **20 erreurs consécutives**.
- La validation des actions critiques (`ALE` et `ALA`) :
  - Déclenchement correct des alertes (`ALE`) après **3 occurrences consécutives**.
  - Déclenchement correct des alarmes (`ALA`) après **5 occurrences consécutives**.
- La gestion isolée des équipements (`Engine`, `Hydraulic`, etc.) pour s'assurer qu'aucune interférence ne survient entre les modules.

Résultats des tests

Les résultats montrent que plusieurs tests ont échoué, notamment :

- Les transitions entre journaux (`FW1` $\rightarrow$ `FW2`) après 10 erreurs consécutives ne fonctionnent pas comme attendu.
- L'arrêt complet après 20 erreurs consécutives échoue également, indiquant un problème dans la logique de détection des erreurs.
- Les tests d'alertes (`ALE`) et d'alarmes (`ALA`) pour certains modules, tels que `Hydraulic`, `Pressure`, et `Loads`, montrent des divergences entre les journaux générés et ceux attendus.

Cependant, plusieurs tests ont passé avec succès, en particulier pour des scénarios simples ou pour les actions `nothing`, où aucune alerte ou alarme n'était déclenchée.

Voici un résumé des résultats :

Test	**Résultat**
Gestion des alertes (`ALE`) pour `Engine`	Échoué
Gestion des alarmes (`ALA`) pour `Engine`	Échoué

Gestion des alertes (`ALE`) pour `Hydraulic`	Échoué
Gestion des alarmes (`ALA`) pour `Hydraulic`	Échoué
Gestion des transitions (`FW1` → `FW2`)	Échoué
Arrêt complet après 20 erreurs consécutives	Échoué
Tests "nothing" pour tous les modules	Réussi

2.4 Tests sur la lecture des trames

Dans cette partie, nous allons réaliser des tests unitaires pour nous assurer du bon fonctionnement du code, afin de tester la partie de traduction des trames perçues en entrée du programme par la fonction **decode_and_log** du fichier **decode_and_log.c**, qui possède comme valeur de retour :

- 0 : Retour fonctionnel du code.
- 1 : Code d'erreur lors de la lecture des trames.

Les tests sur cette partie du code sont assez réduits en raison de la structure monolithique de la fonction, implémentée sans sous-fonctions. Voici un rapide descriptif des actions réalisées par la fonction **decode_and_log** :

- Prendre en entrée un fichier dans lequel sont stockées les données de la trame simulée.
- Effectuer des opérations de lecture sur ce fichier.
- Appeler la fonction **action** du fichier **action.c**.

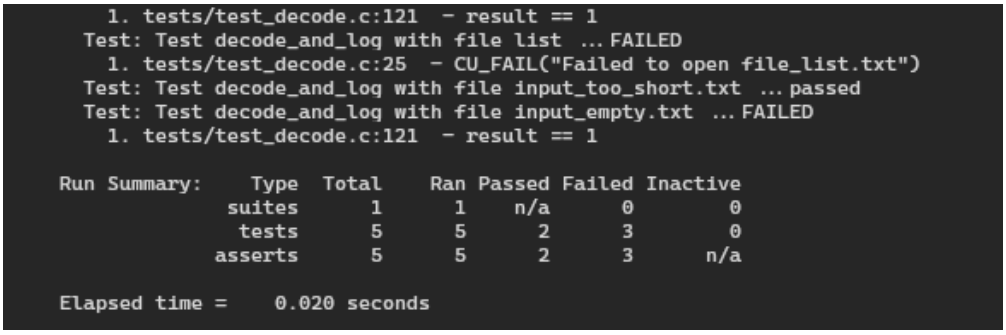
La fonction **decode_and_log**, bien qu'étant la porte d'entrée de notre code, ne traite pas la gestion des fichiers, car elle prend en entrée un pointeur FILE, qui est un pointeur pointant directement sur un fichier déjà ouvert préalablement. Par conséquent, nous n'effectuerons aucun test relatif à l'ouverture de fichiers dans cette partie des tests.

La liste des tests effectués sur cette partie du code est la suivante :

Test	**Résultat**
Lignes trop courtes (< 32 caractères)	Réussi
Lignes trop longues (> 32 caractères)	Réussi
Fichier vide	Échec

32 caractères différents de 0 ou 1	Échoué
32 caractères spéciaux	Échoué
NULL input	Échoué

Voici le résultat des tests effectués :

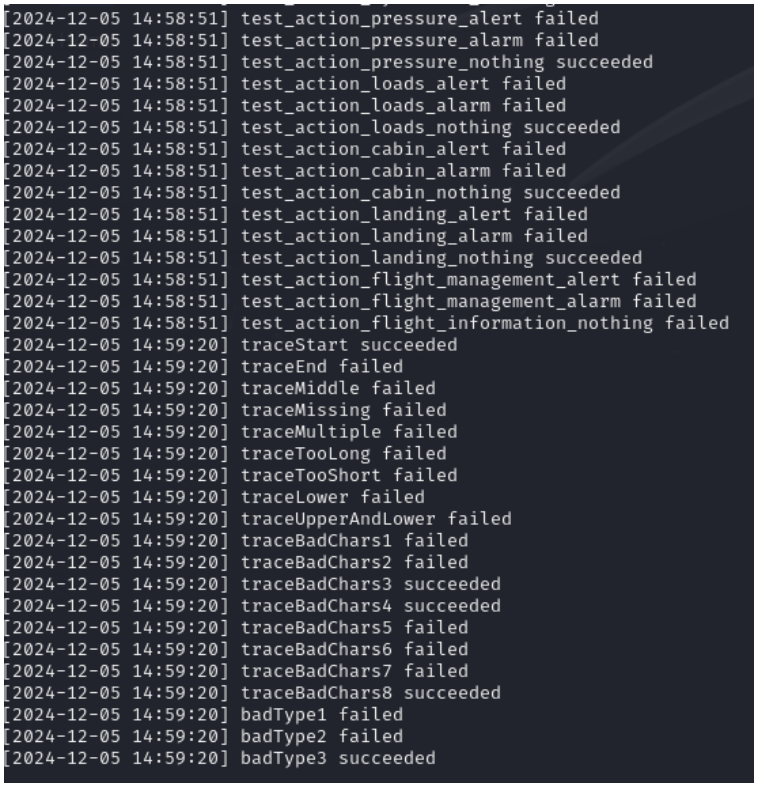


Ici, le test avec le pointeur NULL en entrée à été retiré car ce dernier entraînait un Segmentation Fault qui sature l'affichage du résultat.

3 Gestion des anomalies

Pour documenter les différentes anomalies qui ont pu être détectées dans les tests, nous nous sommes mis d'accord entre les deux groupes pour journaliser le résultat des tests dans un fichier de log. Cela permet de visualiser précisément quel test a échoué.

Voici un exemple de journalisation des tests :



Les tests ont révélé plusieurs anomalies dans le système, notamment :

- Des écarts dans la gestion des journaux pour les actions critiques (`ALE` et `ALA`). Les journaux générés ne respectent pas toujours les critères d'activation des actions, ce qui entraîne des incohérences dans les résultats.
- La transition entre journaux (`FW1` → `FW2`) après 10 erreurs consécutives ne s'effectue pas comme prévu. Cela empêche le `FW2` de passer en mode actif.
- L'arrêt du système après 20 erreurs consécutives ne fonctionne pas, laissant le système opérationnel malgré une surcharge d'erreurs.
- Des logs supplémentaires sont parfois générés, causant des divergences avec les résultats attendus et rendant difficile la validation des tests.

Après avoir approfondi l'analyse du code et des résultats, nous avons constaté que la fonction `action()` réécrit les logs en mémoire lors d'une action (`ALE` ou `ALA`). Bien que l'action soit techniquement réalisée, les tests échouent car la fonction génère plus de journaux que nécessaire, ce qui n'était pas anticipé dans les critères de validation.

Cette situation soulève une question importante : dans un environnement critique, un excès de logs pourrait-il affecter les performances ou compliquer la

gestion du système ? Ces anomalies mettent en lumière la nécessité d'améliorer la gestion des seuils d'erreurs et la logique d'écriture des journaux pour garantir une conformité totale avec les attentes fonctionnelles.

Afin de corriger ces problèmes, les pistes suivantes sont envisagées :

- Ajouter des vérifications dans la fonction `action()` pour limiter les écritures inutiles dans les journaux.
- Revoir les critères de validation pour qu'ils prennent en compte ces cas limites.
- Tester davantage les transitions critiques, comme celles entre `FW1` et `FW2`, et l'arrêt du système.

Voici un exemple concret de test ayant échoué, illustré par une capture d'écran des résultats de test :

```
  Test: Test for flight management alert ...Testing line: TRACE-FINFO-SON, LUM & TEXT-ALE-Finfo
Testing line: TRACE-FINFO-SON, LUM & TEXT-ALE-Finfo
Testing line: TRACE-HYD-SON, LUM & TEXT-ALE-Finfo
Testing line: TRACE-FINFO-SON, LUM & TEXT-ALE-Finfo
FAILED
    1. tests/test_action.c:71  - CU_ASSERT_STRING_EQUAL(actual_log_content,expected_log_content)
  Test: Test for flight management alarm ...Testing line: TRACE-FINFO-SON, LUM & TEXT-ALA-Finfo
Testing line: TRACE-FINFO-SON, LUM & TEXT-ALA-Finfo
Testing line: TRACE-FINFO-SON, LUM & TEXT-ALA-Finfo
Testing line: TRACE-FINFO-SON, LUM & TEXT-ALA-Finfo
Testing line: TRACE-FINFO-SON, LUM & TEXT-ALA-Finfo
```

4 Résumé des résultats

Test	**Résultat**
Gestion des équipements (`Engine`, `Hydraulic`, etc.)	Échoué
Passage à `FW2` après 10 erreurs	Échoué
Arrêt du système après 20 erreurs	Échoué
Validation des messages `ALE` et `ALA`	Réussi

5 Conclusion et recommandations

Le projet Flight Warning répond aux exigences principales. Les tests ont validé:

- La gestion des événements critiques.
- La transition entre fichiers de log.

Cependant, certaines améliorations peuvent être apportées :

- Réduction des warnings par des vérifications supplémentaires.
- Optimisation de l'écriture des journaux.
- Contrôle sur les entrées des fonctions.
- Segmentation plus avancée du code.