

Gestion d'erreur

INF3173 – Principes des systèmes d'exploitation
Automne 2024

Francis Giraldeau
Giraldeau.francis@uqam.ca

Université du Québec à Montréal



Agenda

- Code de retours v.s. exceptions
- Traiter les codes de retour
- Affichage à l'utilisateur
- Utilisation efficace de goto

Gestion d'erreur

- En Java une méthode retourne soit la valeur, ou se termine avec une exception
- En C, la convention est d'utiliser la valeur de retour pour indiquer une erreur
 - Appel système: valeur < 0 signifie erreur
- Si on ne vérifie pas le code de retour, le programme peut continuer dans un mauvais chemin!

Mauvaise pratique : appel à exit()

- Si un appel système échoue, alors on retourne l'erreur à la fonction appelant (parent)
- Si on fait un appel à exit(), alors le processus se termine **immédiatement**
- Exemple: on tente d'ouvrir un fichier dans un éditeur, mais le fichier n'existe pas. Le programme doit afficher l'erreur, et non pas quitter!

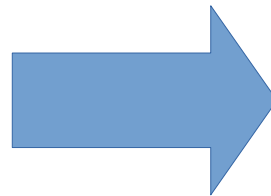
Afficher un message d'erreur

- Fonction `perror()` : affiche à la sortie d'erreur (`stderr`) le plus récent message d'erreur
- Fonction `strerror()` : obtenir la chaîne qui correspond à la dernière erreur
- Variable `errno` : variable globale qui représente la dernière erreur qui est survenue
- Permet d'informer l'utilisateur de l'erreur sans avoir à réinventer la roue
- Le message est traduit dans la langue active (locale)

Utilisation du goto

```
int alloc_sans_goto() {  
    // Allouer 3 entiers  
    int *i1 = malloc(sizeof(int));  
    if (i1 == NULL) {  
        return 1;  
    }  
    int *i2 = malloc(sizeof(int));  
    if (i2 == NULL) {  
        free(i1);  
        return 1;  
    }  
    int *i3 = malloc(sizeof(int));  
    if (i3 == NULL) {  
        free(i1);  
        free(i2);  
        return 1;  
    }  
  
    // Libérer toutes les allocations  
    free(i1);  
    free(i2);  
    free(i3);  
    return 0;  
}
```

Code de libération
duplicué!



```
int alloc_avec_goto() {  
    // Valeur de retour par défaut : erreur  
    int ret = 1;  
  
    // Allouer 3 entiers  
    int *i1 = malloc(sizeof(int));  
    if (i1 == NULL) {  
        goto err1;  
    }  
    int *i2 = malloc(sizeof(int));  
    if (i2 == NULL) {  
        goto err2;  
    }  
    int *i3 = malloc(sizeof(int));  
    if (i3 == NULL) {  
        goto err3;  
    }  
  
    // Succès  
    ret = 0;  
  
    // Libérer toutes les allocations  
    // Ordre inverse des allocations  
    free(i3);  
  
err3:  
    free(i2);  
err2:  
    free(i1);  
err1:  
    return ret;  
}
```

Code de libération
unique

Vérification fuite mémoire

- Mémoire allouée, mais jamais libérée
 - Si survient dans une boucle, alors peut épuiser la mémoire de l'ordinateur
 - Instrumentation de malloc/free, bilan à la fin
 - Analyse statique v.s. analyse à l'exécution
- AddressSanitizer
 - Instructions ajoutées à la compilation
 - Compilation et édition de liens: `-fsanitize=address`
 - Nécessite le code source du programme
- Valgrind
 - Exécute le programme sous observation
 - Ne nécessite pas le code source
 - Généralement plus lent que AddressSanitizer