

Université de Mons

Faculté des Sciences

US-MC-INFO60-014-C — Systèmes d'exploitation

Synthèse:

Systèmes d'exploitation

Auteur:
Baptiste GROSJEAN

Directeur:
Seweryn DYNEROWICZ

Numéro d'étudiant:
232732

Résumé

Juin 2024

Sommaire

1	Introduction	4
1.1	Vue d'ensemble	4
1.2	Abstractions de programmation	4
1.3	Processeur	4
1.3.1	Exécution de programme	5
1.3.2	Ordonnancement de programmes	5
1.4	Mémoire	5
1.4.1	Protection de la mémoire	5
1.4.2	Répartition de la mémoire	6
1.5	Stockage	6
1.6	Typologie des systèmes	6
1.7	Architecture	6
2	Evénements	7
2.1	Composition et exécution de programme	7
2.2	Concept de base – événement	8
2.3	Privilèges	8
2.4	Typologie	8
2.4.1	Exceptions (événements synchrones)	8
2.4.2	Interruptions (événements asynchrones)	9
2.4.3	Traitements	9
2.5	Conséquences possibles pour le programme	9
2.5.1	Evenement récupérable	9
2.5.2	Evenement irrécupérable	10
2.6	Mécanismes de déroutement	10
2.6.1	Registre de cause	10
2.6.2	Vectorisation	10
2.6.3	Niveaux de privilège	11
2.6.4	Basculement de stack	11
2.7	Multiplicité des occurrences	11
2.8	Gestion ré-entrante des événements	12
2.9	Priorités et masques d'interruptions	12
3	Mémoire	13
3.1	Projection et protection	13
3.1.1	Adressage absolu	13
3.1.2	Adressage relatif	13
3.1.3	Segmentation	13
3.1.4	Pagination	13
3.2	Gestion de la mémoire physique	13

3.2.1	Bitmpas	13
3.2.2	Listes chaînées	13
4	Processus	14
4.1	Constitution	14
4.1.1	Table des processus	14
4.1.2	Cycle de vie	14
4.2	Ordonnancement	14
4.2.1	Systèmes batch	14
4.2.2	Systèmes interactifs	14
4.2.3	Systèmes en temps réel	14
4.3	Multi-threading	14
5	Concurrence	15
6	Systèmes de fichiers	16
7	Entrées et sorties	17
	Figures	18
	Tableaux	19
	Algorithmes	20

1 Introduction

1.1 Vue d'ensemble

Un système d'exploitation est constitué d'un noyau auquel vient s'ajouter le code d'une distribution. Le développeur d'un système d'exploitation doit abstraire le fonctionnement des périphériques afin d'offrir une interface de programmation au programmeur. La gestion des ressources et les communications avec les différents périphériques est gérée par le système d'exploitation.

1.2 Abstractions de programmation

Un système d'exploitation peut être perçu comme exposant une interface de programmation système aux programmes qui tournent sur une machine.

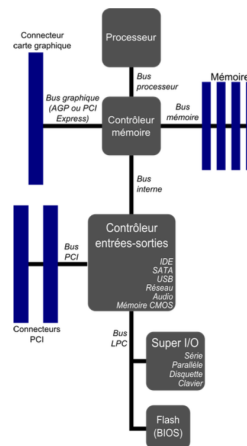


Figure 1: View

1.3 Processeur

Le **processeur** exécute les instructions et réalise les traitements. Il est caractérisé par son jeu d'instructions (Instruction Set Architecture), qui définit les instructions exécutables et leur format d'encodage (opcode et opérandes). Les instructions sont classées en plusieurs types :

- Arithmétique (e.g. `add`, `div`)
- Logique (e.g. `and`)
- Accès mémoire (e.g. `lw`, `sw`)
- Instructions système

Les processeurs modernes utilisent une structure de pipeline, permettant à différentes instructions d'occuper simultanément différentes parties du pipeline. Deux types principaux d'architecture de jeu d'instructions existent :

- RISC (Reduced Instruction Set Computer)
- CISC (Complex Instruction Set Computer)

Les responsabilités de gestion du processeur en collaboration avec le système d'exploitation incluent :

- Surveillance de l'exécution des instructions
- Répartition du temps processeur entre différents programmes

1.3.1 Exécution de programme

La première responsabilité de gestion consiste à surveiller l'exécution des instructions d'un programme utilisant le processeur. Le système d'exploitation et les programmes se partagent le processeur, nécessitant la suspension de l'exécution en cours lors d'un évènement particulier. Le processeur peut réagir de différentes manières :

- **Mauvais comportement** : déroutement vers une fonction du système d'exploitation pour traiter des erreurs telles qu'une division par zéro, un accès mémoire invalide ou une tentative d'exécuter une instruction privilégiée.
- **Anomalie non fatale** : correction des circonstances ayant provoqué l'anomalie et reprise de l'exécution.
- **Appel système** : branchement vers une fonction du système d'exploitation pour des demandes comme l'obtention de mémoire ou l'écriture dans un fichier.
- **Interruption externe** : traitement d'une interruption provenant d'un périphérique externe. Par exemple, une interruption du clavier nécessite de communiquer avec le contrôleur du clavier pour identifier la touche concernée et le programme destinataire.

1.3.2 Ordonnancement de programmes

La seconde responsabilité du système d'exploitation est de répartir le temps processeur entre les programmes. L'ordonnancement sélectionne le prochain programme à exécuter lorsque le processeur est libre. Différentes stratégies sont utilisées selon le type de système et la charge de travail. Les métriques d'évaluation incluent la minimisation du temps d'exécution moyen et l'adéquation entre les besoins du programme et le temps processeur alloué.

1.4 Mémoire

La mémoire stocke les données de travail des programmes et est organisée en une hiérarchie :

- **Caches** (e.g. L1, L2) : Intermédiaires entre les registres et la mémoire principale, stockant temporairement les données fréquemment accédées (principe de localité).
- **Mémoire principale** (e.g. RAM, GPU) : Stocke les données de travail (e.g. tableaux, variables, stack) lorsqu'il n'y a pas de registres disponibles.
- **Mémoire secondaire** (e.g. HDD, SSD) : Utilisée comme espace temporaire (swap) lorsque la mémoire principale est pleine.

L'allocation dynamique permet de gérer la mémoire de manière flexible. La mémoire virtuelle charge/décharge les données non actives pour pallier les limitations de la mémoire physique (e.g. adressage 32 bits, 0x00000000 à 0xffffffff).

1.4.1 Protection de la mémoire

Le système d'exploitation contrôle les accès mémoire pour garantir qu'un programme n'accède qu'à sa propre mémoire, évitant ainsi la lecture ou la modification de données sensibles (e.g. clefs de chiffrement). Le processeur vérifie les accès mémoire selon les descriptions des espaces mémoire accessibles à chaque programme, mises à jour par le système d'exploitation.

- **Adresse valide** : Accessible au programme, l'accès est accordé (e.g. `lw`, `sw`).
- **Adresse indisponible** : Accessible mais non présente en mémoire principale, nécessite un chargement depuis la mémoire secondaire.
- **Adresse invalide** : Hors de l'espace accessible, entraîne l'arrêt du programme.

1.4.2 Répartition de la mémoire

La seconde responsabilité de gestion est la répartition de la mémoire principale entre les différents programmes. Cela inclut la représentation et le suivi des portions de mémoire (libres ou non) et l'identification d'une portion adéquate pour satisfaire une demande d'allocation dynamique.

1.5 Stockage

Le stockage préserve les données produites par un programme au-delà de son exécution. Une fois le programme terminé, sa mémoire est libérée et peut être allouée à un autre programme. La notion de fichier assure la pérennité des données, malgré la diversité des supports physiques adaptés à différents scénarios d'utilisation :

- **HDD** : Disque dur, bon marché, accès lent, organisé en plateaux, cylindres, têtes, et secteurs.
- **SSD** : Disque à état solide, rapide, coût plus élevé, organisé en blocs et pages.
- **Bande magnétique** : Grande capacité, fiable, utilisé pour les backups.

Un système de fichiers met en place une abstraction pour accéder au stockage, en représentant les fichiers et répertoires et leurs méta-données (e.g. taille, dates, permissions). Il garantit également la cohérence des écritures et réduit la possibilité de corruption des données.

1.6 Typologie des systèmes

Les types de systèmes influencent la conception des systèmes d'exploitation :

- **Mainframe** : Exécute des jobs réguliers avec des E/S intensives (e.g. calcul des fiches de paie). Supporte un grand volume de transferts et offre une gestion de timesharing pour plusieurs utilisateurs.
- **Serveur** : Fournit des services aux utilisateurs distants (e.g. serveur d'impression), gère les ressources et la facturation.
- **Personal Computer** : Interaction humaine via interface graphique, exécution simultanée de plusieurs programmes, interactivité centrale (mesurée par la latence entre l'action et la réaction).
- **Système mobile** (e.g. tablette, smartphone) : Contraintes de ressources, économie d'énergie cruciale, processeur et mémoire limités, alimentation par batterie.
- **Système embarqué** (e.g. TV, voiture) : Logiciel prédéterminé pour une machine spécifique, pas de flexibilité pour l'installation d'applications supplémentaires.
- **Système de senseurs** : Surveillance des paramètres environnementaux (e.g. température, humidité), transmet les données à un central, optimisé pour faible consommation d'énergie.
- **Système temps-réel** : Respect des échéances crucial, priorisation des tâches, gestion précise du temps pour aligner les traitements avec l'environnement.

1.7 Architecture

Un système d'exploitation inclut le code, les données, et la stack, similaires à un programme classique. Il utilise des instructions système nécessitant des niveaux de privilège spécifiques, et le noyau gère les ressources.

- **Noyau monolithique** : Code dans un fichier unique, exécuté au plus haut niveau de privilège. Exemple : système avec configuration matérielle fixe.
- **Noyau modulaire** : Base toujours en mémoire, modules chargés/déchargés selon les besoins. Exemple : Linux.
- **Microkernel** : Composantes exécutées à des niveaux de privilège réduits pour éviter les compromissions. Un serveur de ré-incarnation gère les erreurs fatales. Exemple : MINIX3.
- **Exokernel** : Gestion des ressources laissée aux programmes, le noyau assure uniquement la protection et l'isolation. Exemple : machines virtuelles.

2 Événements

2.1 Composition et exécution de programme

Un programme est composé de trois parties principales :

- **Code machine** : Instructions représentant la logique des traitements.
- **Espace de données** : Contient les données manipulées par le programme (variables globales, tableaux, variables dynamiques obtenues via `malloc`).
- **Stack** : Héberge les données de travail courantes (variables locales) et les informations d'appels de fonctions imbriquées.

Ces composantes sont liées par des adresses, utilisées pour les branchements, appels et retours de fonction, accès aux cases d'un tableau, et manipulations de la stack. Pour s'exécuter, elles doivent être placées en mémoire principale.

Le processeur suit un cycle *fetch-decode-execute* :

1. **Fetch** : Récupération de l'instruction à exécuter depuis la mémoire.
2. **Decode** : Décodage des détails de l'instruction pour déterminer le traitement.
3. **Execute** : Exécution des opérations requises par l'instruction
 - (a) Mise à jour des registres, de la stack, des données.
 - (b) Mise à jour (incrément) du Pointeur d'Instruction

L'*instruction pointer* (ou *program counter*) contient l'adresse de l'instruction courante et est mis à jour après chaque instruction (soit par incrémentation, soit par branchement). L'état du programme à un moment donné est déterminé par :

- Les valeurs des registres
- Le contenu de la stack
- Le contenu de l'espace de données
- L'*instruction pointer*

Une sauvegarde de ces éléments représente un instantané de l'état du programme, permettant de reprendre l'exécution là où elle s'est arrêtée.

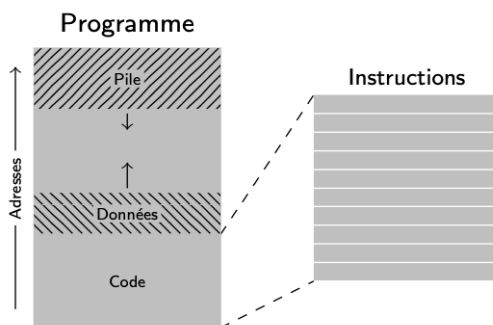


Figure 2: Program view

2.2 Concept de base – évènement

Lorsqu'un programme est exécuté par le processeur, deux questions se posent :

- **Erreur d'instruction** : Doit-on redémarrer la machine ou gérer l'anomalie pour limiter les dégâts ?
- **Activité de périphérique externe** (e.g. frappe au clavier) : Peut-on suspendre temporairement le programme en cours pour traiter l'activité ?

Ces situations sont résolues par la gestion des **évènements**, mécanismes permettant au processeur de brancher vers du code spécifique en réponse à un évènement.

Les exceptions et interruptions sont des mécanismes implémentés par le processeur permettant d'effectuer un branchement en réaction à un évènement.

Les différents types d'évènements auxquels un OS peut être confronté sont définis par l'architecture du processeur qui les exécute.

Le système d'exploitation dispose d'un gestionnaire d'évènements qui prend le relais en cas d'interruption ou d'exception lors de l'exécution d'un programme.

2.3 Privilèges

La vérification du niveau de privilèges est nécessaire pour :

1. Modifier la configuration des interruptions/exceptions
2. Modifier la configuration de la mémoire
3. Modifier le niveau de privilège courant

Dans une architecture x86, il ya 4 anneaux de protection

1. Ring 0: Noyau du os
2. Ring 1 & 2 : Services de l'OS
3. Ring 3 : Programmes

Dans une architecture MIPS, il y a 2 modes de fonctionnement:

- Kernel Mode : os
- User Mode : programmes

2.4 Typologie

2.4.1 Exceptions (évènements synchrones)

Origine interne, résultant de l'exécution d'une instruction problématique :

- **Code opératoire invalide** : Opcode non reconnu par le processeur.
- **Division par zéro** : Nécessité d'un dénominateur non-nul.
- **Instruction privilégiée** : Niveau de privilège insuffisant pour exécuter l'instruction.
- **Instruction non-chargée en mémoire** : Le système d'exploitation charge la partie manquante en mémoire.

2.4.2 Interruptions (événements asynchrones)

Origine externe, provoquées par un signal de périphérique :

- **Frappe de clavier** : Fermeture d'un circuit électronique, générant une interruption.
- **Fichier prêt pour lecture** : Le contrôleur de disque génère une interruption après lecture des données.
- **Réception d'un paquet réseau** : La carte réseau génère une interruption après réception complète d'un paquet.
- **Erreur du matériel** : Anomalies détectées (e.g. température excessive).
- **Signal d'horloge** : Interruption générée 32.768 fois par seconde pour suivre le temps.

2.4.3 Traitements

Multiplicité des événements (interruptions) => priorité Evenements pendant le traitement d'événements (exceptions) => Réantrance et préemption

2.5 Conséquences possibles pour le programme

Le système d'exploitation doit gérer les événements survenus pendant l'exécution d'un programme. Selon la nature de l'événement, deux décisions peuvent être prises :

- **Évènement récupérable** : Après traitement, il est possible de continuer l'exécution du programme.
- **Évènement irrécupérable** : La faute ne peut être corrigée, et il est nécessaire de terminer le programme.

2.5.1 Evenement récupérable

Lorsqu'un événement récupérable survient, il est crucial de préserver l'état du processeur. Le gestionnaire d'événements doit sauvegarder le contexte du programme suspendu (registres et stack) avant de traiter l'évènement. Cette sauvegarde est similaire à celle d'un appel de fonction.

1. Sauvegarde
2. Traitement
 - Reprise
 - (a) Identifier
 - (b) Charger
 - Poursuite
 - (a) Récupérer le code
 - (b) Notifier
 - (c) Transfert du code vers le programme
3. Restauration
4. La continuation peut être :
 - **Reprise** : Retenter l'exécution de l'instruction qui a causé l'évènement. A l'adresse de l'instruction corrélée à cet événement.
 - **Poursuite** : Passer à l'instruction suivante. A l'adresse de l'instruction suivante.

Les interruptions sont généralement des événements récupérables avec poursuite, car elles n'affectent pas l'instruction en cours. Les exceptions comme les appels systèmes (e.g. `syscall`, `int`) sont également des événements récupérables avec poursuite.

Par exemple, une interruption causée par une frappe au clavier est traitée en lisant le code de la touche depuis un port d'entrée/sortie, le traduisant en caractère, puis le transférant au programme destinataire, souvent la fenêtre active dans une interface graphique.

2.5.2 Evenement irrécupérable

Lorsqu'un évènement irrécupérable survient, le programme ne peut plus continuer. Une sauvegarde complète du contexte d'exécution est effectuée pour le débogage.

Le programme acquiert diverses ressources (mémoire, descripteurs de fichiers, sockets) au cours de son exécution. Ces ressources doivent être libérées lorsque le programme est terminé.

1. Sauvegarde
2. Libération des ressources
3. Traitement Les étapes de gestion d'un évènement irrécupérable incluent :
 - Affichage d'une notification à l'utilisateur décrivant le problème rencontré.
 - Sauvegarde du contexte dans un fichier (e.g. `coredump`) ou envoi par mail à un serveur de rapports de crash.
 - Libération du processeur, permettant la continuation d'un autre programme via un *context switch*.
4. Abandon

2.6 Mécanismes de déroutement

Chaque architecture de processeur spécifie et implémente son mécanisme de déroutement pour la gestion des évènements. Deux techniques principales sont utilisées :

- **Déroutement avec registre de cause** : Un registre indique la cause de l'évènement, permettant au gestionnaire d'évènements de déterminer l'action appropriée.
- **Vectorisation** : Différents types d'évènements sont associés à des adresses spécifiques dans une table de vecteurs, chaque adresse pointant vers le code de gestion correspondant.

La technique utilisée influence l'organisation et les responsabilités du code du gestionnaire d'évènements.

2.6.1 Registre de cause

Le registre de cause encode les évènements survenus lors du cycle d'exécution courant, mettant à jour les bits pour les exceptions et interruptions. Si au moins un bit est positionné, le processeur réalise un déroutement vers une adresse précise.

Le gestionnaire d'évènements doit :

- Lire le contenu du registre de cause.
- Tester et traiter les bits des interruptions en attente (bits à 1).
- Traiter les exceptions stipulées par le code d'exception.

Cette technique permet au système d'exploitation de déterminer l'ordre de priorité des traitements des différents évènements. La fonction principale du gestionnaire d'évènements doit être placée à l'adresse de déroutement définie par le processeur au démarrage du système.

Pour permettre la continuation du programme, l'adresse de l'indicateur d'instruction courante est sauvegardée dans un registre dédié (e.g. Exception Program Counter en MIPS32).

2.6.2 Vectorisation

Dans la vectorisation, chaque type d'évènement est associé à un numéro unique (e.g. interrupt vector). Ce numéro identifie une entrée dans une table de descripteurs (e.g. interrupt descriptor table, IDT) qui référence la fonction de traitement spécifique à invoquer (interrupt service routine, ISR).

À la fin d'un cycle d'exécution, le processeur :

- Consulte les numéros d'événements survenus.
- Invoque les fonctions de gestion des événements correspondantes dans un ordre déterminé.

Le code du gestionnaire d'événements se concentre uniquement sur l'événement correspondant. Au démarrage du système, l'OS initialise le contenu de la table des descripteurs et la référence au processeur via une instruction privilégiée (e.g. `lidt` en x86).

Lorsqu'un événement survient :

- Le numéro d'événement sert d'indice dans l'IDT.
- Le descripteur correspondant (interrupt gate) contient l'adresse de la fonction de gestion vers laquelle brancher (offset).

2.6.3 Niveaux de privilège

Le jeu d'instructions d'un processeur inclut toutes les instructions reconnaissables par son architecture, comprenant les instructions arithmétiques, logiques et systèmes. Les instructions systèmes modifient la configuration du processeur et doivent être exécutées uniquement par le système d'exploitation.

Un niveau de privilège contrôle quelles instructions peuvent être exécutées et quand. La spécification du processeur décrit la dynamique de changement de ce niveau de privilège et son utilisation pour les vérifications.

2.6.4 Basculement de stack

Lorsqu'un événement survient, le processeur déroute vers une fonction du système d'exploitation, modifiant le contenu des registres et utilisant une stack pour stocker les valeurs temporaires. Il est crucial de décider si cette stack sera celle du programme suspendu ou une stack dédiée du système d'exploitation.

Le processeur peut basculer vers une stack du système d'exploitation, en sauvegardant les informations nécessaires pour explorer la stack du programme suspendu (e.g. stack pointer). Par exemple, lors d'un appel système, les arguments peuvent être passés via la stack, et le système d'exploitation doit pouvoir les localiser.

Après le traitement de l'événement, la stack courante redevient celle du programme. La figure 1.11 illustre ce processus :

- Avant le déroutement : `sp(pgm)`
- Après le déroutement : `sp(os)`, contenant :
 - Sauvegarde du pointeur de stack précédent `sp(pgm)`
 - Indicateur d'instruction courante `ip(pgm)`
 - Éventuel code d'erreur décrivant l'événement

Ces informations permettent de revenir au programme pour une éventuelle continuation.

2.7 Multiplicité des occurrences

Il est possible que plusieurs événements se produisent pendant un même cycle d'exécution du processeur. Les règles d'agréations entre événements sont les suivantes :

- **Reprise** l'emporte sur une **poursuite**.
- **Abandon** l'emporte sur une **reprise**.

Lors de deux événements similaires (deux poursuites, deux reprises, ou deux abandons), la conséquence est respectivement une poursuite, une reprise, ou un abandon.

Pour des événements de types différents :

- **Reprise et poursuite** : La reprise l'emporte, les deux traitements doivent être réalisés. Si l'évènement avec poursuite est une interruption, il peut être traité en priorité.
- **Poursuite et abandon** : L'abandon l'emporte. Si l'évènement avec poursuite est une exception, son traitement n'est pas nécessaire. Si c'est une interruption, il doit être traité.
- **Reprise et abandon** : L'abandon l'emporte. Le traitement de l'évènement avec reprise n'est pas nécessaire car le programme ne continuera plus.

2.8 Gestion ré-entrante des événements

Un événement peut survenir pendant n'importe quel cycle d'exécution, y compris pendant la gestion d'un autre événement (exception ou interruption). Le gestionnaire d'événements doit être adapté pour gérer cette imbrication correctement.

Un gestionnaire ré-entrant permet de gérer de nouveaux événements survenant pendant l'exécution du gestionnaire actuel. Les considérations sont similaires à la gestion des appels de fonctions récursives.

Pour garantir une sauvegarde cohérente des contextes imbriqués :

- Utiliser la stack pour sauvegarder le contexte du gestionnaire d'événements suspendu.
- Désactiver les interruptions pendant les phases critiques (sauvegarde du contexte, restauration du contexte, continuation), mettant ainsi en attente les interruptions.
- Préserver l'adresse de l'instruction corrélée à chaque niveau d'imbrication.

2.9 Priorités et masques d'interruptions

Dans un système typique, il est crucial de respecter l'ordre de priorité des interruptions. Un gestionnaire ré-entrant seul ne suffit pas pour éviter l'inversion de priorité, où une interruption de basse priorité pourrait interrompre le traitement d'une interruption de haute priorité.

La solution consiste à utiliser un masque d'interruptions, permettant une granularité dans l'activation et la désactivation des interruptions. Ce masque contrôle quelles interruptions sont actives (ou inactives) en utilisant un bit par type d'interruption. Le masque est combiné avec les bits des interruptions en attente à la fin de chaque cycle d'exécution.

Lors du traitement d'une interruption, le gestionnaire :

- Positionne les bits du masque pour autoriser uniquement les interruptions plus prioritaires.
- Après traitement, restaure l'état du masque à son état précédent.

3 Mémoire

3.1 Projection et protection

3.1.1 Adressage absolu

3.1.2 Adressage relatif

3.1.3 Segmentation

3.1.4 Pagination

3.2 Gestion de la mémoire physique

3.2.1 Bitmap

3.2.2 Listes chaînées

4 Processus

4.1 Constitution

4.1.1 Table des processus

4.1.2 Cycle de vie

4.2 Ordonnancement

4.2.1 Systèmes batch

4.2.2 Systèmes interactifs

4.2.3 Systèmes en temps réel

4.3 Multi-threading

5 Concurrency

6 Systèmes de fichiers

7 Entrées et sorties

Figures

1	View	4
2	Program view	7

Tableaux

Algorithmes

