FSGA
Faculté des Sciences,
de Génie et d'Architecture

1/30/2026

Rapport Technique

Conception et Implémentation d'un Mini-OS (LearnixMiniOS)

Université Quisqueya

Cours : Noyau des Systèmes d'Exploitation (INF412)

Professeur : Pr. Jean Andris ADAM

Karlsen PAUL - PA190102
UNIVERSITÉ QUISQUEYA – FSGA (GÉNIE INFORMATIQUE)

REMERCIEMENTS

Je tiens à exprimer ma sincère gratitude envers toutes les personnes qui ont contribué à la réalisation de ce projet.

Mes remerciements s'adressent tout d'abord au Professeur Jean Andris ADAM pour son enseignement du cours Noyau des Systèmes d'Exploitation, sa bienveillance, ses explications sans oublier ses conseils précieux qui m'ont permises d'être à la hauteur pour réaliser ce projet.

Je remercie également la Faculté des Sciences de Génie et d'Architecture de l'Université Quisqueya d'avoir engagé ce genre de professeur qualifié pour dispenser un cours aussi important pour les futurs ingénieurs de l'UniQ.

Enfin, je remercie mes collègues du cours pour les échanges constructifs et le soutien mutuel durant la réalisation de ce travail.

RÉSUMÉ

Ce rapport présente la conception et l'implémentation de LearnixMiniOS, un système d'exploitation éducatif simplifié développé dans le cadre du cours Noyau des Systèmes d'Exploitation (INF412).

L'objectif principal de ce projet est de mettre en pratique les concepts fondamentaux des systèmes d'exploitation : gestion des processus, ordonnancement CPU, gestion de la mémoire et interaction avec le matériel.

LearnixMiniOS est un noyau monolithique 32 bits qui démarre depuis un bootloader personnalisé écrit en assembleur x86, effectue la transition du mode réel vers le mode protégé, et offre un environnement interactif via un Shell de commandes.

Les principales fonctionnalités implémentées comprennent : la gestion des processus avec une structure PCB (Process Control Block) supportant quatre états (READY, RUNNING, BLOCKED, TERMINATED), deux algorithmes d'ordonnancement (FCFS et Round Robin avec préemption), un timer système pour le suivi temporel, un journal d'exécution des événements, une gestion mémoire simplifiée, et un Shell interactif avec plus d'une douzaine de commandes.

Le développement a été réalisé en langage C et assembleur x86, compilé avec un cross-compiler i686-elf-gcc, et testé sur l'émulateur QEMU.

Les résultats obtenus démontrent la faisabilité d'implémenter un noyau fonctionnel capable d'illustrer les mécanismes fondamentaux d'un système d'exploitation moderne.

Table des matières

1 INTRODUCTION

1.1 Contexte

Dans le domaine de l'informatique, le système d'exploitation constitue la couche logicielle fondamentale qui fait le lien entre le matériel et les applications utilisateur. La compréhension de son fonctionnement interne est essentielle pour tout ingénieur en informatique.

Le cours de Noyau des Systèmes d'Exploitation vise à approfondir cette compréhension à travers l'étude théorique et pratique des mécanismes internes d'un OS : gestion des processus, ordonnancement, gestion de la mémoire et interaction avec le matériel.

Ce projet s'inscrit dans cette démarche pédagogique en proposant la conception et l'implémentation d'un système d'exploitation simplifié nommé LearnixMiniOS.

1.2 Problématique

La programmation système bare-metal (sans système d'exploitation sous-jacent) présente des défis particuliers :

1. Comment démarrer un ordinateur et initialiser le processeur ?
2. Comment passer du mode réel 16 bits au mode protégé 32 bits ?
3. Comment gérer plusieurs processus avec des ressources limitées ?
4. Comment implémenter un ordonnancement équitable du CPU ?
5. Comment interagir avec le matériel (écran, clavier) sans pilotes ?

Ces questions constituent le cœur de la problématique abordée dans ce projet.

1.3 Objectifs

L'objectif général est de concevoir et implémenter un mini système d'exploitation fonctionnel démontrant les concepts fondamentaux vu dans le cours.

De cet objectif général, plusieurs objectifs spécifiques en découlent comme :

1. Développer un bootloader capable de charger le noyau en mémoire
2. Implémenter la transition vers le mode protégé 32 bits
3. Créer une structure de gestion des processus (PCB)
4. Implémenter les algorithmes FCFS et Round Robin
5. Développer un timer système et un journal d'exécution
6. Concevoir un Shell interactif pour l'utilisateur

1.4 Méthodologie

1. Le développement suit une approche incrémentale :
2. Étude de la documentation technique x86 et des ressources OSDev
3. Implémentation du bootloader en assembleur NASM
4. Développement du noyau en C avec cross-compiler
5. Tests itératifs sur l'émulateur QEMU
6. Débogage avec les outils GDB et les logs QEMU
7. Documentation et rédaction du rapport

Environnement de développement :

Table 1- Environnement de développement

Composant	Description
Système hôte	Ubuntu Linux / WSL2
Compilateur C	i686-elf-gcc (cross-compiler)
Assembleur	NASM (Netwide Assembler)
Éditeur de liens	i686-elf-ld
Emulateur	QEMU (qemu-system-i386)
Contrôle version	Git / GitHub
Automatisation	GNU Make

1.5 Plan du rapport

Ce rapport est organisé comme suit :

1. Chapitre 2 présente l'état de l'art sur les systèmes d'exploitation et les algorithmes d'ordonnancement.
2. Chapitre 3 détaille la conception et l'architecture globale de LearnixMiniOS.
3. Chapitre 4 décrit l'implémentation technique des différents composants du système.
4. Chapitre 5 présente les résultats obtenus et leur analyse.
5. Chapitre 6 discute des difficultés rencontrées et des solutions apportées.
6. Chapitre 7 conclut le rapport et propose des perspectives d'amélioration.

2 ÉTAT DE L'ART

2.1 Systèmes d'exploitation existants

Un système d'exploitation (OS) est un ensemble de programmes qui gère les ressources matérielles d'un ordinateur et fournit des services aux applications. Les OS modernes comme

Linux, Windows et MacOs sont des systèmes complexes comprenant des millions de lignes de code.

Dans le domaine éducatif, plusieurs projets de mini-OS existent :

- MINIX : Créé par Andrew Tanenbaum à des fins pédagogiques, il a inspiré le développement de Linux.
- xv6 : Développé au MIT, c'est une réimplémentation de Unix v6 pour l'architecture x86, utilisée dans les cours d'OS.
- OS/161 : Utilisé à Harvard, il fournit un cadre pour l'apprentissage des concepts d'OS.

LearnixMiniOS s'inscrit dans cette tradition de projets éducatifs, avec un focus sur la simplicité et la clarté du code.

2.2 Concepts fondamentaux

2.2.1 Le processus

Un processus est un programme en cours d'exécution. Il est caractérisé par :

- Un espace d'adressage (code, données, pile)
- Un état (prêt, en exécution, bloqué, terminé)
- Un contexte (registres, compteur programme)

Le Process Control Block (PCB) est la structure de données qui contient toutes les informations relatives à un processus.

2.2.2 L'ordonnancement

L'ordonnanceur (Scheduler) est le composant de l'OS qui décide quel processus s'exécute à chaque instant. Les critères d'évaluation sont :

- Utilisation CPU : Pourcentage de temps où le CPU est actif
- Débit : Nombre de processus terminés par unité de temps
- Temps de rotation : Temps total entre soumission et terminaison
- Temps d'attente : Temps passé dans la file d'attente
- Temps de réponse : Temps avant la première exécution

2.2.3 Modes du processeur x86

L'architecture x86 définit plusieurs modes de fonctionnement :

- Mode réel (16 bits) : Mode de démarrage, accès direct à 1 Mo

- Mode protégé (32 bits) : Segmentation, protection mémoire
- Mode long (64 bits) : Extension 64 bits (non utilisé ici)

2.3 Algorithmes d'ordonnancement

Table 2- Algorithmes d'ordonnancement

Algorithme	Principe	Avantages	Inconvénients
2.3.1 FCFS (First Come, First Served)	Les processus sont exécutés dans leur ordre d'arrivée	▪ Simple à implémenter ▪ Pas de famine (tout processus finit par s'exécuter)	▪ Effet convoi (processus courts bloqués par processus longs) ▪ Temps d'attente moyen potentiellement élevé
2.3.2 Round Roubin	Chaque processus reçoit un quantum de temps fixe. Après expiration, il est préempté et placé en fin de file.	▪ Temps de réponse garanti ▪ Équitable en termes de temps CPU ▪ Adapté aux systèmes interactifs	▪ Overhead des changements de contexte ▪ Choix du quantum critique (trop petit = overhead, trop grand = FCFS)

2.3.3 Autres algorithmes (non implémentés)

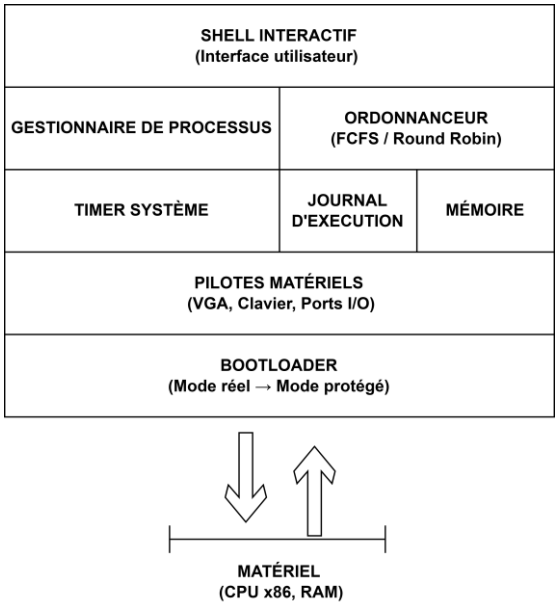
- SJF (Shortest Job First) : Exécute le processus le plus court
- SRTF (Shortest Remaining Time First) : Version préemptive de SJF
- Priorité : Chaque processus a une priorité assignée
- MLFQ (Multi-Level Feedback Queue) : Plusieurs files avec priorités

3 CONCEPTION ET ARCHITECTURE

3.1 Architecture globale

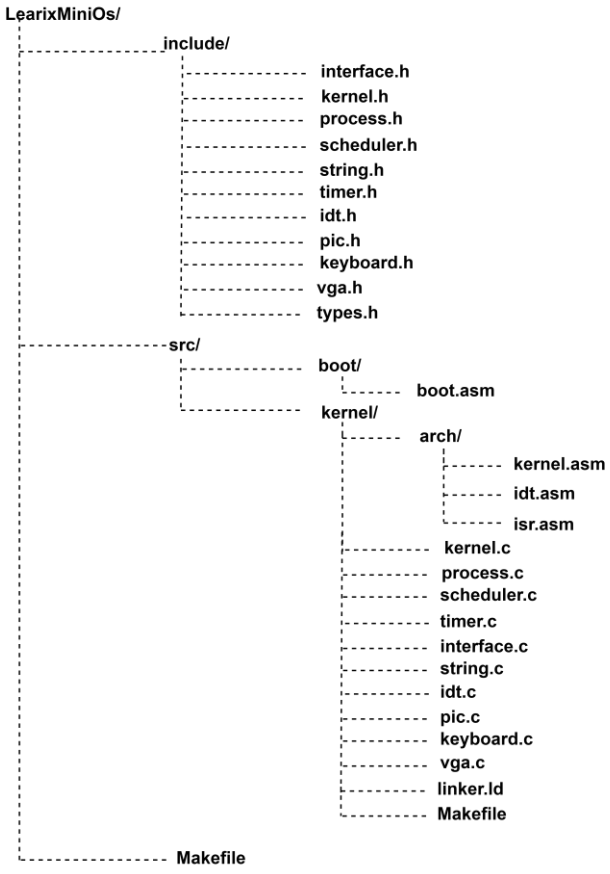
LearnixMiniOS adopte une architecture monolithique où tous les composants s'exécutent dans le même espace d'adressage en mode noyau.

Figure 1- Architecture en couche



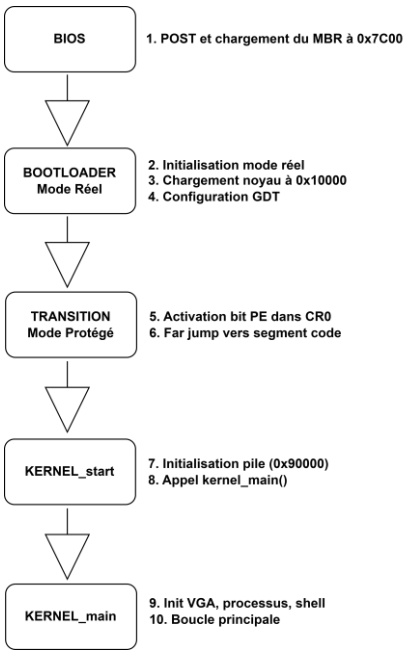
3.2 Organisation des fichiers

Figure 2- Arborescence du projet



3.3 Flux de démarrage

Figure 3- Flux de démarrage



4 IMPLÉMENTATION

4.1 Le Bootloader

Le bootloader est le premier code exécuté. Il occupe exactement 512 octets (un secteur) et se termine par la signature 0xAA55.

Responsabilités :

1. Initialiser les registres de segment (DS, ES, SS = 0)
2. Configurer la pile à 0x7C00
3. Afficher un message via INT 10h
4. Charger le noyau depuis le disque via INT 13h
5. Préparer la GDT
6. Effectuer la transition vers le mode protégé

4.2 Passage en mode protégé

La Global Descriptor Table (GDT) définit les segments mémoire :

Table 3- Structure de la GDT

GLOBAL DESCRIPTOR TABLE (GDT)
Entrée 0 : Descripteur NULL (obligatoire) ▪ 8 octets de zéros
Entrée 1 : Segment de CODE (sélecteur 0x08) ▪ Base : 0x00000000

- Limite : 0xFFFFF (4 Go avec granularité 4K)
- Type : Code, exécutable, lisible
- DPL : 0 (niveau noyau)

Entrée 2 : Segment de DATA (sélecteur 0x10)

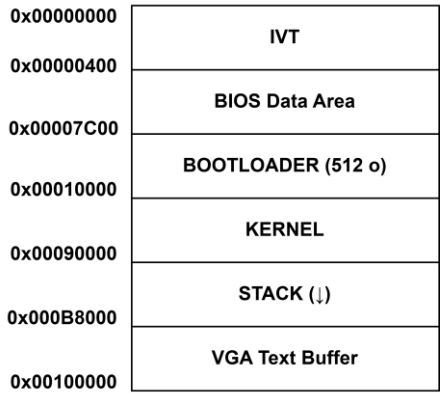
- Base : 0x00000000
- Limite : 0xFFFFF (4 Go avec granularité 4K)
- Type : Données, lecture/écriture
- DPL : 0 (niveau noyau)

Séquence de transition :

1. CLI (désactiver interruptions)
2. LGDT [gdt_descriptor]
3. MOV EAX, CR0 / OR EAX, 1 / MOV CR0, EAX
4. JMP 0x08:init_pm (far jump)
5. Charger sélecteurs de données (DS, ES, SS = 0x10)

Carte mémoire :

Figure 4- Carte Mémoire



4.3 Gestion des processus

4.3.1 Structure PCB

Table 4- Structure du PCB

PROCESS CONTROL BLOCK (PCB)		
Champ	**Type**	**Description**
pid	uint32_t	Identifiant unique
name	char[12]	Nom du processus
state	enum	READY, RUNNING, BLOCKED, TERMINATED
burst_time	uint32_t	Temps d'exécution total
remaining_time	uint32_t	Temps restant

wait_time	uint32_t	Temps d'attente cumulé

Le système supporte 8 processus maximum dans un tableau statique.

4.3.2 États et transitions

Figure 5- Diagramme d'états

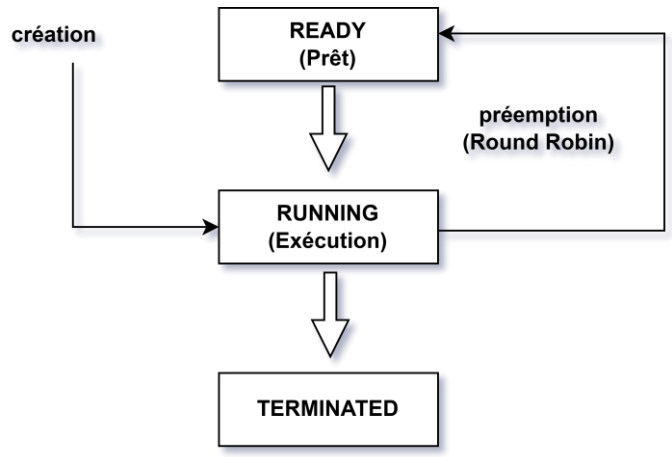


Table 5- Transitions d'états

Transition	Déclencheur	Description
→ READY	create_proc()	Création d'un processus
READY → RUNNING	sched_step()	Élection par l'ordonnanceur
RUNNING → READY	quantum atteint	Préemption Round Robin
RUNNING → TERMINATED	remaining = 0	Fin d'exécution
* → TERMINATED	kill_proc()	Terminaison manuelle

4.4 Ordonnancement

4.4.1 Algorithme FCFS (First Come, First Served)

Les processus s'exécutent dans l'ordre d'arrivée jusqu'à terminaison. Aucune préemption n'est effectuée.

4.4.2 Algorithme Round Robin

Chaque processus reçoit un quantum de 2 ticks. L'implémentation utilise une file circulaire avec un pointeur last_proc pour éviter la ré-élection immédiate du même processus après préemption.

Algorithme de sélection :

start = (last_proc + 1) % nombre_processus

Pour i de 0 à nombre_processus :

idx = (start + i) % nombre_processus

Si processus[idx].état == READY :

Élire processus[idx]

Sortir

4.5 Fonctionnalités systèmes

4.5.1 Timer système

Compteur de ticks incrémenté à chaque pas d'ordonnancement. Utilisé pour horodater les événements et calculer les métriques.

4.5.2 Gestion mémoire

Allocation par blocs fixes :

- Mémoire totale : 1024 Ko (simulée)
- Mémoire noyau : 128 Ko (réservée)
- Par processus : 24 Ko

4.5.3 Journal d'exécution

Buffer circulaire de 15 entrées enregistrant les événements :

- Création/terminaison de processus
- Préemptions
- Changements de Scheduler

4.5.4 Shell interactif

Table 6- Commandes du Shell

Commande	Description et utilisation
help	Affiche la liste des commandes disponibles
clear	Efface le contenu de l'écran
about	Affiche les informations sur le projet
ps	Affiche la liste des processus avec leurs états
run X [N]	Crée processus X avec burst N (N est optionnel, défaut = 5) Exemple : run test 8
kill	Termine le processus ayant le PID N Exemple : kill 3
fcfs	Active l'algorithme d'ordonnancement FCFS
rr	Active l'algorithme d'ordonnancement Round Robin
demo	Lance une démonstration automatique du scheduler

step	Exécute un pas d'ordonnancement
time	Affiche le temps système en ticks
mem	Affiche l'état mémoire
log	Affiche le journal d'exécution
exit	Arrête proprement le système

5 RÉSULTATS ET ANALYSE

5.1 Tests fonctionnels

- ✓ Le système a été testé avec succès pour les scénarios suivants :
- ✓ Démarrage complet depuis le bootloader
- ✓ Affichage du Shell et saisie clavier
- ✓ Création et terminaison de processus
- ✓ Exécution des deux algorithmes d'ordonnancement
- ✓ Affichage correct des états et métriques
- ✓ Fonctionnement du journal d'exécution
- ✓ Arrêt propre du système

5.2 Comparaison FCFS vs Round Robin

Test avec 3 processus : P1 (burst=3), P2 (burst=4), P3 (burst=2)

Figure 6- Timeline FCFS

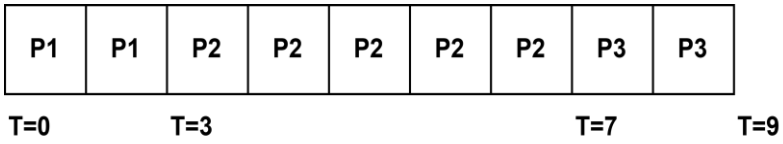


Figure 7- Timeline Round Robin (quantum=2)

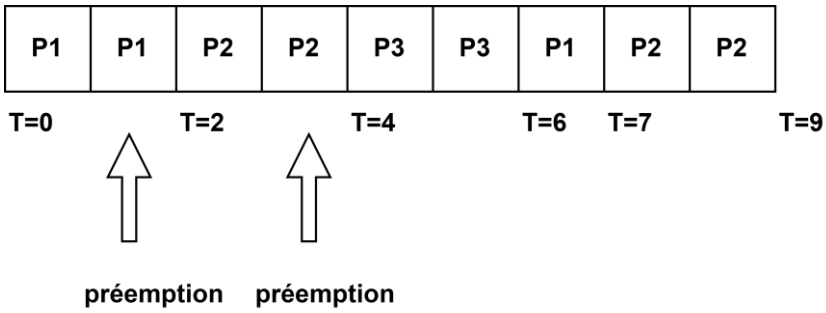


Table 7- Analyse comparative

Critère	FCFS	Round Robin
Temps total	9 ticks	9 ticks
Temps attente P1 **Temps attente P2** **Temps attente P3**	0 3 7	4 3 3
Moyenne attente	3.33 ticks	3.33 ticks
Temps réponse P1 **Temps réponse P2** **Temps réponse P3**	0 3 7	
Moyenne réponse	3.33 ticks	2.00 ticks
Préemptions	0	3

5.3 Analyse des performances

Observations :

1. Le temps total d'exécution est identique (9 ticks) car la charge de travail est la même.
2. Round Robin offre un meilleur temps de réponse moyen (2.00 vs 3.33 ticks), important pour l'interactivité.
3. FCFS n'a aucun overhead de préemption, idéal pour le batch.
4. Avec FCFS, P3 attend 7 ticks avant de commencer (effet convoi). Avec Round Robin, P3 commence à T=4.

Conclusion : Round Robin est plus adapté aux systèmes interactifs où le temps de réponse est critique. FCFS convient aux traitements batch où les processus sont similaires en durée.

6 DISCUSSION

6.1 Difficultés rencontrées

Au cours du développement de LearnixMiniOS, plusieurs défis techniques ont été rencontrés :

1. Transition mode réel → mode protégé

   La configuration de la GDT et le far jump initial ont nécessité une compréhension approfondie de l'architecture x86 et plusieurs sessions de débogage avec QEMU.
2. Chargement du noyau

   Le positionnement du noyau à l'adresse 0x10000 et la lecture des secteurs via INT 13h ont demandé plusieurs itérations pour fonctionner correctement.
3. Bug de réélection Round Robin

L'implémentation initiale causait la réélection immédiate du même processus après préemption, car la recherche recommençait toujours à l'index 0.

4. Absence de librairie standard

   Toutes les fonctions utilitaires (strlen, strcmp, itoa, affichage) ont dû être ré implémentées sans aucune bibliothèque externe.

6.2 Solutions apportées

1. Pour la transition mode protégé :

   Étude approfondie du manuel Intel et des ressources OSDev Wiki.

   Tests incrémentaux avec affichage de caractères de debug.

2. Pour le chargement du noyau :

   Utilisation du numéro de lecteur passé par le BIOS dans DL.

   Vérification de la taille du noyau avec hexdump.

3. Pour le bug Round Robin :

   Implémentation d'un pointeur last_proc qui mémorise le dernier processus exécuté. La recherche commence à (last_proc + 1) % n, garantissant une rotation équitable.

4. Pour les fonctions utilitaires :

   Implémentation minimale de chaque fonction nécessaire, en privilégiant la simplicité et la clarté du code.

7 CONCLUSION ET PERSPECTIVES

7.1 Bilan du travail

LearnixMiniOS implémente toutes les fonctionnalités obligatoires mentionnés dans le cahier des charges du document de projet final du cours Noyau des Systèmes d'exploitation (INF412) :

- ✓ Démarrage du système via bootloader personnalisé
- ✓ Passage en mode protégé 32 bits
- ✓ Gestion des processus avec PCB et 4 états
- ✓ Ordonnancement FCFS (non préemptif)
- ✓ Ordonnancement Round Robin (préemptif, quantum = 2)
- ✓ Timer système pour le suivi temporel
- ✓ Journal d'exécution des événements
- ✓ Shell interactif avec 14 commandes
- ✓ Gestion mémoire simplifiée

Ce projet a permis d'acquérir une compréhension pratique et approfondie des mécanismes internes d'un système d'exploitation, au-delà de la théorie vue en cours.

7.2 Limites

Le système présente certaines limitations inhérentes à sa nature éducative :

- Timer logique : Le temps n'avance que lors des appels à sched_step(), pas en temps réel.
- Pas de vraies interruptions : Le clavier utilise le polling, pas les IRQ.
- Mémoire simulée : Pas de vraie allocation dynamique ni de pagination.
- Processus simulés : Pas de vrai changement de contexte avec sauvegarde des registres.
- Mono-utilisateur : Pas de gestion des droits ou de protection.

7.3 Améliorations futures

Plusieurs extensions sont envisageables :

1. Interruptions matérielles
   Implémenter l'IDT et les IRQ pour un timer réel (PIT) et une gestion clavier par interruption.
2. Système de fichiers
   Ajouter un système de fichiers simple (FAT12 ou custom) pour la persistance des données.
3. Multitâche réel
   Implémenter la sauvegarde/restauration du contexte pour de vrais changements de tâche.
4. Mémoire paginée
   Remplacer l'allocation fixe par une gestion avec pagination et protection mémoire via la MMU.
5. Algorithmes supplémentaires
   Implémenter SJF, SRTF, priorités et MLFQ.
6. Interface graphique
   Passer en mode VGA graphique pour une interface plus riche.

8 BIBLIOGRAPHIE / RÉFÉRENCES

[1] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3: System Programming Guide*, Intel Corporation, 2021.

[2] OSDev Community, "Bare Bones," OSDev Wiki. [En ligne]. Disponible: https://wiki.osdev.org/Bare_Bones

[3] OSDev Community, "GDT Tutorial," OSDev Wiki. [En ligne]. Disponible: https://wiki.osdev.org/GDT_Tutorial

[4] OSDev Community, "Protected Mode," OSDev Wiki. [En ligne]. Disponible: https://wiki.osdev.org/Protected_Mode

[5] OSDev Community, "Scheduling Algorithms," OSDev Wiki. [En ligne]. Disponible: https://wiki.osdev.org/Scheduling_Algorithms

[6] J. Duntemann, Assembly Language Step-by-Step: Programming with Linux, 3rd ed., Hoboken, NJ: Wiley, 2009.

[7] QEMU Project, "QEMU Documentation," QEMU. [En ligne]. Disponible: https://www.qemu.org/documentation/

9 ANNEXES

9.1 ANNEXE A

Le code source complet du projet est disponible sur GitHub :

Lien du dépôt GitHub : https://github.com/paoloart7/LearnixMiniOS.git

Le dépôt contient :

- Code source complet
- Les instructions de compilation
- La documentation
- L'historique des commits