RAPPORT TECHNIQUE FINAL
myos-i686

Mini-système d'exploitation éducatif x86 32 bits (i686)

Projet	Mini-système d'exploitation x86
Version	1.0
Date	Session Septembre 2025
Auteur	Claud Edmond Charles
Architecture	i686 (32 bits)
Cible	QEMU / Bare-métal
Boot	Multiboot (GRUB)

Table des matières

Table des figures

1. Résumé exécutif

Le rapport repose sur cinq hypothèses de travail : **architecture monolithique** (shell dans le même espace que le noyau), **commutation de contexte simulée** (logique ordonnanceur/PCB sans sauvegarde ASM complète), gestion mémoire via un pool allocator statique de 64 KB avec bitmap et blocs fixes, timer PIT configuré à 100 Hz (1 tick = 10 ms) et démarrage conforme Multiboot 1 (magic 0x1BADB002). Alors **myos-i686** est un mini-noyau éducatif 32 bits conçu pour démontrer le boot Multiboot, la gestion des interruptions (IDT/ISR/IRQ/PIC), le timer PIT, la gestion des processus (PCB + états), l'ordonnancement multi-algorithmes, l'IPC par mailboxes, la synchronisation (mutex/sémaphores), la mémoire par pool allocator, ainsi qu'un shell interactif permettant de valider ces fonctionnalités sous **QEMU**.

2. Introduction et objectifs

2.1 Contexte

Développer un noyau 'from scratch' impose de travailler en mode freestanding (sans libc) et d'interagir avec les mécanismes bas niveau. L'objectif est pédagogique : concrétiser les concepts (interruptions, scheduling, IPC, synchronisation, mémoire).

2.2 Périmètre fonctionnel

Fonctionnalités obligatoires :

- Démarrage Mini-OS (Multiboot)
- PCB + états READY/RUNNING/BLOCKED/TERMINATED
- Ordonnancement FCFS et Round Robin
- Timer système + journal d'exécution
- Mini-Shell de commandes

Fonctionnalités bonus (note maximale) :

- Priorité, SJF, SRTF, MLFQ
- IPC par mailboxes
- Synchronisation (mutex / sémaphore)
- Mémoire minimale (pool allocator)
- Scenarios de test + démo vidéo

3. Environnement et outillage

Outil	Rôle	Details
GCC i686	Compilation C	i686-linux-gnu-gcc (C11 freestanding)
NASM	Assembleur	Bootstrap et routines bas niveau
LD	Link	linker.ld (placement noyau, sections)

QEMU	Emulation	qemu-system-i386
Make	Build	cibles build/run + automatisation

Commandes minimales :

```
#création du fichier binaire
make rebuild

# éxecution
make run
```

4. Architecture générale

4.1 Vue d'ensemble (monolithique modulaire)

Le noyau est organisé en modules partageant le même espace d'adressage. Le Shell sert d'interface de validation et de démo.

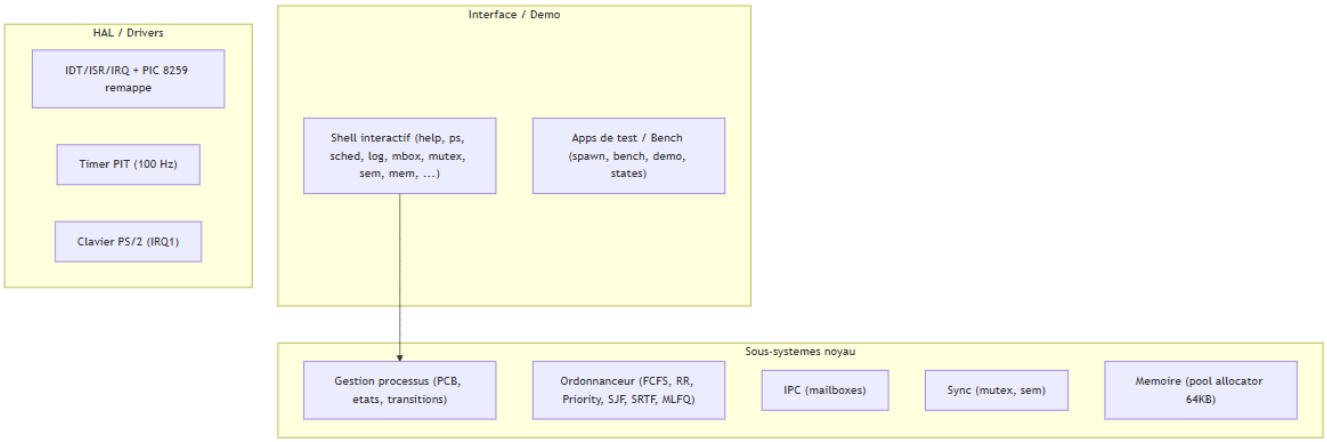

Figure 1- Architecture en couches

5. Boot et initialisation

5.1 Flux d'amorçage (Multiboot 1)

Le flux d'amorçage suit Multiboot 1 : GRUB charge le noyau en mémoire, détecte l'en-tête Multiboot (0x1BADB002) et passe les informations de démarrage. Le point d'entrée _start (dans boot.asm) désactive les interruptions, initialise la pile, puis appelle kernel_main(). Le noyau configure ensuite IDT/ISR/IRQ, remappe le PIC, initialise le PIT (tick), prépare les sous-systèmes (processus/ordonnanceur, IPC, sync, mémoire), active les interruptions (sti) et lance le shell.

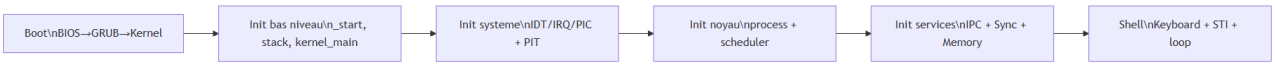

Figure 2 – Boot Sequence

6. Interruptions et timer PIT

6.1 IDT / ISR / IRQ / PIC

Le système installe une IDT (256 entrées) et des handlers pour exceptions CPU (0-31) et IRQ matériels (remappes). IRQ0 : PIT (tick), IRQ1 : clavier PS/2.

6.2 Timer PIT (100 Hz)

- 1 tick = 10 ms.
- Incremente l'uptime et alimente la logique de preemption (quantum).
- Le comportement des algorithmes est observable via exec_log (commande log).

7. Gestion des processus (PCB + états)

7.1 PCB

Chaque processus est represente par un PCB (process_t) : pid, name, state, priority, metriques (ticks, quantum) et contexte logique.

7.2 Machine a états

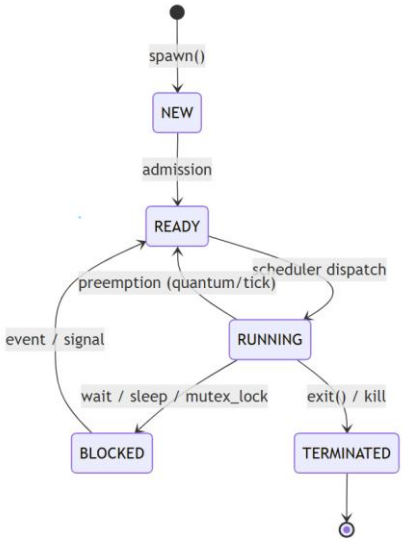

Figure 3 - États & transitions

8. Ordonnancement

8.1 Algorithmes

Algorithme	Type	Principe	Parametres clés
FCFS	Non-préemptif	1er arrive, 1er servi	Ordre d'arrivée
RR	Préemptif	Rotation par quantum	10 ticks
Priority	Préemptif	Plus haute priorité d'abord	Plage priorité
SJF	Non-préemptif	Job le plus court	Durée éstimée
SRTF	Préemptif	Temps restant minimal	Remaining time
MLFQ	Préemptif	Files multi-niveaux	2/4/8 ticks

8.2 MLFQ (3 niveaux)

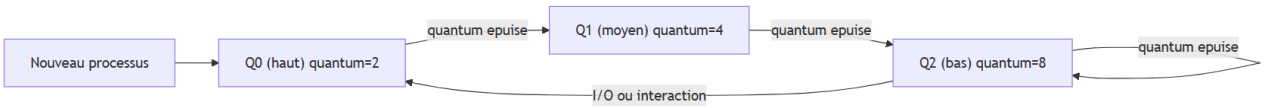

Figure 4 - Schema MLFQ (Mermaid)

8.3 Tick -> décision -> switch (preuve)

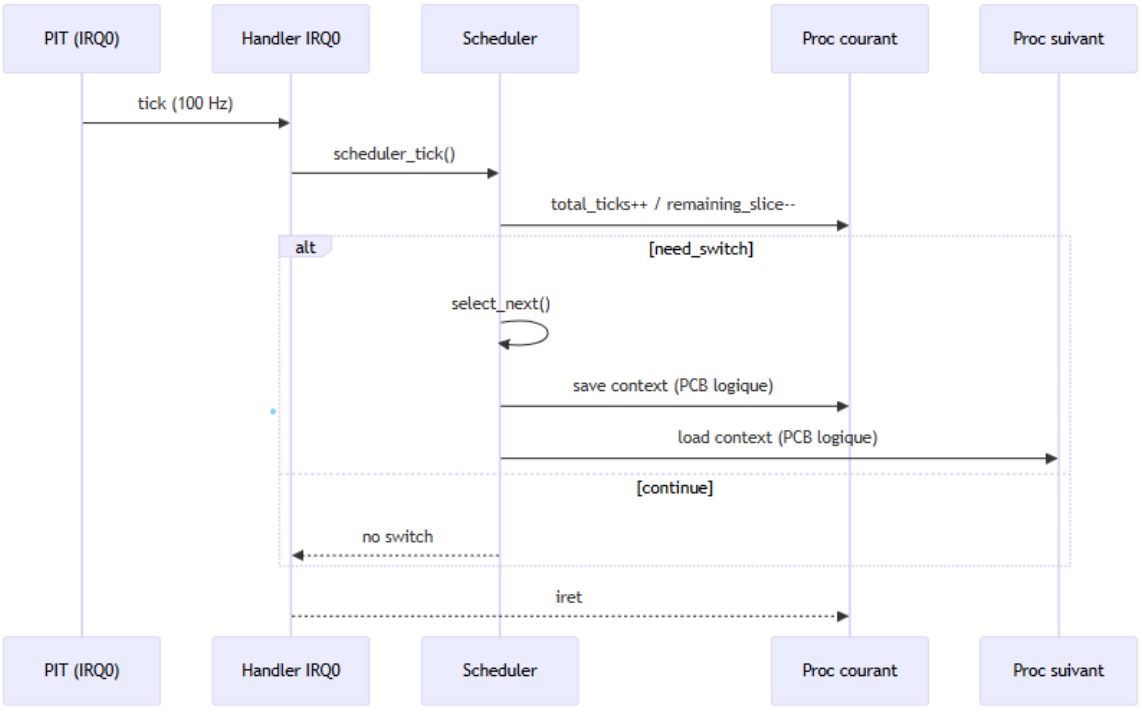

Figure 5 - Timer -> Scheduler

8.4 Journal d'exécution (exec_log)

Le Scheduler enregistre les exécutions (PID, début/fin en ticks, durée). La commande shell log fournit une preuve visuelle de l'alternance et de l'equite.

9. IPC (mailboxes)

9.1 Principe

L'IPC est basée sur des mailboxes (buffer circulaire) : create, send, recv. Le sender peut être bloqué si plein ; le receiver peut être bloqué si vide.

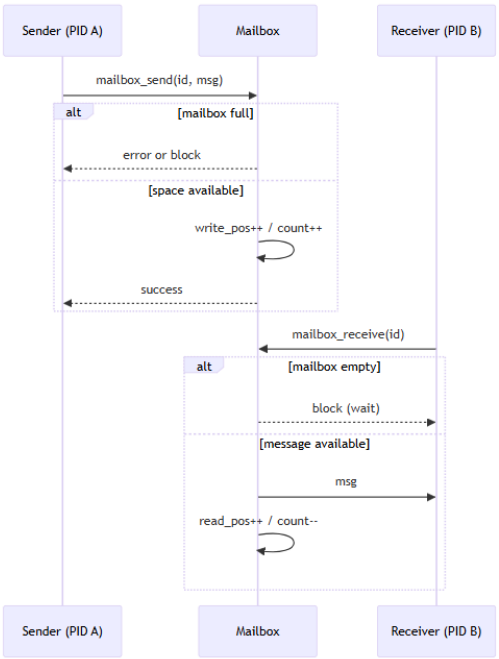

Figure 6 - IPC mailbox

10. Synchronisation (mutex & sémaphores)

10.1 Mutex

Un mutex assure l'exclusion mutuelle : locked, owner_pid, lock_count (optionnel). Les processus concurrents peuvent être places en BLOCKED.

10.2 Sémaphores

Un sémaphore régule l'accès a une ressource comptable via des operations wait/signal (P/V).

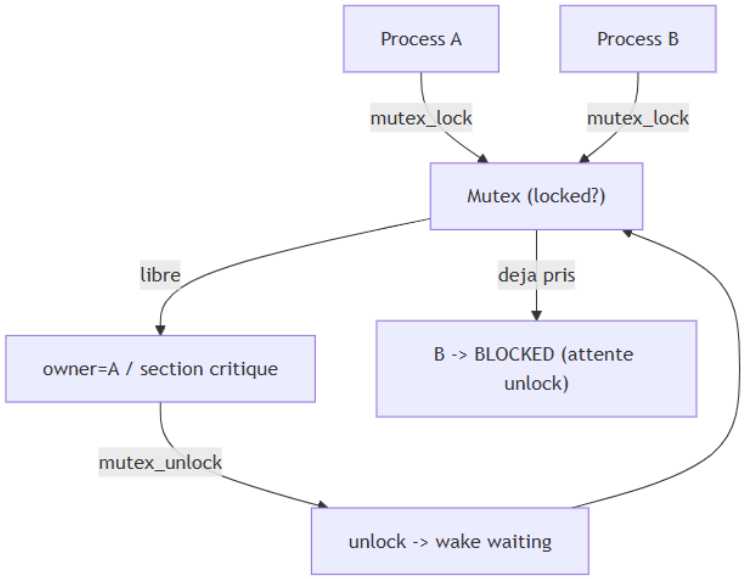

Figure 7 - Mutex

11. Mémoire (pool allocator)

11.1 Design

- Pool statique : 64KB.
- Taille de bloc : 64 bytes.
- Suivi via bitmap.
- API : kmalloc, kfree, kcalloc, stats/map.

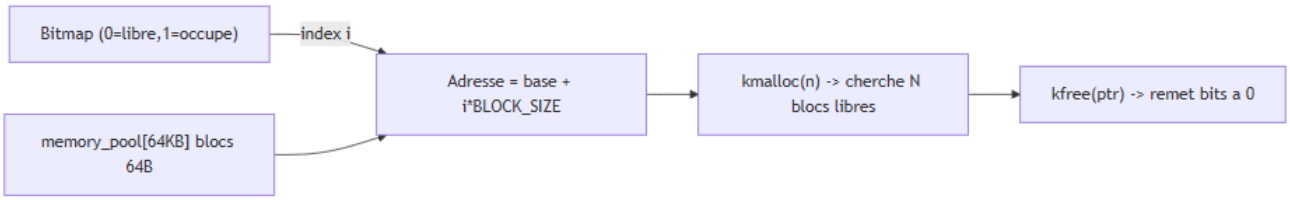

Figure 8 - Pool + bitmap

12. Shell et commandes

12.1 Commandes (categories)

Categorie	Commandes
Base	help, clear, info, uptime
Processus	ps, spawn, kill, block, unblock
Ordonnancement	sched, queue, log
Tests	bench, demo, states

IPC	mbox
Sync	mutex, sem
Memoire	mem

13. Tests et validation

13.1 Tests de build

```
make clean && make
```

13.2 Matrice de tests (shell)

ID	Composant	Commande	Attendu
T01	Boot	make run	prompt shell visible
T02	Process	spawn test1	PID + etat READY/RUNNING
T03	Etats	block <pid>; ps; unblock <pid>; ps	BLOCKED puis READY
T04	RR	sched rr; bench 3; log	alternance reguliere (quantum=10)
T05	Priority	sched priority; demo; log	prio haute avant basse
T06	IPC	mbox create x; mbox send 1 "test"; mbox recv 1	message recu
T07	Mutex	mutex create m; mutex lock 1; mutex unlock 1	lock/unlock OK
T08	Memoire	mem test / mem stats	alloc OK + stats

13.3 Scenarios de preuve (courts)

Scenario FCFS :

```
sched fcfs
bench 3
log
# attendu : ordre strict d'arrivee
```

Scenario RR :

```
sched rr
bench 3
log
# attendu : alternance selon quantum
```

Scenario MLFQ :

```
sched mlfq
bench 3
queue
log
# attendu : mouvements entre files (2/4/8)
```

15. Limites et améliorations

15.1 Limites

- Pas de pagination : pas d'isolation mémoire forte.
- Context switch reel non complet (approche logique / simulation).
- Single-core (pas SMP).
- Pas de systeme de fichiers persistants.

15.2 Améliorations

- Pagination (VMM) + séparation ring0/ring3.
- Syscalls + ABI stable.
- FS simple (ramdisk/initrd).
- Context switch assembleur complet.

16. Conclusion

myos-i686 couvre les mécanismes essentiels : boot, interruptions, timer, processus, ordonnancement avance, IPC, synchronisation, mémoire minimale et Shell de validation. La modularité et la démonstration par commandes facilitent l'évaluation et l'extension.

Annexe A - Matrice de couverture (exigences -> preuve -> test)

Exigence	Statut	Module / fichier	Commande	Attendu
Boot Multiboot	OK	boot.asm / linker.ld	make run	shell demarre
PCB + etats	OK	process.c/h	ps ; states	transitions visibles
FCFS	OK	scheduler.c/h	sched fcfs ; bench 3 ; log	ordre FCFS
Round Robin	OK	scheduler.c/h	sched rr ; bench 3 ; log	alternance quantum
Timer + journal	OK	timer.c/h + scheduler.c	uptime ; log	ticks + exec_log
Shell	OK	shell.c/h + keyboard.c	help	commandes disponibles
Priorite	OK	scheduler.c	sched priority ; demo	prio haute d'abord
SJF / SRTF	OK	scheduler.c	sched sjf/srtf ; bench	ordre par duree
MLFQ	OK	scheduler.c	sched mlfq ; queue ; log	3 niveaux 2/4/8
IPC mailbox	OK	ipc.c/h	mbox ...	send/recv OK
Mutex / sem	OK	sync.c/h	mutex ... ; sem ...	lock/wait OK
Memoire minimale	OK	memory.c/h	mem test/stats	alloc + stats