Linux Scheduler Simulator – Full Documentation

License MIT Version 1.0.0 Status Production Ready

Overview

A complete simulator of process scheduling algorithms with a modern web UI (Next.js/React/TypeScript) and a C backend. You can run it from the browser or directly via CLI.

Key features

- 6 scheduling algorithms: FIFO, Priority (preemptive), Round Robin, Multilevel, Multilevel Dynamic (aging), SRT
- Interactive Gantt chart, CPU utilization graph, queue visualization, detailed per-process stats
- Upload or auto-generate configuration files; default `sample_config.txt` preloaded
- Play/Pause/Step controls, dark theme, responsive UI

Tech stack

- **Frontend**: Next.js 16, React 19, TypeScript, Tailwind CSS, Radix UI, Recharts
- **Backend**: C11 (GCC), CLI binary `ordonnanceur`
- **Tooling**: pnpm, Make, Git

Priority convention (important)

- **Small value = higher priority** (Unix convention).
- CLI default: **descending** (bigger number = higher) because `prio_mode=1` in `main.c` if you do not pass a flag.
- API default: **ascending** (smaller number = higher) because `/api/schedule` sends `--prio-order asc`.
- Force mode: `--prio-order asc|desc`.

Prerequisites

Linux (Debian/Ubuntu)

```
# Update package manager
sudo apt update && sudo apt upgrade -y

# Install build tools
sudo apt install -y \
  build-essential \
  gcc \
  make \
  git

# Install Node.js 18+ and pnpm
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh |
bash
source ~/.bashrc
nvm install 18
npm install -g pnpm@8
```

macOS

```
# Install Xcode Command Line Tools
xcode-select --install

# Install Homebrew (if not installed)
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Install dependencies
brew install node@18 make git

# Install pnpm
npm install -g pnpm@8
```

Windows (WSL2)

```
# Open PowerShell as Administrator and run:
wsl --install -d Ubuntu-22.04

# Then in WSL terminal:
sudo apt update && sudo apt upgrade -y
sudo apt install -y build-essential gcc make git curl

# Install Node.js
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh |
bash
source ~/.bashrc
nvm install 18
npm install -g pnpm@8
```

Project setup

```
# Clone the repository
git clone https://github.com/arijsebai/Projet-Ordonnancement-Linux.git
cd Projet-Ordonnancement-Linux
```

Install Node dependencies

```
# Recommended
pnpm install
# Alternatively
npm install
```

Build the C backend

```
make clean
make all
# Validate binary
ls -la ordonnanceur
```

Quick structure check

```
ls -la config/sample_config.txt
cat config/sample_config.txt
```

Usage

Mode 1: Web UI (recommended)

Start dev server:

```
pnpm dev
# Available at http://localhost:3000
```

What you get:

- Default `sample_config.txt` auto-loaded
- Generate random processes or upload your own `.txt`
- Select algorithm (FIFO, Priority, RR, Multilevel, Multilevel Dynamic, SRT)
- Dynamic params (quantum for RR/Multilevel Dynamic)
- Run and visualize: live Gantt, CPU usage, queue, detailed stats (tables + charts)

Config file example:

```
# name arrival execution priority
P1 0 5 1
P2 2 3 2
P3 4 2 1
P4 6 4 2
P5 8 2 1
```

Mode 2: CLI (C backend)

```
# Direct file
./ordonnanceur config/sample_config.txt

# Interactive (menu: generate or pick file, then choose algorithm)
./ordonnanceur
```

CLI flow:

1. Load config file (either provided or generated)
2. Choose algorithm from menu (FIFO, Priority, RR, Multilevel, Multilevel Dynamic, SRT)
3. Enter params if needed (quantum)
4. Simulation prints Gantt + stats in console

Sample FIFO output (truncated):

```
Time  Executing  Ready Queue
0     P1        []
1     P1        []
2     P1        [P2]
...
Average Wait Time: 2.33
Makespan: 10
```

Tests

C tests

```
make clean && make all
gcc -Iinclude tests/test_fifo.c policies/fifo.c -o test_unit && ./test_unit
gcc -Iinclude tests/test_roundrobin.c policies/roundrobin.c -o test_unit &&
./test_unit
gcc -Iinclude tests/test_priority.c policies/priority_preemptive.c -o
test_unit && ./test_unit
gcc -Iinclude tests/test_multilevel.c policies/multilevel.c -o test_unit &&
./test_unit
gcc -Iinclude tests/test_multilevel_dynamic.c policies/multilevel_dynamic.c
-o test_unit
&& ./test_unit
```

Web checks

```
pnpm build
pnpm start   # serve production build
```

Configuration file format

```
# Comment (optional)
# NAME ARRIVAL EXECUTION PRIORITY
P1 0 5 1
P2 2 3 2
P3 4 2 1
```

Field	Type	Description	Example
NAME	String	Process identifier	P1, Task_A
ARRIVAL	Int	Arrival time (≥0)	0, 5
EXECUTION	Int	Execution time (>0)	5, 10
PRIORITY	Int	Static priority (small = high)	0 (high), 5 (low)

Parsing rules: blank lines ignored; `#` comments ignored (including end-of-line); 4 tokens required per line.

Scheduling algorithms

- **FIFO** – simplest, non-preemptive; best for batch; convoy effect risk.
- **Priority (preemptive)** – priority-driven; starvation risk; small value high priority (default API).
- **Round Robin** – fair time-slicing; needs quantum (e.g., 2–4); context-switch overhead.
- **Multilevel (static)** – multiple queues + RR; configurable quantum; no priority aging.
- **Multilevel Dynamic (aging)** – multilevel with dynamic priority boosts to prevent starvation; requires quantum.
- **SRT** – shortest remaining time first; great average wait; long jobs may starve.

Metrics

- Arrival time, Execution time, Finish time
- Waiting time = finish – arrival – execution
- Turnaround time = finish – arrival
- Average waiting time, Makespan

Troubleshooting

“Backend binary not found”

```
make clean && make all
ls -la ordonnanceur
```

“Parse error in config file”

```
cat config/sample_config.txt
# Ensure: NAME ARRIVAL EXECUTION PRIORITY
```

Port 3000 already in use

```
pnpm dev -- -p 3001
# or
lsof -i :3000
kill -9 <PID>
```

pnpm not found

```
corepack enable
corepack prepare pnpm@latest --activate
# or: npm install -g pnpm
```

Provided configs

- `config/sample_config.txt` — default test set
- `config/sample_config_<timestamp>.txt` — generated during web runs

Contribution

1. Fork the repo
2. Create a branch (`git checkout -b feature/my-change`)
3. Commit (`git commit -m "Add my change"`)
4. Push (`git push origin feature/my-change`)
5. Open a PR

License

MIT License — see LICENSE.

Authors

Academic project (Advanced OS). Team: Arij Sebai, Balkis Hanafi, Hadil Hasni, Aya Sakroufi, Wiem Ayari.

Support

- Docs: see `Documentation.md`
- Issues: open a GitHub issue with steps, environment, and logs
- Contact: create an issue for questions/suggestions

Educational use cases

Great for teaching OS scheduling, live demos, student projects, and technical presentations.

Roadmap (future ideas)

- NUMA-aware scheduling
- CPU/memory profiling
- Export results (PDF, CSV)
- Comparative benchmarking
- Plugin for real-system scheduling
- Mobile UX polish; theme toggle

Last updated: December 2025 · Version: 1.0.0

```
```