

# MODIFICATIONS TO THE LINUX SCHEDULER

**04JEZOQ** Operating Systems

## Group 9

Moretti Simone Marcello 345922

Sarcuni Sabina 338926

Trombotto Edoardo 343403

# IMPLEMENTING A NEW SCHEDULING ALGORITHM

## POLICY

**Lottery scheduling:** When a process is created, it is assigned a number of tickets, which are used to determine the process's chance of being scheduled. The scheduler randomly picks a ticket from the available tickets and runs the process that holds that ticket.

### 1. Define a new scheduling class

```
DEFINE_SCHED_CLASS(lottery) = {  
    .enqueue_task = enqueue_task_lottery,  
    .dequeue_task = dequeue_task_lottery,  
    // And so on ...  
};
```



# IMPLEMENTING A NEW SCHEDULING ALGORITHM

## 2. Tweak the linker file *'linux/include/asm-generic/vmlinux.lds.h'*

```
#define SCHED_DATA          \
    STRUCT_ALIGN();         \
    __begin_sched_classes = .; \
    *(__idle_sched_class)    \
    *(__fair_sched_class)    \
    *(__rt_sched_class)      \
    *(__dl_sched_class)      \
    *(__stop_sched_class)    \
    *(__lottery_sched_class) \
    __end_sched_classes = .;
```

# IMPLEMENTING A NEW SCHEDULING ALGORITHM

## 3. Create the scheduling policy *'linux/include/uapi/linux/sched.h'*

```
#define SCHED_NORMAL      0
#define SCHED_FIFO        1
#define SCHED_RR          2
#define SCHED_BATCH       3
/* SCHED_ISO: reserved but not implemented yet */
#define SCHED_IDLE        5
#define SCHED_DEADLINE    6
/* New scheduling policy for lottery scheduling */
#define SCHED_LOTTERY      7
```

# IMPLEMENTING A NEW SCHEDULING ALGORITHM

## 4. Initialize the data structures *init\_ltr\_rq*

The **lottery runqueue** is a data structure that keeps track of the tasks that are currently in the lottery scheduling class. It keeps track of:

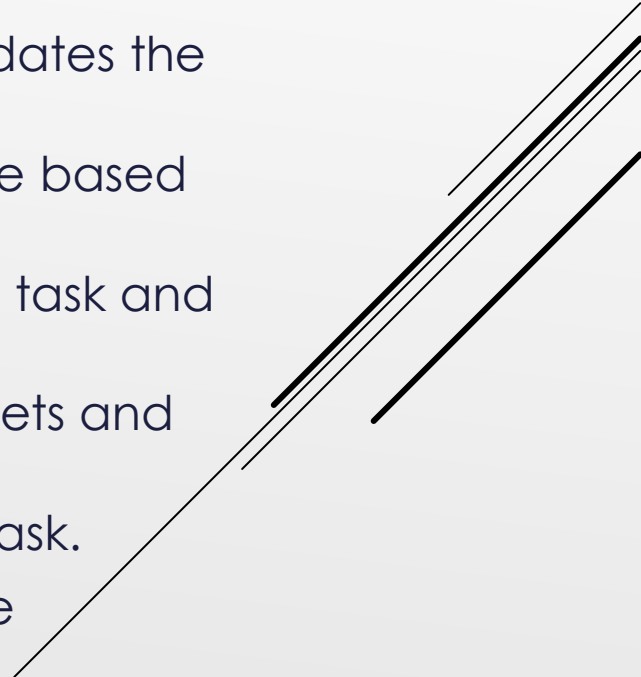
- ▶ The total number of tickets available in the runqueue.
- ▶ The number of tasks in the runqueue.
- ▶ A doubly linked list of tasks in the runqueue.

## 5. Modify the *task\_struct*

The **task\_struct** is the data structure that represents a task in the Linux kernel. We had to modify this kind of data structure to include it in a field of type *sched\_ltr\_entity*, a section that would hold the number of tickets acquired by a process, and in order to do so we modified the file *'linux/include/linux/sched.h'*.

# IMPLEMENTING A NEW SCHEDULING ALGORITHM

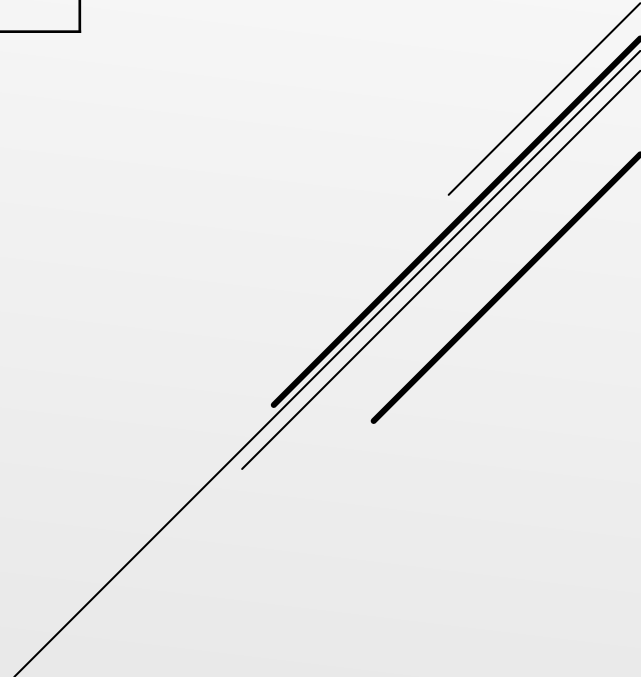
## 6. Implement the scheduling functions

- ▶ **`enqueue_task_lottery`**: It adds the task to the lottery queue and updates the total number of tickets.
  - ▶ **`dequeue_task_lottery`**: It removes the task from the lottery queue and updates the total number of tickets.
  - ▶ **`yield_task_lottery`**: It updates the task's state and may trigger a reschedule based on the lottery tickets.
  - ▶ **`check_preempt_curr_lottery`**: It compares the lottery tickets of the current task and the new task and decides whether to preempt or not.
  - ▶ **`pick_next_task_lottery`**: It randomly selects a ticket from the available tickets and returns the task associated with that ticket.
  - ▶ **`set_next_task_lottery`**: It updates the current task pointer to the selected task.
  - ▶ **`task_tick_lottery`**: It checks if the task should be rescheduled based on the lottery tickets and updates the task's state accordingly.
- 

# IMPLEMENTING A NEW SCHEDULING ALGORITHM

## 7. Modify the Makefile *'linux/kernel/sched'*

```
obj-y += lottery.o
```




# SCRIPTING

When dealing with such big projects it's fundamental to have scripts to automate the process of building and testing the kernel. We made two scripts:

`'compile.sh'` : Compiles the kernel and header files.

`'qemu_launch.sh'` : locates required files and caches them in `~/.qemu_config.sh` for future runs. A flag option called `'-refresh'` is present to force a new search of files.

A series of three parallel diagonal lines in the bottom right corner of the slide, starting from the right edge and extending towards the bottom left.



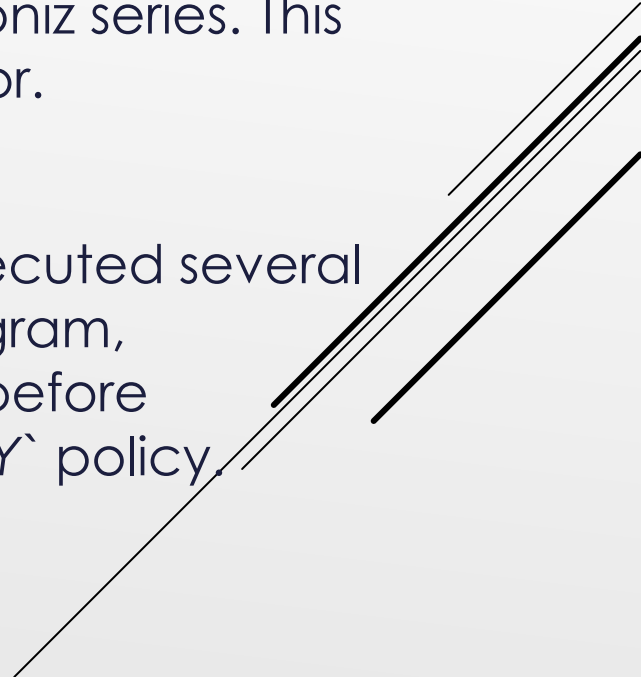
# BENCHMARKING THE SCHEDULER

## 1. Program Generation for Scheduling Analysis

We developed a simple yet computationally intensive C program, ``dummy_program.c``, which calculates the value of  $\pi$  using the Leibniz series. This ensured enough CPU load to effectively test the scheduler's behavior.

## 2. Concurrent Execution Under Different Scheduling Policies

To evaluate how the scheduler manages multiple processes, we executed several instances of ``dummy_program.c`` concurrently using another C program, ``dummy_launcher.c``. This launcher modified the scheduling policy before execution, switching from the default to our custom ``SCHED_LOTTERY`` policy.

A series of three parallel diagonal lines in the bottom right corner of the slide, consisting of a thin black line, a medium-thick grey line, and a thick black line.

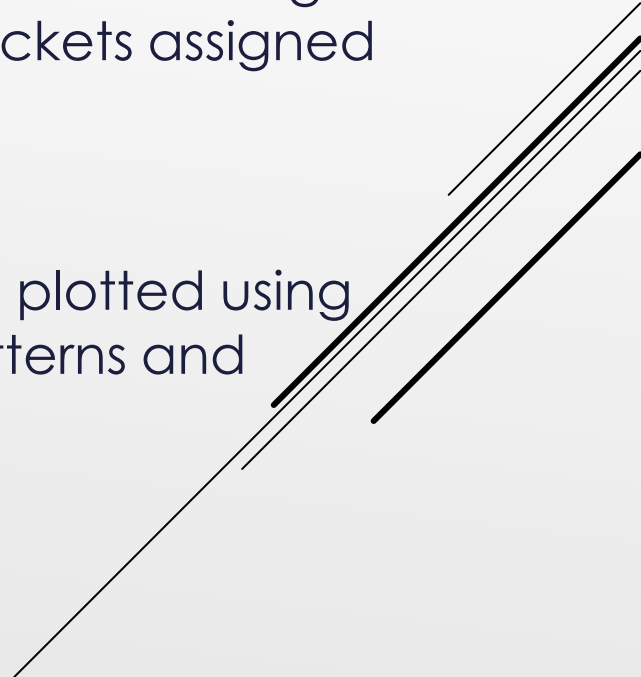
# BENCHMARKING THE SCHEDULER

## 3. Logging Scheduling Decisions

Each time a process was selected by the scheduler, the kernel logged a message containing the PID, the timestamp of selection, and the number of tickets assigned to that process.

## 4. Data Extraction and Visualization

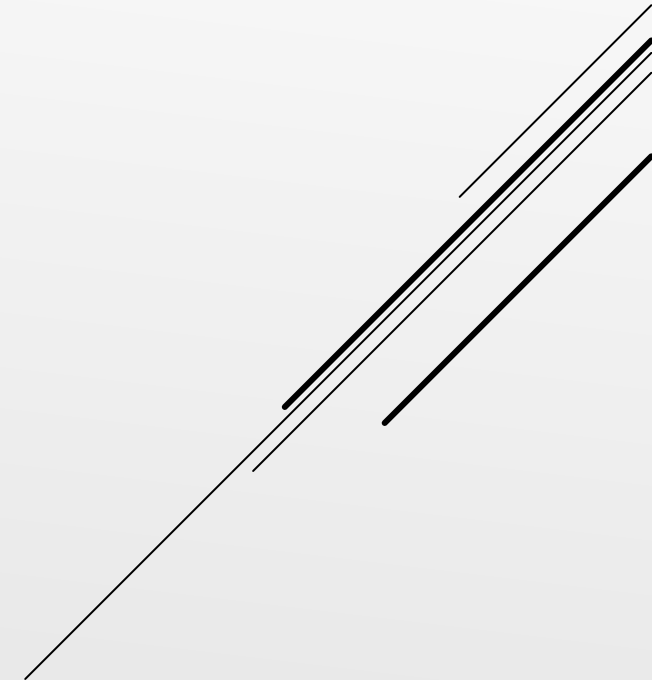
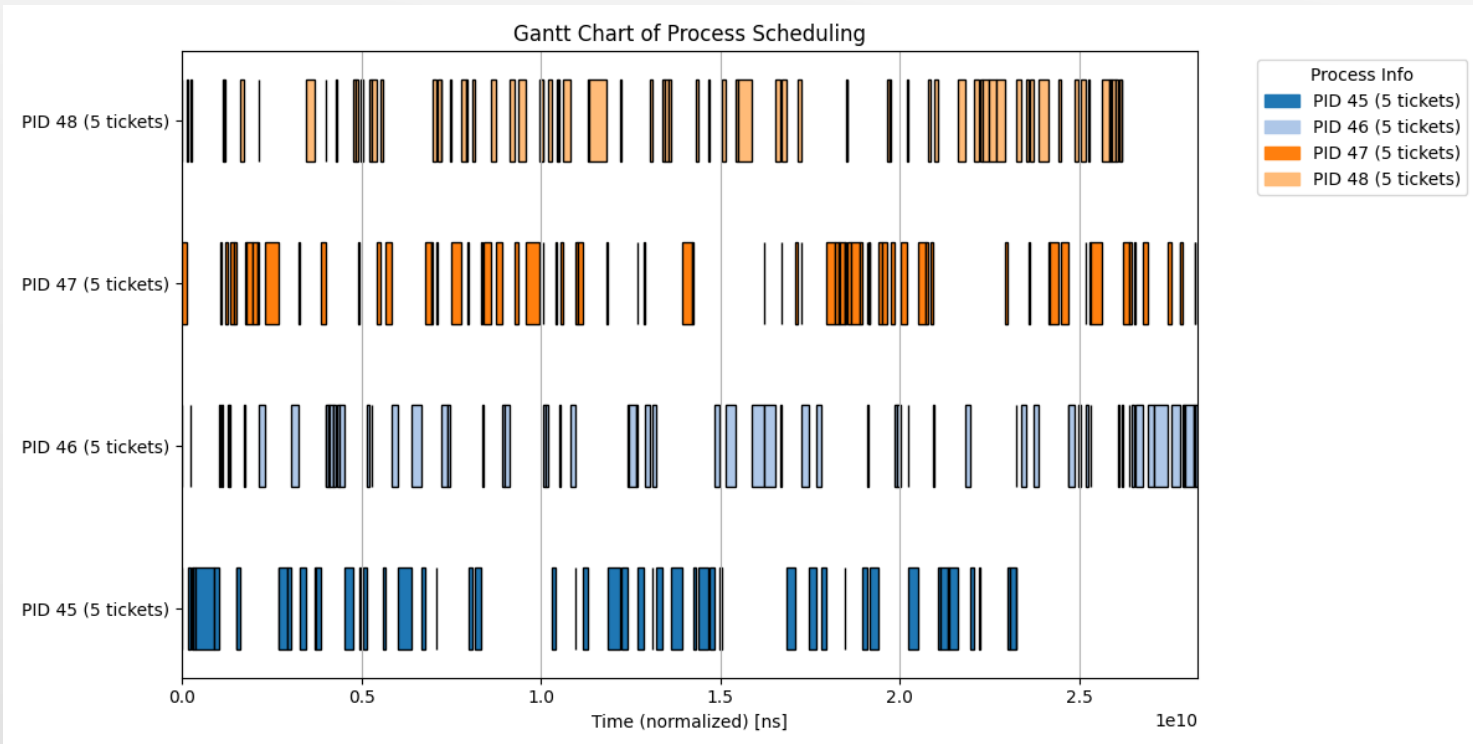
Those data were extracted and stored in a file. The results were then plotted using the Python script `monitor.py`, allowing us to analyze scheduling patterns and waiting times visually.

A series of three parallel diagonal lines in the bottom right corner of the slide, sloping upwards from left to right. The lines are black and vary slightly in thickness and position, creating a modern, abstract graphic element.

# PERFORMANCE METRICS

The results were obtained in two separate conditions regarding the tickets assigned for each process inside the file `linux/kernel/fork.c`:

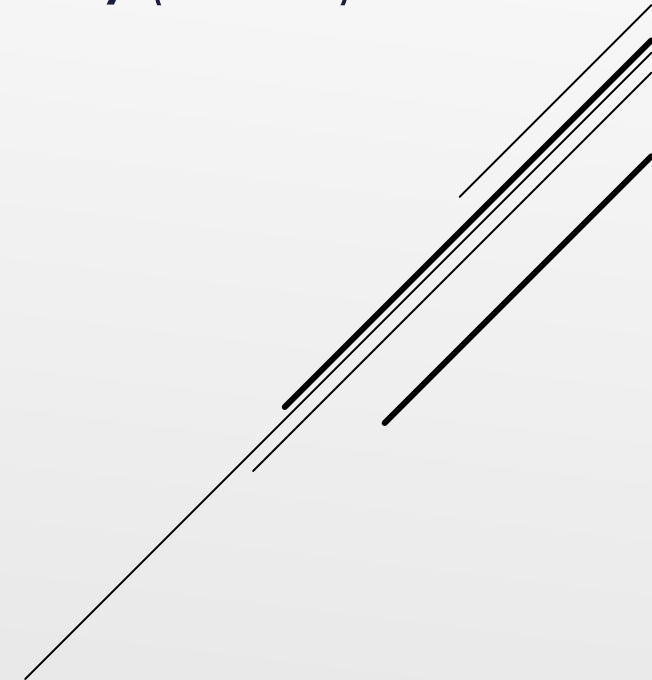
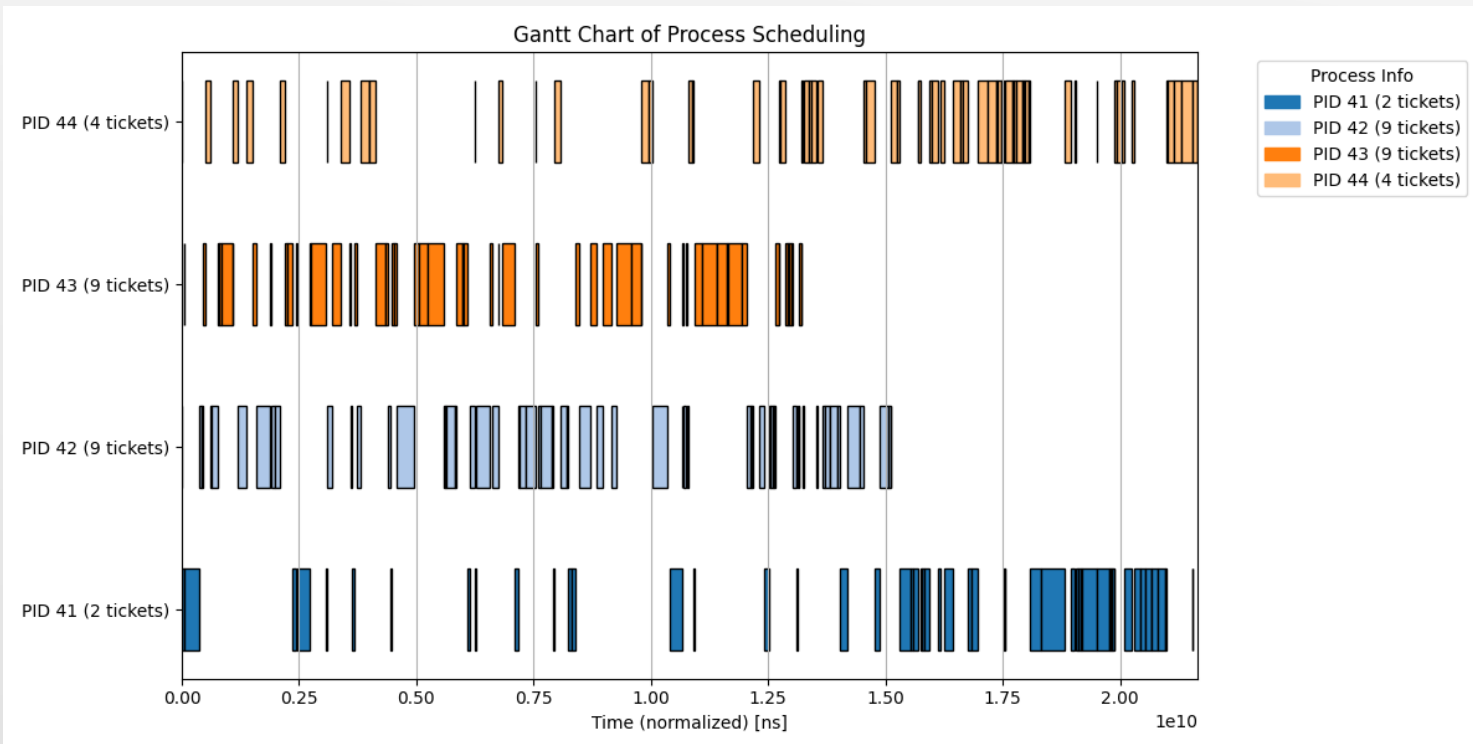
1. With a **fixed** number of tickets per process (5)



# PERFORMANCE METRICS

The results were obtained in two separate conditions regarding the tickets assigned for each process inside the file `linux/kernel/fork.c`:

2. With a **variable** number of tickets per process, assigned **randomically** (1 to 15)



# THANK YOU

## **Group 9**

Moretti Simone Marcello 345922

Sarcuni Sabina 338926

Trombotto Edoardo 343403