Modify Linux Kernel Scheduler: Implementation and Evaluation of Lottery, WRR, and SRTF

ZHANG Yuning, Cui Ningyu
School of Data Science
Chinese University of Hong Kong, Shenzhen
Email: 225040144@link.cuhk.edu.cn, 225040147@link.cuhk.edu.cn

***Abstract*—Process scheduling is a core function of operating systems, determining how CPU time is allocated among multiple tasks in a multitasking environment. Based on the xv6-riscv operating system, this paper implements three classic process scheduling algorithms: Lottery Scheduling, Weighted Round Robin (WRR), and Shortest Remaining Time First (SRTF). We complete the kernel-level implementation of these three algorithms by extending the process structure, modifying the core scheduler logic, and adding corresponding system calls. To verify the correctness of the algorithms, we design standard test programs and employ a Python-based real-time visualization tool to monitor scheduling behavior. Experimental results demonstrate that Lottery Scheduling achieves proportional CPU time allocation, WRR provides deterministic weighted round-robin scheduling, and SRTF significantly optimizes the response time of short processes.**

xv6, process scheduling, lottery scheduling, weighted round robin scheduling, shortest remaining time first, Linux kernel

I. Introduction

Process scheduling policies directly affect system responsiveness, throughput, and fairness. Modern operating systems adopt different scheduling algorithms for different scenarios: general-purpose systems emphasize fairness, real-time systems focus on determinism, and servers pursue high throughput [1].

xv6 is a simplified Unix-like teaching operating system widely used in operating system courses [2]. This study implements three representative scheduling algorithms:

1) **Lottery Scheduling**: Achieves proportional resource allocation through a random drawing mechanism
2) **Weighted Round Robin Scheduling**: Introduces weight support based on round-robin scheduling
3) **Shortest Remaining Time First Scheduling**: Optimizes response time for short tasks

The main contributions of this paper include: (1) Complete kernel-level implementation of the three scheduling algorithms; (2) Addition of corresponding system call interfaces; (3) Construction of a real-time visualization monitoring system; (4) Experimental verification of the correctness and performance characteristics of each algorithm.

II. Background and Related Work

A. Lottery Scheduling

Lottery Scheduling, proposed by Waldspurger and Weihl [3], is a probabilistic scheduling algorithm. The core idea is that each process is allocated a certain number of lottery tickets, and the scheduler randomly draws a winning ticket; the process holding the corresponding ticket obtains the CPU. Over long-term operation, the CPU time share is proportional to the number of tickets held. The advantages of this algorithm lie in its simple implementation, rapid response, and graceful handling of priority issues.

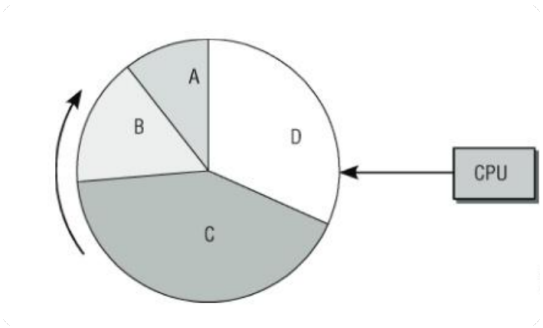

Fig. 1. Lottery Scheduling

B. Weighted Round Robin Scheduling

Weighted Round Robin is an extension of standard Round Robin scheduling [4], assigning a weight to each process. The scheduler maintains a time budget for each process and allocates CPU time in proportion to the weights. Compared to Lottery Scheduling, WRR provides deterministic scheduling behavior, making it suitable for scenarios requiring stable performance guarantees.

C. Shortest Remaining Time First Scheduling

SRTF is the preemptive version of Shortest Job First (SJF), which selects the process with the shortest remaining execution time to run each time [5]. Theoretically, SRTF minimizes average turnaround time but may lead to starvation of long processes. This algorithm requires prior knowledge or prediction of process execution times.

III. Experimental Environment Configuration

A. Development Environment

This study adopts the following development environment:

- **Host System**: Windows 11
- **Virtual Machine**: Ubuntu 22.04 LTS
- **Target OS**: xv6-riscv operating system
- **Emulator**: QEMU (single CPU mode, RISC-V architecture)
- **Toolchain**: RISC-V GCC cross-compiler

IV. Scheduling Algorithm Implementation

A. Lottery Scheduling Implementation

1) Process Structure Extension: Extend `struct proc` in `kernel/proc.h` by adding the tickets field:

```
struct proc {
    // ... existing fields ...
    int tickets;
// Lottery tickets owned by process
};
```

Initialize tickets to the default value of 1 in `allocproc()`. Add random number generator support and implement the `rand()` function.

2) Scheduler Modification: `scheduler()` function is modified to implement the lottery drawing algorithm, as shown in Algorithm 1.

Algorithm 1 Lottery Scheduling Algorithm

```
Traverse all RUNNABLE processes and calculate the total number of tickets total
Generate a random number winner ← random(0, total)
Initialize sum ← 0
for each process p do
  if p is in RUNNABLE state then
    sum ← sum + p.tickets
    if sum > winner then
      Select p to run
      break
    end if
  end if
end for
Output structured scheduling log: SCHEDLOG
```

3) System Call: settickets: To allow user-space processes to set the number of tickets, add a system call:

1) Assign a system call number in `kernel/syscall.h`:define SYS_settickets 22
2) Register the function pointer in `kernel/syscall.c`
3) Implement `sys_settickets()` in `kernel/sysproc.c`:

```
uint64 sys_settickets(void) {
    int n;
    struct proc *p = myproc();
    if(argint(0, &n) < 0 || n < 1)
        return -1;
    acquire(&p->lock);
    p->tickets = n;
    release(&p->lock);
    return 0;
}
```

4) Add the user-space wrapper function in `user/user.h`

B. Weighted Round Robin Scheduling Implementation

1) Data Structure Extension: Add the weight and budget fields to `struct proc`:

```
struct proc {
// ... existing fields ...
int weight; // Scheduling weight
int budget; // Remaining quota in current round
};
```

Initialize weight=1 and budget=1 in `allocproc()`. Allow child processes to inherit the parent process's weight in `fork()`.

2) WRR Scheduling Algorithm: The scheduler maintains a global variable `last_wrr_idx` to record the position of the last scheduled process. The algorithm flow is as follows:

1) Check whether there exists a RUNNABLE process with budget > 0
2) If all RUNNABLE processes have budget equal to 0, reset budget = weight for all processes
3) Start cyclic scanning from `last_wrr_idx+1`, selecting the first process that is RUNNABLE and has budget > 0
4) Decrement the budget of the selected process and switch to run it
5) Update `last_wrr_idx` to the index of the selected process

C. Shortest Remaining Time First Scheduling Implementation

1) Process Structure Extension: Add the burst_time and remaining_time fields:

```
struct proc {
    // ... existing fields ...
    int burst_time;
// Total CPU time required
    int remaining_time; // Remaining CPU time
};
```

2) Remaining Time Update: In the clock interrupt handler `usertrap()`, when the trap type is a clock interrupt, decrement the remaining_time of the current process. If remaining_time becomes 0, mark the process as ZOMBIE state.

3) SRTF Scheduling Algorithm: The scheduler each time selects the process with the smallest remaining_time that is in the RUNNABLE state to run. The implementation traverses all processes, maintains the index of the minimum value, and switches context after selection.

4) System Call: setburst: Similar to settickets, add the `setburst()` system call to allow user processes to set the burst time while also initializing the remaining_time.

V. Test Programs and Visualization Tools

A. Test Program Design

To verify the correctness of each scheduling algorithm, standard test programs were designed:

1) Lottery Scheduling Test: The parent process creates three child processes via `fork()`, calling `settickets(10)`, `settickets(20)`, and `settickets(30)` respectively. Each child process executes a CPU-intensive loop for a fixed duration. The expected scheduling count ratio is 1:2:3.

2) WRR Test: The weight allocation is PID 4→2, PID 5→1, PID 6→3, with an expected CPU time ratio of 2:1:3.

3) SRTF Test: The burst time allocation is 30, 20, and 10 time units. The expected completion order is short process first (10 → 20 → 30).

B. Real-time Visualization System

A GUI visualization tool was built using Python:

- **UI Framework**: Tkinter
- **Charting Library**: Matplotlib
- **Functions**:
  - Automatically launch QEMU to run xv6
  - Read SCHEDLOG logs output by the scheduler in real time
  - Dynamically update scheduling count charts and timeline charts
  - Display current ticket/weight/remaining time distribution

The scheduler outputs structured logs at each scheduling event:

```
SCHEDLOG: pid=4 tickets=10 state=RUNNING
```

VI. Experimental Results and Analysis

A. Lottery Scheduling Results

Figure 1 presents the experimental results of Lottery Scheduling. After long-term operation, the scheduling count ratio of the three processes (with ticket numbers 10:20:30) approaches 1:2:3. Short-term fluctuations are caused by randomness, which is consistent with the theoretical expectation of probabilistic scheduling.

B. WRR Scheduling Results

The WRR scheduler exhibits deterministic weighted allocation behavior. In each complete scheduling round, processes with weights 2, 1, and 3 receive corresponding numbers of scheduling opportunities. The experimental results are completely consistent with the theoretical values, verifying the correctness of the algorithm.

Fig. 2. Scheduling count distribution of Lottery Scheduling

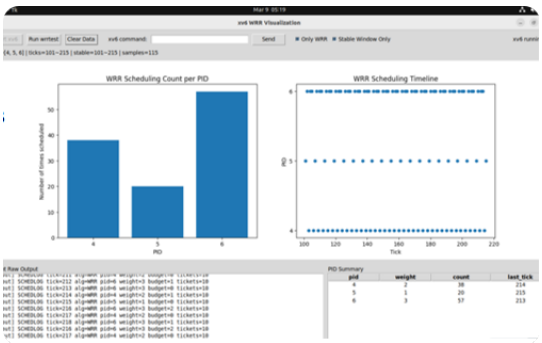

Fig. 3. Scheduling count distribution of WRR Scheduling

C. SRTF Scheduling Results

The SRTF scheduling results are shown in Table I. The process with the shortest remaining time (PID 6, remaining 10) receives the highest number of scheduling opportunities and completes first. The long process (PID 4, remaining 100) receives almost no scheduling after the middle stage until all short processes complete. This aligns with the design goal of the SRTF algorithm to prioritize short tasks.

TABLE I
SRTF Scheduling Experimental Results

PID	Burst Time	Remaining Time	Scheduling Count
4	30	30	5
5	20	20	12
6	10	10	28

D. Discussion

The three scheduling algorithms each have their advantages and disadvantages:

- **Lottery Scheduling**: Simple implementation and high flexibility, but randomness leads to short-term unfairness
- **WRR**: Good determinism, suitable for scenarios requiring stable performance guarantees
- **SRTF**: Optimal average response time, but may cause starvation of long processes, requiring additional mechanisms for prevention

Practical systems can select appropriate algorithms based on application scenarios, or combine the advantages of multiple algorithms to design hybrid schedulers.

VII. Conclusion

This paper fully implements three scheduling algorithms —Lottery Scheduling, Weighted Round Robin Scheduling, and Shortest Remaining Time First Scheduling—in the xv6-riscv operating system. Through kernel extension, scheduler refactoring, and system call addition, the correctness and effectiveness of each algorithm are verified. The main contributions include:

1) Implementation of kernel-level code for three scheduling algorithms, which can serve as teaching examples for operating system courses
2) Addition of `settickets()` and `setburst()` system calls, enriching the system call interface of xv6
3) Construction of a Python-based real-time visualization tool to intuitively display scheduling behavior
4) Experimental measurement of scheduling count distribution for each algorithm, validating theoretical expectations

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 10th ed. Wiley, 2018.

[2] R. Cox, F. Kaashoek, and R. Morris, *xv6: A simple, Unix-like teaching operating system*, 2020.

[3] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proc. 1st USENIX Symp. Operating Systems Design and Implementation (OSDI)*, 1994, pp. 1–11.

[4] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Pearson, 2014.

[5] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 2018.