SimpleShell Scheduler

Group: 50
Members: Satvik Arya(2023493) and Sanjeev(2023483)

Github link-: https://github.com/Satvik1924/SimpleScheduler

This project implements a basic shell, *SimpleShell*, with process scheduling features. It allows users to submit commands, manage process execution with a priority-based scheduling system, and view statistics on process waiting and execution times. Additionally, it supports functionality for handling multiple CPU cores and setting custom time slices.

Compilation-:
To compile the code, use the following command:
gcc -o SimpleShell SimpleShell.c

Execution-:
Start the program by running the compiled binary.
./SimpleShell
The program will prompt for the number of CPUs and the time slice duration in milliseconds.

1. **NCPU -** number of CPUs to use for concurrent process scheduling.
2. **TSLICE -** Time slice duration for each process in milliseconds.

Commands-:
submit ./<command> <priority> - Submits a new command to be scheduled by the shell. Commands are assigned a priority (default is 1 if not specified). Priority levels can be as low as 1 and as high as 4.
schedule - Triggers the scheduling of processes in the priority queue,

assigning CPU time to processes based on their priority levels. ***history -*** Displays the history of all submitted commands. ***stats -*** displays the detailed stats of the process scheduling

Process Scheduling and Priority Queue:

Each process has the following fields:

- pid: Process ID.
- priority: Process priority (1-4).
- last_stop_time and last_start_time: Timestamps to track execution intervals.

The priorityqueue data structure is implemented as a linked list. Higher-priority processes are enqueued at the front, ensuring that processes with priority 1 execute before those with lower priorities.

Enqueue/Dequeue Functions:

1. **enqueue**: Adds processes to the queue based on priority.
2. **dequeue**: Removes the front process from the queue, which has the highest priority.

Time Slice Calculation:

The time slice for each process is calculated based on priority:

```
int calculate_time_slice(int priority) {
    if (priority == 1) { return TIME_QUANTUM * 1; } if
    (priority == 2) { return TIME_QUANTUM * 1.25; } if
    (priority == 3) { return TIME_QUANTUM * 1.5; } if
    (priority == 4) { return TIME_QUANTUM * 2; }
    return -1;
}
```

Signal Handling:

The shell uses signal handling to manage various functions, including interrupting processes and scheduling:

- **SIGINT**: Triggered by CTRL+C, exits the shell and displays all history.
- **SIGQUIT**: Initiates the process scheduling mechanism.
- **SIGALRM**: Triggers time quantum expiration for preempting processes.

Timer Handling:

The timerHandler function triggers when the time quantum expires, setting time_quantum_expired to 1 to indicate that a process's time slice has been expired

History:

The shell maintains a history of all submitted commands. Each history record contains:

- command: Command string.
- args: Command arguments.
- pid: Process ID.
- priority: Process priority.
- start_time and end_time: Timestamps for execution duration tracking.
- waiting_time: Total waiting time (in milliseconds). ● duration: Total execution time.

Displaying Statistics:

When history is entered, the shell displays the command history and aggregated statistics on execution and waiting times by priority.

Main Functions:

main: Initializes shell parameters and enters the shell loop to process commands. **parse_input**: Parses user input to identify commands and arguments.

launch: Creates a new process for the specified command.

enqueue / dequeue: Manage the process queue based on priority.

schedule_the_Processes: Main scheduler function to allocate CPU time slices.

simple_scheduler_daemon: A background function that schedules processes based on signals.

sigintHandler and sigQuitHandler: Manage cleanup and scheduling triggers.

Contributions:

Sanjeev - parse_input, launch, enqueue, dequeue, display_stats

Satvik - main, schedule_the_Process, simple_scheduler_daemon