Simple Scheduler

SimpleShell

This SimpleShell implementation allows the user to submit jobs (executables) for execution using a SimpleScheduler. The shell maintains a job queue using shared memory and interacts with the scheduler to manage job execution. The shell takes NCPU (number of CPUs) and TSLICE (time slice for scheduling) as input arguments. Jobs are scheduled based on Round Robin scheduling by the scheduler, and their states are updated. This SimpleShell works by interacting with a SimpleScheduler that manages the execution of jobs using signals and shared memory. Each job waits for a signal from the scheduler to begin execution and can be stopped using SIGSTOP. The shell ensures smooth communication with the scheduler and maintains a job queue in shared memory, allowing proper resource management.

Function-wise Explanation

1. is_command_valid(char *input)
    - Purpose: Validates user input by ensuring the command doesn't contain pipes ('|').
    - Returns:
        - true if the command is valid.
        - false if the command contains pipes (since pipes are not allowed).

2. initialize_shared_resources()
    - Purpose: Initializes shared memory and semaphores to manage shared resources between the shell and scheduler.
    - Flow:
        1. Creates a shared memory segment using shm_open for the job queue.
        2. Maps the shared memory segment using mmap.
        3. Creates a second shared memory segment to store the number of jobs.
        4. Opens a semaphore to handle synchronization between processes.

3. cleanup_shared_resources()
    - Purpose: Releases shared memory and semaphore resources during termination.
    - Flow:
        1. Unmaps and unlinks the shared memory.
        2. Closes and unlinks the semaphore to free resources.

4. parse_job_name(char *input)
    - Purpose: Extracts the job name from the user's input.
    - Returns: The extracted job name after the submit keyword.

5. get_current_time_ms()
    - Purpose: Fetches the current time in milliseconds using gettimeofday.

6. Completed_jobs()
    - Purpose: Handles completed jobs by:
        1. It checks if a job has terminated.
        2. It marks it as failed if it exits with a non-zero status.
        3. Updating the job queue by removing completed or failed jobs.
        4. Adds job details to the history.

7. handle_SIGCHLD(int signum)
    - Purpose: A signal handler for SIGCHLD, which is triggered when a child process (job) terminates.
    - Flow: Calls Completed_jobs() to handle the termination.

8. print_job_info()
    - Purpose: Displays the command history and detailed information about executed jobs by iterating over
    - the shared memory queue.

9. add_job_to_queue(int pid, const char *job_name)
    - Purpose: Adds a new job to the queue (in shared memory).
    - Flow:
        1. Allocates memory for a new command object.
        2. Initializes the job's properties (e.g., PID, status, start time).
        3. Adds the job to the end of the queue in shared memory.

10. submit_job(char *job_name)

- Purpose: Forks a new child process to execute the submitted job.
- Flow:
    1. Forks a new process.
    2. In the Child process:
        - Adds itself to the queue.
        - Pauses until resumed by the scheduler.
        - Executes the job using execvp.
    3. In the Parent process: Manages foreground jobs and updates the job count.

11. terminate_shell(int sig)
- Purpose: Handles shell termination (on SIGINT), cleans up resources, and exits.

12. shell_loop()
- Purpose: Runs the interactive shell and processes user commands.
- Flow:
    1. Displays a command prompt (SimpleShell$).
    2. Accepts user input using getline().
    3. Validates the input:
        - If the command is submit <job_name>, it calls submit_job().
        - If the command is exit, it terminates the shell.
    4. On exit, calls terminate_shell() to clean up resources.

13. main(int argc, char *argv[])
- Purpose: Initializes resources, forks the scheduler, and starts the shell.
- Flow:
    1. Validates input arguments (NCPU and TSLICE).
    2. Initializes shared resources.
    3. Forks the scheduler process.
    4. Scheduler process: Executes the SimpleScheduler.
    5. Shell process: Starts the shell_loop() to accept user commands.

14. addToHistory(char* name, pid_t pid, int wait_time, int comp_time)

- Purpose: Once the jobs are completed, they are added to a list of all the completed jobs.
- Flow:
    1. It allocates the memory for a job.
    2. Then, it initializes the job.
    3. The job is then added to the list and its head and tail pointers are updated.

15. handle_SIGINT(int sig)

- Purpose: To handle the SIGINT(Ctrl+c) signal for shell termination.
- Flow:
    1. The handler is set up and declared in the shell_loop so that whenever a ctrl+c command is input the signal handler i.e. handle_SIGINT is called.
    2. The handler terminates the shell_loop and checks the job status and if all jobs are complete. The stats are printed for all the jobs.
    3. Then the scheduler is terminated by sending a signal(SIGTERM) using kill().
    4. All the shared resources are cleaned up.

This implementation of SimpleScheduler works closely with SimpleShell to manage job scheduling using shared memory, signals, and semaphores.

Functions in SimpleScheduler

1. reconnect_resources_after_exec()

- Purpose: It re-establishes shared memory and resources if the child process uses exec() (as it overwrites the current process).
- Flow:
    1. Opens shared memory for the job queue using shm_open.
    2. Maps the memory using mmap to make it accessible.
    3. Reattaches the shared memory segment storing the number of jobs using shmget and shmat.

2. cleanup_shared_resources()

- Purpose: Cleans up shared memory and semaphores when the scheduler is terminated.
- Flow:
    1. Unmaps the shared memory and closes the shared memory file descriptor.
    2. Unlinks the shared memory and semaphores to release the resources.

3. get_current_time_ms()

- Purpose: Gets the current time in milliseconds using gettimeofday.
- Use: Helps track job start, end, and completion times for scheduling.

4. is_completed(pid_t pid)

- Purpose: Checks whether a process has completed execution.
- Flow:
    1. Uses waitpid with WNOHANG to non-blockingly check if the process has exited.
    2. Returns:
        - 1 if the job has completed.
        - 0 otherwise.

5. get_next_job()

- Purpose: Fetches the next READY job from the linked list.
- Flow:
    1. Starts from the head of the job queue and iterates through it.
    2. Returns: A pointer to the first job with status == READY or NULL if no such job exists.

6. move_n_nodes_to_end()

- Purpose: Moves the first NCPU jobs in the queue to the end.
- Flow:
    1. Traverses the queue to identify the first NCPU nodes.
    2. Adjusts the pointers to move these nodes to the end of the list.

7. start_scheduler(int n_cpu, int time_slice)

- Purpose: Manages job scheduling using Round Robin with support for NCPU parallel jobs.
- Flow:
    1. Checks if there are jobs in the queue using (*number_of_jobs).
    2. If jobs exist:
        - Fetches up to NCPU jobs from the front of the queue.
        - Signals them using SIGCONT to resume execution.
        - Waits for the TSLICE duration using usleep().
    3. After TSLICE:
        - Signals the jobs with SIGSTOP to pause execution.
        - Moves the executed jobs to the end of the queue using move_n_nodes_to_end().

8. main()

- Purpose: Entry point for the scheduler.
- Flow:
    1. Validates the input arguments (NCPU and TSLICE).
    2. Reconnects shared memory using reconnect_resources_after_exec().
    3. Starts the scheduling loop by calling start_scheduler().

Code Flow for a command:

Shell Startup:

- SimpleShell initializes shared resources (shared memory and semaphores).
- The shell forks a scheduler process to run SimpleScheduler.

Job Submission:

The user enters the executable along with submit as the prompt.

- submit_job() is called, which:
    1. Forks a child process for the job.
    2. The child process:
        - Adds itself to the shared queue using add_job_to_queue().
        - Pauses itself using SIGSTOP until resumed by the scheduler,.
    3. The parent continues accepting further user inputs.

Scheduler Operation:

- Scheduler enters the infinite loop in start_scheduler().

- It checks if any READY jobs exist in the shared queue.
- NCPU jobs are fetched using get_next_job().
    - If jobs are available:
        - They are resumed using SIGCONT.
        - Scheduler waits for TSLICE duration using usleep().
        - After the TSLICE:
            - Jobs are paused using SIGSTOP.
            - Their burst times are updated.
            - Jobs are moved to the rear of the queue.

Job Completion:

- If a job completes during its TSLICE:
    - It is marked as COMPLETED using is_completed().
    - Its completion and wait times are updated.

Scheduler Sleeping:

- If the job queue becomes empty, the scheduler sleeps using usleep(500000) (500 ms).
- It continuously checks for new jobs until termination.

Shell Termination:

- When the user exits the shell (exit command):
    - terminate_shell() is called to:
        - Terminate the scheduler and all jobs.
        - Cleanup resources (shared memory and semaphores).
    - The shell and scheduler terminate gracefully.

Contribution:
1. Abhinav Kashyap, 2023022: get_next_job(), move_n_nodes_to_end(), start_scheduler(), cleanup_shared_resources(SimpleShell.c), initialize_shared_resources(), handle_SIGCHLD(), add_job_to_queue, submit_job, terminate_shell.

2. Harsh Sharma, 2023233: get_next_job(), move_n_nodes_to_end(), start_scheduler(), cleanup_shared_resources(SimpleScheduler.c), reconnect_resources_after_exec(), handle_SIGINT(), Completed_jobs, submit_job, addToHistory.