CSE 231: Operating System **Assignment-3 (Section-A)**

simple scheduler: A Process Scheduler in C from Scratch

Group Details

Group Member 1:

Name: Himanshu
Roll no: 2023241
Gmail ID : himanshu23241@iiitd.ac.in

Group Member 2 :

Name: Harsh Gupta
Roll no : 2023229
Gmail ID : harsh23229@iiitd.ac.in

Contribution

Himanshu :

1) Function to add a process to the queue
2) Function to remove a process from the queue
3) Function to check if the queue is empty
4) Handling in assignment 2 code
5) Bash.h and makefile

Harsh Gupta :

1) Main function
2) Dummy main function
3) function to start the process
4) Function for submitting the processes
5) function for handle processed process(remove from queue after executing)
6) main function for process scheduler

Code Documentation

We use our shell code in simpleshell.c with the main function to implement the scheduler. We done this assignment with the bonus part.

=> main function

-We declare the main function in simpleshell.c, it starts our shell named Dam_Shell with the current working directory, using getcwd() to get the current path. Our shell's while loop starts with error-checking if the signal is SIGINT or not using sigint_handler(). If this setup fails, an error message is printed, and the shell exits.

-Reading and Parsing Commands.

-Command Processing

  function process_submit(): This function likely handles submission tasks specific to this Shell.

  Scheduler and Grandchild Process: Forks a new process for the scheduler.

  calls dummy_main(argc, argv);, which presumably executes a specific task.

  error checking for grandchild.

-Command (not "submit" or "sched"), it calls execute_piped_commands(cmd);, likely handling general shell commands with or without pipes.

-Frees allocated memory for cmd and line after each command execution.

=> add_to_queue

  Adds a new process to the ready_run_queue

  Check if the queue is full before adding

  Updates the rear index of the queue in a circular fashion

=> remove_from_queue

  Removes a process from the front of the ready_run_queue

  Check if the queue is empty before removing

  Returns the removed process or a blank process (with pid = -1) if the queue is empty

=> is_queue_empty

  Check if the ready_run_queue is empty by looking at its size.

  Returns 1 if empty, 0 otherwise.

Semaphore Declarations

```
sem_t *mutexLock;
sem_t *itemsFull;
sem_t *itemsEmpty;
```

ncpu — Number of CPUs.

tslice — Time slice in milliseconds for the round-robin scheduler.

=> Process Handling and Scheduling Functions

start_processes():

-Starts up to ncpu processes from the ready_run_queue

-Each process is get back from the queue, marked with a start time, and sent a SIGCONT signal to

resume execution.
process_submit(char **command):
-Adds a new process to the ready_run_queue.
-Uses fork() to create a new process, which stops it immediately using SIGSTOP.
-execvp() then executes the given command in the new process.
-If the fork fails, an error message is displayed.

handle_finished_processes():
-Checks for any finished processes using waitpid() with the WNOHANG option. Once a process finishes, it is removed from the ready_run_queue.
-Calculates metrics like execution time (burst_time) and wait time for the finished process.

process_scheduler():
-Main scheduling loop that runs until ready_run_queue is empty.
-Controls process execution and termination based on semaphores. start_processes() starts processes if there are any ready in the queue.
-Waits for a time slice (tslice) and then calls handle_finished_processes() to check and remove completed processes.

Main Scheduler Initialization
dummy_main(int argc, char **argv): Entry point for the scheduler, initialized with ncpu and tslice and sets up semaphores (mutexLock, itemsFull, itemsEmpty) after that enters an infinite loop calling process_scheduler(), managing processes in a round-robin fashion and cleans up semaphores upon exit.

dummy_main.h
In dummy_main function we implement the code and logic according to the given in the assignment.
Bash.h
In bash.h we write all the structs, functions, headers file and global variables.

Bonus
In the bonus part, we implemented some advanced functionalities, like priority scheduling. Using SimpleScheduler, by allowing the user to specify a priority value in the range 1-4 (e.g., priority value 2) as a command line parameter while submitting the job as follows: **SimpleShell$ submit ./a.out 2**
If the priority is not specified by the user, the job will have the default priority of 1.

Github private repository link:

https://github.com/himanshu23241/Operating_System-CSE231