Schedular

Guide – Mr. Vivek Kumar

Collaborators – Aahan , Kartikeya

Introduction

This documentation provides an overview and explanation of the C code provided. The code represents a simple shell and Simple Schedular program that allows users to run any executable according to schedular scheme.

Code Overview

There is an implementation of shell with multi-CPU process scheduling, process management, and history logging functionality.

Here's an overview of the major components and functionalities in the code:

Data Structures:

1) struct MyQueue and struct pdt_node: These structures define a queue and its nodes to manage processes in the shell.

2) The pdt_node represents a process with various attributes.

3) struct MyQueue_CPU and struct CPU_NODE: These structures are used to implement CPU scheduling.

4) The CPU_NODE represents a CPU core and the process currently running on it.

Shared Memory (shm):

The code uses shared memory (via shm_open and mmap) to store data structures and information that can be accessed and modified by different processes.

shm_t is a structure that encapsulates shared memory and contains queues for managing processes and CPUs, synchronization mechanisms (sem_t), and various other variables.

Signal Handling:

The signal_handler function is defined to handle signals like SIGINT (Ctrl+C). It prints the history of executed commands and cleans up shared memory before exiting.

Process Execution Functions:

Work, launch, and execution_block functions are used to execute a block of commands. They handle forking, creating pipelines, and managing background processes.

The execution history of commands is logged in a file named "history.txt."

CPU Core Management:

CPU cores are managed using a queue (MyQueue_CPU) for running processes. The code schedules processes to run on multiple CPU cores, and each core can run a different process concurrently.

Process Scheduling:

do_execution function simulates process execution on CPU cores for a fixed time slice. It updates the state and execution time of processes.

PROCESS_HANDLER is the main function for process scheduling, which manages the execution of processes in the ready queue, distributes them among CPU cores, and keeps track of their execution.

Submit Processes:

The code allows users to submit processes with associated priorities, and these are added to the ready queue. This is done via the "**submit**" command.

Shared Memory Cleanup:

The code includes a cleanup function (clean_up) to unmap shared memory, close file descriptors, and unlink shared memory objects before exiting.

Main Function:

In the main function, the code initializes shared memory and starts the demon process for scheduling and managing submitted processes.

The user can interact with the shell by entering commands. The shell supports submitting processes, running scripts, and sending signals like Ctrl+C (SIGINT) to terminate the shell gracefully

Working of Basic And Advanced Functionality and Testcase →

For Basic Functionality "make" command in ubuntu terminal will make all the executables ready along with the SimpleSchedular Code and then we just have to run "./schedular $NCPU $TSLICE" (exe will be made by name schedular.exe) and enter the NCPU for which this has to be run and TSLICE that should be used in schedular.

Shell with integrated schedular will just run Use "submit ./pi" commands to put the processes in the queue with a default priority of 1 and this will be stores in a global queue named q and q will store the process details added by the user (here pi is a process)

And after every 30 seconds schedular will run its while loop and run the processes and show the output on the STDOUT.

For advanced Functionality

We Just put a number between 1-4 , user is not allowed to put any number outside this range , this would show error in exec command and the program will not run as expected.

So as our own heuristic we would RUN the processes with low number first .

for example processes in the queue are

| (p1 , 1) | (p2 , 2) | (p3 , 1) | with a single CPU ,

So First p1 will be completed , then p3 and then p2.

As p1 and p3 have the lowest numbers as their priority.

So this is our own heuristic for the effect of priority on the scheduling. If a number is not provided after the submit command then it will by default take its priority as 1. And runs accordingly

Github Repository