Embedded System
Course Project

INSTRUCTOR: DR. Binod Kumar

AIM

To show Scheduling Techniques in RTOS for better analyzing.

Work Done

The current Project is implemented on four common scheduling Techniques in RTOS.

- Earliest Deadline First (EDF)
- Rate Monotonic (RM)
- Round Robin (RR)
- First Come First Serve (FCFS)
- Latest Deadline First (LDF)

First Two algorithms i.e. EDF and RM and LDF are written in Python Language while the other two are implemented using C++ language.

Contributors

The project is done by:

Harpal Sahil Santosh (B19CSE107)

Harshit Gupta (B19CSE108)

How to Execute?

For EDF & RM

Use this command to execute :-

1) python .\EDF.py
2) python .\RM.py

The environment should have normal python installation and along with these following libraries :-

1) Tabulate

   To install use " pip install tabulate "

2) Matplotlib

   To install use " pip install matplotlib "

To run algorithm first give input in the input.txt file in the following format

Task Name with space separated
Execution Time with space separated (ci)
Deadline with space separated (di)
Time Period with space separated (ti)

Example

T1 T2 T3
1 2 3
4 6 12
4 6 12

Then run programs in the same way python programs are executed.

For RR

Use this command to execute :-

3) g++ .\RR.cpp -o .\rr.exe; .\rr.exe
4) python .\rr_gantt_chart.py

Open rr_input.txt file.

In the first line mention 2 space separated integer values: 1) Number of tasks 2) Time quantum.

Then give arrival and burst time for every task. For every task we gave one line containing 2 space separated integer values: 1) Arrival time 2) Burst time

Open RR.cpp file and run the program in the same way c++ programs are executed.

NOTE:

It will generate 2 output files:

- rr_output.txt (Summary of scheduling)
- rr_timeline.txt (Timeline output)

Now to generate gantt chart:

Open rr_gantt_chart.py file & run in the same way python programs are executed.

For FCFS

Use this command to execute :-

1) g++ .\FCFS.cpp -o .\fcfs.exe; .\fcfs.exe
2) python .\fcfs_gantt_chart.py

Open fcfs_input.txt file.

In the first line give one integer value of Number of tasks.

Then give arrival and burst time for every task. For every task we gave one line containing 2 space separated integer values: 1) Arrival time 2) Burst time

Open FCFS.cpp file and run the program in the same way c++ programs are executed.

NOTE:

It will generate 2 output files:

- fcfs_output.txt (Summary of scheduling)
- fcfs_timeline.txt (Timeline output)

Now to generate gantt chart:

Open fcfs_gantt_chart.py file & run in the same way python programs are executed.

For LDF

Use this command to execute :-

1) python .\LDF.py

The environment should have normal python installation and along with these following libraries :-

1) Networkx :- To install use " pip install networkx "

Theory and Algorithm

1) Earliest Deadline First: In this scheduling algorithm, priorities are assigned based on their relative deadlines and deadlines which are closed are assigned higher priority which means they will be executed first. In order to schedule this utilization factor is calculated using this formula.

$$U = \sum_{k=1}^{n} \frac{C_k}{T_k} <= 1$$

Algorithm used to code

As we know here the deadline is dynamic in nature and we always want that process which will have a deadline closest, thus to solve this problem in the fastest way I have used, Heap this will give O(log(n)) deletion and insertion of process and thus will be efficient.

I have taken two heaps, heap1 and heap2, In heap1 we will be storing the active process which needs to be scheduled within the deadline and time-bound, heap2 will store those processes that have been executed for the current deadline but will be executed in the future again when their time comes.

a) Heap1 is storing elements in this form
(deadline, execution_time,task_name)
b) Heap2 is storing elements in this form
(new_deadline, execution_time,task_name)

Two HashMap are also created to maintain the Heap, and instead of letters task_name is used in number and with help of HashMap are converted Back

2) Rate Monotonic : In this scheduling algorithm, priorities are static and are assigned according to their time period and tasks with less period are assigned higher priority and thus executed first. In order to schedule this utilization factor is calculated using this formula.

$$U = \sum_{k=1}^{n} \frac{C_k}{T_k} <= n\,(2^{\frac{1}{n}} - 1)$$

Algorithm used to code

In this Algorithm we know deadline will be static and priority is judged on Time period, relation is lesser the time period more will be the priority. Again i have used Heap and formed two Heaps: Heap1 and Heap2.

In heap1 we will be storing the active process which needs to be scheduled, heap2 will store those processes that have been executed and will come after their period again.

a) Heap1 is storing elements in this form (time_period,entry_time , execution_time,task_name)
b) Heap2 is storing elements in this form (new_entry_time, time_period ,execution_time,task_name)

Two HashMap are also created to maintain the Heap, and instead of letters task_name is used in number and with help of HashMap are converted Back

3) Round Robin :

It is a CPU scheduling algorithm which executes each task that is assigned to the CPU for a fixed time (time quantum) in a cyclic way. It is preemptive in nature as we perform context switching after every time quantum. It solves the problem of starvation by giving equal share of CPU to every process.

Algorithm used to code

First read input from the rr_input.txt and create task objects using a structure named **task**. Sort all the tasks in the increasing order of their arrival time and store them in the waiting queue.

Used a time variable to keep track of time. Everytime remove all the tasks from the waiting queue whose arrival time <= current time and move them to the ready queue. pop the first task of the ready queue and execute it for time = min(time_quantum,remaining_burst).

Check whether the remaining burst time of the current task is equal to zero or not. If it is zero then mark that task as completed and update finish time of that particular task and remove it from the ready queue. Else if the remaining burst is greater than zero then add that task again to the ready queue.

Perform above steps till the ready queue becomes empty or till all tasks are not completed.

4) First come first serve :

It executed a given set of tasks according to their arrival time. The task arrived earlier is executed first irrespective of its burst time. The core idea behind this algorithm is to maintain a FIFO queue. It is non-preemptive in nature.

Algorithm used to code

First read input from the fcfs_input.txt and create task objects using a structure named **task**. Store all the tasks in the array task_list and sort them in the increasing order of their arrival time.

Used a time variable to keep track of time. Every time pick a new unexecuted task and run that task till it is completed. Update its finish time and compute turnaround and waiting time for that task.

Update the timeline according to the task executed at that point. Write the result to the output files.

Results

After executing the algorithm two files are generated, one is a gantt chart of the scheduled processes and other generalized report consisting of information for each second.