Homework 2

Scheduling Simulator

2020310083 손형준

Objective

Goal:

Develop a scheduler that simulates real-time task execution under various scheduling algorithms.

Requirements:

- Single-processor environment and synchronous activation
- Support for prioritization policies: **FCFS, SJF, RM, EDF**
- Both **preemptive** and **non-preemptive** scheduling.

Output:

- Identify deadline misses or confirm success (index or 0).
- Match input and output file lines.

Input and Output file example

Input:

```
3 0.5 0 970 93 970 512 110 512 748 142 748
3 0.5 0 710 53 710 759 260 759 512 42 512
3 0.5 0 111 27 111 987 171 987 319 25 319
3 0.5 0 212 1 212 868 120 868 484 174 484
3 0.5 0 637 156 637 994 16 994 360 86 360
3 0.5 0 972 29 972 626 144 626 384 92 384
3 0.5 0 211 67 211 457 43 457 246 21 246
3 0.5 0 1000 3 1000 935 58 935 316 137 316
3 0.5 0 143 28 143 104 24 104 344 24 344
```

100 task sets

- Number of task, sum of utilization, is explicit deadline are first 3 elements
- Created by HW1 python file

Output:

```
0
0
1
1
0
0
0
0
0
0
0
0
0
```

The task index of first deadline miss

(0 if no deadline miss until 100,000 time units)

Get user input and validate

```python
"""
function to get user input and validate the input arguments
"""


def get_user_input():

    # initialize error messages list to store any validation errors
    error_msgs = []

    # check if the user provided exactly 3 parameters
    if len(sys.argv) != 4:
        print(
            "Usage: python3 2020310083_HW2.py input_file.txt [FCFS|SJF|RM|EDF] [p|np]"
        )
        sys.exit(1)

    # check if the input file exists
    if not os.path.exists(sys.argv[1]):
        error_msgs.append("Error: Input file does not exist.")

    # validate the scheduling algorithm
    if sys.argv[2] not in {"FCFS", "SJF", "RM", "EDF"}:
        error_msgs.append(
            "Error: Invalid scheduling algorithm. Choose from [FCFS|SJF|RM|EDF]."
        )

    # validate the priority type
    if sys.argv[3] not in {"p", "np"}:
        error_msgs.append("Error: Invalid priority type. Choose from [p|np].")

    # print all error messages and exit if any errors exist
    if error_msgs:
        for message in error_msgs:
            print(message)
        sys.exit(1)

    return sys.argv[1], sys.argv[2], sys.argv[3] == "p"
```

Get 3 parameters for scheduling simulation

- Input file name (.txt)
- Scheduling Algorithm
- Preemptive option ('p' for preemptive, 'np' for non-preemptive)

Validation of user input

- If the number of user input parameters != 3, exit with error message
- Check if each input meet the conditions
    A. If input file exist
    B. If scheduling algorithm is among the [FCFS, SJF, RM, EDF]
    C. If preemptive option is among [p, np]
- If not append the error message to the list and print out (exit)

Load task sets from input file

```python
"""
function to load the task sets from the input file
"""


def load_tasks(input_file):
    tasks = []
    with open(input_file, "r") as file:
        for i, line in enumerate(file):
            # skip empty lines
            line = line.strip()
            if not line:
                continue

            # read metadata from the first line
            if i == 0:
                metadata = line.split()[:3]
                num_tasks = int(metadata[0])

            # read task set from the subsequent lines
            data = list(map(int, line.split()[3:]))
            task_set = [
                (data[j * 3], data[j * 3 + 1], data[j * 3 + 2])
                for j in range(num_tasks)
            ]

            # append the task set to the tasks list
            tasks.append(task_set)

    return tasks
```

Using strip(), skip the empty line

Prevent error from trailing new line

Read metadata of task sets from first 3 elements

- Number of task
- Sum of utilization (not used)
- Is explicit deadline (not used)

Map the task set to list and return whole task sets list

TaskScheduler Class: Initialization

```python
class TaskScheduler:
    """
    function to initialize the task scheduler with the given task set and scheduling parameters
    """

    def __init__(self, task_set, scheduling_algorithm, preemptive):

        # scheduling parameters
        self.scheduling_algorithm = scheduling_algorithm
        self.preemptive = preemptive

        # task set parameters
        self.task_set = task_set
        self.num_tasks = len(task_set)
        self.periods = [task[0] for task in task_set]
        self.wcets = [task[1] for task in task_set]
        self.relative_deadlines = [task[2] for task in task_set]

        # initialize runtime parameters
        self.activation_times = [0] * self.num_tasks
        self.absolute_deadlines = self.relative_deadlines[:]
        self.remain_execution_times = self.wcets[:]
        self.next_activations = self.periods[:]
        self.is_active = [True] * self.num_tasks

        # put all tasks in the ready queue (synchronous activation)
        self.ready_queue = [
            (self.get_task_priority(i)[0], i) for i in range(self.num_tasks)
        ]
        heapify(self.ready_queue)
        self.current_task = None
```

Initializes the scheduler with task parameters

- Scheduling parameters
  (scheduling algorithm, preemptive)
- Task set parameters
  (task set, number of tasks, periods WCETs, relative deadlines)
- Runtime parameters - for time 0
  (activation time, absolute deadlines, remain exec time, …)

Sets up the ready queue

- Put all the task in task sets to the ready queue
- Heapify using heapq library with the priority
  (based on scheduling algorithm)
- Put only the task index for efficient memory usage and faster execution

TaskScheduler Class: get task priority

```python
class TaskScheduler:

    """
    function to get the priority of a task based on the scheduling algorithm
    """

    def get_task_priority(self, index):
        if self.scheduling_algorithm == "EDF":
            # min absolute deadline
            return self.absolute_deadlines[index], index
        elif self.scheduling_algorithm == "RM":
            # min period
            return self.periods[index], index
        elif self.scheduling_algorithm == "SJF":
            # min remaining execution time
            return self.remain_execution_times[index], index
        else:  # FCFS
            # earliest activation time
            return self.activation_times[index], index
```

Based on the scheduling Algorithm, get the priority

- Each **EDF, RM, SJF, FCFS** has their own priority
- If the tasks have same priority based on scheduling algorithm, the lower task index is assigned
- This used when heapify ready queue or pushing task to heap

TaskScheduler Class: execute and next period

```python
class TaskScheduler:
    """
    function to update the task parameters for the next period
    """

    def next_period(self, current_time):
        for i in range(self.num_tasks):
            if current_time >= self.next_activations[i]:
                self.activation_times[i] = self.next_activations[i]
                self.absolute_deadlines[i] = (
                    self.activation_times[i] + self.relative_deadlines[i]
                )
                self.remain_execution_times[i] = self.wcets[i]
                self.is_active[i] = True
                self.next_activations[i] += self.periods[i]

                # Add to ready queue with updated priority
                heappush(self.ready_queue, (self.get_task_priority(i)[0], i))

    """
    function to execute a task for one time unit
    """

    def execute_task(self, index):
        self.remain_execution_times[index] -= 1

        # deactivate the task if execution is complete for the current period
        if self.remain_execution_times[index] <= 0:
            self.is_active[index] = False
```

Update task information for next period

- For all tasks in task set, check the next activation time is reached
- Update the task information and set the task active status to True
- Push updated tasks to the ready queue with task priority (use get task priority helper func.)

Execute the task for 1 time unit

- Reduce remain execution time with 1
- Change task active status to False if remain execution time reaches 0

TaskScheduler Class: simulate the schedule

```python
class TaskScheduler:
    """
    function to calculate the hyperperiod (LCM of periods) of the task set
    https://labex.io/tutorials/python-calculating-least-common-multiple-13682
    """

    def calculate_hyperperiod(self):
        return reduce(lambda x, y: x * y // math.gcd(x, y), self.periods)

    """
    function to simulate the task scheduler
    ready queue is sorted based on the scheduling algorithm
    """

    def simulate(self):
        current_time = 0
        time_limit = min(self.calculate_hyperperiod(), 100000)

        while current_time < time_limit:
            # Check deadline misses
            missed_deadlines = []
            for i in range(self.num_tasks):
                if self.is_active[i] and current_time >= self.absolute_deadlines[i]:
                    missed_deadlines.append((self.get_task_priority(i)[0], i))

            if missed_deadlines:
                # Sort by priority and return the highest priority task index + 1
                return sorted(missed_deadlines)[0][1] + 1

            # Update tasks at their periods more efficiently
            self.next_period(current_time)

            # Non-preemptive scheduling
            if not self.preemptive:
                self._non_preemptive_step()

            # Preemptive scheduling
            else:
                self._preemptive_step()

            current_time += 1

        return 0
```

Calculate LCM of task periods for early stopping
(compare with 100,000 and use smaller one as time limit)

Simulate the task set scheduling

- Simulate until reaching time limit

1. Each iteration, check if the deadline miss occur
   (if so, return the highest priority task index that deadline is missed and move to the next task set)

2. Each iteration, update the task information whose next activation time is reached

3. Split the case of **preemptive** case and **non-preemptive** case and execute the task

TaskScheduler Class: step based on preemptive

```python
class TaskScheduler:
    """
    Non-preemptive scheduling logic.
    """

    def _non_preemptive_step(self):

        if self.current_task is not None:
            # Execute the current task
            self.execute_task(self.current_task)
            # Reset if task is not active
            if not self.is_active[self.current_task]:
                self.current_task = None
        elif self.ready_queue:
            # Pick the next task from the ready queue
            _, task_index = heappop(self.ready_queue)
            self.current_task = task_index
            self.execute_task(task_index)
            if not self.is_active[self.current_task]:
                self.current_task = None

    """
    Preemptive scheduling logic.
    """

    def _preemptive_step(self):
        # Remove completed tasks from queue
        while self.ready_queue and not self.is_active[self.ready_queue[0][1]]:
            heappop(self.ready_queue)
        if self.ready_queue:
            _, task_index = heappop(self.ready_queue)
            self.execute_task(task_index)
            if self.is_active[task_index]:
                heappush(
                    self.ready_queue,
                    (self.get_task_priority(task_index)[0], task_index),
                )
```

For Non-preemptive case

- **If current task executing task exist**
  - Execute the current task for 1 time unit
  - Check if the current task become inactive, and if so, set the current task to None
- **If current executing task is None**
  - Get the highest priority task from ready queue and set it as current task
  - Execute the current task for 1 time unit
  - Check if the current task become inactive, and if so, set the current task to None

For preemptive case

- If ready queue is empty, skip this time unit
- If not, get the highest priority task from ready queue and execute for 1 time unit
- Check if the task become inactive and if not, push the task in toe ready queue again

TaskScheduler Class: multiprocessing

```python
"""
function to process the task set using the task scheduler
used for multiprocessing
"""


def process_task_set(args):
    task_set, scheduling_algorithm, preemptive = args
    scheduler = TaskScheduler(task_set, scheduling_algorithm, preemptive)
    return scheduler.simulate()
```

```python
# set the arguments for multiprocessing
task_set_args = [
    (task_set, scheduling_algorithm, preemptive) for task_set in task_sets
]

# process the task sets using multiprocessing using imap (efficient memory usage)
results = []
with Pool() as pool:
    for result in pool.imap(process_task_set, task_set_args):
        results.append(result)

# with Pool() as pool:
#     results = pool.map(process_task_set, task_set_args)

# write the results to the output file
with open(filename, "w") as file:
    for result in results:
        file.write(f"{result}\n")
```

Use Pool.imap from multiprocessing

- **imap** is memory efficient for big task set, so use instead of map
    - **imap** returns an iterator, meaning it doesn’t load all results into memory at once)
    - Results for task sets are written to the output file in the same order as the input task sets, maintaining consistency
- Multiprocessing enable the program execute much faster (help to meet the time condition of homework)