Lab # 2

Due: 2024-10-04

By: Omar Abdul, Anas Taimah, Yusuf Khan
300228700, 300228842, 300293842

Course: CSI3120

Group: 40

Code explanation:

Step 1:
Our group started by defining a type called “job” which stores 3 integer variables (start_time, duration, priority).

Step 2:
Created a function which converts the start time hours and minutes input to only being in minutes. It is done by multiplying the # of hours by 60 minutes then by adding the rest of the minutes.

The reverse function takes the total minutes and separates it back into hour minute form for formatting. Done by dividing the total minutes by 60 to get the hours as a whole number and then doing taking the total minutes again and mod 60 to get the remainder of minutes.

Helper function:
The read_int function that is defined asks the user a prompt, reads the input from the user and then returns the input as an integer

Step 3:

A helper function is defined which reads the details of a single job from the user, it uses the read_int function from before to get the input of the start time in hours and minutes, the duration and the priority of the job. The start time is converted into one variable by calling the step 2 function which leaves the three variables that are all returned.

The main step 3 function reads the number of jobs specified by the user (num_jobs) and will call the helper function that many times to read the details of each job. “aux 1 []” is initially called (n = 1) as long as the num_jobs doesn’t equal 0, the base case is not met and instead it calls the helper to get details about job n and then makes a recursive call. The job is added to the accumulator list and n is incremented 1. This recursive function ends once n > num_jobs in which case the accumulator list is returned in reverse order.

Step 4:

Strategy 1:
This function schedules the jobs in a way that prevents any overlap. First it starts by sorting all the jobs in order of start time from earliest to latest using (List.sort). A recursive function is defined with an initial call of “aux [] sorted_jobs” ([] is an empty list which will store the final schedule, and the sorted_jobs are the remaining jobs). If the remaining jobs are empty, return the schedule job list, if not the first job “job” is split from the rest of the list “rest” and the first job is placed in the scheduled job list. The code then recursively checks the last job that was scheduled in the list if its end time ends before the start time of the next job in the remaining list. If it does it is added to the scheduled jobs and if it doesn’t the job is skipped. This stops once there’s no remaining jobs left.

Strategy 2:
This function works the same as strategy 1 except instead of initially sorting the jobs based on start time, it sorts them based on the highest priortity.

Strategy 3:
This function works the same as strategy 1 but schedules jobs to minimize idle times between the jobs. It adds the possible jobs that can fit with no idle after the first job by using (List.Filter). If no jobs are found it schedules the next job in the sorted remaining list. If there is a job in the possible list that can start right after the current end time it is scheduled. The function is recursively called until no remaining jobs are left.

Step 5:
This function returns all the job results in a friendly format:

Scheduled Jobs (*Order the user chose*):
Job scheduled: Start Time = x, Duration = y minutes, Priority = z
Job scheduled: Start Time = x, Duration = y minutes, Priority = z
Job scheduled: Start Time = x, Duration = y minutes, Priority = z

Step 6:
The main function of our program which asks the user the # of jobs as well as the scheduling strategy to use. The default strategy is "No overlaps" if a faulty input is entered. The main function then calls the other functions to return the results.

Test cases:

NO OVERLAP

```
How many jobs do you want to schedule? 3
For job 1, please enter the following details:
- Start Time (hours): 1
- Start Time (minutes): 0
- Duration (minutes): 60
- Priority: 1
For job 2, please enter the following details:
- Start Time (hours): 1
- Start Time (minutes): 30
- Duration (minutes): 15
- Priority: 2
For job 3, please enter the following details:
- Start Time (hours): 3
- Start Time (minutes): 30
- Duration (minutes): 30
- Priority: 3
Choose a scheduling strategy (1 for No Overlaps, 2 for Max Priority, 3 for Minimize Idle Time):
Scheduled Jobs (Minimize Idle Time):
Job scheduled: Start Time = 1:00, Duration = 60 minutes, Priority = 1
Job scheduled: Start Time = 3:30, Duration = 30 minutes, Priority = 3
```

MAX PRIORITY

```
How many jobs do you want to schedule? 3
For job 1, please enter the following details:
- Start Time (hours): 1
- Start Time (minutes): 0
- Duration (minutes): 60
- Priority: 1
For job 2, please enter the following details:
- Start Time (hours): 1
- Start Time (minutes): 30
- Duration (minutes): 15
- Priority: 2
For job 3, please enter the following details:
- Start Time (hours): 3
- Start Time (minutes): 30
- Duration (minutes): 30
- Priority: 3
Choose a scheduling strategy (1 for No Overlaps, 2 for Max Priority, 3 for Minimize Idle Time): 2
Scheduled Jobs (Max Priority):
Job scheduled: Start Time = 3:30, Duration = 30 minutes, Priority = 3
Job scheduled: Start Time = 1:30, Duration = 15 minutes, Priority = 2
```

MINIMINZE IDLE

```
How many jobs do you want to schedule? 3
For job 1, please enter the following details:
- Start Time (hours): 1
- Start Time (minutes): 0
- Duration (minutes): 60
- Priority: 1
For job 2, please enter the following details:
- Start Time (hours): 1
- Start Time (minutes): 30
- Duration (minutes): 15
- Priority: 2
For job 3, please enter the following details:
- Start Time (hours): 3
- Start Time (minutes): 30
- Duration (minutes): 30
- Priority: 3
Choose a scheduling strategy (1 for No Overlaps, 2 for Max Priority, 3 for Minimize Idle Time):
Scheduled Jobs (Minimize Idle Time):
Job scheduled: Start Time = 1:00, Duration = 60 minutes, Priority = 1
Job scheduled: Start Time = 3:30, Duration = 30 minutes, Priority = 3
```