Homework 4b

2021315385
이건 / Gun Daniel Lee

Problem explanation

- EDF Scheduling

- Given 5 processors and a set of tasks, find a correct task-processor mapping such that:
  - Every task is assigned to one of the processors
  - A subset of tasks assigned to each processor satisfies the above condition

Problem explanation

Input:

- Number of tasks (10 <= n <= 30)
- Execution time (C) and inter-release time(T) of each task

Problem explanation

Output:

- The processor number running each task

Solution explanation

- EDF algorithm already prioritizes the job / task with an earlier deadline

- Main focus:

- Sum of C / T for all the tasks in a single processor <= 1

- As long as the main focus is true, the processor will handle the rest

Solution explanation

- Required global variables (adjusted to the problem)

```
#define ONE 1.000000000001
#define MAXTASK 30 //max possible tasks

typedef struct Task
{
  int C, T; //c and t for each task
  double U; //u = c / t
  int count; //user's input order
  int processor_num; //processor number
}Task;

Task tasks[MAXTASK];   //all the task inputs
int task_num; //number of tasks in current case
```

Solution explanation

- Priority function: sort tasks in descending order of C/T

```
//sort task such that U is in descending order
void priority()
{
  for(int i = 0; i < task_num; i++)
  {
    for(int j = i + 1; j < task_num; j++)
    {
      if(tasks[i].U < tasks[j].U)
      {
        Task temp = tasks[i];
        tasks[i] = tasks[j];
        tasks[j] = temp;
      }
    }
  }
}
```

Solution explanation

- Is_task_allowed function

- Boolean if task can be placed in processor

- Must succeed two conditions:

1. Sum including current task <= 1
2. While following EDF standards, task can still reach deadline

Solution explanation

```
//check whether task is allowed
int is_task_allowed(int priority_num, int processor_num)
{
  int idx[priority_num];
  int total_C = 0;
  double total_U = 0;
  int count = 0;
  int cond1 = 1, cond2 = 1;

  for(int i = 0; i <= priority_num; i++)
  {
    if(tasks[i].processor_num == processor_num || i == priority_num)
    {
      idx[count++] = i;
      total_C += tasks[i].C;
      total_U += tasks[i].U;
    }
  }
  if(total_U > ONE)
  {
    cond1 = 0;
  }
```

```
  for(int i = 0; i < count; i++)
  {
    int curr_T = tasks[idx[i]].T;
    int curr_C = tasks[idx[i]].C;
    int edf_exec = 0;

    for(int j = 0; j < count; j++)
    {
      if(tasks[idx[j]].T < curr_T)
      {
        edf_exec += tasks[idx[j]].C;
      }
      else if(tasks[idx[j]].T == curr_T && j != i)
      {
        edf_exec += tasks[idx[j]].C;
      }
    }
    if(curr_T < edf_exec + curr_C)
    {
      cond2 = 0;
      break;
    }
  }

  return cond1 & cond2;
}
```

Solution explanation

- Divide_tasks function

- Splits the tasks into different processors

- Loops the following for all tasks:

1. Finds the processor with minimum total sum of C/T

    *if a sum results to exactly 1.0, this is prioritized

2. calls is_task_allowed function to check whether task can be placed

3. If task cannot be placed at any processor, exit program

Solution explanation

```
void divide_tasks()
{
  for(int i = 0; i < task_num; i++)
  {
    int count = 0;
    int unallowed[5] = {0};
    double minSum = 10000000;
    int minProcessor = 1;
    for(int j = 1; j <= 5; j++)
    {
      int allowed = 1;
      if(count != 0)
      {
        for(int k = 0; k < count; k++)
        {
          if(unallowed[k] == j)
          {
            allowed = 0;
            break;
          }
        }
      }
      if(allowed == 0)
      {
        continue;
      }
```

```
      double sum = 0;
      for(int k = 0; k < i; k++)
      {
        if(tasks[k].processor_num == j)
        {
          sum += tasks[k].U;
        }
      }
      if(sum < minSum)
      {
        minSum = sum;
        minProcessor = j;
      }
      if(sum + tasks[i].U == 1.0)
      {
        minSum = sum;
        minProcessor = j;
        break;
      }
    }
```

```
    if(is_task_allowed(i, minProcessor))
    {
      tasks[i].processor_num = minProcessor;
    }
    else
    {
      if(count == 4)
      {
        printf("Error! Cannot split the tasks properly.\n");
        exit(1);
      }
      unallowed[count++] = minProcessor;
      i--;
    }
  }
}
```

Solution explanation

- Main function

- Get inputs
- Call priority and divide_tasks function
- Print results

Solution explanation

```c
int main()
{
  scanf("%d", &task_num);

  for(int i = 0; i < task_num; i++)
  {
    scanf("%d %d", &tasks[i].T, &tasks[i].C);

    tasks[i].U = (double)tasks[i].C / (double)tasks[i].T;
    tasks[i].count = i + 1;
  }

  priority();
  divide_tasks();
```

```c
  for(int i = 0; i < task_num; i++)
  {
    for(int j = 0; j < task_num; j++)
    {
      if(i + 1 == tasks[j].count)
      {
        printf("%d ", tasks[j].processor_num);
        break;
      }
    }
  }
  return 0;
}
```

Thank you!