<u>OS Project</u>

Student Name : Debasis Jana
Student ID : 11701186
Email Address: debasisjana93@gmail.com
GitHub Link : https://github.com/deba19/OS_Project

===

1. Question Description

- **Question No 1:**
  Considering 4 processes with the arrival time and the burst time requirement of the processes the scheduler schedules the processes by interrupting the processor after every 3 units of time and does consider the completion of the process in this iteration. The schedulers then checks for the number of processes waiting for the processor and allots the processor to the process but interrupting the processor after every 6 units of time and considers the completion of the process in this iteration. The scheduler after the second iteration checks for the number of processes waiting for the processor and now provides the processor to the process with the least time requirement to go in the terminated state.The inputs for the number of requirements, arrival time and burst time should be provided by the user.Consider the following units for reference.

Process	Arrival time	Burst time
P1	0	18
P2	2	23
P3	4	13
P4	13	10

  Develop a scheduler which submits the processes to the processor in the above defined scenario, and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time.

- **Question No 28.**
  Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Write a program in C which will print the mapping of processes with memory partitions for both the algorithms.

===

2. Algorithm:

- **Question No1.**

```
1- Create an array rem_bt[] to keep track of remaining
  burst time of processes. This array is initially a
  copy of bt[] (burst times array)
2- Create another array wt[] to store waiting times
  of processes. Initialize this array as 0.
3- Initialize time : t = 0
4- Keep traversing the all processes while all processes
   are not done. Do following for i'th process if it is
   not done yet.
    a- If rem_bt[i] > quantum
       (i)  t = t + quantum
       (ii) bt_rem[i] -= quantum;
    c- Else // Last cycle for this process
(i)  t = t + bt_rem[i];
(ii) wt[i] = t - bt[i]
(ii) bt_rem[i] = 0; // This process is over
```

- **Question No 28.**

For BestFit:

```
1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the
   minimum block size that can be assigned to
   current process i.e., find min(bockSize[1],
   blockSize[2],.....blockSize[n]) >
   processSize[current], if found then assign
   it to the current process.
4- If not then leave that process and keep checking
   the further processes.
```

For FirstFit:

```
1- Input memory blocks with size and processes with size.
2- Initialize all memory blocks as free.
3- Start by picking each process and check if it can
   be assigned to current block.
4- If size-of-process <= size-of-block if yes then
   assign and check for next process.
5- If not then keep checking the further blocks.
```

For Worst Fit:

```
1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the
   maximum block size that can be assigned to
   current process i.e., find max(bockSize[1],
   blockSize[2],.....blockSize[n]) >
   processSize[current], if found then assign
   it to the current process.
4- If not then leave that process and keep checking the further process.
```

==

3. Description (Purpose of Use):

- **Question No 1. :**

CPU scheduling algorithm is a process where each process is assigned a fixed time slot in a cyclic way.

- It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
- One of the most commonly used technique in CPU scheduling as a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.

- **Question No 28.:**

The concept is of Memory Patition.

When there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.
Below are the various partition allocation schemes :

1. First Fit: In the first fit, the partition is allocated which is first sufficient block from the top of Main Memory.

2. Best Fit Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.

3. Worst Fit Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.

---

4. Given Test Cases:

- **Question No. 1:**

Process	Arrival time	Burst time
P1	0	18
P2	2	23
P3	4	13
P4	13	10

- **Question No. 28:**

Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600KB

Processes Size : 212 KB, 417 KB, 112 KB, and 426 KB (in order)

---

5. Code snippet:

- **Question No. 1 :**

  Refer this link

  https://github.com/deba19/OS_Project/blob/master/scheduler.c

- **Question No. 28 :**

  Refer this link:

  https://github.com/deba19/OS_Project/blob/master/FirstFit_BestFit_WorstFit.c

Thank You