Reading Input Data for Process Scheduling

Here, you can see some suggestions on how to read scheduling input data for your scheduling lab assignment.

Sample Input Files:

SJF:

Filename: `sjf_input.txt`	**What does each line mean**
5 0 2 3 4 5 5 2 7 4 5	Total number of processes Arrival times in sequential order Burst times in sequential order

Round Robin:

Filename: `rr_input.txt`	**What does each line mean**
4 20 53 17 68 24	Total number of processes Time Quantum Burst times in sequential order

Priority:

Filename: `priority_input.txt`	**What does each line mean**
5 0 14 3 9 7 15 5 10 22 16 2 4 0 3 1	Total number of processes Arrival times in sequential order Burst times in sequential order Priorities in sequential order

We only need to utilize these functions to read input data: **`fopen, fscanf, fclose`**

Here is how it is done for SJF scheduling:

1. First, we open the input file with read access:

```
FILE* fp = fopen("sjf_input.txt", "r");
```

2. The first value in the value is the **process count**, we read it using **`fscanf`** with a **`%d`** format specifier (since all values are integers):

```
int process_count = 0;
fscanf(fp, "%d", &process_count);
```

3. We need a **structure** to store individual process data called **Process** (This will hold relevant data for each process, like *arrival time, burst time*. And other stuff like *remaining burst time, finish time, turnaround time, waiting time* etc.). Assuming you have already written the struct, we make any **array of that structure** with a length of **process count**. Remember, index 0 stores process info related to P1, index 1 stores P2, and so on and so forth.

```
struct Process processes[process_count];
```

4. For the SJF input data txt file, the next line after process count contains the **arrival time** of each process in sequential order (we don't need to do anything to shift to the next line, **`fscanf`** does it for you). As there are multiple values in this line whose count equates to **process count**, we call **`fscanf`** in a *for loop* **process count** number of times, and in each iteration, **store the value** we just have read to the corresponding process's **arrival time** property in each slot in the processes array.

```
for (int i = 0; i < process_count; i++) {
    int temp;
    fscanf(fp, "%d", &temp);
    processes[i].arrival_time = temp;
}
```

5. The line after that contains **burst time**. Do the same for this but now store into the **burst time** property of each process.

```
for (int i = 0; i < process_count; i++) {
    int temp;
    fscanf(fp, "%d", &temp);
    processes[i].burst_time = temp;
}
```

You probably now have gotten the idea on how to read process input data. Now do the same for **round robin** and **priority** scheduling as well by analyzing their input file structure. And don't forget to close the file after reading:

```
fclose(fp);
```