Dominic Groshong
May 15, 2019
CS365 Operating Systems

CPU Scheduling

This lab report satisfies all tasks given.

Creation

I created my CPU scheduler simulation using C#. As an object-oriented language, the first step was to create the process class (Fig 1.). This class would contain all the variables for keeping track of the various process states, namely: the burst, arrival, completion, start, turnaround, waiting, and response time for each process.

Next, I created several helper methods for generating random numbers (Fig 2.) to use for the burst and arrival time during the creation of each process. As stated in the lab document, the minimum burst time was 2ms, and as such, our maximum time was 42ms. Thus in process creation, I used the random methods for each process creation for assigning the burst and arrival values, process creation happened up to N times (Fig. 3)

The next step was to create a scheduling simulation for SJF and FCFS. To start since FCFS and SJF only differ in the ordering of when jobs are performed, I started by creating a Scheduler method (Fig. 4) which would handle all my calculations on any IEnumerable passed into the method. Then for each item in that IEnumerable, I calculated the correct values for each variable using the descriptions from class. Once they were calculated, I could use the numbers to find the averages for the turnaround time, throughput, and CPU utilization. Once that was done, I used a print method for printing each item from the sorted list and the resulting averages.

Using Linq, I sorted the Process list so they would be ordered by the Arrival variable for FCFS and created another IEnumerable in which I sorted the same list by Burst time. Once each was sorted, I could pass the resulting IEnumerable into the scheduler method. The results can be seen in Figure 5 & 6.

```
class Process
{
    // Process Variables
    2 references
    public string Name { get; set; }
    9 references
    public int Burst { get; set; }
    11 references
    public int Arival { get; set; }
    11 references
    public int Completion { get; set; }
    7 references
    public int Start { get; set; }

    // Scheduling Criteria
    7 references
    public int Turnaround { get; set; }
    3 references
    public int Waiting { get; set; }
    3 references
    public int Response { get; set; }
}
```

Figure 1: Process Class

```
// Generate a random number up to a maximum
1 reference
public int Rand(int max)
{
    Random random = new Random();
    return random.Next(max);
}

// Generate a random number between two numbers
1 reference
public int Rand(int min, int max)
{
    Random random = new Random();
    return random.Next(min, max);
}
```

Figure 2: Random Number Helpers

```
public List<Process> createProcess(int time)
{
    // create list of jobs to add processes to.
    List<Process> jobs = new List<Process>();

    for (int i = 0; i < time; i++)
    {
        Process p = new Process
        {
            Name = "P" + (i + 1),
            Burst = Rand(2, 42),
            Arival = Rand(time)
        };
        jobs.Add(p);
    }

    return jobs;
}
```

Figure 3: Create Processes

Dominic Groshong
May 15, 2019
CS365 Operating Systems

```
public void Scheduler(IEnumerable<Process> processes, int time)
{

    // Varables
    var turnaroundCount = 0;
    var burstCount = 0;
    // Title
    Console.WriteLine("Shortest Job First: \n");
    // Algorithem start time
    int startTime = processes.First().Arival;

    foreach (var item in processes)
    {

        // Preform scheduling on processes
        item.Start = startTime;
        item.Completion = item.Start + item.Burst;
        item.Turnaround = item.Completion - item.Arival;
        item.Waiting = item.Turnaround - item.Burst;
        item.Response = item.Start - item.Arival;
        startTime = item.Completion + item.Arival;

        turnaroundCount += item.Turnaround;
        burstCount += item.Burst;

    }

    // Calculate Averages / CPU
    double averageTurnaround = (double)(turnaroundCount / processes.Count());
    double throughput = (double)(processes.Last().Completion / processes.Count()) / 1000;
    double CPU = (double)burstCount / processes.Last().Completion;

    // Print statements
    PrintList(processes);
    Console.WriteLine("\n");
    Console.WriteLine("CPU Utilization: " + CPU);
    Console.WriteLine("Average Thoughput: " + throughput + " seconds");
    Console.WriteLine("Average Turnaround: " + averageTurnaround + "\n");
}
```

Figure 4: Scheduler Method

Dominic Groshong
May 15, 2019
CS365 Operating Systems

First Come First Serve:

Process	Arival	Burst	Start	Finish	Turnaround	Waiting	Response
P6	2	37	2	39	37	0	0
P8	3	30	41	71	68	38	38
P5	4	19	74	93	89	70	70
P3	6	20	97	117	111	91	91
P4	7	37	123	160	153	116	116
P10	7	20	167	187	180	160	160
P1	9	20	194	214	205	185	185
P2	9	17	223	240	231	214	214
P7	9	38	249	287	278	240	240
P9	9	4	296	300	291	287	287

CPU Utilization: 0.806666666666667
Average Thoughput: 0.03 seconds
Average Turnaround: 164

Figure 6: FCFS Results

Shortest Job First:

Process	Arival	Burst	Start	Finish	Turnaround	Waiting	Response
P9	9	4	9	13	4	0	0
P2	9	17	22	39	30	13	13
P5	4	19	48	67	63	44	44
P1	9	20	71	91	82	62	62
P3	6	20	100	120	114	94	94
P10	7	20	126	146	139	119	119
P8	3	30	153	183	180	150	150
P4	7	37	186	223	216	179	179
P6	2	37	230	267	265	228	228
P7	9	38	269	307	298	260	260

CPU Utilization: 0.788273615635179
Average Thoughput: 0.03 seconds
Average Turnaround: 139

Figure 5: SJF Results

Results

By analyzing the results we can see that the shortest job first was better across the board, from CPU Utilization and Turnaround time both being shorter then FCFS. The throughputs where the same on this particular test and when running it with other processes Throughput was generally within a few ms of each other. As we know SJF is an provably the optimal solution, providing the shortest wait times which we can see by looking at the waiting column, The downsides of SJF being the time taken by a process must be known by the CPU beforehand, which is not possible, plus longer processes will have more waiting time, eventually leading to starvation of these processes.