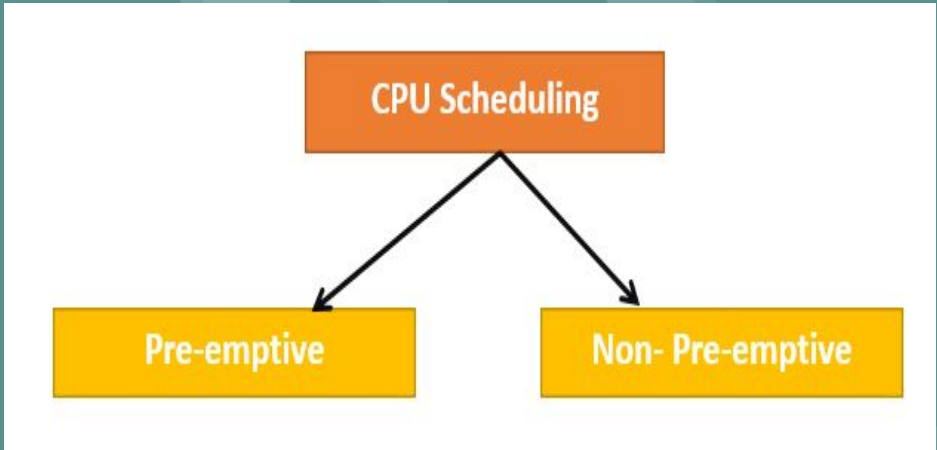
CS4310.01: Group Assignment #2



Benjamin Mai
Minh Huynh
Bao Nguyen
4/09/2021 (Submitted Early)

Assignment Task

We are being asked to simulate OS scheduling algorithms that include:

1.) FCFS (first come first serve)
2.) SJF (shortest job first)
3.) SRT (shortest remaining time first)

Coding Implementation

- We have a int variable K that represents all processes that arrive within time [0-k] uniform random
- A double variable D represents: CPU times of processes have average D
- A double variable V represents the sigma V from a Gaussian distribution
- A int variable N represents the number of processes

Coding Implementation

- We have a boolean variable = false initially, but users can set it to true to see more output
- Create 3 copies of the same system that are represented by a 2D array
- For the parameters of K, D, V, and N, we take those from user input
- If there are no valid parameters, we print out "One or more simulation params are invalid"

Coding Implementation

- Once all the parameters are valid, we initialize the system: One for First Come First Serve, Shortest Job First, and Shortest Remaining Time
- After this, we want to see if the simulation parameters are valid:

  D, V, N > 0, K > 0

- We return true if parameters are valid, otherwise we return false

Coding Implementation

- Initialize the system based on the simulation params
- For each of the N processes; the system will have the following info:

  Active : 0/1 (1 at process arrival and 0 on termination)

  Arrival Time : Uniform Random in [0, K)

  Total CPU time : CPU time with mean D, deviation V (gaussian distribution)

  Turnaround time : Finish time - arrival time (computed at the end)

  Left CPU time : initially Total CPU time

Coding Implementation

Deriving n total CPU times with Gaussian Distribution:

- int totalCPUTime = (int) (rand.nextGaussian()*V + D) + 1;
- We add 1 so the CPU time is never 0
- Each turnaround time will be computed after the simulation

FCFS Algorithm

- Check if any process arrived?
- If multiple processes arrive at the same time, the one with smaller i goes first
- If no process is running, check FIFO queue. Otherwise, no process has arrived yet, so continue on
- Current process runs for 1 time step, then decrement remaining time
- If the process is finished, find the turnaround time and consider it inactive
- For anyRemaining method, it returns true if any Ri != 0 in the system, otherwise return false

SJF Algorithm

```java
// SJF: the shortest job always runs first (non-preemptive)
private static void simulateSJF() {
    int t = 0;
    int currentProcess = -1;
    LinkedList<Integer> queue = new LinkedList<>();
    int[][] system = system2;

    while (anyRemaining(system)) {
        if (t > 100) break;
        // check if any process arrived?
        for (int i = 0; i < N; i++) {
            int arrTime = system[i][1];
            if (t == arrTime) {
                queue.addLast(i);
            }
        }

        // SJF: select the job with shortest CPU time
        if (currentProcess < 0) {
            if (!queue.isEmpty()) {
                int leastCPU = Integer.MAX_VALUE;
                int shortestJob = -1;
                for (int p : queue) {
                    int CPU = system[p][2];
                    if (CPU < leastCPU) {
                        shortestJob = p;
                        leastCPU = CPU;
                    }
                }
                if (shortestJob != -1) {
                    currentProcess = shortestJob;
                    queue.removeFirstOccurrence(shortestJob);
                }
            } else {
                // no process has arrived yet
                t++;
                continue;
            }
        }

        if (DEBUG) {
            System.out.printf("Time: %d Process: %d\n", t, currentProcess);
        }

        // current process runs for 1 time step
        system[currentProcess][3]--;                    // decrement remaining time

        if (system[currentProcess][3] == 0) {           // process finished?
            int arrTime = system[currentProcess][1];
            system[currentProcess][4] = t - arrTime + 1; // turnaround time
            system[currentProcess][0] = 0;              // inactive
            currentProcess = -1;
        }
        t++;
    }
}
```

SRT Algorithm

```java
// SRT: the process with least remaining time runs next (preemptive)
    private static void simulateSRT() {
        int t = 0;
        int currentProcess = -1;
        LinkedList<Integer> queue = new LinkedList<>();
        int[][] system = system3;

        while (anyRemaining(system)) {

            // check if any process arrived?
            for (int i = 0; i < N; i++) {
                int arrTime = system[i][1];
                if (t == arrTime) {
                    queue.addLast(i);
                }
            }

            // this is preemptive: at every time step chose the
            // process with the least rem. time.
            int leastRemTime = Integer.MAX_VALUE;
            int leastRemJob  = -1;
            for (int p : queue) {
                int remTime = system[p][3];
                if (remTime < leastRemTime) {
                    leastRemTime = remTime;
                    leastRemJob = p;
                }
            }
            if (leastRemJob >= 0) {
                queue.removeFirstOccurrence(leastRemJob);

                if (currentProcess >= 0 && currentProcess != leastRemJob) {
                    // currently running process got preempted
                    queue.addLast(currentProcess);
                }
                currentProcess = leastRemJob;
            }

            if (currentProcess < 0) {
                // processes haven't arrived yet
                t++;
                continue;
            }

            if (DEBUG) {
                System.out.printf("Time: %d Process: %d\n", t, currentProcess);
            }

            // current process runs for 1 time step
            system[currentProcess][3]--;                    // decrement remaining
                time

            if (system[currentProcess][3] == 0) {           // process finished?
                int arrTime = system[currentProcess][1];
                system[currentProcess][4] = t - arrTime + 1; // turnaround time
                system[currentProcess][0] = 0;              // inactive
                currentProcess = -1;
            t++;
        }
    }
```

Coding Implementation: Printing the Output

```java
private static void printResults(String algo, int[][] system) {
        int total = 0;
        System.out.printf("\n**** %s ****\n", algo);
        System.out.println("No.| A| Arr| CPU| Rem| TT |");
        System.out.println("---------------------------");
        for (int i = 0; i < N; i++) {
            int active = system[i][0];
            int arrTim = system[i][1];
            int CPUTim = system[i][2];
            int remTim = system[i][3];
            int TTTime = system[i][4];
            total += TTTime;
            System.out.printf("%-3d %1d %4d %4d %4d %4d\n", i, active, arrTim, CPUTim,
                remTim, TTTime);
        }
        System.out.printf("Average TT: %.4f\n", total * 1.0 / N);
    }
```

```
Enter max arrival time for all processes (k): 5
Enter Avg. CPU time (D): 4
Enter Std Dev. of CPU times (V): 2
Enter number of processes (N): 5

**** FCFS ****
No.| A| Arr| CPU| Rem| TT |
---------------------------
0   0    4    2    0   29
1   0    3    9    0   15
2   0    1    8    0    8
3   0    3    6    0   21
4   0    3    7    0   28
Average TT: 20.2000

**** SJF ****
No.| A| Arr| CPU| Rem| TT |
---------------------------
0   0    4    2    0    7
1   0    3    9    0   30
2   0    1    8    0    8
3   0    3    6    0   14
4   0    3    7    0   21
Average TT: 16.0000

**** SRT ****
No.| A| Arr| CPU| Rem| TT |
---------------------------
0   0    4    2    0    3
1   0    3    9    0   30
2   0    1    8    0   18
3   0    3    6    0   11
4   0    3    7    0   25
Average TT: 17.4000
```

Sample Simplified Output

Sample Full Output (Next Slide)

```
Enter max arrival time for all processes (k): 5
Enter Avg. CPU time (D): 4
Enter Std Dev. of CPU times (V): 2
Enter number of processes (N): 5
Time: 2 Process: 1
Time: 3 Process: 1
Time: 4 Process: 1
Time: 5 Process: 1
Time: 6 Process: 2
Time: 7 Process: 2
Time: 8 Process: 2
Time: 9 Process: 2
Time: 10 Process: 2
Time: 11 Process: 3
Time: 12 Process: 3
Time: 13 Process: 3
Time: 14 Process: 3
Time: 15 Process: 3
Time: 16 Process: 4
Time: 17 Process: 4
Time: 18 Process: 4
Time: 19 Process: 4
Time: 20 Process: 4
Time: 21 Process: 4
Time: 22 Process: 4
Time: 23 Process: 0
Time: 24 Process: 0
Time: 25 Process: 0
Time: 26 Process: 0
Time: 27 Process: 0
Time: 2 Process: 1
Time: 3 Process: 1
Time: 4 Process: 1
Time: 5 Process: 1
Time: 6 Process: 2
Time: 7 Process: 2
Time: 8 Process: 2
Time: 9 Process: 2
Time: 10 Process: 2
Time: 11 Process: 3
Time: 12 Process: 3
Time: 13 Process: 3
Time: 14 Process: 3
Time: 15 Process: 3
Time: 16 Process: 0
Time: 17 Process: 0
Time: 18 Process: 0
Time: 19 Process: 0
Time: 20 Process: 0
Time: 21 Process: 4
Time: 22 Process: 4
Time: 23 Process: 4
Time: 24 Process: 4
Time: 25 Process: 4
Time: 26 Process: 4
Time: 27 Process: 4
Time: 2 Process: 1
Time: 3 Process: 2
Time: 4 Process: 1
Time: 5 Process: 2
Time: 6 Process: 1
Time: 7 Process: 2
Time: 8 Process: 1
Time: 9 Process: 2
Time: 10 Process: 3
Time: 11 Process: 2
Time: 12 Process: 3
Time: 13 Process: 0
Time: 14 Process: 3
Time: 15 Process: 0
Time: 16 Process: 3
Time: 17 Process: 0
Time: 18 Process: 3
Time: 19 Process: 0
Time: 20 Process: 4
Time: 21 Process: 0
Time: 22 Process: 4
Time: 23 Process: 4
Time: 24 Process: 4
Time: 25 Process: 4
Time: 26 Process: 4
Time: 27 Process: 4

**** FCFS ****
No.| A| Arr| CPU| Rem| TT |
---------------------------
0   0    4    5    0   24
1   0    2    4    0    4
2   0    2    5    0    9
3   0    2    5    0   14
4   0    2    7    0   21
Average TT: 14.4000

**** SJF ****
No.| A| Arr| CPU| Rem| TT |
---------------------------
0   0    4    5    0   17
1   0    2    4    0    4
2   0    2    5    0    9
3   0    2    5    0   14
4   0    2    7    0   26
Average TT: 14.0000

**** SRT ****
No.| A| Arr| CPU| Rem| TT |
---------------------------
0   0    4    5    0   18
1   0    2    4    0    7
2   0    2    5    0   10
3   0    2    5    0   17
4   0    2    7    0   26
Average TT: 15.6000
```

D vs D/ATT Test Values

Row 1 - FCFS

Row 2 - SJF

Row 3 - SRT

D value: 4

D	ATT	D/ATT
4	14.4	0.278
4	14	0.286
4	15.6	0.256

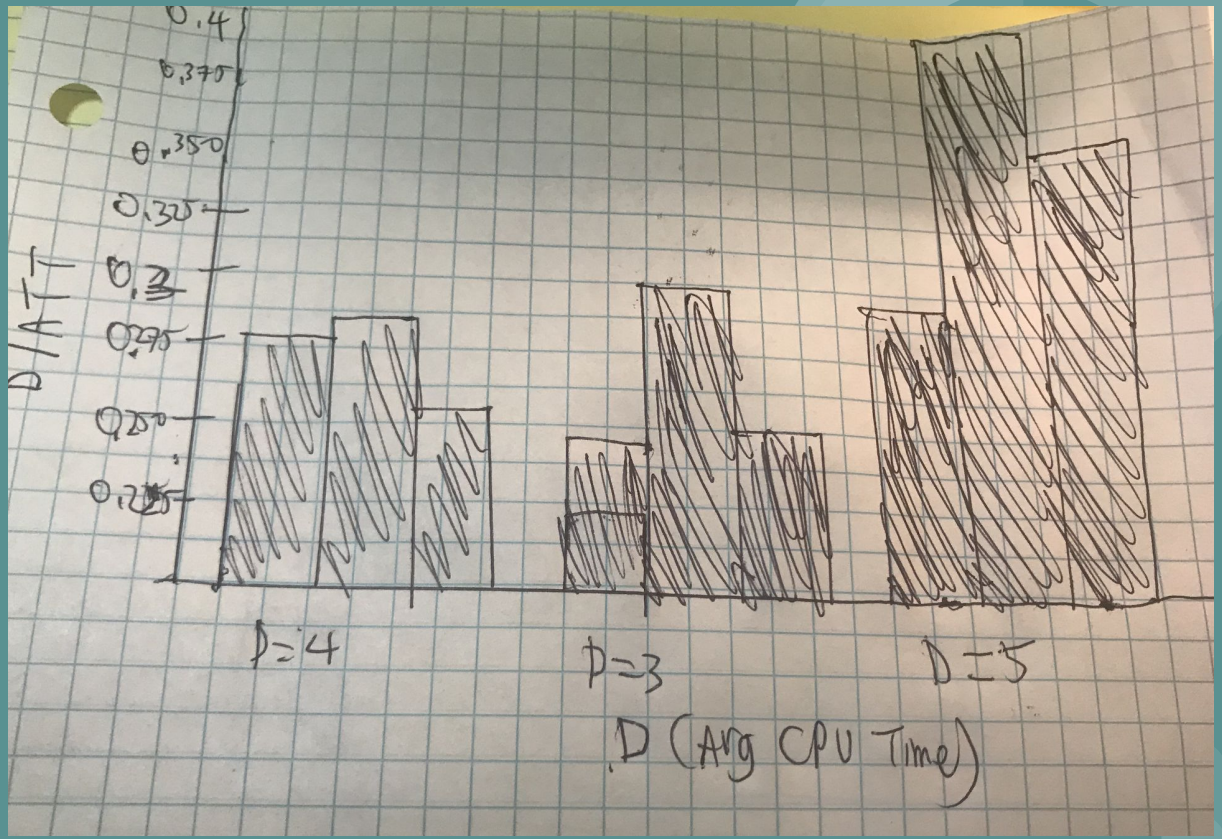
D value 3:

D	ATT	D/ATT
3	12.2	0.246
3	10	0.3
3	12	0.25

D value 5:

D	ATT	D/ATT
5	17.2	0.2907
5	11.8	0.4237
5	13.8	0.3623

D vs D/ATT Graph



Learning Outcome

- We learned how to apply OS scheduling theory to practical usage with the code
- Competition for the CPU slows down all the processes
- Learned how to simulate the different processes at different times
- How to collaborate and debug for issues within a short period of time

Thank you for listening, Professor!