Operating System Project

University ID	Students	Section
	Abdulrahman Alfrihidi	CS1
	Ali Abdullah Al-Hebshi	CS1
	Sultan Shaikh Omar	CS1

Prepared to Prof. Asaad

Project Overview

Introduction

This group project focuses on designing and implementing a modular Operating Systems Scheduling Simulator. The project highlights CPU scheduling, job admission, memory allocation, and event handling using object-oriented programming (Java). The simulator models core OS behavior through queues (like holdQ1, holdQ2, readyQ), schedulers (FCFSScheduler, DynamicRRScheduler), and event-driven execution (managed by SimulationController and Main).

Learning Objectives

- Apply OS scheduling and memory management concepts in a practical project.
- Develop a modular and object-oriented software architecture in Java (e.g., PrManager, OtherKerServices).
- Design and analyze scheduling strategies such as **Dynamic/Static RR** and **FCFS** .

Key Features

- **Dynamic and FCFS CPU Scheduling**: Implements DynamicRRScheduler (which calculates quantum based on remaining work) and FCFSScheduler (which runs processes to completion). The PrManager can switch between them.
- **Job admission Using Hold Queues HQ1 and HQ2**: PrManager uses two hold queues. holdQ1 is a PriorityQueue for P=1 jobs, sorted by requested memory. holdQ2 is a FIFO Deque for P=2 jobs.
- **Memory allocation with pre-allocation policy**: OtherKerServices tracks total and available memory. PrManager only admits jobs from hold queues if their memory (reqMemory) and device (reqDevices) needs can be met.
- **Event-driven simulation (internal vs external events)**: The Main class reads external events (like 'A' for arrival, 'S' for scheduler switch) from input.txt. PrManager manages internal events by calculating nextInternalEvent (time slice end or process completion).
- **Display and statistics reporting**: PrManager maintains a List<FinishedRecord> to track turnaround time and weighted turnaround time for completed processes.

System Design

Component Structure

- **SimulationController**: (Inferred from Main.java) Manages the main simulation loop, parses events from the input file, and advances the simulation clock. It delegates tasks to PrManager and OtherKerServices.
- **PrManager**: Handles the complete process lifecycle, manages all queues (readyQ, holdQ1, holdQ2), selects the active CPU scheduler, and handles CPU dispatching.
- **OtherKerServices**: Manages system resources, specifically tracking availMemory and availDevices and providing methods to allocate and deallocate them.
- **Scheduler**: (Inferred from implementations) An interface implemented by DynamicRRScheduler and FCFSScheduler that provides a unified chooseQuantum method.

Data Structures

- **Submit Queue**: Represented by the input.txt file, which provides a timed sequence of jobs for admission.
- **Hold Queue 1 (HQ1)**: A PriorityQueue<Job> in PrManager. It sorts P=1 jobs ascending by reqMemory, using a serial for FIFO tie-breaking.
- **Hold Queue 2 (HQ2)**: An ArrayDeque<Job> in PrManager. It holds P=2 jobs in FIFO order.
- **Ready Queue (readyQ)**: An ArrayDeque<Process> in PrManager. It holds processes that have been allocated resources and are waiting for the CPU in FIFO order.
- **Process Table**: This role is fulfilled by the combination of the running process variable, the readyQ, holdQ1, holdQ2, and the finished list, all managed within PrManager.
- **Memory Counters**: Maintained inside OtherKerServices, which tracks totalMemory and availMemory.

Final Results Tables

Below are the final "Finished Jobs" tables from the simulation runs for FCFS, Static RR, Dynamic RR, and a combination of all.

FCFS

(Sorted by Job ID)

Job ID	Burst Time	ArrivalTime	CompleteTime	TurnaroundTime	WeightedTime	WaitingTime
1	8	10.00	18.00	8.00	1	0.00
3	5	15.00	23.00	8.00	1.60000	3.00
4	7	19.00	30.00	11.00	1.57143	4.00
5	4	21.00	34.00	13.00	3.25000	9.00
6	8	28.00	42.00	14.00	1.75000	6.00
7	12	34.00	54.00	20.00	1.66667	8.00
8	8	35.00	62.00	27.00	3.37500	19.00
Average				**14.43**	**2.03**	**7.00**

Static RR (Fixed Quantum = 16) (Note: TEAM Number=6, Quantum = 10 + 6 = 16)

(Sorted by Job ID)

Job ID	**Burst Time**	**ArrivalTime**	**CompleteTime**	**TurnaroundTime**	**WeightedTime**	**WaitingTime**
1	8	10.00	18.00	8.00	1	0.00
3	5	15.00	23.00	8.00	1.60000	3.00
4	7	19.00	30.00	11.00	1.57143	4.00
5	4	21.00	34.00	13.00	3.25000	9.00
6	8	28.00	42.00	14.00	1.75000	6.00
7	12	34.00	54.00	20.00	1.66667	8.00
8	8	35.00	62.00	27.00	3.37500	19.00
Average				**14.43**	**2.03**	**7.00**

Dynamic RR

(Sorted by Job ID)

Job ID	**Burst Time**	**ArrivalTime**	**CompleteTime**	**TurnaroundTime**	**WeightedTime**	**WaitingTime**
1	8	10.00	18.00	8.00	1	0.00
3	5	15.00	23.00	8.00	1.60000	3.00
4	7	19.00	38.75	19.75	2.82143	12.75
5	4	21.00	32.50	11.50	2.87500	7.50
6	8	28.00	49.03	21.03	2.62891	13.03
7	12	34.00	62.00	28.00	2.33333	16.00
8	8	35.00	50.00	15.00	1.87500	7.00
Average				**15.90**	**2.16**	**8.47**

FCFS at t=9.00, then switched to STATIC at t=20.00, then switched to DYNAMIC at t=30.00

(Sorted by Job ID)

Job ID	**Burst Time**	**ArrivalTime**	**CompleteTime**	**TurnaroundTime**	**WeightedTime**	**WaitingTime**
1	8	10.00	18.00	8.00	1.00000	0.00
3	5	15.00	23.00	8.00	1.60000	3.00
4	7	19.00	30.00	11.00	1.57143	4.00
5	4	21.00	34.00	13.00	3.25000	9.00
6	8	28.00	42.00	14.00	1.75000	6.00
7	12	34.00	54.00	20.00	1.66667	8.00
8	8	35.00	62.00	27.00	3.37500	19.00
Average				14.43	2.03	7.00

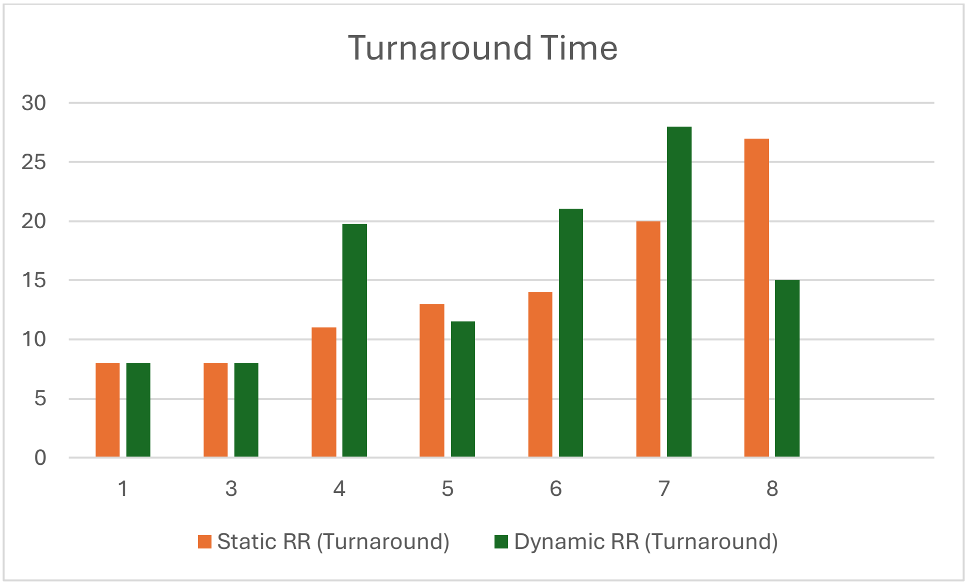
The resulted table is the same as FCFS result. Due to the non-preemtive feature in both FCFS and STATIC RR shcedulers. And for the DYNAMIC RR used at the end, the calculated average was always larger than the burst time for all processes in the DYNAMIC RR portion.

Comparative Study: Dynamic RR vs. Static RR

This study analyzes the performance of two CPU scheduling algorithms, Static Round Robin (SRR) and Dynamic Round Robin (DRR), using the provided test file. The primary metrics for comparison are **Turnaround Time** and **WaitingTime**.

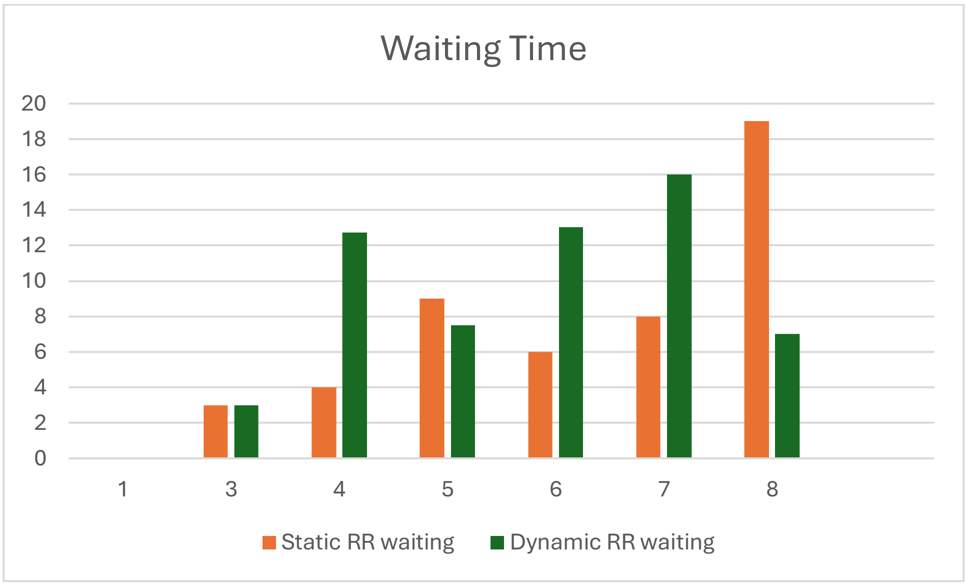
Graphing Turnaround and Wating time (STATIC x DYNAMIC)

Turnaround Time Graph:



Dynamic RR's frequent preemptions (context switching) helped shorter jobs like Job 5 and Job 8 finish faster. However, it severely delayed longer jobs like Job 4, Job 6, and Job 7, which ultimately led to a worse (higher) average turnaround time for the entire system.

Waiting Time Graph:



This graph shows that the "simple" Static RR (FCFS) was more efficient for jobs in the middle of the run, but the "complex" Dynamic RR was much fairer to jobs that arrived at the end.

Conclusion

For this specific workload, characterized by a set of relatively short jobs, the “Static Round Robin scheduler” **which behaved the same as FCFS** because each process had a Burst Time that is smaller than the qantum time (16). The static RR was the most efficient. The overhead of frequent context switching in the “Dynamic Round Robin” scheduler was detrimental, delaying several jobs and leading to a higher average turnaround time and poorer overall system throughput.

Implementation Code

```
/*
* Group Members:
 * Abdulrahman Alfrihidi (2338090)
* Ali Abdullah Al-Hebshi (2343235)
* Sultan Shaikh Omar (2338115)
*
* Compiler: [javac 11.0.25]
 * Hardware: [ Apple M3 Pro, 18GB RAM]
* Operating System: [macOS Sequoia 15.7.1]
*/


import java.io.*;

public class Main {
   public static void main(String[] args) {
      if (args.length < 1 || args.length > 2) {
         System.err.println("Usage: NO ARGS, java Main <input.txt> [output.txt]");
         System.exit(1);
      }
      String inputPath = args[0];
      String outputPath = (args.length == 2) ? args[1] : null;

      try (BufferedReader br = new BufferedReader(new FileReader(inputPath));
          PrintWriter out = (outputPath == null)
                ? new PrintWriter(System.out, true)
                : new PrintWriter(new FileWriter(outputPath))) {

         SimulationController controller = new SimulationController(br, out);
         controller.run();

      } catch (IOException e) {
         System.err.println("I/O Error:");
         e.printStackTrace(System.err);
         System.exit(2);
      }
   }
}
```

```
import java.util.Deque;

public class DynamicRRScheduler implements Scheduler {
  @Override
  public double chooseQuantum(double now, Deque<Process> readyQ, Process running, double
SR, int readyCount) {
    // Include the running process into the average
    double totalRemaining = SR + running.remService;
    int totalCount = readyCount + 1;

    double tq = totalRemaining / totalCount;

    if (tq <= 0) {
      return running.remService;
    }

    return tq;
  }
}
```

```
import java.util.Deque;

public class FCFSScheduler implements Scheduler {
  @Override
  public double chooseQuantum(double now, Deque<Process> readyQ, Process running,
                double SR, int readyCount) {
    return running.remService; // to completion
  }
}
```

```
public class Job {
  public final int jobId;
  public final double arrivalTime;
  public final int reqMemory;
  public final double serviceTime;
  public final int reqDevices;
  public final int priority;

  // used to break ties (HQ1 FIFO after mem)
  public long serial = 0L;
  public Job(int jobId, double arrivalTime, int reqMemory, double serviceTime, int reqDevices, int
priority) {
    this.jobId = jobId;
    this.arrivalTime = arrivalTime;
    this.reqMemory = reqMemory;
    this.serviceTime = serviceTime;
    this.reqDevices = reqDevices;
    this.priority = priority;
  }
}
```

```
public class OtherKerServices {
  private int totalMemory = 0;
  private int availMemory = 0;

  private int totalDevices = 0;
  private int availDevices = 0;

  public void configure(int totalMem, int totalDev) {
    this.totalMemory = totalMem;
    this.availMemory = totalMem;
    this.totalDevices = totalDev;
    this.availDevices = totalDev;
  }

  public int getTotalMemory() { return totalMemory; }
  public int getAvailMemory() { return availMemory; }

  public int getTotalDevices() { return totalDevices; }
  public int getAvailDevices() { return availDevices; }

  public boolean canAllocate(int mem, int dev) {
    return mem <= availMemory && dev <= availDevices;
  }

  public void allocateMemory(int units) {
    availMemory -= units;
    if (availMemory < 0) availMemory = 0;
  }

  public void deallocateMemory(int units) {
        availMemory += units;
        if (availMemory > totalMemory) availMemory = totalMemory;
    }

    public void reserveDevice(int count) {
        availDevices -= count;
        if (availDevices < 0) availDevices = 0;
    }

    public void releaseDevice(int count) {
        availDevices += count;
        if (availDevices > totalDevices) availDevices = totalDevices;
    }
}
```


```
import java.util.*;

public class PrManager {
    private final OtherKerServices kernel;

    // Queues
    private final Deque<Process> readyQ = new ArrayDeque<>();
    private final PriorityQueue<Job> holdQ1; // asc by requested memory, then FIFO
    private final Deque<Job> holdQ2 = new ArrayDeque<>();

    // Scheduler variants
    private Scheduler dynRR;   // default
    private Scheduler staticRR;
    private Scheduler fcfs;

    // Which CPU scheduler to use (default dynamic RR)
    private Scheduler cpuScheduler;

    // Internal clock concept (kept for requirement 3.7.4)
    private double internalClock = 0.0;

    // Currently running process & its time slice end
    private Process running = null;
    private double nextInternalEvent = Double.POSITIVE_INFINITY; // time for slice end or completion

    // SR/AR tracking for dynamic RR
```

```
private double SR = 0.0;
private int readyCount = 0;

// Finished records
public static class FinishedRecord {
  int jobId;
  double arrivalTime, completeTime, turnaround, weightedTurnaround;
}
private final List<FinishedRecord> finished = new ArrayList<>();

// FIFO tie-breaker for HQ1
private long arrivalSerial = 0;

public PrManager(OtherKerServices kernel) {
  this.kernel = kernel;
  this.holdQ1 = new PriorityQueue<>((a, b) -> {
    if (a.reqMemory != b.reqMemory) return Integer.compare(a.reqMemory, b.reqMemory);
    return Long.compare(a.serial, b.serial);
  });
}

public void setSchedulers(Scheduler dyn, Scheduler stat, Scheduler f) {
  this.dynRR = dyn;
  this.staticRR = stat;
  this.fcfs = f;
  this.cpuScheduler = this.dynRR; // default dynamic
}

@SuppressWarnings("unused")
public void setCpuToDynamicRR() { this.cpuScheduler = this.dynRR; }
@SuppressWarnings("unused")
public void setCpuToStaticRR()  { this.cpuScheduler = this.staticRR; }
@SuppressWarnings("unused")
public void setCpuToFCFS()      { this.cpuScheduler = this.fcfs; }

public boolean hasRunnableOrQueuedWork() {
  return running != null || !readyQ.isEmpty() || !holdQ1.isEmpty() || !holdQ2.isEmpty();
}

public void cpuTimeAdvanceTo(double t) {
  if (t > internalClock) internalClock = t;
}

// Called by controller when a job arrival line is seen
public void procArrivalRoutine(double now, Job j) {
  j.serial = (++arrivalSerial);

  // Reject rule: if job requests more than system contains (not only available)
```

```
    if (j.reqMemory > kernel.getTotalMemory() || j.reqDevices > kernel.getTotalDevices()) {
      // rejected (do nothing)
      return;
    }

    // If enough available memory & devices ⇒ allocate & move to ready
    if (kernel.canAllocate(j.reqMemory, j.reqDevices)) {
      kernel.allocateMemory(j.reqMemory);
      kernel.reserveDevice(j.reqDevices);
      Process p = new Process(j);
      p.arrivalOnReady = now;
      pushReady(p);
      tryStartCpuIfIdle(now);
    } else {
      // Go to Hold queues based on priority
      if (j.priority == 1) holdQ1.add(j);
      else holdQ2.addLast(j);
    }
  }

  // Internal event handling: complete time slice or completion
  public void dispatch(double now) {
    // complete any slice/termination already scheduled
    if (running == null) {
      tryStartCpuIfIdle(now);
      return;
    }

    // Advance running by the planned quantum
    double runFor = Math.min(running.tqPlanned, running.remService);
    running.remService = round2(running.remService - runFor);

    if (running.remService <= 1e-9) {
      // Terminate
      finishProcess(now);
      // free resources & move jobs from holds if possible
      admitFromHolds(now);
      // Start next
      tryStartCpuIfIdle(now);
    } else {
      // Time slice expired, preempt and go back to ready :)
      running.lastEnqueueTime = now;
      pushReady(running);
      running = null;

      // After requeue, try start next immediately
      tryStartCpuIfIdle(now);
    }
    // schedule next internal event timestamp accordingly inside tryStartCpuIfIdle()
}

@SuppressWarnings("unused")
public double getNextDecisionTime(double now) {
    // if CPU is running, nextInternalEvent is already set; else +∞ :O
    return nextInternalEvent;
}

private void tryStartCpuIfIdle(double now) {
    if (running != null) return;

    Process next = pickNextReady();
    if (next == null) {
        nextInternalEvent = Double.POSITIVE_INFINITY;
        return;
    }

    running = next;
    running.waitAccum += (now - running.arrivalOnReady);
    running.arrivalOnReady = now;

    // Choose quantum based on scheduler rules
    double tq = cpuScheduler.chooseQuantum(now, readyQ, running, SR, readyCount);
    running.tqPlanned = tq;

    // schedule next internal event (slice end or completion)
    double runFor = Math.min(tq, running.remService);
    nextInternalEvent = now + runFor;
}

private void finishProcess(double now) {
    // Free resources
    kernel.deallocateMemory(running.reqMemory);
    kernel.releaseDevice(running.reqDevices);

    // Stats
    FinishedRecord fr = new FinishedRecord();
    fr.jobId = running.jobId;
    fr.arrivalTime = running.arrivalTime;
    fr.completeTime = now;
    fr.turnaround = now - running.arrivalTime;
    fr.weightedTurnaround = fr.turnaround / running.serviceTime;
    finished.add(fr);

    // Remove from SR counts (running was not in SR while it ran; SR tracked only readyQ)
    running = null;
}
```

```
private void admitFromHolds(double now) {
  boolean moved = true;
  while (moved) {
    moved = false;
    // First HQ1 (priority 1)
    while (!holdQ1.isEmpty()) {
      Job j = holdQ1.peek();
      if (kernel.canAllocate(j.reqMemory, j.reqDevices)) {
        holdQ1.poll();
        kernel.allocateMemory(j.reqMemory);
        kernel.reserveDevice(j.reqDevices);
        Process p = new Process(j);
        p.arrivalOnReady = now;
        pushReady(p);
        moved = true;
      } else break;
    }
    // Then HQ2 FIFO
    while (!holdQ2.isEmpty()) {
      Job j = holdQ2.peekFirst();
      if (kernel.canAllocate(j.reqMemory, j.reqDevices)) {
        holdQ2.pollFirst();
        kernel.allocateMemory(j.reqMemory);
        kernel.reserveDevice(j.reqDevices);
        Process p = new Process(j);
        p.arrivalOnReady = now;
        pushReady(p);
        moved = true;
      } else break;
    }
  }
}

private void pushReady(Process p) {
  readyQ.addLast(p);
  SR += p.remService;
  readyCount++;
}

private Process pickNextReady() {
  Process p = readyQ.pollFirst();
  if (p != null) {
    SR -= p.remService;
    readyCount--;
  }
  return p;
}
```

```
    // Snapshots for display
    public List<Process> snapshotReady() {
        return new ArrayList<>(readyQ);
    }
    public List<Job> snapshotHQ1() {
        return new ArrayList<>(holdQ1);
    }
    public List<Job> snapshotHQ2() {
        return new ArrayList<>(holdQ2);
    }
    public List<FinishedRecord> snapshotFinished() {
        return new ArrayList<>(finished);
    }

    private static double round2(double v) {
        // minor rounding to mitigate FP noise
        return Math.abs(v) < 1e-9 ? 0.0 : v;
    }

    // Public wrapper to start CPU
    public void tryStartCpuIfIdlePublic(double now) {
        tryStartCpuIfIdle(now);
    }

}
```

```
public class Process {
    public final int jobId;
    public final double arrivalTime;
    public final int reqMemory;
    public final int reqDevices;

    public final double serviceTime;
    public double remService;

    // runtime bookkeeping
    public double tqPlanned = 0.0;
    public double waitAccum = 0.0;
    public double arrivalOnReady = 0.0;
    public double lastEnqueueTime = 0.0;
```

```
  public Process(Job j) {
    this.jobId = j.jobId;
    this.arrivalTime = j.arrivalTime;
    this.reqMemory = j.reqMemory;
    this.reqDevices = j.reqDevices;
    this.serviceTime = j.serviceTime;
    this.remService = j.serviceTime;
  }
}
```

```
import java.util.Deque;

public interface Scheduler {
  // This is for choosing the slot slices :)
  double chooseQuantum(double now, Deque<Process> readyQ, Process running,
             double SR, int readyCount);
}
```

```
import java.io.*;
import java.util.*;

public class SimulationController {
  private final PrintWriter out;
  private final OtherKerServices kernel;
  private final PrManager prManager;

  private double curTime = 0.0;
  private int teamNumber = 0;

  // ---- Event container for sorted queue ----
  private static class Event {
    double time;
```

```
        char type;
        String line;
        long order;

        Event(double time, char type, String line, long order) {
            this.time = time;
            this.type = type;
            this.line = line;
            this.order = order;
        }
    }

    // Sort by time, then C->A->D, then file order
    private final PriorityQueue<Event> eventQueue = new PriorityQueue<>((a, b) -> {
        int t = Double.compare(a.time, b.time);
        if (t != 0) return t;

        int pa = (a.type == 'C') ? 0 : (a.type == 'A' ? 1 : 2);
        int pb = (b.type == 'C') ? 0 : (b.type == 'A' ? 1 : 2);
        if (pa != pb) return pa - pb;

        return Long.compare(a.order, b.order);
    });

    public SimulationController(BufferedReader in, PrintWriter out) {
        this.out = out;
        this.kernel = new OtherKerServices();
        this.prManager = new PrManager(kernel);
        loadExternalEvents(in);
    }

    // Read all external events first
    private void loadExternalEvents(BufferedReader in) {
        try {
            String line;
            long order = 0;
            while ((line = in.readLine()) != null) {
                line = line.trim();
                if (line.isEmpty()) continue;

                char c = Character.toUpperCase(line.charAt(0));
                if (c != 'A' && c != 'C' && c != 'D' && c != 'S') continue;


                String[] parts = line.split("\\s+");
                double t = Double.parseDouble(parts[1]);

                eventQueue.add(new Event(t, c, line, order++));
            }
        } catch (Exception ignored) {}
    }

    public void run() {
        while (!eventQueue.isEmpty() || prManager.hasRunnableOrQueuedWork()) {

            double e = prManager.getNextDecisionTime(curTime); // next internal
            double i = eventQueue.isEmpty() ? Double.POSITIVE_INFINITY : eventQueue.peek().time;

            double next = Math.min(e, i);
            if (Double.isInfinite(next)) break;

            if (Math.abs(e - i) < 1e-9) {
                curTime = e;
                prManager.cpuTimeAdvanceTo(curTime);
                prManager.dispatch(curTime);
                handleOneExternal();
            }
            else if (e < i) {
                curTime = e;
                prManager.cpuTimeAdvanceTo(curTime);
                prManager.dispatch(curTime);
            }
            else {
                curTime = i;
                handleOneExternal();
            }
        }

        out.flush();
    }

    private void handleOneExternal() {
        if (eventQueue.isEmpty()) return;

        Event ev = eventQueue.poll();
        String line = ev.line;
        char type = ev.type;

        try {
            switch (type) {
                case 'C':
                    parseSystemConfig(line);
                    break;

                case 'A':
                    prManager.procArrivalRoutine(curTime, parseArrival(line));
                    prManager.cpuTimeAdvanceTo(curTime);
                    prManager.tryStartCpuIfIdlePublic(curTime);
                    break;

                case 'D':
                    printDisplay();
                    break;

                case 'S':
                    handleSchedulerCommand(line);
                    break;
            }
        } catch (Exception e) {
            out.println("ERROR processing: " + line);
        }
    }

    private void parseSystemConfig(String line) {
        String[] toks = line.split("\\s+");
        double startTime = Double.parseDouble(toks[1]);

        if (curTime < startTime) curTime = startTime;

        int mem = kernel.getTotalMemory();
        int dev = kernel.getTotalDevices();

        for (int k = 2; k < toks.length; k++) {
            String part = toks[k];
            if (part.startsWith("M=")) mem = Integer.parseInt(part.substring(2));
            else if (part.startsWith("S=")) dev = Integer.parseInt(part.substring(2));
            else if (part.startsWith("TEAM=")) teamNumber = Integer.parseInt(part.substring(5));
        }

        kernel.configure(mem, dev);
        prManager.setSchedulers(new DynamicRRScheduler(), new StaticRRScheduler(10 +
teamNumber), new FCFSScheduler());
    }

    private Job parseArrival(String line) {
        String[] toks = line.split("\\s+");
        double t = Double.parseDouble(toks[1]);
        int id = 0, m = 0, dev = 0, p = 1;
        double svc = 0.0; // service time (R)

        for (int k = 2; k < toks.length; k++) {
            String tok = toks[k];
            if (tok.startsWith("J=")) id = Integer.parseInt(tok.substring(2));
            else if (tok.startsWith("M=")) m = Integer.parseInt(tok.substring(2));
            else if (tok.startsWith("S=")) dev = Integer.parseInt(tok.substring(2));   // S = devices
            else if (tok.startsWith("R=")) svc = Double.parseDouble(tok.substring(2)); // R = service time
(Cycles)
            else if (tok.startsWith("P=")) p = Integer.parseInt(tok.substring(2));
        }

        return new Job(id, t, m, svc, dev, p);
    }

    // ---------------- DISPLAY  ----------------
    private void printDisplay() {
        out.println("\n\n--------------------------------------------------------");
        out.println("System Status:                                   ");
        out.println("--------------------------------------------------------");
        out.printf("         Time: %.2f%n", curTime);
        out.printf("  Total Memory: %d%n", kernel.getTotalMemory());
        out.printf(" Avail. Memory: %d%n", kernel.getAvailMemory());
        out.printf(" Total Devices: %d%n", kernel.getTotalDevices());
        out.printf("Avail. Devices: %d%n", kernel.getAvailDevices());
        out.println();

        out.println("\nJobs in Ready List                               ");
        out.println("--------------------------------------------------------");
        List<Process> rq = prManager.snapshotReady();
        if (rq.isEmpty()) out.println("  EMPTY");
        else for (Process p : rq)
            out.printf("Job ID %d, %.2f Cycles left to completion.%n", p.jobId, p.remService);
        out.println();

        out.println("\nJobs in Long Job List                            ");
        out.println("--------------------------------------------------------");
        out.println("  EMPTY\n");

        out.println("\nJobs in Hold List 1                              ");
        out.println("--------------------------------------------------------");
        List<Job> h1 = prManager.snapshotHQ1();
        if (h1.isEmpty()) out.println("  EMPTY");
        else for (Job j : h1)
            out.printf("Job ID %d, %.2f Cycles left to completion.%n", j.jobId, j.serviceTime);
        out.println();

        out.println("\nJobs in Hold List 2                              ");
        out.println("--------------------------------------------------------");
        List<Job> h2 = prManager.snapshotHQ2();
        if (h2.isEmpty()) out.println("  EMPTY");
        else for (Job j : h2)
            out.printf("Job ID %d, %.2f Cycles left to completion.%n", j.jobId, j.serviceTime);
        out.println();
        out.println("\nJobs in Hold List 3                              ");
        out.println("-------------------------------------------------------");
        out.println("  EMPTY\n\n");

        out.println("Finished Jobs (detailed)                          ");
        out.println("-------------------------------------------------------");
        out.println("  Job   ArrivalTime   CompleteTime   TurnaroundTime   WeightedTime");
        out.println("-----------------------------------------------------------------------");

        List<PrManager.FinishedRecord> done = prManager.snapshotFinished();
        if (done.isEmpty()) {
            out.println("  EMPTY");
        } else {
            for (PrManager.FinishedRecord fr : done)
              out.printf("  %d%12.2f%18.2f%18.2f%18s%n",
                    fr.jobId, fr.arrivalTime, fr.completeTime,
                    fr.turnaround, format(fr.weightedTurnaround));
            out.printf("Total Finished Jobs:          %d%n", done.size());
        }

        out.println("\n");
    }

    private String format(double v) {
        if (Math.abs(v - Math.round(v)) < 1e-9) return String.valueOf((int)Math.round(v));
        return String.format(Locale.US, "%.5f", v);
    }

    private void handleSchedulerCommand(String line) {
        // formats:
        // S <time> STATIC for static RR
        // S <time> FCFS for FCFS xD
        // The config call can be:
        // C= <Number> M= <Number> S= <Number> OPTIONAL:)==> TEAM= <Number>
        // This is for you Abdulrahman, so you can know :)


        String[] toks = line.trim().split("\\s+");
        if (toks.length < 3) {
            out.println(">> Unknown scheduler command: " + line);
            return;
        }

        String mode = toks[2].toUpperCase(Locale.ROOT);

        if (mode.startsWith("STAT")) {
            prManager.setCpuToStaticRR();
```

```
            out.println(">> Scheduler switched to STATIC RR at t=" + String.format(Locale.US, "%.2f",
curTime));
        } else if (mode.startsWith("DYN")) {
            prManager.setCpuToDynamicRR();
            out.println(">> Scheduler switched to DYNAMIC RR at t=" + String.format(Locale.US, "%.2f",
curTime));
        } else if (mode.startsWith("FCFS")) {
            prManager.setCpuToFCFS();
            out.println(">> Scheduler switched to FCFS at t=" + String.format(Locale.US, "%.2f",
curTime));
        } else {
            out.println(">> Unknown scheduler mode in: " + line);
        }
    }

}
```

```
import java.util.Deque;

public class StaticRRScheduler implements Scheduler {
    private final double quantum;

    public StaticRRScheduler(int q) {
        this.quantum = Math.max(1, q);
    }

    @Override
    public double chooseQuantum(double now, Deque<Process> readyQ, Process running,
                        double SR, int readyCount) {
        // Fixed time quantum = 10 + teamNumber (as file said)
        return quantum;
    }
}
```

```
------------------------------------------------------------------------------------------------------------------------------------
```

Appendix A: Program Output (FCFS)

```
>> Scheduler switched to FCFS at t=9.00


----------------------------------------------------------
System Status:
----------------------------------------------------------
        Time: 36.00
  Total Memory: 45
 Avail. Memory: 1
 Total Devices: 12
Avail. Devices: 3


Jobs in Ready List
-----------------------------------------------------------
Job ID 7, 12.00 Cycles left to completion.


Jobs in Long Job List
-----------------------------------------------------------
  EMPTY


Jobs in Hold List 1
-----------------------------------------------------------
Job ID 8, 8.00 Cycles left to completion.


Jobs in Hold List 2
-----------------------------------------------------------
  EMPTY


Jobs in Hold List 3
-----------------------------------------------------------
  EMPTY


Finished Jobs (detailed)
```

```
-------------------------------------------------------
 Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
----------------------------------------------------------------------
 1     10.00          18.00           8.00            1
 3     15.00          23.00           8.00         1.60000
 4     19.00          30.00          11.00          1.57143
 5     21.00          34.00          13.00          3.25000
Total Finished Jobs:          4




-------------------------------------------------------
System Status:
-------------------------------------------------------
       Time: 99999.00
 Total Memory: 45
 Avail. Memory: 45
 Total Devices: 12
Avail. Devices: 12


Jobs in Ready List
-------------------------------------------------------
 EMPTY


Jobs in Long Job List
-------------------------------------------------------
 EMPTY


Jobs in Hold List 1
-------------------------------------------------------
 EMPTY


Jobs in Hold List 2
-------------------------------------------------------
 EMPTY


Jobs in Hold List 3
-------------------------------------------------------
 EMPTY


Finished Jobs (detailed)
```

```
------------------------------------------------------
 Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
----------------------------------------------------------------------
 1     10.00          18.00           8.00             1
 3     15.00          23.00           8.00           1.60000
 4     19.00          30.00           11.00           1.57143
 5     21.00          34.00           13.00           3.25000
 6     28.00          42.00           14.00           1.75000
 7     34.00          54.00           20.00           1.66667
 8     35.00          62.00           27.00           3.37500
Total Finished Jobs:          7
```

Appendix B: Program Output (STATIC RR)

```
>> Scheduler switched to STATIC RR at t=9.00


------------------------------------------------------
System Status:
------------------------------------------------------
       Time: 36.00
 Total Memory: 45
 Avail. Memory: 1
 Total Devices: 12
Avail. Devices: 3


Jobs in Ready List
------------------------------------------------------
Job ID 7, 12.00 Cycles left to completion.


Jobs in Long Job List
------------------------------------------------------
 EMPTY


Jobs in Hold List 1
------------------------------------------------------
Job ID 8, 8.00 Cycles left to completion.


Jobs in Hold List 2
------------------------------------------------------
```

```
  EMPTY


Jobs in Hold List 3
------------------------------------------------------
  EMPTY


Finished Jobs (detailed)
------------------------------------------------------
  Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
----------------------------------------------------------------------
   1      10.00          18.00           8.00            1
   3      15.00          23.00           8.00         1.60000
   4      19.00          30.00          11.00          1.57143
   5      21.00          34.00          13.00          3.25000
Total Finished Jobs:           4




------------------------------------------------------
System Status:
------------------------------------------------------
          Time: 99999.00
  Total Memory: 45
 Avail. Memory: 45
 Total Devices: 12
Avail. Devices: 12


Jobs in Ready List
------------------------------------------------------
  EMPTY


Jobs in Long Job List
------------------------------------------------------
  EMPTY


Jobs in Hold List 1
------------------------------------------------------
  EMPTY


Jobs in Hold List 2
------------------------------------------------------
```

```
 EMPTY


Jobs in Hold List 3
--------------------------------------------------------
 EMPTY


Finished Jobs (detailed)
--------------------------------------------------------
 Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
-----------------------------------------------------------------------
 1     10.00          18.00           8.00             1
 3     15.00          23.00           8.00          1.60000
 4     19.00          30.00          11.00           1.57143
 5     21.00          34.00          13.00           3.25000
 6     28.00          42.00          14.00           1.75000
 7     34.00          54.00          20.00           1.66667
 8     35.00          62.00          27.00           3.37500
Total Finished Jobs:         7
```

Appendix C: Program Output (DYNAMIC RR)

```
>> Scheduler switched to DYNAMIC RR at t=9.00


-------------------------------------------------------
System Status:
-------------------------------------------------------
      Time: 36.00
 Total Memory: 45
 Avail. Memory: 9
 Total Devices: 12
Avail. Devices: 4


Jobs in Ready List
--------------------------------------------------------
Job ID 4, 1.50 Cycles left to completion.
Job ID 8, 8.00 Cycles left to completion.


Jobs in Long Job List
--------------------------------------------------------
```

```
  EMPTY


Jobs in Hold List 1
--------------------------------------------------------
  EMPTY


Jobs in Hold List 2
--------------------------------------------------------
Job ID 7, 12.00 Cycles left to completion.


Jobs in Hold List 3
--------------------------------------------------------
  EMPTY


Finished Jobs (detailed)
--------------------------------------------------------
  Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
------------------------------------------------------------------------
  1      10.00          18.00           8.00            1
  3      15.00          23.00           8.00        1.60000
  5      21.00          32.50          11.50        2.87500
Total Finished Jobs:           3




-------------------------------------------------------
System Status:
-------------------------------------------------------
        Time: 99999.00
  Total Memory: 45
 Avail. Memory: 45
 Total Devices: 12
Avail. Devices: 12


Jobs in Ready List
--------------------------------------------------------
  EMPTY


Jobs in Long Job List
--------------------------------------------------------
  EMPTY
```

```
Jobs in Hold List 1
---------------------------------------------------------
 EMPTY


Jobs in Hold List 2
---------------------------------------------------------
 EMPTY


Jobs in Hold List 3
---------------------------------------------------------
 EMPTY


Finished Jobs (detailed)
---------------------------------------------------------
 Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
--------------------------------------------------------------------------
 1     10.00          18.00           8.00             1
 3     15.00          23.00           8.00          1.60000
 5     21.00          32.50          11.50           2.87500
 4     19.00          38.75          19.75           2.82143
 6     28.00          49.03          21.03           2.62891
 8     35.00          50.00          15.00           1.87500
 7     34.00          62.00          28.00           2.33333
Total Finished Jobs:          7
```

Appendix D: Program Output (FCFS, STATIC RR, DYNAMIC RR)

```
>> Scheduler switched to FCFS at t=9.00
>> Scheduler switched to STATIC RR at t=20.00
>> Scheduler switched to DYNAMIC RR at t=30.00


--------------------------------------------------------
System Status:
--------------------------------------------------------
```

```
        Time: 36.00
  Total Memory: 45
 Avail. Memory: 1
 Total Devices: 12
Avail. Devices: 3


Jobs in Ready List
--------------------------------------------------------
Job ID 7, 12.00 Cycles left to completion.


Jobs in Long Job List
--------------------------------------------------------
  EMPTY


Jobs in Hold List 1
--------------------------------------------------------
Job ID 8, 8.00 Cycles left to completion.


Jobs in Hold List 2
--------------------------------------------------------
  EMPTY


Jobs in Hold List 3
--------------------------------------------------------
  EMPTY


Finished Jobs (detailed)
--------------------------------------------------------
  Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
------------------------------------------------------------------------
  1     10.00          18.00           8.00             1
  3     15.00          23.00           8.00         1.60000
  4     19.00          30.00          11.00          1.57143
  5     21.00          34.00          13.00          3.25000
Total Finished Jobs:          4




-------------------------------------------------------
System Status:
-------------------------------------------------------
```

```
       Time: 99999.00
  Total Memory: 45
 Avail. Memory: 45
 Total Devices: 12
Avail. Devices: 12


Jobs in Ready List
--------------------------------------------------------
  EMPTY


Jobs in Long Job List
--------------------------------------------------------
  EMPTY


Jobs in Hold List 1
--------------------------------------------------------
  EMPTY


Jobs in Hold List 2
--------------------------------------------------------
  EMPTY


Jobs in Hold List 3
--------------------------------------------------------
  EMPTY


Finished Jobs (detailed)
--------------------------------------------------------
  Job   ArrivalTime    CompleteTime    TurnaroundTime   WeightedTime
------------------------------------------------------------------------
  1     10.00         18.00          8.00           1
  3     15.00         23.00          8.00        1.60000
  4     19.00         30.00         11.00         1.57143
  5     21.00         34.00         13.00         3.25000
  6     28.00         42.00         14.00         1.75000
  7     34.00         54.00         20.00         1.66667
  8     35.00         62.00         27.00         3.37500
Total Finished Jobs:          7
```