Aufgabe 1:  Real-Time Systems Concept

What are the main differences between general purpose computing and real-time computing? List some critical applications which require a real-time system support.

Solution - Task 1

General purpose computing:

- Tasks do not have timing constraints
- Correctness of system behavior depends only on computing the correct results
- The \textit{goal} is usually to minimize the average response time of a given set of tasks, without any guarantees given for individual timing requirements of each task in all possible scheduling scenarios
- Faster is usually better
- \textit{Applications}: word processing, image processing, etc.

Real-time computing:

- Tasks have timing requirements: soft or hard deadlines, periods, etc.
- Correct behavior depends on \textit{both}: (1) correct computation and (2) time at which results are produced
- The \textit{goal} is to meet the individual timing requirement of each task under all possible execution scenarios
- \textit{System} may have to react within precise timing constraints to events from the environment
- \textit{Applications}: engine controls, anti-lock breaks, electric steering columns, flight management systems, etc.
Aufgabe 2: First Come First Serve Scheduling

(A)
A set of independent tasks A, B, C and D need to execute on a processor. The details are shown in Table 1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Task Arrival and Execution Times

a) Draw the schedule generated by the First Come First Serve (FCFS) algorithm.
b) Determine the following metrics for each task: (a) Waiting Time, (b) Response Time.
c) Determine the average waiting time of all tasks.
d) Is the FCFS algorithm fair? Explain your answer.

Solution - Task 2A

a) The Gantt chart showing FCFS schedule is shown in Fig 1.

![Gantt Chart](image)

Figure 1: Gantt Chart showing FCFS schedule for question 2A.a

b) Waiting times:
   \[ w_A = 0 - 0 = 0 \]
   \[ w_B = 10 - 2 = 8 \]
   \[ w_C = 13 - 3 = 10 \]
   \[ w_D = 15 - 5 = 10 \]

Response Times:

Response time for a task is defined as:

\[ r = \text{Time when the task finishes computing} - \text{Time when the task arrives (is released)} \]
\( r_A = 10 - 0 = 10 \)
\( r_B = 13 - 2 = 11 \)
\( r_C = 15 - 3 = 12 \)
\( r_D = 16 - 5 = 11 \)

c) Average waiting time for all tasks taken together:
\[
\bar{w} = \frac{0 + 8 + 10 + 10}{4} = 7
\]
d) Yes. The FCFS scheduling algorithm is fair in the traditional sense of fairness; i.e., all tasks will eventually get a chance to execute on the processor. However, depending on task arrival times and their task execution times, several tasks with small execution times may wait until a single task with a very long execution time finishes.
A set of independent tasks $A, B, C$ and $D$ need to execute on a processor. A task may need to execute more than once on the processor. Once a task finishes one execution, it is ready (released) for the next execution with a possibly different execution time. The details are shown in Table 2.

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>1st exec. time</th>
<th>2nd exec. time</th>
<th>3rd exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Task Arrival and Execution Times

a) Draw the schedule generated by the First Come First Serve (FCFS) algorithm.
b) Determine the following metrics for each task considering all of its executions: (a) Waiting Time, (b) Response Time.
c) Determine the average waiting time of all tasks.
d) Does the generated schedule exhibit the convoy effect?

Solution - Task 2B

a) The Gantt chart showing FCFS schedule is shown in Fig 2.

![Gantt Chart](image)

Figure 2: Gantt Chart showing FCFS schedule for question 2B.a

b) Waiting Times:

- $w_A = (0 - 0) + (16 - 10) + (25 - 22) = 9$
- $w_B = (10 - 2) + (22 - 13) + (29 - 24) = 22$
- $w_C = 13 - 3 = 10$
- $w_D = (15 - 5) + (24 - 16) = 18$

Response Times:

- $r_A = 29 - 0 = 29$
- $r_B = 31 - 2 = 29$
- $r_C = 15 - 3 = 12$
- $r_D = 25 - 5 = 20$

c) Average waiting time for all tasks taken together:

$$\bar{w} = \frac{9 + 22 + 10 + 18}{4} = 14.75$$
d) Yes. Tasks with short execution times wait for a long-running task (A, for example) to finish, leading to larger overall waiting times.
Aufgabe 3: Shortest Job First Scheduling

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>Exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: Task Arrival and Execution Times

a) For the values shown in Table 3, determine the schedule generated by the Shortest Job First (SJF) scheduling algorithm with no preemptions.
b) Determine the waiting time for each task and the overall average waiting time.
c) Outline a proof that the SJF algorithm leads to the shortest average waiting times. Consider only the case when tasks have equal arrival (release) times and one execution.
d) List practical problems that can arise when implementing the SJF algorithm in modern Operating System kernels.

Solution - Task 3

a) The Gantt chart for the SJF schedule is shown in Fig 3.

![Gantt Chart](image)

Figure 3: Gantt Chart showing SJF schedule for question 3.a

b) Waiting Times:
\[ w_A = 0 - 0 = 0 \]
\[ w_B = 13 - 2 = 11 \]
\[ w_C = 11 - 3 = 8 \]
\[ w_D = 10 - 5 = 5 \]

Average waiting time of all tasks:
\[ \bar{w} = \frac{0 + 11 + 8 + 5}{4} = 6 \]
c) Only an outline is being provided here for tasks with equal arrival times.

The SJF algorithm statically minimizes the average waiting time and thus, requires apriori knowledge of task execution times and that all tasks have arrived when the execution begins. The proof rests on the following observation:

*Given two independent tasks with equal arrival times, $T_1$ and $T_2$, such that the execution times of these tasks, $C_{T_1} > C_{T_2}$, then the average waiting time is minimal if the task with smaller execution time runs first.*

That is, consider tasks $T_1$ and $T_2$ with the given relation between their execution times as shown in Fig 4(a), with an average overall waiting time of $\frac{C_{T_1}}{2}$.

Now consider the sequence in Fig 4(b), with an average overall waiting time of $\frac{C_{T_2}}{2}$.

![Figure 4: Gantt Chart showing FCFS schedule for question 3c)](image)

Since $C_{T_2} < C_{T_1}$, the average waiting time for the case in Fig 4(b) is lower. It follows that for a given set of tasks with different execution times, swapping two neighbouring tasks and ordering them in the increasing order of their execution times will lower the average waiting time.

Now, let $T_1 - T_3 - T_2 - T_6 - T_4 - T_5$ be a random schedule with $C_{T_i} < C_{T_j}$, $\forall i < j$.

Now consider $T_3 - T_2$ pair of tasks, and swap them as shown in Fig 5.

![Figure 5: Swap Task pair $T_3 - T_2$ to get lower average waiting time.](image)

The average waiting time for the case in Fig 5(a) is

$$w_{(a)} = \frac{w_{T_1} + w_{T_3(a)} + w_{T_2(a)} + w_{T_6} + w_{T_4} + w_{T_5}}{6}$$

where $w_{T_2(a)}$, $w_{T_3(a)}$ are the waiting times of $T_2$ and $T_3$ in schedule (a). Respectively, for the case in 5(b):

$$w_{(b)} = \frac{w_{T_1} + w_{T_2(b)} + w_{T_3(b)} + w_{T_6} + w_{T_4} + w_{T_5}}{6}$$

Note that the waiting times of tasks $T_1$, $T_4$, $T_5$, $T_6$ are the same in both schedules. From the preceding discussion, $w_{T_2(a)} + w_{T_3(a)} > w_{T_2(b)} + w_{T_3(b)}$, and therefore, $w_{(a)} > w_{(b)}$. Following similar procedure, all pairs of tasks can be swapped until are processes are sorted in increasing order of their execution times, leading to minimal average waiting time.
d) The SJF algorithm needs to know in advance the runtime execution time of each task in order to make a proper schedule. However, this information is almost never available with sufficient accuracy before executing the task to completion. In such cases, the scheduler may need to estimate the expected task execution times from recent history; which may itself be inaccurate.
### Aufgabe 4: Shortest Remaining Time Next

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; exec. time</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; exec. time</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt; exec. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4: Task Arrival and Execution Times

a) For the values shown in Table 4, determine the schedule generated by the Shortest Remaining Time Next (SRTN) scheduling algorithm with preemptions. A task may need to execute more than once on the processor. Once a task finishes one execution, it is ready (released) for the next execution with a possibly different execution time.

b) Is the SRTN algorithm fair? Justify your answer.

### Solution - Task 4

a) The Gantt chart showing SRTN schedule is shown in Fig 6. Remember that SRTN is a pre-emptive scheduling algorithm.

![Gantt Chart](image)

Figure 6: Schedule for Shortest Remaining Time Next Scheduling algorithm.

b) No. The SRTN algorithm may continuously schedule tasks with the smaller execution times, causing tasks with large execution times to starve and never be executed.
Aufgabe 5: Round Robin Scheduling

Consider four independent tasks with their arrival and execution times as shown in Table 5.

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 5: Task Arrival and Execution Times

For the values shown in Table 5, determine the schedule generated by the Round Robin (RR) scheduling algorithm. Assume a quantum $q$ of 2 time units and processor context switching time of 0 time units.

Solution - Task 5

The Gantt chart for the RR schedule is shown in Fig 7. In the considered implementation of the RR scheduling algorithm, a task yields the processor only if it has finished execution or it has used up the time quantum on the processor. In the second case, the task is reinserted at the end of the ready queue. At the beginning of a new quantum, the scheduler selects the first task in the queue to run (if the queue is not empty). Every new task is appended to the end of the queue.

Figure 7: Schedule for Round Robin Scheduling algorithm.
A real-time system has to execute four tasks $J_1, J_2, J_3, J_4$ with arrival times and deadlines shown in the following table. The observed scheduling function $\sigma(t)$ (as defined in the lecture) is shown in Figure 8.

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrival time</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>deadline</td>
<td>9</td>
<td>10</td>
<td>17</td>
<td>13</td>
</tr>
</tbody>
</table>

Determine:

a) The maximum lateness.

b) The laxity of each task.

c) The processor utilization for this schedule (from time 0 to time 14).

Is the schedule feasible? If not, modify the scheduling function so that the task set is schedulable.
Solution - Task 6

a) The maximum lateness is dominated by task $J_2$ and it equals $f_2 - d_2 = 12 - 10 = 2$ (task $J_2$ is the only task that violates the given constraints and has a positive lateness).

b) The laxities of the tasks $J_1, \ldots, J_4$ are as follows (in the table below are the task computation times):
   (a) $X_1 = d_1 - a_1 - C_1 = 9 - 0 - 4 = 5$
   (b) $X_2 = 10 - 4 - 2 = 4$
   (c) $X_3 = 17 - 2 - 4 = 11$
   (d) $X_4 = 13 - 4 - 3 = 6$

<table>
<thead>
<tr>
<th></th>
<th>$J_1$</th>
<th>$J_2$</th>
<th>$J_3$</th>
<th>$J_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_i$</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

c) The CPU(Utilization) is $13/14 = 0.93$.

A schedule is said to be feasible if all tasks can be completed according to a set of specified constraints. However, the scheduling function suggested in the exercise cannot schedule all tasks within their deadlines.

Figure 9: Violation in the old scheduling function

The schedule can be modified in order to complete the execution of all tasks before their respective deadline. One (of several) scheduling functions is depicted in Fig 10.

Figure 10: New scheduling function
**Aufgabe 7: Design of a Real-Time Scheduling Algorithm**

For real-time system scheduling it is important to know the maximum (worst-case) execution time of each task \textit{a priori}. However, also if this information is given, there are several other problems that may be encountered during the design of a scheduling algorithm for a real-time system. Can you think of some difficulties? What are possible solutions? How can we compare different scheduling algorithms?

---

**Solution - Task 7**

Constraints that are usually given while designing a real-time scheduling algorithm are for example:

1) Timing constraints: the design of the real-time scheduler may be influenced by the possible activation patterns of the tasks. The actual arrival times may not be known in advance;
2) Precedence relations between the tasks: the tasks to be scheduled may have precedence constraints: for example, one task cannot begin its execution, since it needs the result of another task that still has to compute/output it;
3) Mutual exclusion constraints on shared resources need to be met: for example, two tasks accessing the same file should be allowed to do it only in a sequential manner, in order to guarantee consistence and correctness of their respective functions.

Possible solutions to the problems mentioned above are (respectively): 1) Find a way to model all possible task arrival patterns. For instance use a model with jitter (= interval of admissible arrival times); 2) The scheduling algorithm should be able to consider precedence relations; 3) The synchronization mechanism for a shared resource should be able to suspend tasks and determine an order for accessing the shared resource.

Another difficulty is the so-called \textbf{domino effect}. No new tasks should be added at runtime if they jeopardize the schedulability of (previously) guaranteed tasks.

There are different metrics for the performance comparison of various scheduling algorithms. Some of those are: the average response time, total completion time, weighted sum of completion times, maximum lateness, maximum number of late tasks.