Task Scheduling for Real Time Multimedia Streaming

Fahad Ahmed (Student ID: 202207274, Email: x2022fjh@stfx.ca)

1 Introduction

Real-time systems are integral to modern computing environments, demanding precise and predictable timing in the execution of tasks. These systems are characterized by stringent timing constraints, where the correctness of the system's behavior is not only a function of logical correctness but also of the time at which the results are produced. Scheduling algorithms play a pivotal role in ensuring that tasks in real-time systems are executed within their specified deadlines, thereby maintaining system reliability, responsiveness, and predictability. This document explores the significance of various scheduling algorithms in real-time systems, emphasizing how they cater to the unique demands of such environments. Furthermore, it delves into the intersection of real-time systems and multimedia applications, illustrating how principles of real-time scheduling are applied to manage time-sensitive multimedia data processing, thereby ensuring a seamless user experience. This project involves developing a real-time system for scheduling and transmitting both MP3 (music) data and voice data to an Icecast2 server, with the content destined for two separate mount points. To manage and prioritize the transmission of these two types of data, your system employs two scheduling algorithms: a Priority-Driven algorithm and the Earliest Deadline First (EDF) algorithm. In our real-time system project that involves the transmission of MP3 music data and voice data to an Icecast2 server, you've introduced an additional feature: a pause task. This pause task is assigned the highest priority level within the system's task hierarchy, even above the priority of transmitting voice data. In our real-time system project that involves the transmission of MP3 music data and voice data to an Icecast2 server, we've introduced an additional feature: a pause task. This pause task is assigned the highest priority level within the system's task hierarchy, even above the priority of transmitting voice data in our priority-driven scheduling. Simulating the Earliest Deadline First (EDF) algorithm with a variety of tasks, each characterized by its scheduled time, execution time, and deadline, is a robust approach to evaluating and fine-tuning your real-time system's performance, especially for our project involving the transmission of MP3 and voice data to an Icecast2 server. The motivation behind this project is twofold: to deepen the theoretical understanding of real-time scheduling algorithms and to apply these principles in enhancing the reliability, responsiveness, and user experience of multimedia streaming services. Through this endeavor, the project aims to contribute to the broader field of real-time systems, offering insights and solutions that bridge the gap between theory and practical application.

2 Project Description

Our application integrates the power of real-time scheduling algorithms to efficiently manage and prioritize tasks, specifically designed for streaming audio to an Icecast server. This document outlines the basic functionalities, including task management, algorithm selection, and priority handling within our application. Our application supports two main scheduling algorithms namely Priority-Based Scheduling and Earliest Deadline First (EDF) Algorithm. The choice of scheduling algorithm, along with the details such as task deadlines and execution times, is configurable through an XML configuration file. This flexibility allows for easy customization and adaptation to different operational requirements.

2.1 Task Management

Our application handles three main types of tasks, each with unique characteristics and requirements:

1. Aperiodic Task: This involves capturing microphone input and streaming the voice data to the Icecast server. The nature of this task is unpredictable, as it depends on the presence of voice input to initiate the streaming process.

2. Sporadic Task: This task is designed to introduce a pause or a few seconds of sleep within the system's operation. It's used to manage system resources effectively and ensure there's a balance between task processing and system performance.

3. Periodic Task: In this task, our application reads chunks of data from an MP3 file and streams it to the Icecast server. This task occurs at regular intervals, ensuring continuous music playback through the designated mount point on the server.

2.2 Priority Based Scheduling

Priority-based scheduling assigns tasks different levels of importance to manage their execution order effectively, especially in computing systems where task timing is crucial. In the provided example, tasks are ranked from highest to lowest priority as follows:

1. Pause Operation (Sporadic Task): This task receives the highest priority to ensure immediate system response to user commands, enhancing user experience by allowing for instant interruption and resumption of processes.

2. Voice Data Sending (Aperiodic Task): This task is assigned a mid-level priority, recognizing the need for timely and clear voice data transmission in applications like VoIP. By prioritizing voice data over less time-sensitive tasks, the system ensures minimal delay and maintains communication quality.

3. MP3 Data Sending (Periodic Task): Given the lowest priority, this task involves regular transmission of audio streams, such as music. While important for user enjoyment, it is considered less critical than instant system responses or real-time voice communication and thus is prioritized accordingly.

This hierarchical organization allows a computing system to prioritize urgent user inputs and real-time communication over routine data transmission tasks. By doing so, it ensures responsiveness and maintains high-quality communication, thereby improving overall user experience.

2.3 Earliest Deadline First (EDF) Algorithm

The EDF algorithm prioritizes tasks based on their deadlines. This approach is particularly effective in scenarios where tasks have strict timing requirements. Our application calculates the deadlines from the XML configuration file and schedules tasks accordingly to ensure timely execution.

2.4 Streaming Audio to Icecast Server

Once the periodic task (MP3 data sending) is initiated, users can enjoy continuous music playback through the Icecast server's mount point link. The application seamlessly handles the streaming of both voice and MP3 data, ensuring a smooth and uninterrupted listening experience for the audience.

3 Development Environment Set Up

3.1 Ubuntu 64 bit OS installation

1. Prerequisites: Install VirtualBox, download Ubuntu 64-bit ISO.
2. Create VM: In VirtualBox, create a new VM with Linux type and Ubuntu (64-bit) version, allocate RAM, and create a virtual hard disk.
3. Mount ISO: Attach the Ubuntu ISO to the VM's storage settings.
4. Install Ubuntu: Boot VM from ISO, follow installation prompts, and restart without the ISO.
5. Configure Networking: Set VM network to "Bridged Adapter" and select host's network interface.
6. Start VM: Boot the Ubuntu VM with network bridge enable.

3.2 Third party library instllation

Prerequisites:
Before proceeding with the installations, update your system with the following commands:

```
sudo apt update
sudo apt upgrade
```

G++ Installation:
G++ is a compiler in the GNU Compiler Collection (GCC) for compiling C++ code.

```
sudo apt install g++
```

LibShout Installation:
LibShout is used for sending streaming data to an Icecast server.

```
sudo apt install libshout3 libshout3-dev
```

ALSA Installation:
ALSA (Advanced Linux Sound Architecture) provides audio functionality, including voice data detection.

```
sudo apt install libasound2 libasound2-dev
```

IceCast2 Server Installation:
IceCast is a streaming media server supporting various audio and video streams.

```
sudo apt install icecast2
```

TinyXML Installation:
TinyXML is a simple, small, C++ XML parser.

```
sudo apt install libtinyxml2-dev
```

Make Installation:
Make is a build automation tool that reads files called Makefiles to build executable programs.

```
sudo apt install make
```

SSH Server Installation:
OpenSSH server enables secure remote access to your Linux system.

```
sudo apt install openssh-server

sudo systemctl status ssh

sudo systemctl start ssh
```

3.3 ICE cast Server installation

Installation of Icecast2
Begin by updating your system's package list and installing Icecast2. During installation, you will be prompted to configure basic settings, including passwords. Ensure to use secure passwords as recommended.

```
sudo apt update
sudo apt install IceCast2
```

Configuring Icecast2
To configure a mount point in Icecast2, you must edit the `icecast.xml` configuration file, typically located in `/etc/icecast2/`. A mount point allows you to define a specific stream or media content that clients can connect to. Follow these steps:

1. Open the Icecast2 configuration file for editing:

   ```
   sudo nano /etc/icecast2/icecast.xml
   ```

2. Locate the `<mount>` section within the configuration file. If a specific section for your desired mount point does not exist, you can add a new one. Below is an example configuration for a new mount point named `/mystream`:

```
<mount>
    <mount-name>/mystream</mount-name>
    <username>source</username>
    <password>yourpassword</password>
    <max-listeners>100</max-listeners>
    <mime-type>audio/mpeg</mime-type>
    <public>1</public>
    <dump-file>/var/log/icecast2/mystream.dump</dump-file>
</mount>
```

3. In the example above, replace `yourpassword` with a secure password. This configuration sets up `/mystream` as a publicly accessible mount point, with a maximum listener limit of 100, and streams audio in MPEG format. The `<dump-file>` directive specifies a file to save the stream data, useful for debugging or recording.

4. After editing, save the changes and close the editor. Then, restart the Icecast2 service to apply the new configuration:

```
sudo systemctl restart icecast2
```

How to Listen Stream:
First, you need the URL of the Icecast stream. This typically consists of the Icecast server's domain or IP address, the port number, and the mount point path. It usually looks something like this:

```
http://example.com:8000/mountpoint
```

- **http://example.com** is the domain name or IP address of the server where Icecast is running.
- **8000** is the port number on which the Icecast server is broadcasting. The default port for Icecast is 8000, but this can be configured differently.
- **/mountpoint** is the specific mount point you want to listen to. The mount point is defined in the Icecast server's configuration.

To listen to the stream, enter this URL into the address bar of your web browser and press Enter. The browser should begin playing the stream. No additional plugins or software are required, as most modern browsers have built-in support for streaming audio content.

4 Source Code Structure

This document provides a detailed overview of the MainRTS project's source code. The project is organized into several key classes, each responsible for distinct functionalities within the real-time scheduling and streaming application.

4.1 Class Documentation

Following Classes have been created to implement various functionalities for this overall project

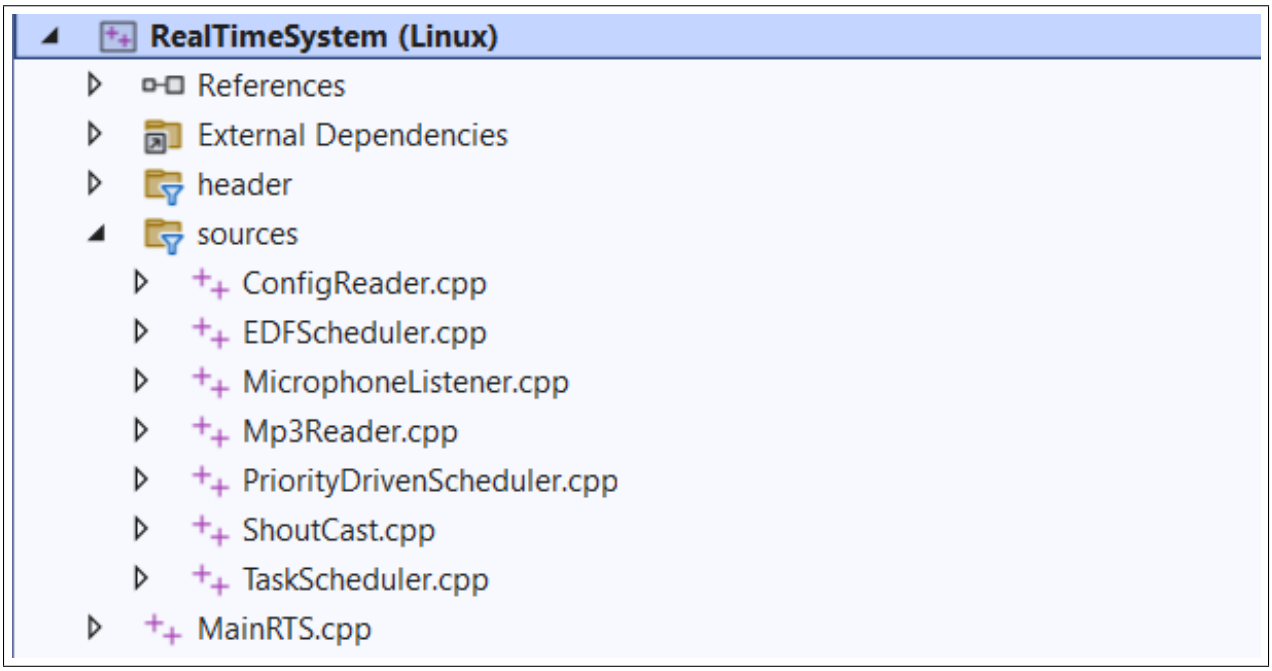


Figure 1: Source Code Structure

TaskScheduler: The `TaskScheduler` class serves as a basic virtual class for task scheduling, defining the interface for scheduler implementations.

PriorityDrivenScheduler: Implements the `TaskScheduler` for a priority-driven scheduling mechanism.

EDFScheduler: Implements the `TaskScheduler` for Earliest Deadline First (EDF) scheduling algorithm.

MicrophoneListener: Detects audio from the microphone and triggers an event upon detection of a significant sound level.

Mp3Reader: Reads MP3 file content in chunks.

ShoutCast: Facilitates streaming of data to the Icecast2 server.

ConfigReader: Responsible for reading task data and parameters, such as task type, priority, and execution time, from a `config.xml` file.

4.2 Main Program (MainRTS.cpp) Structure

Implements the main program logic for the executable named `MainRTS`. Major parts of the code consist of:

Periodic task scheduling(Non-Blocking):

```
//Non Blocking Part
//Launch a thread to execute simulatePeriodicTask every 5 seconds
std::thread periodicThread([&running]() {
    while (running) {


        scheduler->addTask(ConstructTask(periodic, periodicTaskAction));
        std::this_thread::sleep_for(std::chrono::milliseconds(ConfigReader::getInstance().periodicTaskPeriod()));
    }
    });
```

Figure 2: Periodic Task Scheduling

Aperiodic task scheduling(Non-Blocking):

```
MicrophoneListener microphoneDataListener;
microphoneDataListener.startCapturing([](char* data, size_t size)
    {
    //this a Event Drivent function. It will Execute when certain amount of Data is found in microphone
    //std::cout << "-----> Micrpone Audio Detected <--------" << std::endl;

        scheduler->addTask(ConstructTask(aperiodic, aperiodicTaskAction));
    });
```

Figure 3: Aperiodic Task Scheduling

Sporadic task scheduling(Blocking):

```
//blocking the execution here
while (getchar()) {
    //Schedule Sporadic task here
    scheduler->addTask(ConstructTask(sporadic, sporadicTaskAction));
}
```

Figure 4: Sporadic Task Scheduling

5 Source Code Build and Run Process

5.1 Build Process

Navigate to the project directory (where the Makefile is located) and execute the following commands in your terminal.

Clean the Build
To remove any previously compiled files, ensuring a clean state:

```
make clean
```

Compile and Link
To compile the source files and link the object files to create the executable:

```
make
```

5.2 Configuration of the program

This document explains the structure and key elements of the XML configuration file used by our application. The XML configuration file defines the settings and parameters for various tasks and services.

`IsEdfEnabled`

Determines whether the Earliest Deadline First (EDF) scheduling is enabled or not.

- **0:** EDF scheduling is disabled.
- **1:** EDF scheduling is enabled.

`IsPreemptionEnabled`

Indicates if task preemption is allowed in the scheduler.

- **0:** Preemption is disabled.
- **1:** Preemption is enabled.

`ShoutCastServer`

Defines the configuration for connecting to a ShoutCast server.

- **ServerAddress:** The IP address of the ShoutCast server.
- **ServerPort:** The port number on which the ShoutCast server is running.
- **MountPoint:** The mount point URL for streaming.
- **MountPassword:** Password for accessing the mount point.

`Microphone`

Configuration related to the microphone input.

- **RmsThreshold:** The RMS threshold for detecting audio from the microphone.

Task Configuration

Defines settings for different types of tasks: `AperiodicTask`, `SporadicTask`, and `PeriodicTask`.

- **Priority:** Task priority level (0: low, 1: mid, 2: high).
- **ExecutionTime:** The time in milliseconds the task is expected to run.
- **Deadline:** The deadline in milliseconds from the scheduled time within which the task should complete.
- **Period (PeriodicTask only):** The period in milliseconds for the periodic task.

`LocalMusicFileName`

Specifies the name of the local music file to be streamed.

XML Example

Below is an example snippet of the configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
        <IsEdfEnabled>0</IsEdfEnabled>  <!-- disable = 0, enable = 1 -->
        <IsPreemptionEnabled>1</IsPreemptionEnabled>
    <ShoutCastServer>
        <ServerAddress>127.0.0.1</ServerAddress>
        <ServerPort>8000</ServerPort>
        <MountPoint>/music</MountPoint>
        <MountPassword>hackme</MountPassword>
    </ShoutCastServer>
    <Microphone>
        <RmsThreshold>1000</RmsThreshold>
    </Microphone>
    <SporadicTask>
        <Priority>2</Priority>  <!-- low = 0, mid = 1, high =2  -->
        <ExecutionTime>3000</ExecutionTime>  <!-- in milliseconds -->
        <Deadline>3500</Deadline>  <!-- in millisec from Scheduled Time -->
    </SporadicTask>
    <AperiodicTask>
        <Priority>1</Priority>  <!-- low = 0, mid = 1, high =2  -->
        <ExecutionTime>800</ExecutionTime>
        <Deadline>1200</Deadline>
    </AperiodicTask>
    <PeriodicTask>
        <Priority>0</Priority>  <!-- low = 0, mid = 1, high =2  -->
```

```
        <Period>4000</Period>    <!-- in millisec -->
        <ExecutionTime>3000</ExecutionTime>
        <Deadline>4500</Deadline>
    </PeriodicTask>
        <MountPoint>/music</MountPoint>
        <LocalMusicFileName>sample.mp3</LocalMusicFileName>
</Configuration>
```

5.3 How to Run the program

After successfully compiling the project with the provided Makefile, an executable named `MainRTS` will be created in the same directory. This executable is the main program that integrates all the functionalities described in the source code. To run this executable, follow the steps outlined below:

Running the Executable

Ensure that you are in the directory containing the `MainRTS` executable. Open a terminal in this directory. To run the program, execute the following command:

./MainRTS

Upon running this command, the `MainRTS` program will start executing. The program's behavior and output will depend on the specific implementation and configurations set in your code and the XML configuration file.

Expected Output

After starting the program, observe the terminal for any output or prompts from the program. The exact output and behavior will vary based on your program's functionality and how it's configured to interact with the user or other systems.

Troubleshooting

If you encounter any issues running the `MainRTS` executable, consider the following troubleshooting steps:

- Ensure that the executable has the necessary permissions to run. You can set the executable permission with the command `chmod +x MainRTS` if needed.
- Verify that all external dependencies, libraries, and services (like Icecast2 server) are correctly installed and configured.
- Check the XML configuration file for any misconfigurations that might prevent the program from running as expected.

6 Result

6.1 Priority Driven Scheduling

Low-Priority Task:

This is a periodic task that runs regularly. Its function is not specified, but it operates continuously until interrupted.

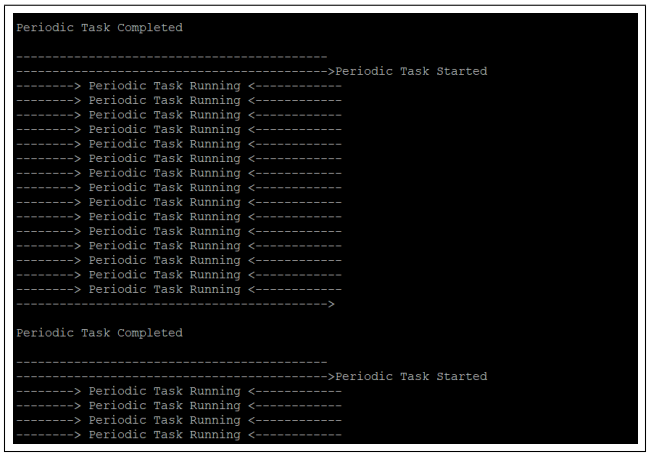


Figure 5: Low Priority Task

Medium-Priority Task:

This task is aperiodic based on a specific trigger— in this case, the detection of a reasonable amount of sound. When this condition is met, the medium-priority task preempts the low-priority task, taking over the processing resources.

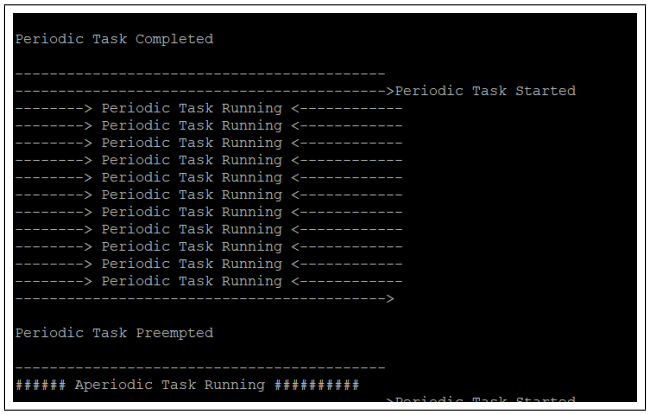
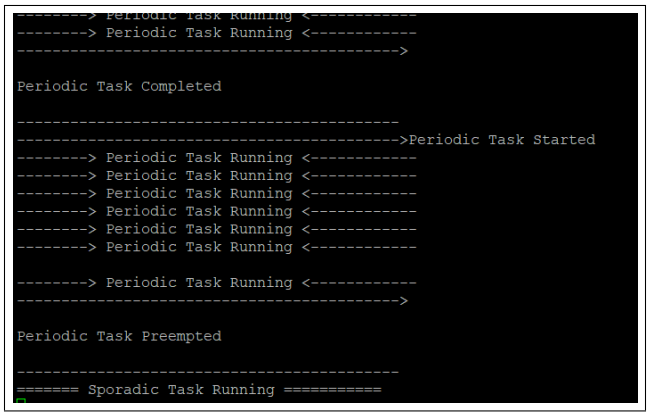


Figure 6: Mid Priority Task

High-Priority Task:

It could be activated by various conditions(— in this case, pressing ”Enter” key on keyboard) deemed to be of utmost importance. Upon activation, this task preempts low-priority periodic task. This hierarchy ensures that the most critical operations are attended to immediately, regardless of the state of lower-priority tasks.



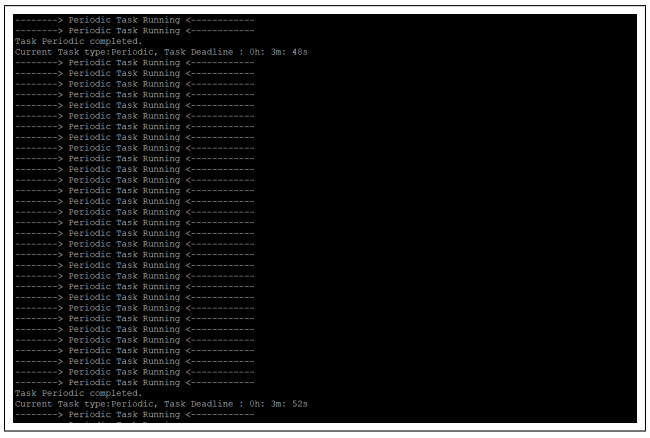
```
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
------------------------------------------->

Periodic Task Completed

-------------------------------------------
------------------------------------------->Periodic Task Started
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------

--------> Periodic Task Running <------------
------------------------------------------->

Periodic Task Preempted

-------------------------------------------
======= Sporadic Task Running ===========
```

Figure 7: High Priority Task

6.2 Earliest Deadline First Scheduling

Deadline Handling:

Each task has an associated deadline. Before execution, the scheduler checks if executing the task would exceed this deadline. If the deadline would be missed, the task is skipped, and an appropriate message is logged.



```
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
Task Periodic completed.
Current Task type:Periodic, Task Deadline : 0h: 3m: 48s
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
--------> Periodic Task Running <------------
Task Periodic completed.
Current Task type:Periodic, Task Deadline : 0h: 3m: 52s
--------> Periodic Task Running <------------
```

Figure 8: Deadline Management

Preemption Logic:

The scheduler supports preemption for periodic tasks if a new task with an earlier deadline arrives. This is checked during the execution of the periodic task. If a higher priority task (earlier deadline) is found, the current task is preempted, and control returns to the main loop to handle the new task.

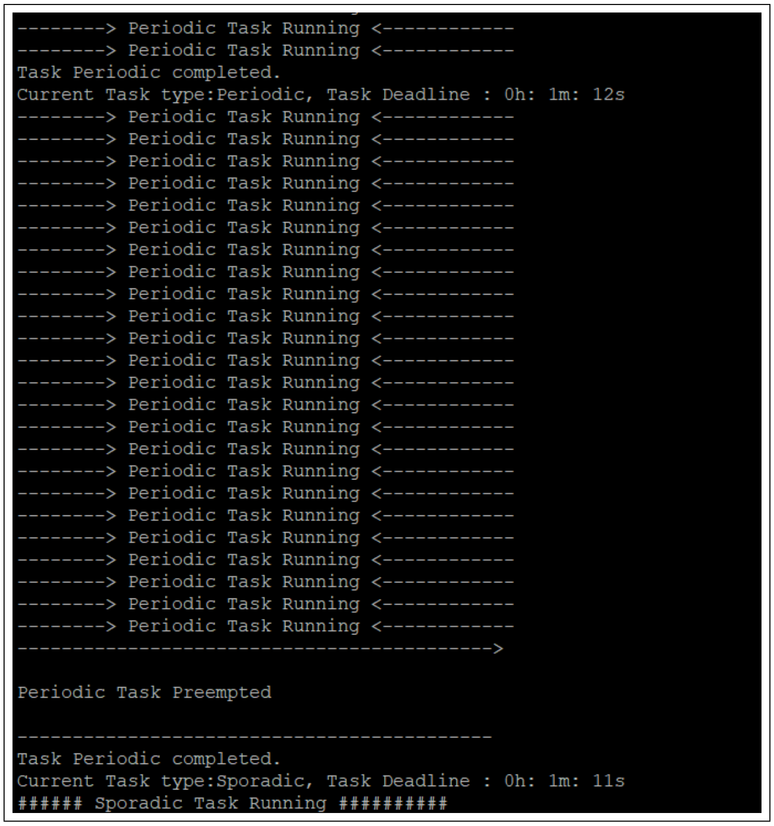


Figure 9: Preemption

7 Future Task

Our ongoing project, which focuses on the real-time scheduling and transmission of MP3 and voice data to an Icecast2 server, is set to undergo further enhancements. These enhancements aim to improve system efficiency, functionality, and user experience. We have outlined several key tasks for future development, including simulating a multi-processor system, exploring new scheduling algorithms, and enhancing the audio clarity of streamed voice data.

Simulate a Multi-Processor System

To better understand how our scheduling algorithms perform under increased loads and in a more realistic computing environment, we plan to simulate a multi-processor system.

Explore Additional Scheduling Algorithms

Our project will also explore additional scheduling algorithms beyond the currently used Priority-Driven and EDF algorithms, such as Rate Monotonic Scheduling (RMS) and Least Laxity First (LLF).

Voice Streaming

Currently, MP3 data streaming is fully functional; however, voice data streaming has yet to be implemented due to challenges associated with converting audio types.

8 Conclusion

This document outlines the integration of real-time scheduling algorithms within a system designed to manage the transmission of MP3 and voice data to an Icecast2 server. It employs Priority-Based and Earliest Deadline First (EDF) scheduling algorithms to handle various task types—aperiodic, sporadic, and periodic—with different priorities to ensure optimal system performance. aperiodic tasks handle real-time voice streaming, sporadic tasks manage system pauses with the highest priority for responsiveness, and periodic tasks ensure continuous MP3 data playback. The project aims to enhance the theoretical and practical understanding of real-time systems, focusing on reliability and user experience in multimedia streaming services. By prioritizing tasks based on urgency and deadlines, the system effectively balances immediate user interactions with ongoing data transmission, demonstrating a practical application of scheduling algorithms that bridge theoretical concepts with real-world multimedia management.