Thread Analysis Report

1. Analysis of Problematic Threads

PassThroughMessageProcessor-107
This thread is part of the pool that processes messages in a pass-through manner.
Its stack trace shows that it is parked inside a blocking call on a LinkedBlockingQueue.take.
It is waiting for a task from the thread pool executor using methods such as LockSupport.park and AbstractQueuedSynchronizer$ConditionObject.await.
A high number of similar threads (e.g. up to PassThroughMessageProcessor-107) indicate that the thread pool may be overprovisioned or misconfigured.
This can lead to resource exhaustion when a payload is sent because many idle threads may delay task acquisition or overwhelm system resources.

PassThroughMessageProcessor-1
This thread is similar in behavior to PassThroughMessageProcessor-107.
It is parked waiting for a new task from the thread pool executor and its call stack follows the exact blocking pattern.
The large accumulation of such threads may further exacerbate resource contention, possibly delaying the processing of incoming payloads.

PollTimer
The PollTimer thread is waiting on an Object.wait call inside a Timer thread loop.
This is standard behavior for timer threads that sleep until a scheduled task needs to run.
There are no indications of a lock problem with PollTimer, as it is not blocking any critical processing directly.

Timer-0
Like PollTimer, Timer-0 is also executing in a Timer thread that waits on Object.wait.
This operation is normal, indicating that this thread is correctly waiting for its next scheduled timer event.
There is no direct evidence that Timer-0 contributes to the payload error.

HTTP-Listener I/O dispatcher-1
This thread is part of the I/O dispatcher for HTTP listeners, waiting on incoming network events.
Its stack trace shows it is in a native polling call (KQueue.poll) and then selecting channels via SelectorImpl.
Its waiting state indicates that it is idle, waiting for network I/O, and it is not directly implicated in the processing error.

2. Relationship of Thread Issues to System Logs

The system logs show an error related to payload processing, specifically:
"Could not get parser from data source for element jsonObject" and "Illegal character: <d".
This indicates that the payload being sent does not properly conform to the expected JSON format.
While the thread dumps show many PassThroughMessageProcessor threads parked in a waiting state, these threads become active when payloads are processed.
If the payload is malformed, the PayloadFactoryMediator fails during the transformation, leading to the XMLStreamException seen in the logs.
The high number of idle PassThroughMessageProcessor threads suggests that when a payload is finally submitted, the task management might be inefficient, potentially contributing to resource exhaustion and poor error recovery during input processing.

3. Overall Diagnosis of the Root Cause

The root cause appears to be twofold.
First, there is a misconfiguration or overprovisioning of the thread pool for PassThroughMessageProcessor threads, which could be leading to resource mismanagement when handling actual payloads.
Second, the payload error ("Illegal character: <d") indicates that the data sent to the endpoint is not valid JSON.
This malformed payload is causing the PayloadFactoryMediator to fail during its parsing attempt, triggering the error in the logs.
The combination of thread pool mismanagement and malformed payload input is leading to the micro integrator error observed.

4. Suggested Solutions to Address the Identified Issues

Review and Adjust Thread Pool Configuration
Audit the configuration settings for the PassThroughMessageProcessor thread pool.
Reduce the maximum number of threads to better match the expected workload and available system resources.
Ensure that the thread pool is properly sized to handle peak loads without causing resource exhaustion.

Validate and Sanitize Payloads
Implement input validation at the client or API gateway level to ensure that payloads conform to the expected JSON format before being sent to the endpoint.
Apply schema or format validation to catch illegal characters or malformed JSON early in the processing pipeline.

Review and Configure Payload Transformation Logic
Examine the PayloadFactoryMediator configuration to ensure it correctly handles the expected JSON structure.
Consider adding error-handling or fallback logic in the transformation process so that malformed payloads do not result in cascading failures.

Monitor System Resource Usage
Set up monitoring to track thread pool activity and resource utilization.
This will help identify when the system is nearing resource exhaustion, allowing proactive adjustments in configuration.

Apply Software Updates and Best Practices
Ensure that the micro integrator and related components are updated to versions that include improved error handling for malformed payloads.
Follow best practices for concurrency management and I/O operations to mitigate the effects of any misconfigurations.

By addressing both the thread pool configuration and the input payload format, the system should be able to process payloads more reliably without exhausting resources or encountering transformation errors.