Revision Notes for Process Management and Diagnostic Script Class

The class focused on the concepts of process management in Linux and the development of a bash script to monitor processes for memory leaks. Here are the key takeaways:

Understanding Processes in Linux

1. **Processes and Threads**:
   - A process in Linux is a running instance of a program. Each process is assigned a unique PID (Process Identifier).
   - Threads are components of processes that execute sequences of programmed instructions.
2. **The `/proc` File System**:
   - `/proc` contains a hierarchy of special files representing the current state of the kernel.
   - Important files within `/proc/<PID>/` that can be analyzed for resource usage include:
     - `maps` - Shows memory allocation.
     - `status` - Provides various statistics about the process.
3. **Diagnostic Artifacts**:
   - Artifacts refer to any machine evidence or data collected for analysis, particularly useful for diagnosing system states, often used in deployment pipelines.

Bash Script Development

The primary task was to write a bash script to monitor memory usage and restart a process if necessary:

1. **Script Requirements**:

  - Option to restart the process when threshold conditions are breached.
  - Log activities with timestamps.

2. **Steps to Implement the Script**:
   - **Input Handling**:
     - Use `getopts` to parse input options for specifying parameters like `-p <pid>`, `-t <threshold>`, `-i <interval>`, `-d <directory>`, and an optional `-r` for restart【6:4†transcript.txt】.
   - **Validation**:
     - Check if all necessary inputs are provided and valid.
     - Verify if the PID exists using `pgrep`.
   - **Monitoring Loop**:
     - Implement a loop to periodically check memory usage.
     - Use `/proc/<PID>/status` to fetch `VmRSS` (Resident Set Size) and calculate memory utilization percentage.
   - **Threshold Detection**:
     - Compare memory consumption against the user-defined threshold.
   - **Evidence Collection**:
     - On breaching thresholds, collect diagnostics by copying relevant files from `/proc/<PID>/` to the specified directory .
   - **Restart Logic**:
     - Optionally restart the process first using a graceful shutdown with `kill -15` and, if unsuccessful, `kill -9` for forced termination .
3. **Handling Edge Cases**:
   - Prevent excessive resource consumption by the monitoring script itself using priority adjustments.
   - Use methods like `nice` and `renice` to manage the priority and CPU allocation .
4. **Logging and Exit Codes**:
   - Log actions with timestamps for tracking and potential alerting.
   - Exit codes are specified for different states like successful execution, input errors, or process non-existence .

Complementary Topics

- **Control Groups (`cgroups`)**: For restricting memory usage of processes beyond simple monitoring with the script, `cgroups` can be used to cap memory and CPU usage in a production setup .

These notes represent a detailed overview of the session, providing both theoretical and practical insights into process management and monitoring in a Linux environment.