Guardian OS

What is Guardian OS

What is Guardian OS?

Guardian OS was initially called T/TOS (Tandem Transactional Operating System) but soon named Guardian for its ability to protect all data from machine faults and software faults. Guardian is the original name for the Tandem NonStop System (TNS) OS, and was renamed (by HP, the current owner of the TNS) to NonStop OS some time ago.

Guardian OS is a specialized, fault-tolerant operating system designed for mission-critical applications that require continuous availability. It runs on HPE NonStop servers (previously Tandem computers) and is fundamentally different from traditional operating systems like Windows, Linux, or Unix.

cont...

- **What it is**: The **native operating system** for HPE NonStop servers.
- **Core design**: Built from the ground up for **fault tolerance and continuous availability**.
- **Process Model**:
  - Uses **message-based inter-process communication (IPC)**.
  - Runs **protected process pairs** (backup process automatically takes over if the primary fails).
- **Reliability**: Every component (CPU, disk, memory, I/O) has a **redundant pair** to avoid single points of failure.
- **Transaction Management**: Tight integration with **NonStop Transaction Manager (TMF)** ensures ACID compliance.
- **Programming Environment**: Supports **TACL**, **TAL**, **COBOL**, **C**, and **modern compilers**.

Open System Services (OSS)

- **OSS = POSIX-compliant environment** within HPE NonStop.
- Provides a **Unix-like shell**, standard APIs, and open-source tools.
- Runs **alongside Guardian OS** for flexibility and reliability.
- Supports programming in **C, C++, Java, and scripting languages**.
- Enables easy **porting of modern applications** to NonStop.
- Builds on **Guardian's fault-tolerant foundation** for high availability.
- Bridges **modern development practices** with **enterprise-class reliability**.

Guardian vs OSS

	Guardian OS	OSS
Type	Proprietary NonStop OS	POSIX-compliant Unix layer on NonStop
Shell	TACL (Transaction Application Control)	Unix-like (bash/ksh)
Languages	TAL, COBOL, C	C, C++, Java, scripting
Focus	Fault-tolerant transaction processing	Developer portability & open-source tools

IPC (Inter process commn)

Concept

- In Guardian, *all processes are isolated* — they don’t share memory or global variables.
- The only way processes communicate is through **message-based IPC**.
- This design enforces **fault isolation** (a failed process can’t corrupt another process’s memory).

Enscribe (File System)

- **Native, record-oriented** file system on NonStop.
- Optimized for **high-volume OLTP**: predictable latency, strong locking, tight **TMF** (Transaction Management Facility) integration.
- Accessed via **Guardian procedure calls** (e.g., FILE_OPEN_, READ, WRITE, READUPD, WRITEUPD, FILE_CLOSE_) or via **COBOL/C** language runtimes.

Enscribe

Core Features

- **Record-oriented**: Files consist of fixed or variable-length records, not raw byte streams.
- **File types**:
  - **Entry-Sequenced** → append records in order (like logs/queues). “Give me the next record written.”
  - **Relative** → records addressed by slot number (RRN). “Read record #1500.”
  - **Key-Sequenced** → records organized in B-tree by a primary key, optional alternate keys. “Retrieve the record where CUSTOMER-ID = 12345.”
- **Auditing** (via TMF): Ensures **ACID transactions** across multiple files.
- **Concurrency**: Record-level locking (READUPD + WRITEUPD).

Enscribe

Record-Oriented Concept

- A **record** is the **fundamental unit of data** in an Enscribe file.
- Instead of treating a file as one long stream of bytes (like Unix/Windows), Enscribe treats it as a **collection of self-contained records**.
- Applications read/write **entire records**, never partial bytes.

Enscribe

File Naming Hierarchy

The **classic Guardian file system** supports a **3-level hierarchy** (sometimes described as 4 parts including the system name).

\system.$volume.subvolume.filename

1. **System Name** (\PROD, \TEST)
   - Identifies the NonStop system (in Expand networks you can reach remote systems).
2. **Volume** ($DATA01, $SYSnn)
   - Physical/logical disk volume. Always starts with $.
3. **Subvolume** (CUSTSUB, SALES)
   - Directory-like grouping (max 63 entries). Not a free-form folder tree — only *one level*.
4. **File Name** (CUSTFILE, ORDER01)
   - The actual file.

FUP Commands

- Create : FUP CREATE \MYNODE.$DATA01.DEMO.MYFILE, REC 50, BLOCK 4, AUDIT OFF
- Edit : TEDIT \MYNODE.$DATA01.DEMO.MYFILE
- Read : FUP INFO \MYNODE.$DATA01.DEMO.MYFILE, DETAIL
- List :
  - Sequential - FUP LIST \MYNODE.$DATA01.DEMO.MYFILE, DETAIL
  - Key - FUP LIST \MYNODE.$DATA01.DEMO.MYFILE, KEY "000001" TO "000100"
- Delete: FUP PURGE \MYNODE.$DATA01.DEMO.MYFILE

Volumes

- List Volumes : FUP INFO \MYNODE.*
- Sub Volumes : FUP INFO \MYNODE.$DATA01.*
- Setting Default Volume : VOLUME $DATA01, SUBVOL CUSTSUB
- Check working directory : STATUS

Enscribe as a Database

1. Why People Call It a DB

- **Record-oriented access**: You can define files with **primary keys** and **alternate keys**, just like indexes in a database.
- **Random access by key**: A READKEY on a key-sequenced file feels like a SELECT … WHERE PK=….
- **Transactions (ACID)**: When files are **AUDIT ON**, all reads/writes participate in **TMF transactions** (commit/rollback).
- **Concurrency**: Record-level locking ensures multiple users/processes can work safely at the same time.

2. Limitations vs RDBMS

- No **SQL interface** natively — you use **Guardian procedure calls** or COBOL/C APIs.
- No **joins**, **foreign keys**, or **query optimizer**.
- Each file is essentially one “table”; relationships must be coded in the application.
- Schema enforcement is minimal: record layout is defined by the program, not the file system.

TACL CheatSheet

STATUS	Show processes and environment status	STATUS
RUN	Start a new process/program	RUN \PROD.$SYSTEM.SUBVOL.MYPROG, PARAMS
STOP	Stop a process	STOP $PROCESS1
FUP	File Utility Program (file mgmt)	FUP INFO MYFILE
TEDIT	Edit/create text files	TEDIT MYFILE

TACL Cheatsheet

SCF	Subsystem Control Facility	SCF STATUS TMF
VOLUME	Set default volume	VOLUME $DATA01
SUBVOL	Set default subvolume	SUBVOL SALES
DO	Run a command file (script)	DO MYJOB
SET	Define a symbol (variable)	SET MYFILE \PROD.$DATA01.SALES.CUSTFILE
!symbol	Expand a defined symbol	FUP INFO !MYFILE

SCF Commands

Command	Purpose	Example
STATUS <subsys>	Show status of a subsystem	STATUS TMF
START <subsys>	Start a subsystem	START TCPIP
STOP <subsys>	Stop a subsystem	STOP TCPIP
INFO <subsys>	Show configuration details	INFO TMF
ALTER <subsys>	Change configuration (parameters)	ALTER TCPIP, MAXSESSIONS 100
ADD <object>	Add resources (like a PATHWAY server)	ADD SERVERCLASS $MYAPP
DELETE <object>	Remove resources	DELETE SERVERCLASS $OLDAPP

Transaction Processing Monitor

- In large OLTP (online transaction processing) systems, you have thousands of tiny requests (banking transactions, POS purchases, telecom calls).

- A **Transaction Processing Monitor (TPM)** is a **middleware layer** that:

  1. **Accepts client requests**.

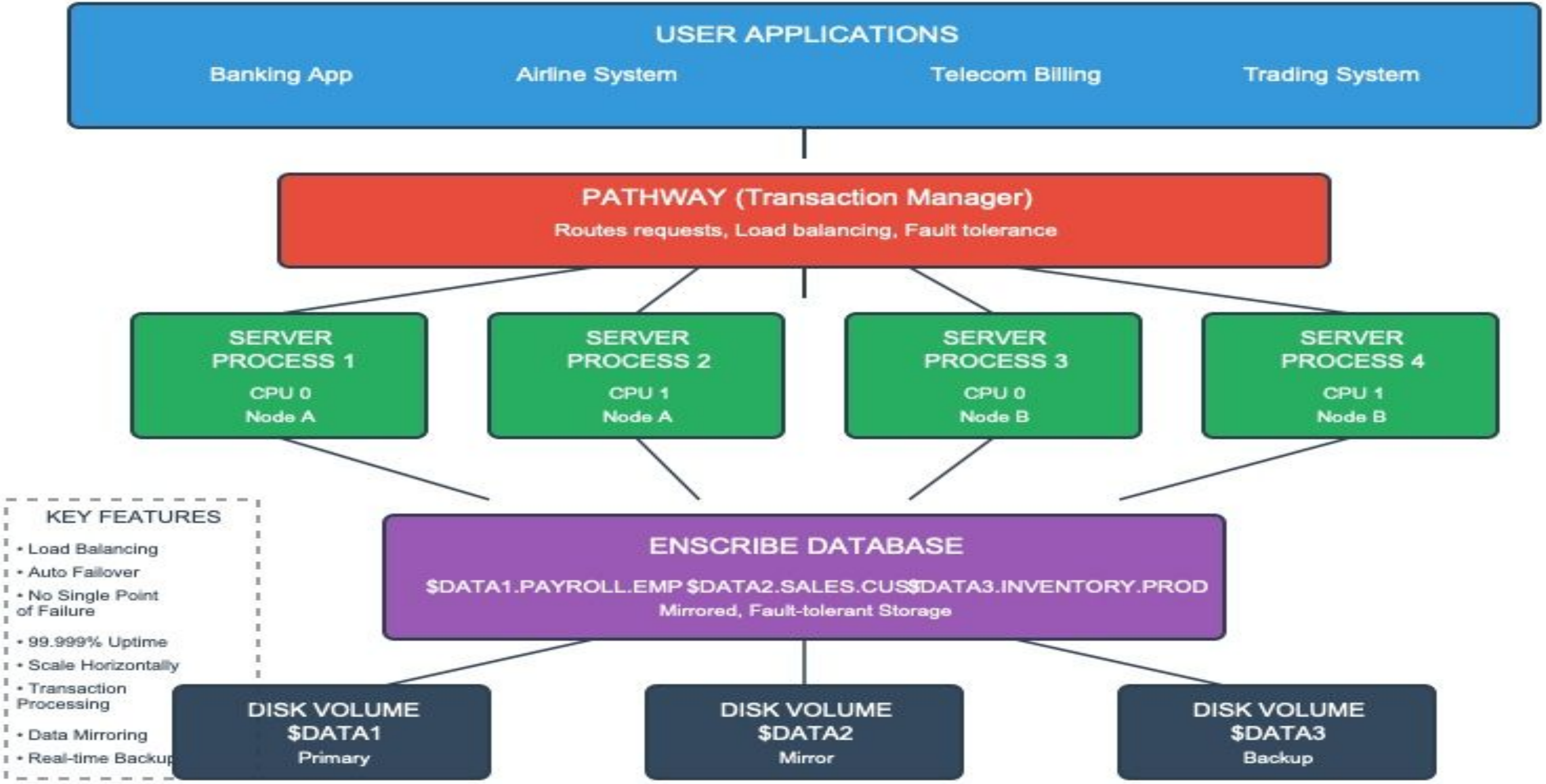
  2. **Dispatches them to available server processes**.

  3. **Balances load** across many servers.

  4. **Handles failures** by restarting crashed servers.

  5. **Ensures transactions are processed reliably** (with TMF for commit/rollback).

NonStop System Pathway



USER APPLICATIONS

Banking App Airline System Telecom Billing Trading System

PATHWAY (Transaction Manager)

Routes requests, Load balancing, Fault tolerance

SERVER PROCESS 1
CPU 0
Node A

SERVER PROCESS 2
CPU 1
Node A

SERVER PROCESS 3
CPU 0
Node B

SERVER PROCESS 4
CPU 1
Node B

ENSCRIBE DATABASE

$DATA1.PAYROLL.EMP $DATA2.SALES.CUS$DATA3.INVENTORY.PROD

Mirrored, Fault-tolerant Storage

DISK VOLUME
$DATA1
Primary

DISK VOLUME
$DATA2
Mirror

DISK VOLUME
$DATA3
Backup

KEY FEATURES

- Load Balancing
- Auto Failover
- No Single Point of Failure
- 99.999% Uptime
- Scale Horizontally
- Transaction Processing
- Data Mirroring
- Real-time Backup

Pathway as TPM on NonStop

- **Pathway** is NonStop’s **native TPM** — built right into Guardian OS.
- Key parts:
  - **Pathmon** (Pathway Monitor) = the “dispatcher” process that manages everything.
  - **Serverclasses** = pools of server processes (like workers).
  - **Requestors** = client programs that call SERVERCLASS_SEND_ to send work.
- **Pathmon routes requests** from requesters to the right server process, waits for reply, and sends it back.

It means Pathway is the **middleware manager** that ensures every client request gets dispatched, balanced, processed, and replied **without downtime**, even under hardware or software failures.

Why Pathway Matters as a TPM

- **Scalability**: you can have dozens or hundreds of server processes in a class — Pathmon spreads requests automatically.
- **Fault Tolerance**: if a server crashes, Pathmon restarts it instantly, and requests are rerouted.
- **Resource Isolation**: each server process handles one request at a time, avoiding interference.
- **Transaction Safety**: integrates with **TMF** so every request runs as a proper ACID transaction.
- **Manageability**: SCF commands (START PATHMON, INFO PATHMON) let ops control the environment.

Basic Check Balance Use case

Requestor (Client)

- Could be an **ATM**, **POS terminal**, or **online app**.
- Client connect to front-end process (TCP IP/Port)
- Sends a **Check Balance request** to the backend using SERVERCLASS_SEND_.

Pathway / Pathmon (TPM)

- **Transaction Processing Monitor (TPM)**:
  - Receives the request.
  - Chooses an available **server process** in the right **serverclass** (e.g., ACCTSRV).
  - Routes the request there.
- Provides **load balancing** and **automatic restart** if a server fails.

Server Process (Application Logic)

- Written in **COBOL, C, or Java**.
- Implements the logic for CheckBalance:
  1. Start a **TMF transaction**.
  2. Read the customer's account record from the **Enscribe file** (or SQL/MP table).
  3. Return the balance as a reply.
  4. End/Commit the transaction.

ATM/POS → TCP Listener (C) → Pathmon (TPM) → AUTHSRV (serverclass) → TMF + Enscribe/SQL → back to ATM.