```
"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
exports.startServer = exports.handleResult = void 0;
const fs_1 = require("fs");
const net_1 = require("net");
const path_1 = require("path");
const perf_hooks_1 = require("perf_hooks");
const file_hasher_1 = require("../../hasher/file-hasher");
const native_1 = require("../../native");
const consume_messages_from_socket_1 = require("../../utils/
consume-messages-from-socket");
const fileutils_1 = require("../../utils/fileutils");
const versions_1 = require("../../utils/versions");
const workspace_context_1 = require("../../utils/workspace-
context");
const workspace_root_1 = require("../../utils/workspace-
root");
const cache_1 = require("../cache");
const socket_utils_1 = require("../socket-utils");
const file_watcher_sockets_1 = require("./file-watching/file-
watcher-sockets");
const handle_hash_tasks_1 = require("./handle-hash-tasks");
const handle_outputs_tracking_1 = require("./handle-outputs-
tracking");
const handle_process_in_background_1 = require("./handle-
process-in-background");
const handle_request_file_data_1 = require("./handle-request-
file-data");
const handle_request_project_graph_1 = require("./handle-
request-project-graph");
const handle_request_shutdown_1 = require("./handle-request-
shutdown");
const logger_1 = require("./logger");
const outputs_tracking_1 = require("./outputs-tracking");
const project_graph_incremental_recomputation_1 = require("./
project-graph-incremental-recomputation");
const shutdown_utils_1 = require("./shutdown-utils");
const watcher_1 = require("./watcher");
let performanceObserver;
let workspaceWatcherError;
let outputsWatcherError;
let numberOfOpenConnections = 0;
const server = (0, net_1.createServer)(async (socket) => {
    numberOfOpenConnections += 1;
    logger_1.serverLogger.log(`Established a connection.
Number of open connections: ${numberOfOpenConnections}`);
    (0, shutdown_utils_1.resetInactivityTimeout)
(handleInactivityTimeout);
    if (!performanceObserver) {
        performanceObserver = new
```

```
perf_hooks_1.PerformanceObserver((list) => {
            const entry = list.getEntries()[0];
            logger_1.serverLogger.log(`Time taken for '$
{entry.name}'`, `${entry.duration}ms`);
        });
        performanceObserver.observe({ entryTypes:
['measure'] });
    }
    socket.on('data', (0,
consume_messages_from_socket_1.consumeMessagesFromSocket)
(async (message) => {
        await handleMessage(socket, message);
    }));
    socket.on('error', (e) => {
        logger_1.serverLogger.log('Socket error');
        console.error(e);
    });
    socket.on('close', () => {
        numberOfOpenConnections -= 1;
        logger_1.serverLogger.log(`Closed a connection. Number
of open connections: ${numberOfOpenConnections}`);
        (0,
file_watcher_sockets_1.removeRegisteredFileWatcherSocket)
(socket);
    });
});
registerProcessTerminationListeners();
async function handleMessage(socket, data) {
    if (workspaceWatcherError) {
        await (0, shutdown_utils_1.respondWithErrorAndExit)
(socket, `File watcher error in the workspace '$
{workspace_root_1.workspaceRoot}'.`, workspaceWatcherError);
    }
    if (daemonIsOutdated()) {
        await (0, shutdown_utils_1.respondWithErrorAndExit)
(socket, `Lock files changed`, {
            name: '',
            message: 'LOCK-FILES-CHANGED',
        });
    }
    (0, shutdown_utils_1.resetInactivityTimeout)
(handleInactivityTimeout);
    const unparsedPayload = data;
    let payload;
    try {
        payload = JSON.parse(unparsedPayload);
    }
    catch (e) {
        await (0, shutdown_utils_1.respondWithErrorAndExit)
(socket, `Invalid payload from the client`, new
Error(`Unsupported payload sent to daemon server: $
{unparsedPayload}`));
    }
    if (payload.type === 'PING') {
        await handleResult(socket, 'PING', () =>
Promise.resolve({ response: JSON.stringify(true), description:
'ping' }));
    }
    else if (payload.type === 'REQUEST_PROJECT_GRAPH') {
        await handleResult(socket, 'REQUEST_PROJECT_GRAPH', ()
=> (0,
handle_request_project_graph_1.handleRequestProjectGraph)());
    }
    else if (payload.type === 'HASH_TASKS') {
        await handleResult(socket, 'HASH_TASKS', () => (0,
handle_hash_tasks_1.handleHashTasks)(payload));
    }
    else if (payload.type === 'REQUEST_FILE_DATA') {
        await handleResult(socket, 'REQUEST_FILE_DATA', () =>
(0, handle_request_file_data_1.handleRequestFileData)());
    }
    else if (payload.type === 'PROCESS_IN_BACKGROUND') {
        await handleResult(socket, 'PROCESS_IN_BACKGROUND', ()
=> (0,
handle_process_in_background_1.handleProcessInBackground)
(payload));
    }
    else if (payload.type === 'RECORD_OUTPUTS_HASH') {
        await handleResult(socket, 'RECORD_OUTPUTS_HASH', ()
=> (0, handle_outputs_tracking_1.handleRecordOutputsHash)
(payload));
    }
    else if (payload.type === 'OUTPUTS_HASHES_MATCH') {
        await handleResult(socket, 'OUTPUTS_HASHES_MATCH', ()
=> (0, handle_outputs_tracking_1.handleOutputsHashesMatch)
(payload));
    }
    else if (payload.type === 'REQUEST_SHUTDOWN') {
        await handleResult(socket, 'REQUEST_SHUTDOWN', () =>
(0, handle_request_shutdown_1.handleRequestShutdown)(server,
numberOfOpenConnections));
    }
    else if (payload.type === 'REGISTER_FILE_WATCHER') {

file_watcher_sockets_1.registeredFileWatcherSockets.push({ soc
ket, config: payload.config });
    }
    else {
        await (0, shutdown_utils_1.respondWithErrorAndExit)
(socket, `Invalid payload from the client`, new
Error(`Unsupported payload sent to daemon server: $
{unparsedPayload}`));
    }
}
async function handleResult(socket, type, hrFn) {
    const startMark = new Date();
    const hr = await hrFn();
    const doneHandlingMark = new Date();
    if (hr.error) {
        await (0, shutdown_utils_1.respondWithErrorAndExit)
(socket, hr.description, hr.error);
    }
    else {
        await (0, shutdown_utils_1.respondToClient)(socket,
hr.response, hr.description);
    }
    const endMark = new Date();
    logger_1.serverLogger.log(`Handled ${type}. Handling time:
${doneHandlingMark.getTime() - startMark.getTime()}. Response
time: ${endMark.getTime() - doneHandlingMark.getTime()}.`);
}
exports.handleResult = handleResult;
function handleInactivityTimeout() {
    if (numberOfOpenConnections > 0) {
        logger_1.serverLogger.log(`There are $
{numberOfOpenConnections} open connections. Reset inactivity
timer.`);
        (0, shutdown_utils_1.resetInactivityTimeout)
(handleInactivityTimeout);
    }
    else {
        (0, shutdown_utils_1.handleServerProcessTermination)({
            server,
            reason: `$
{shutdown_utils_1.SERVER_INACTIVITY_TIMEOUT_MS}ms of
inactivity`,
        });
    }
}
function registerProcessTerminationListeners() {
    process
        .on('SIGINT', () => (0,
shutdown_utils_1.handleServerProcessTermination)({
        server,
        reason: 'received process SIGINT',
    }))
        .on('SIGTERM', () => (0,
shutdown_utils_1.handleServerProcessTermination)({
        server,
        reason: 'received process SIGTERM',
    }))
        .on('SIGHUP', () => (0,
shutdown_utils_1.handleServerProcessTermination)({
        server,
        reason: 'received process SIGHUP',
    }));
}
let existingLockHash;
function daemonIsOutdated() {
    return nxVersionChanged() || lockFileHashChanged();
}
function nxVersionChanged() {
    return versions_1.nxVersion !== getInstalledNxVersion();
}
const nxPackageJsonPath = require.resolve('nx/package.json');
function getInstalledNxVersion() {
    try {
        const { version } = (0, fileutils_1.readJsonFile)
(nxPackageJsonPath);
        return version;
    }
    catch (e) {
        // node modules are absent, so we can return null,
which would shut down the daemon
        return null;
    }
}
function lockFileHashChanged() {
    const lockHashes = [
        (0, path_1.join)(workspace_root_1.workspaceRoot,
'package-lock.json'),
        (0, path_1.join)(workspace_root_1.workspaceRoot,
'yarn.lock'),
        (0, path_1.join)(workspace_root_1.workspaceRoot,
'pnpm-lock.yaml'),
    ]
        .filter((file) => (0, fs_1.existsSync)(file))
        .map((file) => (0, native_1.hashFile)(file));
    const newHash = (0, file_hasher_1.hashArray)(lockHashes);
    if (existingLockHash && newHash != existingLockHash) {
        existingLockHash = newHash;
        return true;
    }
    else {
        existingLockHash = newHash;
        return false;
    }
}
/**
 * When applicable files in the workspaces are changed
```

```
(created, updated, deleted),
 * we need to recompute the cached serialized project graph so
that it is readily
 * available for the next client request to the server.
 */
const handleWorkspaceChanges = async (err, changeEvents) => {
    if (workspaceWatcherError) {
        logger_1.serverLogger.watcherLog('Skipping
handleWorkspaceChanges because of a previously recorded
watcher error.');
        return;
    }
    try {
        (0, shutdown_utils_1.resetInactivityTimeout)
(handleInactivityTimeout);
        if (daemonIsOutdated()) {
            await (0,
shutdown_utils_1.handleServerProcessTermination)({
                server,
                reason: 'Lock file changed',
            });
            return;
        }
        if (err || !changeEvents || !changeEvents.length) {
            let error = typeof err === 'string' ? new
Error(err) : err;
            logger_1.serverLogger.watcherLog('Unexpected
workspace watcher error', error.message);
            console.error(error);
            workspaceWatcherError = error;
            return;
        }
        logger_1.serverLogger.watcherLog((0,
watcher_1.convertChangeEventsToLogMessage)(changeEvents));
        const updatedFilesToHash = [];
        const createdFilesToHash = [];
        const deletedFiles = [];
        for (const event of changeEvents) {
            if (event.type === 'delete') {
                deletedFiles.push(event.path);
            }
            else {
                try {
                    const s = (0, fs_1.statSync)((0,
path_1.join)(workspace_root_1.workspaceRoot, event.path));
                    if (s.isFile()) {
                        if (event.type === 'update') {

updatedFilesToHash.push(event.path);
                        }
                        else {
createdFilesToHash.push(event.path);
                        }
                    }
                }
                catch (e) {
                    // this can happen when the update file
was deleted right after
                }
            }
        }
        (0,
project_graph_incremental_recomputation_1.addUpdatedAndDeleted
Files)(createdFilesToHash, updatedFilesToHash, deletedFiles);
    }
    catch (err) {
        logger_1.serverLogger.watcherLog(`Unexpected workspace
error`, err.message);
        console.error(err);
        workspaceWatcherError = err;
    }
};
const handleOutputsChanges = async (err, changeEvents) => {
    try {
        if (err || !changeEvents || !changeEvents.length) {
            let error = typeof err === 'string' ? new
Error(err) : err;
            logger_1.serverLogger.watcherLog('Unexpected
outputs watcher error', error.message);
            console.error(error);
            outputsWatcherError = error;
            (0, outputs_tracking_1.disableOutputsTracking)();
            return;
        }
        if (outputsWatcherError) {
            return;
        }
        logger_1.serverLogger.watcherLog('Processing file
changes in outputs');
        (0, outputs_tracking_1.processFileChangesInOutputs)
(changeEvents);
    }
    catch (err) {
        logger_1.serverLogger.watcherLog(`Unexpected outputs
watcher error`, err.message);
        console.error(err);
        outputsWatcherError = err;
        (0, outputs_tracking_1.disableOutputsTracking)();
    }
```

```
};
async function startServer() {
    (0, workspace_context_1.setupWorkspaceContext)
(workspace_root_1.workspaceRoot);
    // Persist metadata about the background process so that
it can be cleaned up later if needed
    await (0, cache_1.writeDaemonJsonProcessCache)({
        processId: process.pid,
    });
    // See notes in socket-command-line-utils.ts on OS
differences regarding clean up of existings connections.
    if (!socket_utils_1.isWindows) {
        (0, socket_utils_1.killSocketOrPath)();
    }
    return new Promise(async (resolve, reject) => {
        try {
            server.listen(socket_utils_1.FULL_OS_SOCKET_PATH,
async () => {
                try {
                    logger_1.serverLogger.log(`Started
listening on: ${socket_utils_1.FULL_OS_SOCKET_PATH}`);
                    // this triggers the storage of the lock
file hash
                    daemonIsOutdated();
                    if (!(0,
shutdown_utils_1.getWatcherInstance)()) {
                        (0,
shutdown_utils_1.storeWatcherInstance)(await (0,
watcher_1.watchWorkspace)(server, handleWorkspaceChanges));

logger_1.serverLogger.watcherLog(`Subscribed to changes
within: ${workspace_root_1.workspaceRoot} (native)`);
                    }
                    if (!(0,
shutdown_utils_1.getOutputWatcherInstance)()) {
                        (0,
shutdown_utils_1.storeOutputWatcherInstance)(await (0,
watcher_1.watchOutputFiles)(handleOutputsChanges));
                    }
                    return resolve(server);
                }
                catch (err) {
                    await handleWorkspaceChanges(err, []);
                }
            });
        }
        catch (err) {
            reject(err);
        }
    });
```

```
}
exports.startServer = startServer;
```