|  |  |
| --- | --- |
| **Katedra Oprogramowania Systemy Operacyjne** | Realizacja projektu: 14-21.05.2023 |
| Temat projektu: **Demon monitorujący katalog**  Przygotowali: Michał Niedźwiecki i Sebastian Syperek | Prowadzący: mgr inż. Daniel Reska |

1. **Treść zadania do realizacji.**

Celem usługi jest monitorowanie zmian plików w podanym katalogu. Program startowy otrzymuje co najmniej dwa argumenty: ścieżkę źródłową i ścieżkę docelowa . Jeżeli któraś ze ścieżek nie jest katalogiem program powraca natychmiast z komunikatem błędu. W przeciwnym wypadku staje się demonem. Demon wykonuje następujące czynności:

* przy pierwszym uruchomieniu skanuje katalog w poszukiwaniu plików i tworzy strukturę danych zawierającą ich aktualny stan (więcej poniżej);
* proces demon usypia domyślnie na pięć minut (czas spania można zmieniać przy pomocy dodatkowego opcjonalnego argumentu);
* po obudzeniu skanuje ponownie katalog i porównuje aktualny stan plików ze stanem zapisanym na starcie, wykonuje niezbędne operacje i usypia ponownie.

Możliwe jest natychmiastowe obudzenie się demona poprzez wysłanie mu sygnału SIGUSR1. Komunikaty demona, jak informacja o każdej akcji typu uśpienie/obudzenie się (naturalne lub w wyniku sygnału) czy wykonanie operacji na plikach, są przesyłane do logu systemowego (syslog). Operacje porównania stanu działają wg następujących zasad:

* napotkanie nowego plik w monitorowanym katalogu powinno spowodować zalogowanie informacji o nim do logu systemowego, podobnie w przypadku zniknięcia pliku istniejącego na starcie;
* w przypadku zmiany daty modyfikacji lub rozmiaru istniejącego już pliku, jego nowa wersja powinna być skopiowana do katalogu docelowego;
* demon powinien zaktualizować informacje o zmodyfikowanym pliku, aby przy kolejnym obudzeniu nie trzeba było wykonać kopii (chyba że plik w katalogu źródłowym zostanie ponownie zmieniony);
* pozycje, które nie są zwykłymi plikami są ignorowane (np. katalogi i dowiązania symboliczne);
* operacje kopiowania mają być wykonane za pomocą niskopoziomowych operacji typu copy\_file\_range lub sendfile.

Wersja podstawowa**: 15p**. Dodatkowo:

* Zamiast wielkości i daty modyfikacji demon sprawdza sumy kontrolne plików (np. algorytmem z rodziny SHA). Uwaga: można skorzystać tutaj z bibliotek zewnętrznych (np. biblioteki openssl), natomiast same operacje odczytu plików powinny opierać się na API niskopoziomowym**(5p)**.
* Opcja -R pozwalająca na rekurencyjną synchronizację katalogów (teraz pozycje będące katalogami nie są ignorowane). Przy kopiowaniu zmienionych plików struktura katalogów powinna być zachowana w katalogu docelowym **(5p).**

1. **Sposób uruchomienia demona.**

Po rozpakowaniu archiwum ***Final-daemon-main.zip***, należy przejść do folderu z wypakowanymi plikami projektowymi. W otwartym folderze należy otworzyć terminal i stworzyć plik wykonywalny ***daemon*** poleceniem ***make***, które doprowadzi do skompilowania programu. Po kompilacji (wyświetlenie ***gcc daemon.c fileoperations.c -o daemon -lssl -lcrypto***, oraz brak informacji o zakończeniu z kodem błędu) możliwe jest uruchomienie programu.

Aby poprawnie uruchomić program, należy użyć komendy:

|  |
| --- |
| ***./daemon <ścieżka folderu źródłowego> <ścieżka folderu docelowego>*** |

Niepoprawne użycie komendy wywoła na ekran następującą wiadomość:

|  |
| --- |
| ***Program usage: ./daemon <source\_path> <destination\_path> [options] Check available options by: ./daemon --help.*** |

Użycie komendy ***./daemon –help***wyświetli na ekranie poprawne użycie programu wraz z opcjonalnymi argumentami:

|  |
| --- |
| ***Program usage: ./daemon <source\_path> <destination\_path> [options]***  ***Available options:***  ***-t <number>: Sets new sleep interval for Daemon [seconds] (300sec by default).***  ***-R: Recursive synchronization of subdirectiories included.*** |

Aby program działał, podane katalogi muszą istnieć.

1. **Opis stosowanych funkcji.**

Zastosowane w programie funkcje zostały odseparowane w oddzielnym pliku ***fileoperations.c***.

***- struct FileEntry \*createFileEntry(const char \*name, const char \*path, enum FileType type)*** – funkcja *createFileEntry*() tworzy nowy obiekt struktury *FileEntry*, inicjalizuje go wartościami przekazanymi jako argumenty (nazwa pliku, ścieżka pliku, typ pliku) i zwraca ten obiekt jako wynik funkcji.

- ***void insertFileEntry(struct FileEntry \*\*head, struct FileEntry \*entry)*** – funkcja insertFileEntry umieszcza nowy wpis pliku (*entry*) na końcu listy, niezależnie od jej rozmiaru i aktualnej zawartości.

- ***void calculateMD5(const char \*filePath, unsigned char \*md5sum)*** – funkcja otwiera plik, odczytuje go porcjami, aktualizuje sumę kontrolną MD5 na podstawie odczytanych danych i zwraca wynik w postaci tablicy bajtów md5sum.

- ***int compareHashes(const unsigned char \*hash1, const unsigned char \*hash2)*** – funkcja porównuje dwie sumy kontrolne MD5 i zwraca 1, jeśli są identyczne, lub 0, jeśli są różne.

- ***void traverseDirectory(const char \*path, struct FileEntry \*\*head)*** – ta funkcja odpowiedzialna za rekurencyjne przeglądanie katalogu. Przegląda zawartość katalogu o podanej ścieżce (path) i tworzy wpisy plików (struktury FileEntry) dla każdego pliku i podkatalogu w tym katalogu. Wpisy są dodawane do listy (head), która jest wskaźnikiem na wskaźnik do struktury FileEntry.

- ***void freeLinkedList(struct FileEntry \*head)*** – funkcja zwalnia pamięć zajmowaną przez listę jednokierunkową, której głową jest wskaźnik na strukturę FileEntry (head).

- ***int isElementInLinkedList(struct FileEntry\* element, struct FileEntry\* head)*** – funkcja sprawdza, czy dany element (wskazywany przez wskaźnik element) znajduje się w liście jednokierunkowej, której głową jest wskaźnik head do struktury FileEntry. Funkcja porównuje pola name, path i type elementu element z odpowiadającymi polami w każdym elemencie listy.

- ***char\* absoluteToRelative(const char\* absolutePath, const char\* currentFolder)*** – funkcja przekształca ścieżkę bezwzględną (absolutePath) na ścieżkę względną w stosunku do określonego folderu bieżącego (currentFolder).

- ***void getDestinationFilePath(char\* temp\_path, char\* destination\_path, char\* current\_path, char\* source\_path)*** – funkcja służy do uzyskania pełnej ścieżki pliku docelowego na podstawie ścieżki aktualnego pliku, ścieżki źródłowej oraz ścieżki docelowej.

- ***void copyFile(char\* copyFromPath, char\* copyToPath)*** – funkcja kopiuje zawartość pliku źródłowego copyFromPath do pliku docelowego copyToPath, korzystając z podanych ścieżek.