

***Systemy Operacyjne***

**Dokumentacja Projektu**

Wykonujący ćwiczenie:

* **Mateusz Mogielnicki**
* **Dominik Mierzejewski**
* **Przemysław Rutkowski**
* **Jakub Matyszak**

Studia dzienne Kierunek: Informatyka Semestr: IV

Grupa zajęciowa: PS6

Prowadzący ćwiczenie: mgr inż. Tomasz Kuczyński Data wykonania projektu: 27.04.2023r.

**Treść Projektu**

***Demon synchronizujący dwa podkatalogi [12p.]***

Program który otrzymuje co najmniej dwa argumenty: ścieżkę źródłową oraz ścieżkę docelową . Jeżeli któraś ze ścieżek nie jest katalogiem program powraca natychmiast z komunikatem błędu. W przeciwnym wypadku staje się demonem. Demon wykonuje następujące czynności: śpi przez pięć minut (czas spania można zmieniać przy pomocy dodatkowego opcjonalnego argumentu), po czym po obudzeniu się porównuje katalog źródłowy z katalogiem docelowym. Pozycje które nie są zwykłymi plikami są ignorowane (np. katalogi i dowiązania symboliczne). Jeżeli demon (a) napotka na nowy plik w katalogu źródłowym, i tego pliku brak w katalogu docelowym lub (b) plik w katalogu źrodłowym ma późniejszą datę ostatniej modyfikacji demon wykonuje kopię pliku z katalogu źródłowego do katalogu docelowego - ustawiając w katalogu docelowym datę modyfikacji tak aby przy kolejnym obudzeniu nie trzeba było wykonać kopii (chyba ze plik w katalogu źródłowym zostanie ponownie zmieniony). Jeżeli zaś odnajdzie plik w katalogu docelowym, którego nie ma w katalogu źródłowym to usuwa ten plik z katalogu docelowego. Możliwe jest również natychmiastowe obudzenie się demona poprzez wysłanie mu sygnału SIGUSR1.

Wyczerpująca informacja o każdej akcji typu uśpienie/obudzenie się demona (naturalne lub w wyniku sygnału), wykonanie kopii lub usunięcie pliku jest przesłana do logu systemowego. Informacja ta powinna zawierać aktualną datę.

**Dodatkowo:**

1. [10p.] Dodatkowa opcja -R pozwalająca na rekurencyjną synchronizację katalogów (teraz pozycje będące katalogami nie są ignorowane). W szczególności jeżeli demon stwierdzi w katalogu docelowym podkatalog którego brak w katalogu źródłowym powinien usunąć go wraz z zawartością.
2. [12p.] W zależności od rozmiaru plików dla małych plików wykonywane jest kopiowanie przy pomocy read/write a w przypadku dużych przy pomocy mmap/write (plik źródłowy) zostaje zamapowany w całości w pamięci. Próg dzielący pliki małe od dużych może być przekazywany jako opcjonalny argument.

**Opis poszczególnych modułów Demona**

***„functions.h”***

// IMPORT POTRZEBNYCH BIBLIOTEK

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdbool.h>

#include <sys/types.h>

#include <dirent.h>

#include <time.h>

#include <limits.h>

#include <string.h>

#include <utime.h>

#include <sys/stat.h>

#include <unistd.h>

#include <signal.h>

#include <sys/mman.h>

#include <errno.h>

#include <syslog.h>

/\*

OPIS ZMIENNYCH GLOBALNYCH

- `recursive` jest zmienną typu `bool`, która przechowuje informację, czy synchronizacja ma być rekurencyjna czy nie. Domyślnie ustawiona jest na `false`.

- `timeSleep` jest zmienną typu `unsigned long`, która przechowuje czas w sekundach, jaki demon ma odczekać między kolejnymi synchronizacjami. Domyślnie ustawiona jest na 5 minut (czyli 5 \* 60 sekund).

- `forcedSynchro` jest zmienną typu `bool`, która przechowuje informację, czy synchronizacja została wymuszona przez sygnał `SIGUSR1`. Domyślnie ustawiona jest na `false`.

- `mmapThreshold` jest zmienną typu `int`, która przechowuje wartość graniczną w MB dla plików, które zostaną podzielone na mniejsze fragmenty podczas kopiowania. Domyślnie ustawiona jest na wartość `100`, co oznacza, że pliki o rozmiarze powyżej 100MB zostaną podzielone na mniejsze fragmenty.

- `mmapThreshold\_DEFAULT` jest stałą, która definiuje domyślną wartość dla zmiennej `mmapThreshold`.

- `MAX\_COPYING\_BUFFER\_SIZE` jest stałą, która definiuje maksymalny rozmiar bufora do kopiowania danych podczas synchronizacji. Domyślnie ustawiona jest na `65536` bajtów (czyli 64KB).

\*/

bool recursive = false;

*unsigned* *long* timeSleep = 5 \* 60;

bool forcedSynchro = false;

#define mmapThreshold\_DEFAULT 100

*int* mmapThreshold = mmapThreshold\_DEFAULT;

#define MAX\_COPYING\_BUFFER\_SIZE 65536

/\* OPIS FUNKCJI

Funkcja "changeModTime" jest używana do zmiany czasu modyfikacji pliku na aktualny czas systemowy. Przyjmuje ona jako argument wskaźnik na łańcuch znaków reprezentujący ścieżkę do pliku, którego czas modyfikacji ma zostać zmieniony.

Funkcja ta korzysta z funkcji systemowej "time", która zwraca aktualny czas systemowy w formacie czasu uniksowego (liczba sekund od 1 stycznia 1970 roku). Następnie tworzy strukturę "utimbuf", która służy do przechowywania czasu dostępu i modyfikacji pliku. W tym przypadku, czas dostępu nie jest zmieniany, więc zostaje ustawiony na aktualny czas systemowy. Czas modyfikacji zostaje również ustawiony na aktualny czas systemowy.

Na koniec, funkcja wywołuje funkcję systemową "utime", która przyjmuje jako argumenty ścieżkę do pliku i strukturę "utimbuf" z ustawionymi czasami dostępu i modyfikacji. Po wywołaniu tej funkcji czas modyfikacji pliku zostanie zmieniony na aktualny czas systemowy.

\*/

*void* changeModTime(*char* \**srcFilePath*);

/\*

Funkcja "currentTime" służy do wyświetlania aktualnego czasu systemowego w formacie "rok-miesiąc-dzień godzina:minuty:sekundy".

Funkcja ta korzysta z funkcji systemowej "time", która zwraca aktualny czas systemowy w formacie czasu uniksowego (liczba sekund od 1 stycznia 1970 roku). Następnie korzysta z funkcji "localtime", która konwertuje czas uniksowy na lokalny czas, w zależności od ustawień strefy czasowej w systemie. Wynikiem tej funkcji jest struktura "tm", która zawiera informacje o roku, miesiącu, dacie, godzinie, minutach i sekundach.

Funkcja "strftime" jest używana do formatowania czasu w odpowiedni sposób. W tym przypadku, formatowanie odbywa się za pomocą ciągu znaków "%Y-%m-%d %H:%M:%S", który oznacza kolejno: rok, miesiąc, dzień, godzinę, minutę i sekundę, oddzielone myślnikami i dwukropkami. Wynik formatowania zostaje zapisany w tablicy "datetime".

Na końcu, funkcja wyświetla sformatowany czas za pomocą funkcji "printf", którą przekazuje łańcuch znaków z formatowanym czasem wewnątrz nawiasów kwadratowych, umieszczając go wraz z dodanym przed nim nawiasem kwadratowym wewnątrz jednego z nawiasów okrągłych. Takie wyświetlenie pozwala na oznaczenie czasu jako znacznik czasowy w różnych kontekstach, na przykład w logach lub w konsoli systemowej.

\*/

*void* currenTime();

/\*

Funkcja "clearTheArray" oczyszcza zawartość tablicy znaków przekazanej jako argument do funkcji.

Argument "arr" to wskaźnik na początek tablicy znaków, którą chcemy wyczyścić.

Wewnątrz funkcji używana jest funkcja "strlen", która zwraca długość napisu, który przekazujemy jako argument (tablica znaków kończy się znakiem null, czyli '\0', więc długość napisu to liczba znaków przed tym znakiem).

Następnie wykorzystywana jest funkcja "memset", która ustawia kolejne bajty pamięci na wartość podaną jako drugi argument (w tym przypadku jest to znak null, czyli '\0'). Funkcja "memset" ustawia wartość w pamięci przez określoną liczbę bajtów, która jest wyliczana jako długość napisu pomnożona przez rozmiar jednego elementu tablicy (w tym przypadku rozmiar jednego znaku, czyli 1 bajt).

W rezultacie, po wywołaniu tej funkcji, tablica "arr" będzie zawierała same znaki null, co oznacza, że zostanie ona w pełni wyczyszczona.

\*/

*void* clearTheArray(*char* \**arr*);

/\*

Funkcja "copyDirectory" kopiuje zawartość katalogu źródłowego (srcPath) do katalogu docelowego (dstPath), włączając w to wszystkie pliki i podkatalogi. W przypadku katalogów, kopiowanie jest rekurencyjne, czyli funkcja wywołuje siebie sama dla każdego podkatalogu.

Funkcja używa biblioteki dirent.h do iteracji przez pliki i podkatalogi w katalogu źródłowym. W każdej iteracji, funkcja sprawdza, czy bieżący plik jest katalogiem, czy plikiem, a następnie wykonuje odpowiednie działania.

Jeśli bieżący plik to plik, funkcja porównuje czas modyfikacji pliku źródłowego i docelowego, aby określić, czy plik źródłowy został zmieniony. Jeśli tak, funkcja kopiowania pliku zostaje wywołana, a czas modyfikacji pliku źródłowego jest ustawiany na czas bieżący. W przeciwnym razie funkcja wyświetla odpowiedni komunikat.

Jeśli bieżący plik to katalog, funkcja wywołuje samą siebie dla katalogu źródłowego i docelowego, aby skopiować zawartość katalogu.

Funkcja również obsługuje błędy, takie jak nieudane otwarcie katalogu źródłowego lub nieudane utworzenie katalogu docelowego. Komunikaty o błędach są wyświetlane na ekranie, a program kończy działanie z kodem błędu.

\*/

*void* copyDirectory(const *char* \**srcPath*, const *char* \**dstPath*);

/\*

Funkcja "syncDirectory" synchronizuje zawartość dwóch katalogów - "srcPath" i "dstPath". Funkcja sprawdza, czy każdy plik i katalog w "dstPath" istnieje w "srcPath", a jeśli nie, to usuwa go z "dstPath". Jeśli plik lub katalog istnieje w obu katalogach, funkcja rekurencyjnie wywołuje się na nich, aby synchronizować ich zawartość.

W szczególności funkcja wykonuje następujące czynności:

- Otwiera katalog docelowy ("dstPath") za pomocą funkcji opendir.

- Iteruje po każdym pliku i katalogu w "dstPath" za pomocą readdir.

- Sprawdza, czy dany plik lub katalog to "." lub "..", które są specjalnymi katalogami systemowymi i pomija je.

- Tworzy ścieżki źródłowe i docelowe dla każdego pliku lub katalogu za pomocą snprintf.

- Sprawdza, czy plik lub katalog w "dstPath" jest plikiem regularnym lub katalogiem za pomocą lstat.

- Jeśli plik regularny nie istnieje w "srcPath", usuwa go z "dstPath" za pomocą unlink.

- Jeśli katalog nie istnieje w "srcPath", usuwa go z "dstPath". Jeśli katalog nie jest pusty, usuwa jego zawartość rekurencyjnie za pomocą rekurencyjnego wywołania syncDirectory z pustym "srcPath". Następnie usuwa pusty katalog za pomocą rmdir.

- Jeśli plik lub katalog istnieje w obu katalogach, rekurencyjnie wywołuje syncDirectory dla ścieżek źródłowej i docelowej.

- Zamyka katalog docelowy za pomocą closedir.

W przypadku wystąpienia błędu podczas otwierania, odczytywania lub zamykania katalogu, usuwania pliku lub katalogu lub wywoływania innych funkcji systemowych, funkcja wypisuje odpowiedni komunikat o błędzie i kończy działanie programu za pomocą exit.

\*/

*void* syncDirectory(const *char* \**srcPath*, const *char* \**dstPath*);

/\*

Funkcja "copy" ma trzy argumenty wejściowe: wskaźniki do łańcuchów znaków "source" i "destination", oraz "mmapThreshold", który określa maksymalny rozmiar pliku, dla którego funkcja będzie używać pamięci mapowanej w celu kopiowania.

Wewnątrz funkcji, na początku, jest wywoływana funkcja "stat" z argumentem "source", aby uzyskać informacje o pliku, takie jak jego rozmiar i właściciel. Jeśli "stat" zwróci wartość różną od zera, funkcja wypisze komunikat "Failed on stat" i zakończy działanie.

Następnie, rozmiar pliku jest przypisywany do zmiennej "fileSize", a zmienna "status" jest zainicjowana.

Jeśli rozmiar pliku jest większy niż "mmapThreshold", funkcja wywoła funkcję "copyUsingMMapWrite" z argumentami "source", "destination" i "fileSize", która kopiuje plik z użyciem pamięci mapowanej. W przeciwnym razie, funkcja wywoła funkcję "copyUsingReadWrite" z takimi samymi argumentami, która kopiuje plik przy użyciu operacji odczytu i zapisu.

Na końcu, jeśli zmienna "status" ma wartość inną niż "EXIT\_SUCCESS", funkcja zwróci wartość tej zmiennej. W przeciwnym razie, funkcja zwróci "EXIT\_SUCCESS".

\*/

*int* copy(*char* \**source*, *char* \**destination*, *int* *mmapThreshold*);

/\*

Funkcja "copyUsingReadWrite" służy do kopiowania pliku z użyciem operacji odczytu i zapisu. Ma trzy argumenty wejściowe: łańcuchy znaków "srcPath" i "dstPath", które określają ścieżki plików źródłowego i docelowego odpowiednio, oraz "bufferSize", który określa rozmiar bufora, który będzie używany podczas kopiowania pliku.

Wewnątrz funkcji, na początku, otwierane są pliki źródłowy i docelowy z użyciem funkcji "open". W przypadku niepowodzenia otwarcia pliku źródłowego, funkcja wypisze komunikat "Error opening source file." i zakończy działanie. W przypadku niepowodzenia otwarcia pliku docelowego, funkcja wypisze komunikat "Target file open error." i zakończy działanie.

Następnie, jest tworzony bufor o rozmiarze "bufferSize". W pętli "while", jest wywoływana funkcja "read", aby odczytać dane z pliku źródłowego i zapisać je w buforze. Jeśli odczytanych zostanie 0 bajtów, pętla zostanie przerwana. Następnie, funkcja "write" jest wywoływana, aby zapisać dane z bufora do pliku docelowego. Jeśli zapisane zostanie mniej bajtów niż odczytane, funkcja wypisze komunikat "Error writing to target file." i zakończy działanie.

Na końcu, funkcja sprawdza, czy ilość odczytanych danych jest równa -1. Jeśli tak, oznacza to, że wystąpił błąd podczas odczytu danych z pliku źródłowego, a funkcja wypisze komunikat "Source file read error." i zakończy działanie.

Na końcu, funkcja zamyka pliki źródłowy i docelowy przy użyciu funkcji "close". Jeśli którykolwiek z plików nie zostanie pomyślnie zamknięty, funkcja wypisze komunikat "File close error." i zakończy działanie.\*/

*void* copyUsingReadWrite(const *char* \**srcPath*, const *char* \**dstPath*, *long* *int* *bufferSize*);

/\*

Funkcja "copyUsingMMapWrite" ma na celu skopiowanie zawartości pliku o nazwie "source" do pliku o nazwie "destination" przy użyciu mapowania pamięci.

Funkcja otwiera pliki źródłowy i docelowy przy użyciu funkcji open(), zwracając wartość EXIT\_FAILURE w przypadku niepowodzenia. Następnie funkcja używa funkcji ftruncate() do ustawienia rozmiaru pliku docelowego na rozmiar pliku źródłowego, co oznacza, że ​​docelowy plik będzie miał ten sam rozmiar co źródłowy plik.

Następnie funkcja używa funkcji mmap() w celu mapowania pamięci źródłowej i docelowej plików. Mapowanie pamięci jest techniką, która umożliwia aplikacjom dostęp do plików jak do pamięci wirtualnej, umożliwiając efektywną wymianę danych między plikami a pamięcią systemu.

Funkcja używa memcpy() do skopiowania zawartości pliku źródłowego do pliku docelowego. Następnie funkcja używa munmap() do zwolnienia mapowanych obszarów pamięci i zamyka pliki źródłowy i docelowy przy użyciu funkcji close().

Funkcja zwraca wartość EXIT\_SUCCESS, gdy skopiowanie zostanie wykonane pomyślnie, a wartość EXIT\_FAILURE, gdy wystąpi błąd.

\*/

*int* copyUsingMMapWrite(*char* \**source*, *char* \**destination*, *long* *int* *fileSize*);

/\*

Funkcja "compareDestSrc" ma na celu porównanie plików znajdujących się w dwóch katalogach - źródłowym i docelowym - i usunięcie plików, które znajdują się w katalogu docelowym, ale nie w katalogu źródłowym.

Funkcja rozpoczyna się od otwarcia katalogów źródłowego i docelowego przy użyciu funkcji opendir(). Następnie funkcja sprawdza, czy katalog źródłowy i docelowy zostały poprawnie otwarte. Jeśli któryś z nich nie został poprawnie otwarty, funkcja wyświetli komunikat o błędzie i zwróci sterowanie.

Funkcja następnie przechodzi przez wszystkie wpisy w katalogu docelowym przy użyciu funkcji readdir(). Dla każdego wpisu funkcja sprawdza, czy jest to plik regularny (DT\_REG) - jeśli tak, to funkcja buduje pełną ścieżkę do pliku źródłowego i sprawdza, czy plik istnieje przy użyciu funkcji access(). Jeśli plik nie istnieje, funkcja buduje pełną ścieżkę do pliku docelowego i usuwa go przy użyciu funkcji unlink(). Funkcja wyświetla informację o usunięciu pliku i datę usunięcia za pomocą funkcji currenTime().

Funkcja kończy się przez zamknięcie katalogów źródłowego i docelowego przy użyciu funkcji closedir().

\*/

*void* compareDestSrc(*char* \**sourcePath*, *char* \**destinationPath*);

/\*

Funkcja "compareSrcDest" porównuje zawartość dwóch katalogów o ścieżkach `sourcePath` i `destinationPath`.

Najpierw otwiera oba katalogi przy użyciu funkcji `opendir()`. Następnie sprawdza, czy otwarcie katalogu `sourcePath` zakończyło się sukcesem. Jeśli nie, funkcja wyświetla informację o błędzie i kończy swoje działanie. Jeśli otwarcie katalogu `destinationPath` zakończyło się niepowodzeniem, funkcja wyświetla informację o błędzie i kończy swoje działanie, po uprzednim zamknięciu katalogu `sourceDir`.

Funkcja korzysta z biblioteki `dirent.h` do przechodzenia przez zawartość katalogów. Iteruje po zawartości katalogu `sourcePath`, odczytując po kolei jego elementy. Jeśli aktualny element jest plikiem zwykłym, funkcja sprawdza, czy istnieje plik o tej samej nazwie w katalogu `destinationPath`. Jeśli taki plik istnieje, funkcja wyświetla informację o znalezieniu pliku o tej samej nazwie, a następnie odczytuje czas ostatniej modyfikacji pliku z katalogu `sourcePath` i `destinationPath`. Jeśli czas ostatniej modyfikacji tych plików jest różny, funkcja kopiuję plik ze ścieżki `sourcePath` do ścieżki `destinationPath` przy użyciu funkcji `copy()`. Następnie ustawia czas ostatniej modyfikacji skopiowanego pliku na czas ostatniej modyfikacji pliku z `sourcePath`. Jeśli plik o tej samej nazwie nie istnieje w katalogu `destinationPath`, funkcja wyświetla informację o nie znalezieniu pliku i kopiuje go ze ścieżki `sourcePath` do ścieżki `destinationPath`.

Funkcja `currenTime()` jest używana do wyświetlenia bieżącej daty i czasu, a funkcja `clearTheArray()` jest używana do wyczyszczenia tablicy znaków.

Funkcja zwraca `void`.

\*/

*void* compareSrcDest(*char* \**sourcePath*, *char* \**destinationPath*);

/\*

Funkcja "recursiveSynchronization" przyjmuje dwa argumenty typu char\*, które reprezentują ścieżki do katalogów, które mają zostać zsynchronizowane.

Funkcja wywołuje dwie inne funkcje: "copyDirectory" i "syncDirectory". Pierwsza z nich kopiuje zawartość katalogu źródłowego do katalogu docelowego, natomiast druga synchronizuje zawartość katalogu źródłowego i katalogu docelowego, czyli aktualizuje pliki w katalogu docelowym, które zostały zmienione w katalogu źródłowym od czasu ostatniej synchronizacji.

Cała funkcja działa rekurencyjnie, co oznacza, że jeśli w katalogu źródłowym znajdują się inne katalogi, to również zostaną one zsynchronizowane z odpowiadającymi katalogami w katalogu docelowym.

W ogólnym zarysie, funkcja ta ma na celu zsynchronizowanie dwóch katalogów, aby zawierały one identyczną zawartość, uwzględniając zmiany, które mogły wystąpić w międzyczasie.\*/

*void* recursiveSynchronization(*char* \**srcPath*, *char* \**dstPath*);

/\*

Funkcja "Demon" przyjmuje tablicę dwóch łańcuchów znaków argv, które reprezentują ścieżkę do źródłowego katalogu (argv[1]) i docelowego katalogu (argv[2]).

W funkcji, najpierw sprawdzane jest czy zmienna "recursive" jest ustawiona na wartość prawda (true). Jeśli tak, wywoływana jest funkcja recursiveSynchronization(), która synchronizuje zawartość źródłowego katalogu z docelowym katalogiem, poprzez skopiowanie plików i katalogów z jednego do drugiego i zaktualizowanie ich zawartości.

Jeśli zmienna "recursive" nie jest ustawiona na wartość prawda, wywoływana jest funkcja compareSrcDest(), która porównuje zawartość źródłowego katalogu z zawartością docelowego katalogu i wyświetla informacje o ewentualnych różnicach między nimi.

W skrócie, funkcja Demon jest punktem wejścia do programu i decyduje, której z dwóch funkcji (recursiveSynchronization() lub compareSrcDest()) należy użyć w zależności od wartości zmiennej "recursive".

\*/

*void* Demon(*char* \*\**argv*);

/\*

Funkcja "options" ma na celu analizę argumentów wejściowych przekazanych do programu i ustalenie wartości zmiennych globalnych, które będą wykorzystywane w innych funkcjach. Funkcja przyjmuje dwa argumenty: "argc" - ilość argumentów przekazanych do programu oraz "argv" - tablica napisów, która przechowuje argumenty.

Funkcja rozpoczyna pętlę "for" od indeksu 3, ponieważ argumenty przekazane do programu o indeksach 0, 1 i 2 zawierają informacje o nazwie programu, źródłowym i docelowym katalogu.

Wewnątrz pętli sprawdzane są kolejne argumenty za pomocą funkcji "strcmp", która porównuje napisy. Jeśli argument odpowiada jednemu z trzech flag: "-r", "-t" lub "-d", to zostaje przypisana odpowiednia wartość do zmiennych globalnych "recursive", "timeSleep" lub "mmapThreshold".

W przypadku flagi "-r", zmienna "recursive" jest ustawiana na wartość "true", co oznacza, że synchronizacja będzie odbywać się rekurencyjnie.

W przypadku flagi "-t", wartość po niej oznacza liczbę sekund, po której program będzie próbował ponownie wybudzic sie ze snu. Wartość ta zostanie przypisana do zmiennej "timeSleep".

W przypadku flagi "-d", wartość po niej oznacza maksymalny rozmiar pliku, który będzie mapowany w pamięci przy użyciu funkcji "mmap". Wartość ta zostanie przypisana do zmiennej "mmapThreshold".

Funkcja nie zwraca żadnej wartości, a jedynie ustawia wartości zmiennych globalnych.

\*/

*void* options(*int* *argc*, *char* \*\**argv*);

/\*

Funkcja "sigusr1\_handler" obslubuje sygnał SIGUSR1, czyli sygnału użytkownika nr 1. Kiedy proces otrzymuje ten sygnał, system wywołuje tę funkcję, która wypisuje bieżący czas oraz komunikat "Demon awakening by signal SIGUSR1." Następnie zmienna forcedSynchro jest ustawiana na wartość true, co oznacza, że proces zostanie wymuszony do natychmiastowej synchronizacji w czasie kolejnej iteracji.

\*/

*void* sigusr1\_handler(*int* *signum*);

/\*

Funkcja "createDemon" tworzy proces demona w systemie operacyjnym. Kod jest napisany w języku C.

Funkcja najpierw otwiera systemowy dziennik zdarzeń za pomocą funkcji "openlog", z ustawieniami, które powodują wyświetlenie identyfikatora procesu (PID) i identyfikatora użytkownika (UID) w logach.

Następnie funkcja wywołuje funkcję "fork", aby utworzyć nowy proces. Proces potomny jest utworzony i kod kontynuuje jego wykonanie, podczas gdy proces rodzicielski kończy swoje działanie za pomocą funkcji "exit". Proces potomny będzie działał jako demon.

Proces demon wywołuje funkcję "umask", aby ustawić maskę uprawnień plików na 0, co oznacza, że ​​tworzone pliki będą miały pełne uprawnienia.

Następnie demon wywołuje funkcję "setsid", aby utworzyć nową sesję. Funkcja ta powoduje, że proces staje się liderem nowej sesji, procesu grupowego i zrywa związek z terminalami kontrolnymi, co oznacza, że ​​demon nie jest już zależny od terminala.

Funkcja "currentTime" wywołuje inną funkcję, która zwraca aktualny czas, który jest wyświetlany na konsoli i zapisywany w dzienniku systemowym.

Następnie demon wyświetla swoje PID na konsoli i zapisuje go w dzienniku systemowym za pomocą funkcji "syslog".

Na koniec demon zamyka standardowe wejście, standardowe wyjście i standardowe wyjście błędów, aby zapobiec niechcianym wyjściom, a następnie zamyka dziennik zdarzeń za pomocą funkcji "closelog".

\*/

*void* createDemon();

/\*

Funkcja `main` jest punktem wejścia programu. Przyjmuje dwa argumenty, `argc`, który jest liczbą argumentów wiersza poleceń przekazanych do programu i `argv`, który jest tablicą łańcuchów zawierających argumenty wiersza poleceń.

Funkcja najpierw sprawdza, czy liczba argumentów jest mniejsza niż trzy, czyli minimalna liczba argumentów wymagana do poprawnego działania programu. Jeśli argumentów jest mniej, na konsoli wyświetlany jest komunikat o błędzie. W przeciwnym razie program kontynuuje ustawianie opcji i obsługę sygnału.

Funkcja `options` jest wywoływana w celu przeanalizowania argumentów wiersza poleceń i ustawienia odpowiednich opcji na podstawie ich wartości. Funkcja `signal` jest wywoływana w celu zarejestrowania procedury obsługi sygnału dla sygnału `SIGUSR1`, który jest używany do wymuszenia synchronizacji między katalogami źródłowym i docelowym.

Następnie funkcja drukuje bieżący czas i identyfikator procesu demona.

Rozpoczyna się pętla, która działa w nieskończoność. Jeśli `forcedSynchro` ma wartość false, funkcja drukuje komunikat wskazujący, że demon się obudził. Następnie wywoływana jest funkcja `Demon` w celu wykonania synchronizacji między katalogami źródłowym i docelowym.

Po zakończeniu synchronizacji funkcja wyświetla komunikat wskazujący, że demon śpi i czeka przez liczbę sekund określoną przez `timeSleep` przed rozpoczęciem kolejnej iteracji pętli. Jeśli flaga `forcedSynchro` jest prawdziwa, demon pomija krok synchronizacji i wraca do snu.

\*/

*int* main(*int* *argc*, *char* \*\**argv*);

***„demon.c”***

// IMPORT NAGŁÓWKÓW FUNKCJI

#include "functions.h"

*void* changeModTime(*char* \**srcFilePath*)

{

// Przypisz ścieżkę pliku do zmiennej file\_path.

*char* \*file\_path = *srcFilePath*;

// Pobierz aktualny czas i przypisz go do zmiennej now.

*time\_t* now = time(NULL);

// Utwórz strukturę utimbuf i przypisz do jej pól czas dostępu i modyfikacji.

*struct* *utimbuf* new\_times;

new\_times.actime = now; // czas dostępu - zostawiamy bieżący

new\_times.modtime = now; // czas modyfikacji - ustawiamy na bieżący

// Wywołaj funkcję utime z parametrami ścieżki pliku i nowych czasów.

utime(file\_path, &new\_times);

}

*void* currenTime()

{

// otwarcie systemowego logu z tagiem "Demon", numerem PID procesu oraz identyfikatorem użytkownika LOG\_USER

openlog("Demon", LOG\_PID, LOG\_USER);

// pobranie aktualnego czasu systemowego w sekundach

*time\_t* current\_time = time(NULL);

// przetworzenie czasu do lokalnej strefy czasowej

*struct* *tm* \*local\_time = localtime(&current\_time);

// utworzenie łańcucha znaków datetime zawierającego datę i godzinę w formacie YYYY-MM-DD HH:MM:SS

*char* datetime[21];

strftime(datetime, 21, "%Y-%m-%d %H:%M:%S", local\_time);

// zapisanie logu do systemowego logu z priorytetem LOG\_INFO, zawierającego datę i godzinę w takim samym formacie jak na wyjściu

// syslog(LOG\_INFO, "[%s] ", datetime);

// wyświetlenie daty i godziny w takim samym formacie jak na wyjściu na standardowym wyjściu

printf("[%s] ", datetime);

// zamknięcie systemowego logu

closelog();

}

*void* clearTheArray(*char* \**arr*)

{

// obliczenie długości łańcucha znaków przy pomocy funkcji strlen()

*size\_t* length = strlen(*arr*);

// wyczyszczenie całego łańcucha znaków przy pomocy funkcji memset(), która wypełnia pierwsze length bajtów wskaźnika arr wartością '\0' (null terminator)

memset(*arr*, '\0', length);

}

*void* copyDirectory(const *char* \**srcPath*, const *char* \**dstPath*)

{

// Uruchomienie systemowego dziennika zdarzeń

openlog("Demon", LOG\_PID, LOG\_USER);

// Otwarcie katalogu źródłowego

*DIR* \*srcDirectory = opendir(*srcPath*);

// Sprawdzenie czy udało się otworzyć katalog źródłowy

if (srcDirectory == NULL)

{

// Funkcja wyświetlająca aktualny czas

currenTime();

printf("Błąd podczas otwierania katalogu źródłowego.");

// Dodanie wpisu do dziennika zdarzeń

syslog(LOG\_ERR, "Błąd podczas otwierania katalogu źródłowego.");

// Zakończenie programu z kodem błędu

exit(EXIT\_FAILURE);

}

if (mkdir(*dstPath*, S\_IRWXU | S\_IRGRP | S\_IXGRP | S\_IROTH | S\_IXOTH) == -1 && errno != EEXIST)

{ // Sprawdzenie czy udało się utworzyć katalog docelowy

currenTime();

printf("Błąd podczas tworzenia katalogu docelowego.");

syslog(LOG\_ERR, "Błąd podczas tworzenia katalogu docelowego.");

exit(EXIT\_FAILURE);

}

// Struktura przechowująca informacje o plikach/katalogach w katalogu źródłowym

*struct* *dirent* \*srcFileInfo;

// Pętla przechodząca przez wszystkie pliki/katalogi w katalogu źródłowym

while ((srcFileInfo = readdir(srcDirectory)) != NULL)

{

// Ignorowanie katalogów "." i ".."

if (strcmp(srcFileInfo->d\_name, ".") == 0 || strcmp(srcFileInfo->d\_name, "..") == 0)

{

continue;

}

// Zmienna przechowująca pełną ścieżkę do pliku/katalogu w katalogu źródłowym

*char* srcFilePath[PATH\_MAX];

// Zmienna przechowująca pełną ścieżkę do pliku/katalogu w katalogu docelowym

*char* dstFilePath[PATH\_MAX];

// Nazwa pliku/katalogu

*char* \*src = srcFileInfo->d\_name;

// Utworzenie pełnej ścieżki do pliku/katalogu źródłowego

snprintf(srcFilePath, sizeof(srcFilePath), "%s/%s", *srcPath*, srcFileInfo->d\_name);

// Utworzenie pełnej ścieżki do pliku/katalogu docelowego

snprintf(dstFilePath, sizeof(dstFilePath), "%s/%s", *dstPath*, srcFileInfo->d\_name);

// Struktura przechowująca informacje o pliku/katalogu źródłowym

*struct* *stat* srcFileInfo;

// Struktura przechowująca informacje o pliku/katalogu docelowym

*struct* *stat* dstFileInfo;

// Zmienna przechowująca czas modyfikacji pliku/katalogu źródłowego

*char* modTimeSrc[20];

// Zmienna przechowująca czas modyfikacji pliku/katalogu docelowego

*char* modTimeDst[20];

// Sprawdzenie statystyk pliku lub katalogu źródłowego i zapisanie ich w strukturze srcFileInfo

if (lstat(srcFilePath, &srcFileInfo) == -1)

{

// Wyświetlenie komunikatu o błędzie na standardowym wyjściu oraz w logu systemowym

currenTime();

printf("Błąd odczytu statystyk pliku/katalogu źródłowego.");

syslog(LOG\_ERR, "Błąd odczytu statystyk pliku/katalogu źródłowego.");

// Wyświetlenie ścieżki do pliku/katalogu źródłowego w logu systemowym

syslog(LOG\_ERR, "%s\n", srcFilePath);

// Wyświetlenie ścieżki do pliku/katalogu docelowego w logu systemowym

syslog(LOG\_ERR, "%s\n", dstFilePath);

// Wyjście z programu z kodem błędu

exit(EXIT\_FAILURE);

}

// Przekształcenie czasu modyfikacji pliku/katalogu źródłowego na format tekstowy

strftime(modTimeSrc, sizeof(modTimeSrc), "%Y-%m-%d %H:%M:%S", localtime(&srcFileInfo.st\_mtime));

// Sprawdzamy czy istnieje plik docelowy

if (lstat(dstFilePath, &dstFileInfo) == -1)

{

if (S\_ISREG(srcFileInfo.st\_mode))

{ // plik

// Poniższy fragment kodu porównuje czas modyfikacji pliku źródłowego i docelowego

// Jeśli są one różne, to następuje kopiowanie pliku, zmiana czasu modyfikacji, logowanie informacji o zdarzeniu oraz wypisanie na ekran informacji o zdarzeniu

// Jeśli czas modyfikacji jest taki sam, to wypisywana jest informacja o tym, że znaleziono plik o takiej samej nazwie

if (strcmp(modTimeSrc, modTimeDst) != 0)

{

copy(srcFilePath, dstFilePath, mmapThreshold);

changeModTime(srcFilePath);

currenTime();

syslog(LOG\_INFO, "Różne czasy modyfikacji: %s\n", src);

printf("Różne czasy modyfikacji: %s\n", src);

currenTime();

syslog(LOG\_INFO, "Plik: %s został pomyślnie skopiowany.\n", src);

printf("Plik: %s został pomyślnie skopiowany.\n", src);

}

else

{

currenTime();

syslog(LOG\_INFO, "Znaleziono plik o takiej samej nazwie: %s\n", src);

printf("Znaleziono plik o takiej samej nazwie: %s\n", src);

}

} // Jeśli plik jest katalogiem, to wywoływana jest funkcja kopiująca katalog, logowane jest zdarzenie oraz wypisywana informacja na ekranie

else if (S\_ISDIR(srcFileInfo.st\_mode))

{

copyDirectory(srcFilePath, dstFilePath);

currenTime();

syslog(LOG\_INFO, "Znaleziono katalog: %s\n", src);

printf("Znaleziono katalog: %s\n", src);

}

}

else

{

// Formatowanie czasu modyfikacji pliku docelowego

strftime(modTimeDst, sizeof(modTimeDst), "%Y-%m-%d %H:%M:%S", localtime(&dstFileInfo.st\_mtime));

if (strcmp(modTimeSrc, modTimeDst) != 0)

{

if (S\_ISREG(srcFileInfo.st\_mode))

{ // plik

if (strcmp(modTimeSrc, modTimeDst) != 0)

{

copy(srcFilePath, dstFilePath, mmapThreshold);

changeModTime(srcFilePath);

currenTime();

syslog(LOG\_INFO, "Różne czasy modyfikacji: %s\n", src);

printf("Różne czasy modyfikacji: %s\n", src);

currenTime();

syslog(LOG\_INFO, "Plik: %s został pomyślnie skopiowany.\n", src);

printf("Plik: %s został pomyślnie skopiowany.\n", src);

}

else

{

currenTime();

printf("Znaleziono plik o takiej samej nazwie: %s\n", src);

syslog(LOG\_INFO, "Znaleziono plik o takiej samej nazwie: %s\n", src);

}

}

else if (S\_ISDIR(srcFileInfo.st\_mode))

{ // katalog

copyDirectory(srcFilePath, dstFilePath);

currenTime();

printf("Znaleziono katalog: %s\n", src);

syslog(LOG\_INFO, "Znaleziono katalog: %s\n", src);

}

}

}

}

// Jeśli zamknięcie katalogu źródłowego się nie powiedzie:

if (closedir(srcDirectory) == -1)

{

// Wywołaj funkcję currenTime() - pobierająca aktualny czas.

currenTime();

// Wyświetl komunikat błędu w konsoli.

printf("Błąd podczas zamykania katalogu źródłowego.");

// Wywołaj funkcję syslog() - logowanie błędów systemowych.

syslog(LOG\_ERR, "Błąd podczas zamykania katalogu źródłowego.");

// Zakończ program z kodem błędu.

exit(EXIT\_FAILURE);

}

closelog();

}

*void* syncDirectory(const *char* \**srcPath*, const *char* \**dstPath*)

{

// Otwieranie logu systemowego z nazwą "Demon", z PID-em oraz użytkownikiem

openlog("Demon", LOG\_PID, LOG\_USER);

// Otwieranie katalogu docelowego za pomocą funkcji opendir()

// Jeśli otwarcie katalogu się nie powiedzie, program zakończy działanie z kodem błędu

*DIR* \*dstDirectory = opendir(*dstPath*);

if (dstDirectory == NULL)

{

// Wywołanie funkcji currenTime() do wypisania aktualnego czasu

currenTime();

printf("Błąd podczas otwierania katalogu docelowego.");

// Zapisanie błędu do logu systemowego za pomocą funkcji syslog()

syslog(LOG\_ERR, "Błąd podczas otwierania katalogu docelowego.");

exit(EXIT\_FAILURE); // Zakończenie działania programu z kodem błędu

}

// Struktura przechowująca informacje o pliku/katalogu w katalogu docelowym

*struct* *dirent* \*FileOrDirectory;

// Pętla while, która będzie wykonywać operacje dopóki kolejny element katalogu docelowego istnieje

// readdir() zwróci NULL, kiedy wszystkie elementy zostaną przeczytane

while ((FileOrDirectory = readdir(dstDirectory)) != NULL)

{

// Sprawdzanie, czy element katalogu nie jest "." ani ".."

// Jeśli tak, to pomiń ten element i przejdź do kolejnego

if (strcmp(FileOrDirectory->d\_name, ".") == 0 || strcmp(FileOrDirectory->d\_name, "..") == 0)

{

continue;

}

// Tworzenie zmiennych przechowujących ścieżki plików/katalogów źródłowych i docelowych

*char* srcFilePath[PATH\_MAX];

*char* dstFilePath[PATH\_MAX];

// Używanie funkcji snprintf() do sklejenia ścieżki źródłowej i nazwy pliku/katalogu

snprintf(srcFilePath, sizeof(srcFilePath), "%s/%s", *srcPath*, FileOrDirectory->d\_name);

// Używanie funkcji snprintf() do sklejenia ścieżki docelowej i nazwy pliku/katalogu

snprintf(dstFilePath, sizeof(dstFilePath), "%s/%s", *dstPath*, FileOrDirectory->d\_name);

// Struktura przechowująca informacje o pliku/katalogu

*struct* *stat* FileOrDirectoryInfo;

// Pobieranie informacji o pliku/katalogu docelowym przy pomocy funkcji lstat()

// Jeśli funkcja zwróci -1, to znaczy, że wystąpił błąd i program zakończy działanie z kodem błędu

if (lstat(dstFilePath, &FileOrDirectoryInfo) == -1)

{

// Wywołanie funkcji currenTime() do wypisania aktualnego czasu

currenTime();

printf("Błąd podczas odczytywania statystyk pliku/katalogu docelowego.");

// Zapisanie błędu do logu systemowego za pomocą funkcji syslog()

syslog(LOG\_ERR, "Błąd podczas odczytywania statystyk pliku/katalogu docelowego.");

exit(EXIT\_FAILURE); // Zakończenie działania programu z kodem błędu

}

// Sprawdzanie czy plik lub katalog istnieje w katalogu źródłowym i usuwanie, jeśli tak

if (S\_ISREG(FileOrDirectoryInfo.st\_mode))

{

// Jeśli to plik regularny, sprawdź czy istnieje w katalogu źródłowym

if (access(srcFilePath, F\_OK) == -1)

{

// Usuwanie pliku

currenTime();

printf("Plik: %s został pomyślnie usunięty.\n", FileOrDirectory->d\_name);

syslog(LOG\_INFO, "Plik: %s został pomyślnie usunięty.\n", FileOrDirectory->d\_name);

if (unlink(dstFilePath) == -1)

{

currenTime();

printf("Błąd usuwania pliku docelowego.");

syslog(LOG\_ERR, "Błąd usuwania pliku docelowego.");

exit(EXIT\_FAILURE);

}

}

}

else if (S\_ISDIR(FileOrDirectoryInfo.st\_mode))

{

if (access(srcFilePath, F\_OK) == -1)

{

// Usuwanie katalogu

currenTime();

printf("Katalog: %s został pomyślnie usunięty.\n", FileOrDirectory->d\_name);

syslog(LOG\_INFO, "Katalog: %s został pomyślnie usunięty.\n", FileOrDirectory->d\_name);

if (rmdir(dstFilePath) == -1)

{

if (errno == ENOTEMPTY)

{

// Synchronizacja zawartości katalogu i usuwanie

syncDirectory("", dstFilePath);

if (rmdir(dstFilePath) == -1)

{

currenTime();

printf("Błąd usuwania katalogu docelowego.");

syslog(LOG\_ERR, "Błąd usuwania katalogu docelowego.");

exit(EXIT\_FAILURE);

}

}

else

{

currenTime();

printf("Błąd usuwania katalogu docelowego.");

syslog(LOG\_ERR, "Błąd usuwania katalogu docelowego.");

exit(EXIT\_FAILURE);

}

}

}

else

{

// Synchronizacja zawartości katalogu

syncDirectory(srcFilePath, dstFilePath);

}

}

}

// Zamykanie katalogu docelowego przy pomocy funkcji closedir()

// Jeśli zamknięcie katalogu się nie powiedzie, program zakończy działanie z kodem błędu

if (closedir(dstDirectory) == -1)

{

// Wywołanie funkcji currenTime() do wypisania aktualnego czasu

currenTime();

printf("Błąd podczas zamykania katalogu docelowego.");

// Zapisanie błędu do logu systemowego za pomocą funkcji syslog()

syslog(LOG\_ERR, "Błąd podczas zamykania katalogu docelowego.");

exit(EXIT\_FAILURE); // Zakończenie działania programu z kodem błędu

}

closelog();

}

*int* copy(*char* \**source*, *char* \**destination*, *int* *mmapThreshold*)

{

// Sprawdzenie informacji o pliku źródłowym

*struct* *stat* fileStat;

if (stat(*source*, &fileStat) == -1)

{

printf("Błąd podczas wykonywania funkcji stat.");

}

*long* *int* fileSize = fileStat.st\_size;

*int* status;

// Sprawdzenie wielkości pliku i wybór metody kopiowania

if (fileSize > *mmapThreshold*)

{

status = copyUsingMMapWrite(*source*, *destination*, fileSize);

}

else

{

copyUsingReadWrite(*source*, *destination*, fileSize);

}

// Obsługa błędów kopiowania

if (status != EXIT\_SUCCESS)

{

return status;

}

return EXIT\_SUCCESS;

}

*void* copyUsingReadWrite(const *char* \**srcPath*, const *char* \**dstPath*, *long* *int* *bufferSize*)

{

// Inicjalizacja demonowego logowania z identyfikatorem PID i źródłem logowania użytkownika

openlog("Demon", LOG\_PID, LOG\_USER);

// Otwarcie pliku źródłowego w trybie tylko do odczytu

*int* srcFile = open(*srcPath*, O\_RDONLY);

if (srcFile == -1)

{

// Obsługa błędu w przypadku niepowodzenia otwarcia pliku źródłowego

currenTime();

printf("Błąd podczas otwierania pliku źródłowego.");

syslog(LOG\_ERR, "Błąd podczas otwierania pliku źródłowego.");

exit(EXIT\_FAILURE);

}

// Otwarcie pliku docelowego w trybie tylko do zapisu, tworząc go jeśli nie istnieje

*int* dstFile = open(*dstPath*, O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH);

if (dstFile == -1)

{

// Obsługa błędu w przypadku niepowodzenia otwarcia pliku docelowego

currenTime();

printf("Błąd otwarcia pliku docelowego.");

syslog(LOG\_ERR, "Błąd otwarcia pliku docelowego.");

exit(EXIT\_FAILURE);

}

// Bufor dla danych z pliku źródłowego i zmienne do przechowywania liczby odczytanych i zapisanych bajtów

*char* buf[*bufferSize*];

*ssize\_t* bytesRead, bytesWritten;

// Pętla odczytuje dane ze źródłowego pliku i zapisuje je do docelowego pliku

while ((bytesRead = read(srcFile, buf, sizeof(buf))) > 0)

{

bytesWritten = write(dstFile, buf, bytesRead);

if (bytesWritten == -1)

{

currenTime();

printf("Błąd zapisu do pliku docelowego.");

syslog(LOG\_ERR, "Błąd zapisu do pliku docelowego.");

exit(EXIT\_FAILURE);

}

}

// Jeśli odczytanie danych ze źródłowego pliku zakończy się niepowodzeniem, to wyświetlamy odpowiedni komunikat błędu i kończymy działanie programu

if (bytesRead == -1)

{

currenTime();

printf("Błąd odczytu pliku źródłowego.");

syslog(LOG\_ERR, "Błąd odczytu pliku źródłowego.");

exit(EXIT\_FAILURE);

}

// Zamykamy otwarte pliki źródłowy i docelowy; jeśli zamknięcie plików zakończy się niepowodzeniem, to wyświetlamy odpowiedni komunikat błędu i kończymy działanie programu

if (close(srcFile) == -1 || close(dstFile) == -1)

{

currenTime();

printf("Błąd zamykania pliku.");

syslog(LOG\_ERR, "Błąd zamykania pliku.");

exit(EXIT\_FAILURE);

}

closelog();

}

*int* copyUsingMMapWrite(*char* \**source*, *char* \**destination*, *long* *int* *fileSize*)

{

// Otwieranie logu systemowego z nazwą demona, dodanie identyfikatora PID i informacja, że logi będą zapisywane w imieniu użytkownika

openlog("Demon", LOG\_PID, LOG\_USER);

*int* fd\_src, fd\_dst;

*void* \*src\_data;

*void* \*dst\_data;

// Otwarcie pliku źródłowego tylko do odczytu

if ((fd\_src = open(*source*, O\_RDONLY)) == -1)

{

currenTime();

printf("Nie udało się otworzyć pliku źródłowego");

syslog(LOG\_ERR, "Nie udało się otworzyć pliku źródłowego");

return EXIT\_FAILURE;

}

// Otwarcie pliku docelowego z możliwością odczytu i zapisu, oraz utworzenie go jeśli nie istnieje

if ((fd\_dst = open(*destination*, O\_RDWR | O\_CREAT, 0644)) == -1)

{

currenTime();

printf("Nie udało się otworzyć pliku docelowego");

syslog(LOG\_ERR, "Nie udało się otworzyć pliku docelowego");

close(fd\_src);

return EXIT\_FAILURE;

}

// Ustawienie rozmiaru pliku docelowego na podany w argumencie

if (ftruncate(fd\_dst, *fileSize*) == -1)

{

close(fd\_src);

close(fd\_dst);

return EXIT\_FAILURE;

}

// Otwarcie pliku zrodlowego do odczytu

src\_data = mmap(NULL, *fileSize*, PROT\_READ, MAP\_SHARED, fd\_src, 0);

// Sprawdzenie czy udalo sie zaalokowac pamiec i obslugiwanie bledu w przypadku niepowodzenia

if (src\_data == MAP\_FAILED)

{

// Wypisanie bledu do konsoli i logu systemowego

currenTime();

printf("Nie udalo sie zmapowac pliku zrodlowego");

syslog(LOG\_ERR, "Nie udalo sie zmapowac pliku zrodlowego");

// Zamkniecie otwartych plikow i zwrocenie wartosci niepowodzenia

close(fd\_src);

close(fd\_dst);

return EXIT\_FAILURE;

}

// Otwarcie pliku docelowego do zapisu

dst\_data = mmap(NULL, *fileSize*, PROT\_WRITE, MAP\_SHARED, fd\_dst, 0);

// Sprawdzenie czy udalo sie zaalokowac pamiec i obslugiwanie bledu w przypadku niepowodzenia

if (dst\_data == MAP\_FAILED)

{

// Wypisanie bledu do konsoli i logu systemowego

currenTime();

printf("Nie udalo sie zmapowac pliku docelowego");

syslog(LOG\_ERR, "Nie udalo sie zmapowac pliku docelowego");

// Zwolnienie zaalokowanej pamieci, zamkniecie otwartych plikow i zwrocenie wartosci niepowodzenia

munmap(src\_data, *fileSize*);

close(fd\_src);

close(fd\_dst);

return EXIT\_FAILURE;

}

// Skopiowanie danych z pliku zrodlowego do pliku docelowego przy pomocy funkcji write()

*size\_t* remaining = *fileSize*;

*char* \*src = src\_data;

*char* \*dst = dst\_data;

while (remaining > 0)

{

*ssize\_t* written = write(fd\_dst, src, remaining);

if (written == -1)

{

// Wypisanie błędu do konsoli i logu systemowego

currenTime();

printf("Nie udało się zapisać danych do pliku docelowego");

syslog(LOG\_ERR, "Nie udało się zapisać danych do pliku docelowego");

// Zwolnienie zaalokowanej pamięci, zamknięcie otwartych plików i zwrot wartości niepowodzenia

munmap(src\_data, *fileSize*);

munmap(dst\_data, *fileSize*);

close(fd\_src);

close(fd\_dst);

closelog();

return EXIT\_FAILURE;

}

remaining -= written;

src += written;

}

// Zwolnienie zaalokowanej pamieci

munmap(src\_data, *fileSize*);

munmap(dst\_data, *fileSize*);

// Zamkniecie otwartych plikow i logu systemowego

close(fd\_src);

close(fd\_dst);

closelog();

// Zwrocenie wartosci powodzenia

return EXIT\_SUCCESS;

}

*void* compareDestSrc(*char* \**sourcePath*, *char* \**destinationPath*)

{

// Inicjalizacja logu systemowego

openlog("Demon", LOG\_PID, LOG\_USER);

// Otwarcie katalogu zrodlowego i katalogu docelowego

*DIR* \*sourceDir = opendir(*sourcePath*);

*DIR* \*destinationDir = opendir(*destinationPath*);

// Sprawdzenie czy udalo sie otworzyc katalog zrodlowy i obslugiwanie bledu w przypadku niepowodzenia

if (sourceDir == NULL)

{

// Wypisanie bledu do konsoli i logu systemowego

currenTime();

printf("Wystapil problem przy otwieraniu katalogu zrodlowego\n");

syslog(LOG\_ERR, "Wystapil problem przy otwieraniu katalogu zrodlowego\n");

return;

}

// Sprawdzenie czy udalo sie otworzyc katalog docelowy i obslugiwanie bledu w przypadku niepowodzenia

if (destinationDir == NULL)

{

// Wypisanie bledu do konsoli i logu systemowego

currenTime();

printf("Wystapil problem przy otwieraniu katalogu docelowego\n");

syslog(LOG\_ERR, "Wystapil problem przy otwieraniu katalogu docelowego\n");

// Zamkniecie katalogu zrodlowego i zwrocenie funkcji

closedir(sourceDir);

return;

}

// Inicjalizacja struktury przechowujacej wpis w katalogu docelowym

*struct* *dirent* \*destinationEntry;

// Inicjalizacja tablicy przechowujacej pelna sciezke wpisu w katalogu

*char* entryPath[PATH\_MAX];

// Pobieranie kolejnych wpisow z katalogu docelowego i przetwarzanie ich w petli

while ((destinationEntry = readdir(destinationDir)) != NULL)

{

// Sprawdzenie czy aktualny wpis jest plikiem regularnym

if ((destinationEntry->d\_type) == DT\_REG)

{

// Utworzenie pelnej sciezki do pliku z katalogu zrodlowego

if (snprintf(entryPath, PATH\_MAX, "%s/%s", *sourcePath*, destinationEntry->d\_name) >= PATH\_MAX)

{

// Wypisanie bledu do konsoli i logu systemowego i przerwanie dzialania programu

currenTime();

printf("Wystapil problem przy pobieraniu pelnej sciezki pliku zrodlowego. Plik nie zostal odnaleziony\n");

syslog(LOG\_ERR, "Wystapil problem przy pobieraniu pelnej sciezki pliku zrodlowego. Plik nie zostal odnaleziony\n");

exit(EXIT\_FAILURE);

}

// Sprawdzenie czy plik zrodlowy istnieje

if (access(entryPath, F\_OK) != 0)

{

// Wyczyszczenie tablicy zawierajacej sciezke do pliku

clearTheArray(entryPath);

// Utworzenie pelnej sciezki do pliku z katalogu docelowego

if (snprintf(entryPath, PATH\_MAX, "%s/%s", *destinationPath*, destinationEntry->d\_name) >= PATH\_MAX)

{

// Wypisanie bledu do konsoli i logu systemowego i przerwanie dzialania programu

currenTime();

printf("Wystapil problem przy pobieraniu pelnej sciezki pliku docelowego. Plik nie zostal odnaleziony\n");

syslog(LOG\_ERR, "Wystapil problem przy pobieraniu pelnej sciezki pliku docelowego. Plik nie zostal odnaleziony\n");

exit(EXIT\_FAILURE);

}

// Usuniecie pliku z katalogu docelowego

if (unlink(entryPath) == 0)

{

// Wypisanie informacji o usunietym pliku do logu systemowego i konsoli

currenTime();

syslog(LOG\_INFO, "Plik: %s zostal pomyslnie usuniety.\n", destinationEntry->d\_name);

printf("Plik: %s zostal pomyslnie usuniety.\n", destinationEntry->d\_name);

}

}

}

// Wyczyszczenie tablicy zawierajacej sciezke do pliku przed przetworzeniem kolejnego wpisu z katalogu

clearTheArray(entryPath);

}

// Zamkniecie katalogow zrodlowego i docelowego oraz logu systemowego

closedir(sourceDir);

closedir(destinationDir);

closelog();

}

*void* compareSrcDest(*char* \**sourcePath*, *char* \**destinationPath*)

{

// Inicjalizacja logowania

openlog("Demon", LOG\_PID, LOG\_USER);

// Otwarcie katalogów źródłowego i docelowego

*DIR* \*sourceDir = opendir(*sourcePath*);

*DIR* \*destinationDir = opendir(*destinationPath*);

// Sprawdzenie czy otwarcie katalogu źródłowego powiodło się

if (sourceDir == NULL)

{

currenTime();

printf("Wystąpił problem podczas próby otwarcia katalogu źródłowego\n");

syslog(LOG\_ERR, "Wystąpił problem podczas próby otwarcia katalogu źródłowego\n");

return;

}

// Sprawdzenie czy otwarcie katalogu docelowego powiodło się

if (destinationDir == NULL)

{

currenTime();

printf("Wystąpił problem podczas próby otwarcia katalogu docelowego\n");

syslog(LOG\_ERR, "Wystąpił problem podczas próby otwarcia katalogu docelowego\n");

closedir(sourceDir);

return;

}

// Struktury zawierające informacje o plikach w katalogach źródłowym i docelowym

*struct* *stat* srcFileInfo;

*struct* *stat* destFileInfo;

// Kontenery na ścieżki do plików w katalogach źródłowym i docelowym

*char* srcFilePathContainer[PATH\_MAX];

*char* destFilePathContainer[PATH\_MAX];

// Wskaźniki na wpisy katalogowe w katalogach źródłowym i docelowym

*struct* *dirent* \*sourceEntry;

*struct* *dirent* \*destinationEntry;

// Kontener na pełną ścieżkę do pliku

*char* entryPath[PATH\_MAX];

// Kontenery na daty modyfikacji plików

*char* modTimeSrc[20];

*char* modTimeDest[20];

// Struktura z czasami modyfikacji pliku źródłowego

*struct* *utimbuf* srcTimes;

// Otwieramy katalog źródłowy i katalog docelowy

while ((sourceEntry = readdir(sourceDir)) != NULL)

{

// Sprawdzamy, czy plik jest zwykłym plikiem (nie katalogiem)

if ((sourceEntry->d\_type) == DT\_REG)

{

// Tworzymy ścieżkę do pliku w katalogu docelowym

if (snprintf(entryPath, PATH\_MAX, "%s/%s", *destinationPath*, sourceEntry->d\_name) >= PATH\_MAX)

{

currenTime();

printf("Problem wystąpił podczas próby pobrania pełnej ścieżki do pliku docelowego\n");

syslog(LOG\_ERR, "Problem wystąpił podczas próby pobrania pełnej ścieżki do pliku docelowego\n");

exit(EXIT\_FAILURE);

}

// Sprawdzamy, czy plik o takiej samej nazwie istnieje już w katalogu docelowym

if (access(entryPath, F\_OK) == 0)

{

currenTime();

syslog(LOG\_INFO, "Plik o takiej samej nazwie został znaleziony: %s\n", sourceEntry->d\_name);

printf("Plik o takiej samej nazwie został znaleziony: %s\n", sourceEntry->d\_name);

clearTheArray(entryPath);

// Tworzymy ścieżkę do pliku w katalogu źródłowym

if (snprintf(entryPath, PATH\_MAX, "%s/%s", *sourcePath*, sourceEntry->d\_name) >= PATH\_MAX)

{

currenTime();

printf("Problem wystąpił podczas próby pobrania pełnej ścieżki do pliku źródłowego\n");

syslog(LOG\_ERR, "Problem wystąpił podczas próby pobrania pełnej ścieżki do pliku źródłowego\n");

exit(EXIT\_FAILURE);

}

// Pobieramy informacje o pliku z katalogu źródłowego

if (stat(entryPath, &srcFileInfo) == -1)

{

currenTime();

printf("Problem wystąpił podczas próby pobrania informacji o pliku (z źródła)\n");

syslog(LOG\_ERR, "Problem wystąpił podczas próby pobrania informacji o pliku (z źródła)\n");

exit(EXIT\_FAILURE);

}

// Zmieniamy format daty i czasu modyfikacji pliku na czytelny dla użytkownika

strftime(modTimeSrc, sizeof(modTimeSrc), "%Y-%m-%d %H:%M:%S", localtime(&srcFileInfo.st\_mtime));

// Pętla while iteruje po wszystkich plikach w katalogu docelowym

// destinationEntry jest wskaźnikiem na strukturę reprezentującą kolejny plik

// w katalogu docelowym, a readdir() zwraca NULL, kiedy wszystkie pliki zostaną już przeczytane

while ((destinationEntry = readdir(destinationDir)) != NULL)

{

// Sprawdzenie, czy plik jest regularny

if ((destinationEntry->d\_type) == DT\_REG)

{

// Porównanie nazw plików

if (strcmp(sourceEntry->d\_name, destinationEntry->d\_name) == 0)

{

// Wyczyszczenie tablicy entryPath

clearTheArray(entryPath);

// sprawdzenie, czy długość pełnej ścieżki do pliku źródłowego nie przekracza PATH\_MAX

if (snprintf(entryPath, PATH\_MAX, "%s/%s", *destinationPath*, destinationEntry->d\_name) >= PATH\_MAX)

{

// zapisanie czasu wystąpienia problemu i wypisanie komunikatu o błędzie

currenTime();

printf("Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku źródłowego\n");

syslog(LOG\_ERR, "Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku źródłowego\n");

// zakończenie programu z kodem błędu

exit(EXIT\_FAILURE);

}

// sprawdzenie informacji o pliku docelowym

if (stat(entryPath, &destFileInfo) == -1)

{

// zapisanie czasu wystąpienia problemu i wypisanie komunikatu o błędzie

currenTime();

printf("Wystąpił problem podczas próby uzyskania informacji o pliku docelowym\n");

syslog(LOG\_ERR, "Wystąpił problem podczas próby uzyskania informacji o pliku docelowym\n");

// zakończenie programu z kodem błędu

exit(EXIT\_FAILURE);

}

// konwersja czasu modyfikacji pliku docelowego na string

strftime(modTimeDest, sizeof(modTimeDest), "%Y-%m-%d %H:%M:%S", localtime(&destFileInfo.st\_mtime));

// Sprawdzanie, czy czas modyfikacji pliku w źródle jest różny od czasu modyfikacji pliku w miejscu docelowym

if (strcmp(modTimeSrc, modTimeDest) != 0)

{

// Tworzenie pełnej ścieżki do pliku źródłowego

if (snprintf(srcFilePathContainer, PATH\_MAX, "%s/%s", *sourcePath*, sourceEntry->d\_name) >= PATH\_MAX)

{

currenTime();

printf("Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku źródłowego\n");

syslog(LOG\_ERR, "Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku źródłowego\n");

exit(EXIT\_FAILURE);

}

// Tworzenie pełnej ścieżki do pliku docelowego

if (snprintf(destFilePathContainer, PATH\_MAX, "%s/%s", *destinationPath*, destinationEntry->d\_name) >= PATH\_MAX)

{

currenTime();

printf("Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku docelowego\n");

syslog(LOG\_ERR, "Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku docelowego\n");

exit(EXIT\_FAILURE);

}

// Wywołanie funkcji kopiującej zawartość pliku źródłowego do pliku docelowego

copy(srcFilePathContainer, destFilePathContainer, mmapThreshold);

// Aktualizacja czasu modyfikacji pliku docelowego na czas modyfikacji pliku źródłowego

currenTime();

printf("Różne czasy modyfikacji: %s\n", sourceEntry->d\_name);

srcTimes.actime = srcFileInfo.st\_atime;

srcTimes.modtime = srcFileInfo.st\_mtime;

// Ustawianie czasów modyfikacji pliku docelowego

if (utime(destFilePathContainer, &srcTimes) < 0)

{

currenTime();

printf("Wystąpił problem podczas próby ustawienia czasu pliku docelowego\n");

syslog(LOG\_ERR, "Wystąpił problem podczas próby ustawienia czasu pliku docelowego\n");

exit(EXIT\_FAILURE);

}

}

}

}

}

}

else

{

// Wywołanie funkcji currenTime, która wypisuje aktualną godzinę w logach.

currenTime();

// Wypisanie informacji o nieznalezieniu pliku o tej samej nazwie w katalogu źródłowym.

printf("Nie znaleziono pliku o tej samej nazwie: %s\n", sourceEntry->d\_name);

// Zapisanie informacji do logów systemowych.

syslog(LOG\_INFO, "Nie znaleziono pliku o tej samej nazwie: %s\n", sourceEntry->d\_name);

// Skonstruowanie ścieżki do pliku źródłowego.

if (snprintf(entryPath, PATH\_MAX, "%s/%s", *sourcePath*, sourceEntry->d\_name) >= PATH\_MAX)

{

// Wywołanie funkcji currenTime, która wypisuje aktualną godzinę w logach.

currenTime();

// Wypisanie informacji o problemie z uzyskaniem pełnej ścieżki do pliku źródłowego.

printf("Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku źródłowego\n");

// Zapisanie informacji do logów systemowych.

syslog(LOG\_ERR, "Wystąpił problem podczas próby uzyskania pełnej ścieżki do pliku źródłowego\n");

// Zakończenie programu z kodem błędu.

exit(EXIT\_FAILURE);

}

// Sprawdzenie informacji o pliku źródłowym.

if (stat(entryPath, &srcFileInfo) == -1)

{

// Wywołanie funkcji currenTime, która wypisuje aktualną godzinę w logach.

currenTime();

// Wypisanie informacji o problemie z dostępem do informacji o pliku źródłowym.

printf("Wystąpił problem podczas próby uzyskania dostępu do informacji o pliku (ze źródła). Plik nie został znaleziony\n");

// Zapisanie informacji do logów systemowych.

syslog(LOG\_ERR, "Wystąpił problem podczas próby uzyskania dostępu do informacji o pliku (ze źródła). Plik nie został znaleziony\n");

// Zakończenie programu z kodem błędu.

exit(EXIT\_FAILURE);

}

// Sprawdzenie, czy długość ścieżki do pliku źródłowego nie przekracza PATH\_MAX

if (snprintf(srcFilePathContainer, PATH\_MAX, "%s/%s", *sourcePath*, sourceEntry->d\_name) >= PATH\_MAX)

{

currenTime();

printf("Problem wystąpił podczas próby uzyskania pełnej ścieżki do pliku źródłowego. Plik nie został znaleziony\n");

syslog(LOG\_ERR, "Problem wystąpił podczas próby uzyskania pełnej ścieżki do pliku źródłowego. Plik nie został znaleziony\n");

exit(EXIT\_FAILURE);

}

// Sprawdzenie, czy długość ścieżki do pliku docelowego nie przekracza PATH\_MAX

if (snprintf(destFilePathContainer, PATH\_MAX, "%s/%s", *destinationPath*, sourceEntry->d\_name) >= PATH\_MAX)

{

currenTime();

printf("Problem wystąpił podczas próby uzyskania pełnej ścieżki do pliku docelowego. Plik nie został znaleziony\n");

syslog(LOG\_ERR, "Problem wystąpił podczas próby uzyskania pełnej ścieżki do pliku docelowego. Plik nie został znaleziony\n");

exit(EXIT\_FAILURE);

}

// Skopiowanie pliku źródłowego do pliku docelowego z wykorzystaniem mmap lub read/write

copy(srcFilePathContainer, destFilePathContainer, mmapThreshold);

// Zapisanie komunikatu o powodzeniu do pliku logów i wyświetlenie go na ekranie

currenTime();

syslog(LOG\_INFO, "Plik %s został skopiowany pomyślnie.\n", sourceEntry->d\_name);

printf("Plik %s został skopiowany pomyślnie.\n", sourceEntry->d\_name);

// Ustawienie czasu dostępu i modyfikacji pliku docelowego na takie same jak źródłowego

srcTimes.actime = srcFileInfo.st\_atime;

srcTimes.modtime = srcFileInfo.st\_mtime;

// Ustawienie czasów dostępu i modyfikacji pliku docelowego przy użyciu funkcji utime

if (utime(destFilePathContainer, &srcTimes) < 0)

{

currenTime();

printf("Problem wystąpił podczas próby ustawienia czasów dostępu i modyfikacji pliku docelowego\n");

syslog(LOG\_ERR, "Problem wystąpił podczas próby ustawienia czasów dostępu i modyfikacji pliku docelowego\n");

exit(EXIT\_FAILURE);

}

}

}

// Wyczyszczenie tablic po każdej iteracji pętli oraz ponowne otwarcie katalogu docelowego

clearTheArray(srcFilePathContainer);

clearTheArray(destFilePathContainer);

clearTheArray(modTimeSrc);

clearTheArray(modTimeDest);

clearTheArray(entryPath);

rewinddir(destinationDir);

}

// Zamknięcie katalogów źródłowego i docelowego, wywołanie funkcji compareDestSrc oraz zamknięcie pliku logów

closedir(sourceDir);

closedir(destinationDir);

compareDestSrc(*sourcePath*, *destinationPath*);

closelog();

}

*void* recursiveSynchronization(*char* \**srcPath*, *char* \**dstPath*)

{

// Wywolujemy funkcje do kopiowania oraz synchronizacji katalogow i plikow

copyDirectory(*srcPath*, *dstPath*);

syncDirectory(*srcPath*, *dstPath*);

}

*void* Demon(*char* \*\**argv*)

{

// Odpowiednio od opcje -r wywolujemy funkcje

if (recursive)

recursiveSynchronization(*argv*[1], *argv*[2]);

else

compareSrcDest(*argv*[1], *argv*[2]);

}

*void* options(*int* *argc*, *char* \*\**argv*)

{

// Iterujemy po tablicy argumentow i odpowiednio ustawiamy wartosci zmiennych globalnych

for (*int* i = 3; i < *argc*; i++)

if (strcmp(*argv*[i], "-r") == 0)

recursive = true;

else if (strcmp(*argv*[i], "-t") == 0)

timeSleep = atoi(*argv*[i + 1]);

else if (strcmp(*argv*[i], "-d") == 0)

mmapThreshold = atoi(*argv*[i + 1]);

}

*void* sigusr1\_handler(*int* *signum*)

{

// Uruchomienie logowania dla demona z nazwą "Demon", dodanie identyfikatora procesu oraz określenie źródła logów jako użytkownik systemu

openlog("Demon", LOG\_PID, LOG\_USER);

// Zapisanie bieżącego czasu do logów i wypisanie informacji na ekranie o obudzeniu demona przez sygnał SIGUSR1

currenTime();

syslog(LOG\_INFO, "Demon obudzony przez sygnał SIGUSR1.\n");

printf("Demon obudzony przez sygnał SIGUSR1.\n");

// Ustawienie flagi wymuszającej synchronizację

forcedSynchro = true;

// Zamknięcie logowania

closelog();

}

*void* createDemon()

{

// Otwieranie logów systemowych z etykietą "Demon", z opcją dołączenia PID oraz opcją identyfikacji użytkownika

openlog("Demon", LOG\_PID, LOG\_USER);

// Tworzenie procesu potomnego

*pid\_t* pid, sid;

pid = fork();

if (pid < 0) // Proces potomny nie został utworzony

{

exit(EXIT\_FAILURE);

}

if (pid > 0) // Proces macierzysty kończy działanie, aby proces potomny mógł działać niezależnie

{

exit(EXIT\_SUCCESS);

}

// Ustawianie maski plików, aby demon nie ograniczał dostępu do plików

umask(0);

// Tworzenie nowej sesji dla procesu potomnego

sid = setsid();

if (sid < 0) // Błąd podczas tworzenia sesji

{

exit(EXIT\_FAILURE);

}

// Wyświetlanie informacji o PID demonu w konsoli i logach systemowych

currenTime();

printf("PID Procesu Demona: %d.\n", getpid());

syslog(LOG\_INFO, "PID Procesu Demona: %d.\n", getpid());

// Zamykanie standardowych deskryptorów plików

close(STDIN\_FILENO);

close(STDOUT\_FILENO);

close(STDERR\_FILENO);

// Zamykanie logów systemowych

closelog();

}

*int* main(*int* *argc*, *char* \*\**argv*)

{

openlog("Demon", LOG\_PID, LOG\_USER);

syslog(LOG\_INFO, " ");

syslog(LOG\_INFO, "---------------------------------\n");

syslog(LOG\_INFO, "------------>>>Hello<<<----------\n");

syslog(LOG\_INFO, "---------->>>DAEMONN!!<<<--------\n");

syslog(LOG\_INFO, "---------------------------------\n");

// Sprawdzenie, czy użytkownik podał poprawną ilość argumentów wejściowych

if (*argc* < 3)

{

currenTime();

syslog(LOG\_INFO, "Nieprawidłowa ilość argumentów wejściowych.\nPrawidłowe użycie: ./demon [KatalogŹródłowy] [KatalogDocelowy]\nOPCJE: -r [SynchronizacjaRekursywna] -t [CzasUśpienia] -d [PrógDzielącyDużePliki]");

printf("Nieprawidłowa ilość argumentów wejściowych.\nPrawidłowe użycie: ./demon [KatalogŹródłowy] [KatalogDocelowy]\nOPCJE: -r [SynchronizacjaRekursywna] -t [CzasUśpienia] -d [PrógDzielącyDużePliki]");

}

{

// Sprawdzenie podanych opcji i przekazanie ich do odpowiedniej zmiennej

options(*argc*, *argv*);

// Tworzenie demona

// createDemon();

currenTime();

printf("PID Procesu Demona: %d.\n", getpid());

// Przypisanie funkcji obsługi sygnału SIGUSR1

signal(SIGUSR1, sigusr1\_handler);

// Pętla nieskończona demona

while (1)

{

// Sprawdzenie, czy demon został obudzony

if (!forcedSynchro)

{

currenTime();

printf("Demon obudzony.\n");

syslog(LOG\_INFO, "Demon obudzony.\n");

forcedSynchro = false;

}

// Wywołanie funkcji synchronizującej katalogi

Demon(*argv*);

// Wypisanie informacji o usypianiu demona

currenTime();

syslog(LOG\_INFO, "Demon uśpiony.");

syslog(LOG\_INFO, "---------------------------------\n");

printf("Demon uśpiony.\n\n");

// Uśpienie demona na zadany czas

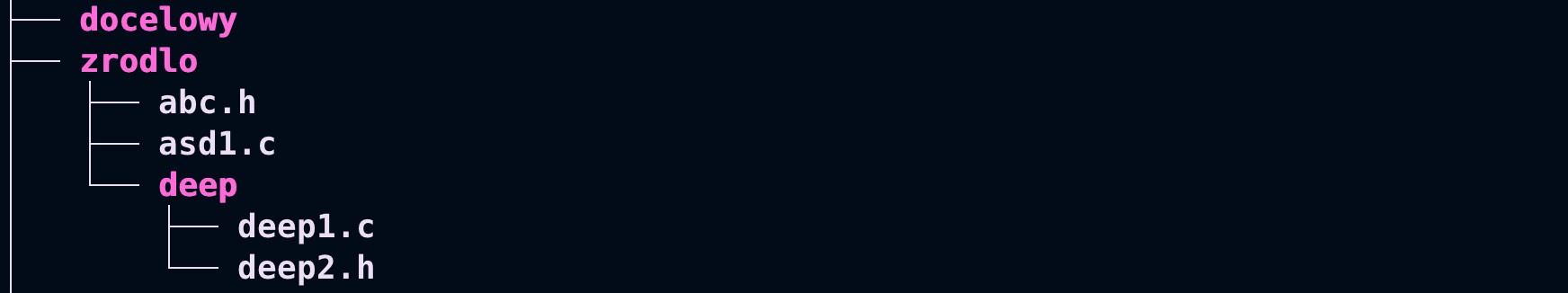
sleep(timeSleep);

}

}

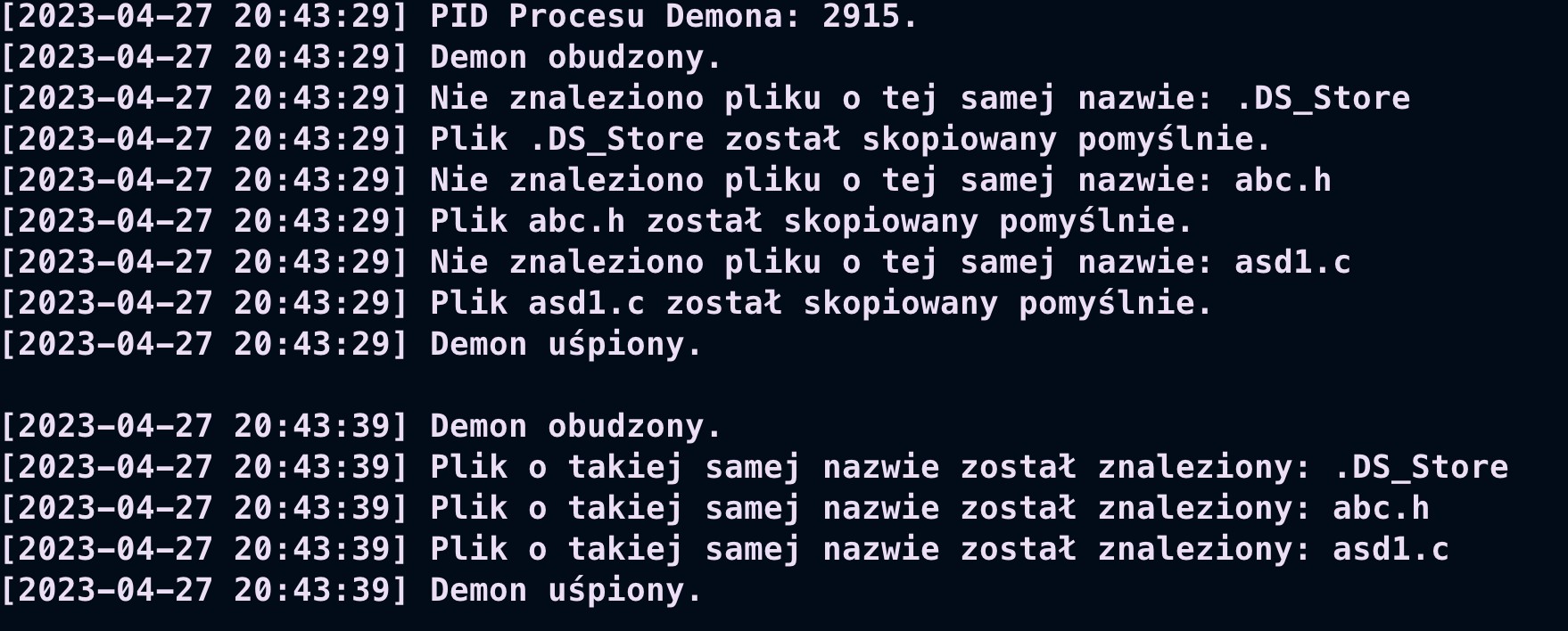
closelog();

}

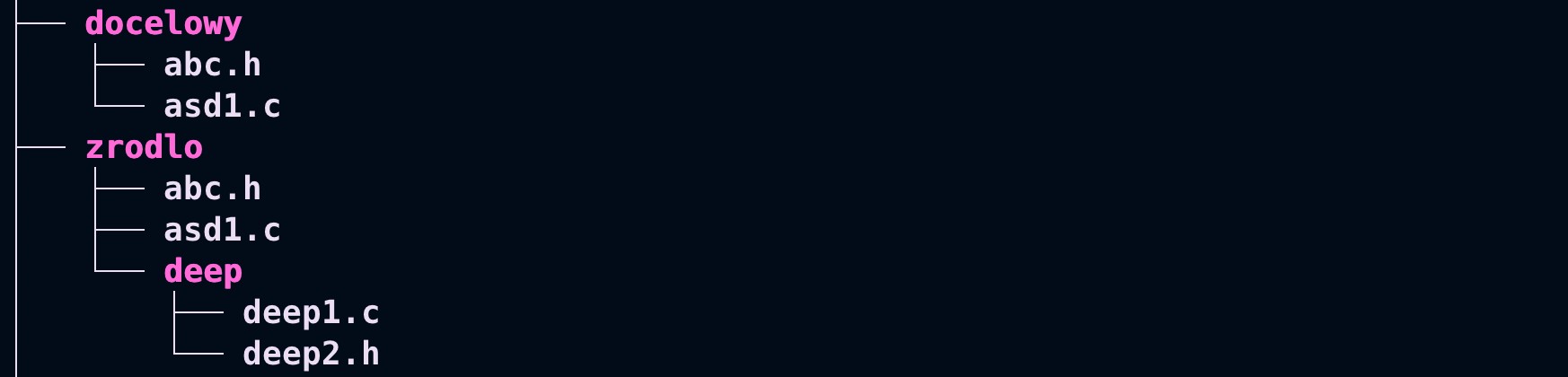
**Demonstracja funkcjonalności Demona**

Struktura katalogów, które będą synchronizowane:

Zgodnie z założeniami zadania włączamy demona ustalając katalog źródłowy i docelowy oraz wszystkie opcje, gdzie opcja -d 100 ustala próg kopiowania dużego pliku, -t 10 czas spania demona.

Otrzymaliśmy taki wynik demona:

Oczywiście wypisujemy takie same logi do logów systemów, lecz nie będziemy już tego pokazywać, ponieważ są identyczne. Demon skopiował wszystkie pliki do katalogu docelowego pomijając katalogi. Możemy również zauważyć demon obudził się po 10 sekundach i nie wykonał żadnego kopiowania tylko wylogował na konsole jakie pliki są zarówno w katalogu docelowym jak i źródłowym.

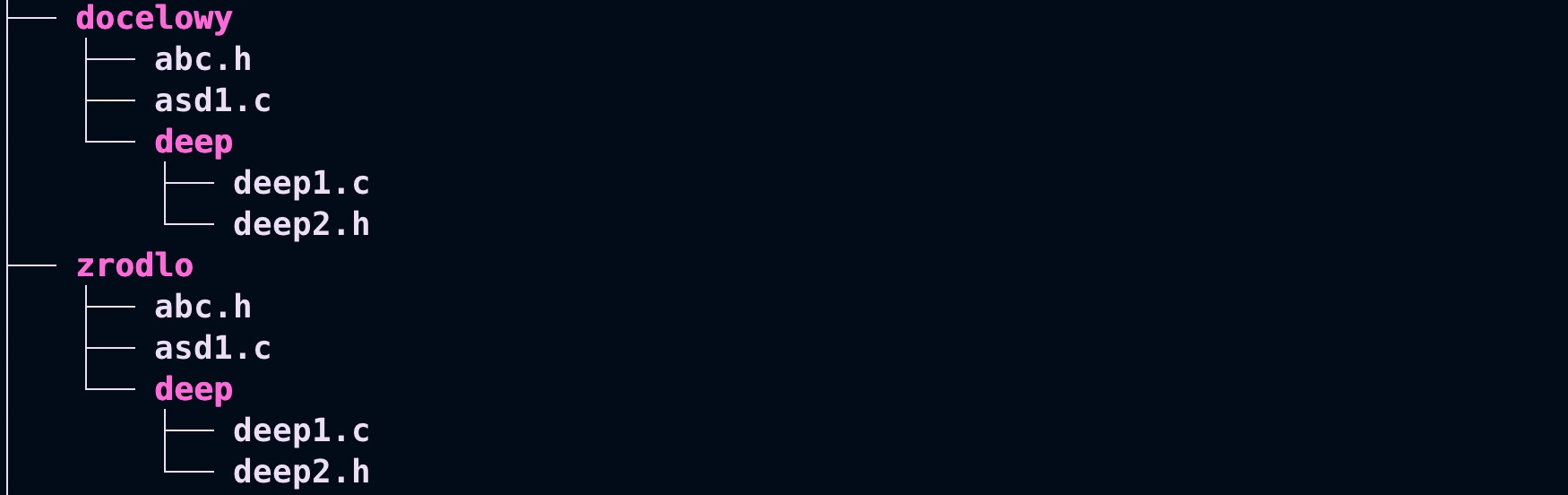
Sprawdźmy zatem czy naprawdę zmieniła się zawartość katalogu docelowego:

Drzewo plików zgadza się z tym co wylogował demon.

Obraz zawierający tekst

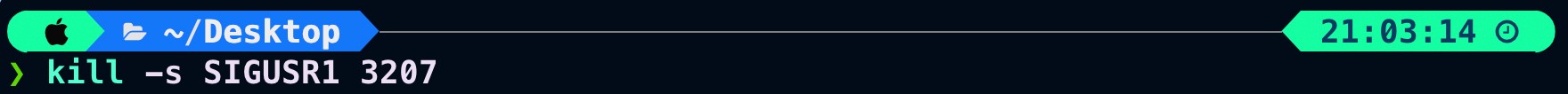
Opis wygenerowany automatycznieUżyjmy teraz demona z opcja -r, powinien on teraz zsynchronizować katalog „deep” do katalogu docelowego.

Stało się tak jak powinno, czyli demon znalazł katalog „deep” i przekopiował jego zawartość do katalogu docelowego. Demon wybudził się po 10 sekundach I znalazł katalog „deep” lecz nic z nim nie zrobił, ponieważ czasy modyfikacji plików były takie same.

Obraz zawierający tekst

Opis wygenerowany automatycznieGdy zmodyfikuje na przykład plik deep2.h powinniśmy otrzymać log, że czasy modyfikacj są różne oraz, że plik został skopiowany.

Wszystko się zgadza z naszym drzewem katalogów.

Stało się tak jak pisałem kilka linijek wyżej oraz teraz synchronizacja jest kompletna, iż katalog źródłowy oraz docelowy maja taka sama zawartość.

Gdy wpiszemy w konsole takie polecenie podczas spania demona:

Obraz zawierający tekst

Opis wygenerowany automatycznieDemon podczas spania powinien przechwycić sygnał SIGUSR1 oraz obudzić się i dalej synchronizować katalogi. Również powinniśmy użyć polecenia: kill [PID procesu demona], aby zakończyć proces, iż w przeciwnym wypadku demon będzie pracować w tle do momentu wyłączenia komputera.

Zmieńmy w katalogu źródłowym nazwę pliku abc.h na abcd.h oraz nazwę pliku deep1.c na deeep1.c. Gdy skompilujemy demona bez opcji -r usunie on tylko plik abc.h z katalogu docelowego i skopiuje plik abcd.h do katalogu docelowego, ponieważ nie mamy włączonej opcji, która rozpatruje katalogi.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznieNatomiast gdy skompilujemy demona z opcja -r to plik deeep1.c zostanie skopiowany do katalogu docelowego oraz zostanie usunięty plik deep1.c

**Opis podziału pracy nad Projektem**

1. ***Mateusz Mogielnicki***
   * + changeModTime()
     + currenTime()
     + Demon()
     + options()
     + sigusr1\_handler()
     + createDemon()
     + main()
     + Wszelkie poprawki związane z błędami w innych funkcjach
     + Logi w konsoli oraz logach systemowych
     + Sprawozdanie oraz dokumentacja w kodzie
2. ***Dominik Mierzejewski***
   * + *Zmienne Globalne*
     + *copy()*
     + *copyUsingReadWrite()*
     + *copyUsingMMapWrite()*
3. ***Przemysław Rutkowski***
   * + *clearTheArray()*
     + *compareDestSrc()*
     + *compareSrcDes()*

1. ***Jakub Matyszak***
   * + *copyDirectory()*
     + *syncDirectory()*
     + *recursiveSynchronization()*