Wydział informatyki Politechniki Białostockiej Przedmiot: Systemy Operacyjne	Data:
Temat: Demon synchronizujący dwa podkatalogi.	Prowadzący:

[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 źródłowym 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:
a) **[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.
b) **[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.
Uwagi: (a) Wszelkie operacje na plikach należy wykonywać przy pomocy API Linuksa a nie standardowej biblioteki języka C (b) kopiowanie za każdym obudzeniem całego drzewa katalogów zostanie potraktowane jako poważny błąd (c) podobnie jak przerzucenie części zadań na shell systemowy (funkcja system).

Plik „main.h”

```c
#ifndef __MAIN_H__
#define __MAIN_H__

#pragma once
#include <dirent.h>
#include <errno.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <time.h>
#include <stdbool.h>
#include <fcntl.h>
#include <signal.h>

#endif
```

Rys. 1 Standardowe biblioteki języka C zawarte w projekcie.

Funkcje zawarte w powyższych bibliotekach były wymagane do poprawnej realizacji projektu które zostały umieszczone w pliku „main.h”.

Plik „functions.h”

```c
//--------FUNCTIONS----------//

char* createPath(char* path, char* next_el);

int isDir(char* path);

int isFile(char* path);

int lastModifyDateCmp(char* path1, char* path2);

int fileCopier(char* sorcePath,char* outputPath,long long fileSize);

void dirMakeEmpty(char* dirPath);

int lastModifyDateSet(char* path1, char* path2);
```

Rys. 2 Funkcje zawarte w pliku functions.h

```c
char* createPath(char* path, char* next_el){
    char* str = malloc(sizeof(char)*(strlen(path)+strlen(next_el)+2));
    strcpy(str,path);
    if(str[strlen(path)-1] != '/')
      strcat(str, "/");
    strcat(str, next_el);
    return str;
}
```

Rys. 3 F-cja "createPath"

Pierwsza stworzona funkcja – createPath tworzy pełną ścieżkę do pliku kopiowanego przez program, ponieważ od użytkownika wymagane jest podanie zaledwie ostatniej części danej ścieżki.

```c
int isDir(char* path)
{
    struct stat info;
    if (stat(path, &info) != 0)return 0;
    if(S_ISDIR(info.st_mode))
        return 1;
    else
      return -1;
}
```

Rys. 4 F-cja isDir

Funkcja isDir wczytuje wartość string mającą być ścieżką i sprawdza, czy ścieżka prowadzi do folderu, jeśli tak zwraca wartość 1 w przeciwnym przypadku zwraca -1.

```c
int isFile(char* path)
{
    struct stat info;
    if (stat(path, &info) != 0)return 0;
    if(S_ISREG(info.st_mode))
        return 1;
    else
        return -1;
}
```

Rys. 5 F-cja isFile

Funkcja ta sprawdza czy ścieżka prowadzi do pliku, jeśli tak zwraca wartość 1, w przeciwnym przypadku -1.

```c
int fileCopier(char* sourcePath,char* destinyPath,long long fileSize)
{
    struct stat info;

    char buffor[BUFF_SIZE];
    int count = 1;
    int file = open(sourcePath, O_RDONLY, 0666);
    int file2 = open(destinyPath,O_WRONLY | O_CREAT, 0666);
    if (stat(sourcePath, &info) != 0)return 0;
    if(info.st_size<fileSize)
    {
        while(1){
        count = read(file,buffor,BUFF_SIZE);
        if(count!=0)write(file2,buffor,count);
        else break;
        }
    }
    else{
        char *file_in_memory = mmap(NULL,info.st_size,PROT_READ,MAP_PRIVATE,file,0);
        for(int i=0;i<info.st_size;i++)
        {
            write(file2,&file_in_memory[i],sizeof(char));
        }
        munmap(file_in_memory,info.st_size);

    }
    close(file);
    close(file2);
    return 1;
}
```

Rys. 6 F-cja fileCopier

Funkcja kopiująca pojedynczy plik. Wczytuje ścieżkę prowadzącą do pliku kopiowanego oraz do docelowego folderu a także wielkość pliku który powinien zostać skopiowany za pomocą mapowania. Zawiera instrukcje warunkowe sprawdzające czy plik jest odpowiednio mały by skopiować go za pomocą bufora, czy jest wymagane mapowanie. Gdyby wystąpił błąd funkcja zwraca 0, w przypadku powodzenia – zwraca 1.

```c
void dirMakeEmpty(char* dirPath){
    DIR *dir;
    struct dirent *entry;
    time_t t;
    struct tm *tm;

    if ((dir = opendir(dirPath)) == NULL) perror("opendir() error");
    else {
        while ((entry = readdir(dir)) != NULL){
            if(strcmp(entry->d_name,".")==0)continue;
            if(strcmp(entry->d_name,"..")==0)continue;

            char* fullPath = createPath(dirPath, entry->d_name);

            if(isDir(fullPath)==1){
                dirMakeEmpty(fullPath);
                rmdir(fullPath);
                t = time(NULL);
                tm = localtime(&t);
                syslog (LOG_INFO, "%s: Removing directory %s", asctime(tm), fullPath);
            }
            else {
                remove(fullPath);
                t = time(NULL);
                tm = localtime(&t);
                syslog (LOG_INFO, "%s: Removing %s", asctime(tm), fullPath);
            }
            free(fullPath);
        }
    }
}
```

Rys. 7 F-cja dirMakeEmpty

Funkcja ta odpowiada za rekurencyjne usunięcie wszystkich plików i folderów zawartych we wskazanym katalogu. Wykorzystywana jest wtedy, gdy demon znajdzie pliki które nie powinny się znajdować w folderze docelowym.

```c
int lastModifyDateCmp(char* path1, char* path2)
{
    struct stat info;

    if (stat(path1, &info) != 0)
        perror("stat() error");

    time_t time1 = info.st_mtime;

    if (stat(path2, &info) != 0)
        perror("stat() error");

    time_t time2 = info.st_mtime;
    double diff = difftime(time1, time2);
    if(diff<0)return -1;
    if(diff == 0)return 0;
    return 1;
}
```

Rys. 8 F-cja lastModifyDateCmp

Powyższa funkcja odpowiada za sprawdzenie daty modyfikacji pliku. Wczytuje ścieżki do dwóch porównywanych ze sobą plików. Funkcja zwraca -1 gdy 2 plik jest starszy, 0 jeśli data modyfikacji jest taka sama oraz 1 jeśli pierwszy plik jest starszy.

```c
int lastModifyDateSet(char* path1, char* path2){
    struct utimbuf ubuf;
    struct stat info;

    stat(path2,&info);
    ubuf.modtime = info.st_mtime;  /* set modification time */
    time(&ubuf.actime);
    if (utime(path1, &ubuf) != 0){
        perror("utime() error");
        return 0;
    }
    return 1;
}
```

Rys. 9 F-cja lastModifyDateSet

Funkcja odpowiada za ostawienie daty modyfikacji aktualnie zmienianaego pliku na datę modyfikacji pliku źródłowego. Zwraca 1 w przypadku udanej modyfikacji, a 0 w przypadku wystąpienia błędu.

Plik „main.c”:

```c
#include "main.h"
#include "functions.h"


void filesCopy(char* source, char* destiny, long long fileSize, int R){
    DIR *dir; struct dirent *entry; time_t t; struct tm *tm;
    if ((dir = opendir(source)) == NULL) perror("opendir() error");
    else {
        while ((entry = readdir(dir)) != NULL){
            if(strcmp(entry->d_name,".")==0)continue;
            if(strcmp(entry->d_name,"..")==0)continue;
            char* fullSorcePath = createPath(source, entry->d_name);
            char* fullDestinyPath = createPath(destiny, entry->d_name);
            if(isFile(fullSorcePath)==1){ //IF REGULAR FILE
                t = time(NULL);     tm = localtime(&t);
                if(isFile(fullDestinyPath)==1){ //IF FILE ALREADY EXSISTS
                    if(lastModifyDateCmp(fullSorcePath,fullDestinyPath)!=0)
                    {
                        remove(fullDestinyPath);
                        fileCopier(fullSorcePath,fullDestinyPath,fileSize);
                        lastModifyDateSet(fullSorcePath,fullDestinyPath);
                        syslog (LOG_INFO, "%s: Copying %s to %s", asctime(tm), fullSorcePath, fullDestinyPath);
                    }
                }
                else {
                    fileCopier(fullSorcePath,fullDestinyPath,fileSize);
                    syslog (LOG_INFO, "%s: Copying %s to %s", asctime(tm), fullSorcePath, fullDestinyPath);
                }
            }
            else if(R && isDir(fullSorcePath)==1){ //IF DIRECTORY
                t = time(NULL);     tm = localtime(&t);
                if(isDir(fullDestinyPath)==1){ //IF DIR ALREADY EXSISTS
                    filesCopy(fullSorcePath,fullDestinyPath, fileSize, R);
                }
                else {
                    mkdir(fullDestinyPath, 0777);
                    syslog (LOG_INFO, "%s: Create directory %s", asctime(tm), fullDestinyPath);
                    filesCopy(fullSorcePath,fullDestinyPath, fileSize, R);
                }
            }
            free(fullSorcePath);    free(fullDestinyPath);
        }
    }
    closedir(dir);
}
```

Rys. 10 F-cja "filesCopy"

Plik „main” rozpoczyna się załączeniem biblioteki oraz bibliotekami zawartymi w main.h, Pierwszą wykorzystywaną przez program funkcją jest *filesCopy* której zadaniem jest skopiowanie plików zawartych w określonym przez użytkownika folderze do folderu docelowego również podanym przez użytkownika. Poza ścieżkami do folderów, funkcja wczytuje również wielkość plików od której są one kopiowane za pomocą mapowania oraz wskaźnik czy program wywołano z rozszerzeniem rekurencji.

Funkcja ta sprawdza czy ścieżki wskazują na folder czy plik, oraz czy powinien pliki kopiować. Na końcu zwalnia pamięć zalokowaną do wyznaczenia pełnych ścieżek do pliku/folderu źródłowego oraz docelowego i zamyka deskryptor, aby uniknąć wycieków pamięci.

```c
static void demon()
{
    pid_t pid;

    pid = fork();

    if (pid < 0)exit(EXIT_FAILURE);
    if (pid > 0)exit(EXIT_SUCCESS);
    if (setsid() < 0)exit(EXIT_FAILURE);

    signal(SIGCHLD, SIG_IGN);
    signal(SIGHUP, SIG_IGN);

    pid = fork();

    if (pid < 0)exit(EXIT_FAILURE);
    if (pid > 0)exit(EXIT_SUCCESS);

    umask(0);
    chdir("/");

    int x;
    for (x = sysconf(_SC_OPEN_MAX); x>=0; x--)close (x);

     syslog (LOG_INFO, "DAEMON CREATED pid: %d",getpid());
}
```

Rys. 11 F-cja „demon”

Kluczowa funkcja projektu, „demon”. Jej zadaniem jest wyłączenie programu z konsoli i aktywowanie go jako programu działającego w tle co definiuje go jako demon. Uruchomienie tej funkcji zapisuje się w logach wraz z pidem pozwalającym na wyselekcjonowanie tego procesu.

```c
void handler()
{
    syslog (LOG_INFO, "DAEMON RECIVED SIGUSR1 pid: %d",getpid());
}
```

Rys. 12 F-cja "handler"

Funkcja ta zapisuje do logów informacje o otrzymaniu polecenia poprzez SIGUSR1 wraz z Pid-em procesu. Jest to handler dla sygnału którego zadaniem jest budzenie demona.

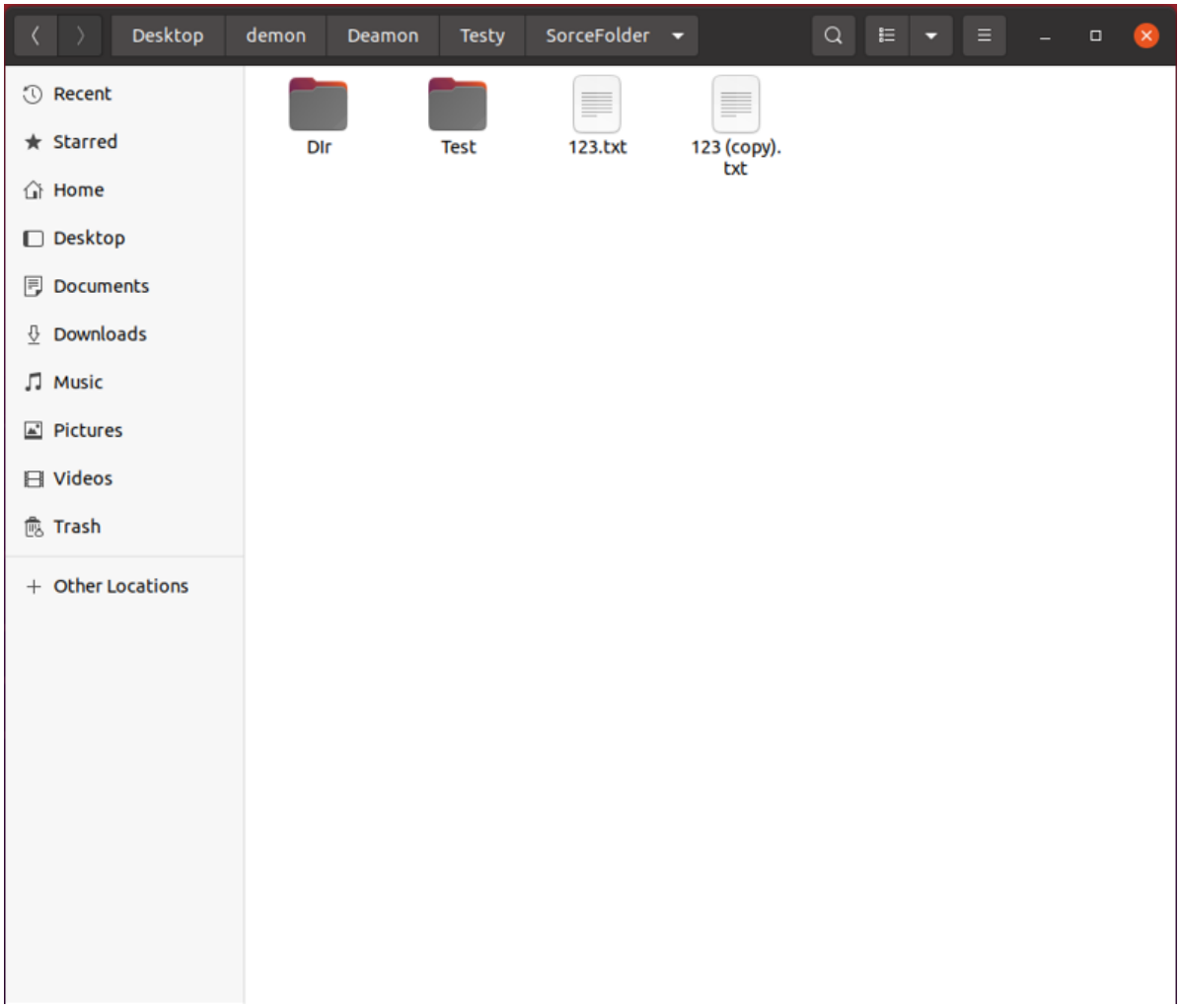
```c
int main(int argc, char* argv[])
{
    char *source =argv[1], char *destiny =argv[2];
    int waitTime = 300, long long fileSize = 1024, int R = 0;
    for(int i = 3; i<argc; i++)
    {
        char* x = argv[i];
        switch(x[1])
        {
            case 'R':
            {
                R=1;
                break;
            }
            case 'T':
            {
                waitTime = atoi(argv[++i]);
                if(waitTime==0)exit(0);
                break;
            }
            case 'S':
            {
                fileSize = atoll(argv[++i]);
                if(fileSize==0)exit(0);
                break;
            }
        }
    }

    signal(SIGUSR1, handler);
    demon();
    while (1)
    {
        syslog (LOG_INFO, "DAEMON RUNNING COPING");
        filesRemove(source,destiny, R);
        filesCopy(source,destiny, fileSize, R);
        syslog (LOG_INFO, "DAEMON GO TO SLEEP");
        sleep(waitTime);
    }
    syslog (LOG_INFO, "DAEMON TRREMINATED");
    closelog();
    return EXIT_SUCCESS;
}
```

Rys. 13 F-cja main

Funkcja main programu. Wczytuje od użytkownika dowolną ilość argumentów i odpowiednio na nie reaguje. Pierwszymi argumentami są ścieżki do plików – najpierw źródłowego, następnie docelowego, a potem program wczytuje „aspekty” jak ma zostać wywołany (Uwaga, ważne jest określenie parametru z wykorzystaniem „-„ – Wpisanie litery R informuje program o działaniu rekurencyjnym, litera T zmienia ilość czasu co jaki demon budzi się samoczynnie, po literze T powinno się podać czas w sekundach, a podanie litery S informuje o zmianie granicy wielkości plików do kopiowania za pomocą mapowania, po literze S powinno się podać wielkość pliku w bajatach. Funkcja main po zinterpretowaniu argumentów

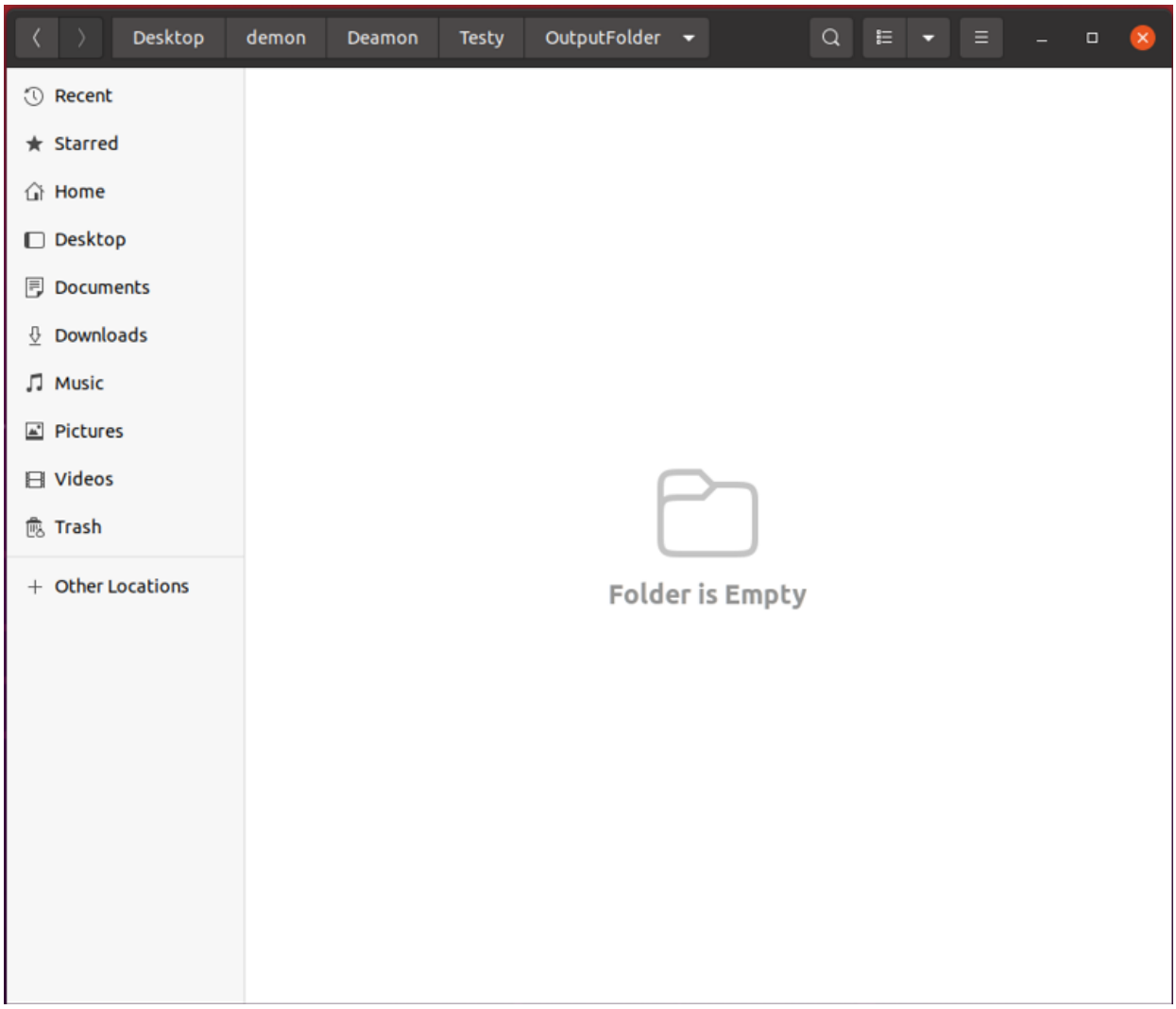
wstępnych definiuje sygnał użytkownika – SIGUSR1 i budzi demona wykorzystując funkcję demon(). Zadanie demona znajduje się w nieskończonej pętli while – jest to zapis do logów systemowych i wykonanie synchronizacji wykorzystując zdefiniowane wcześniej funkcje – filesRemove() i filesCopy(). Po zakończeniu działania demon zapisuje do logów kiedy został zabity i program się kończy.

Zrzuty ekranu z działającego programu:



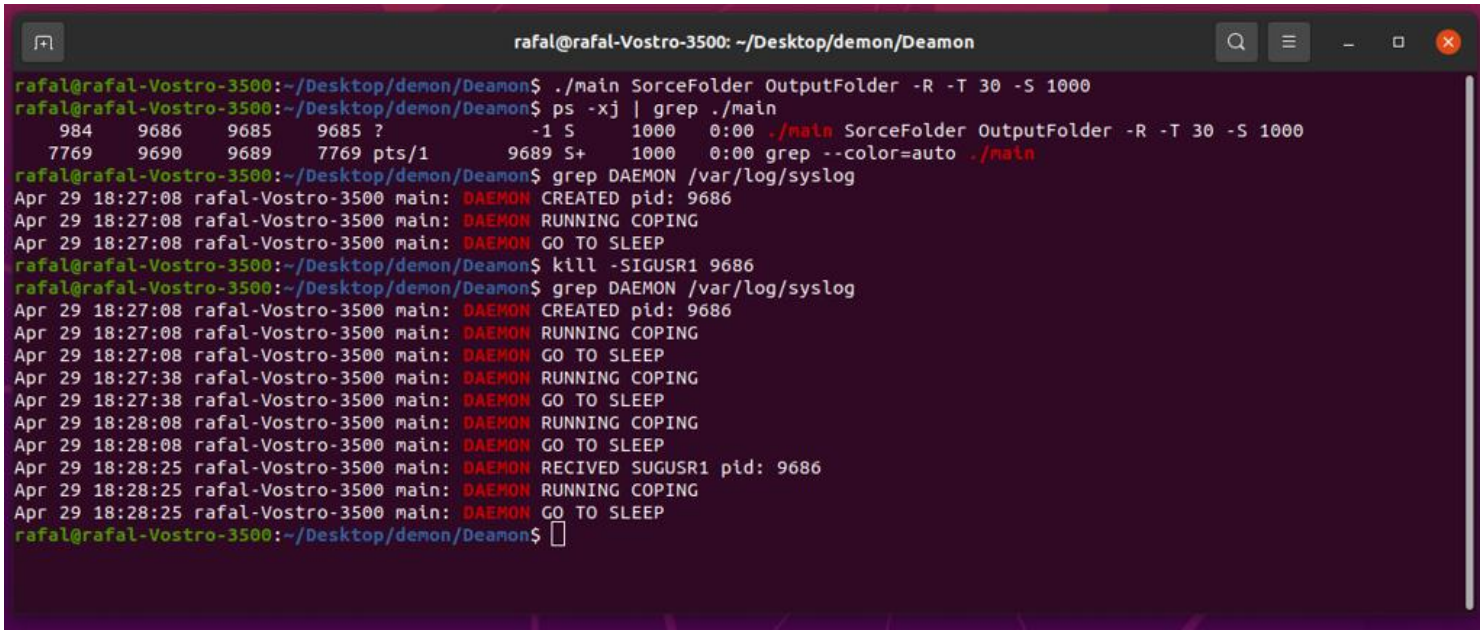
Rys. 14 SourceFolder przed uruchomieniem demona

Plik źródłowy z testowymi plikami i folderami.



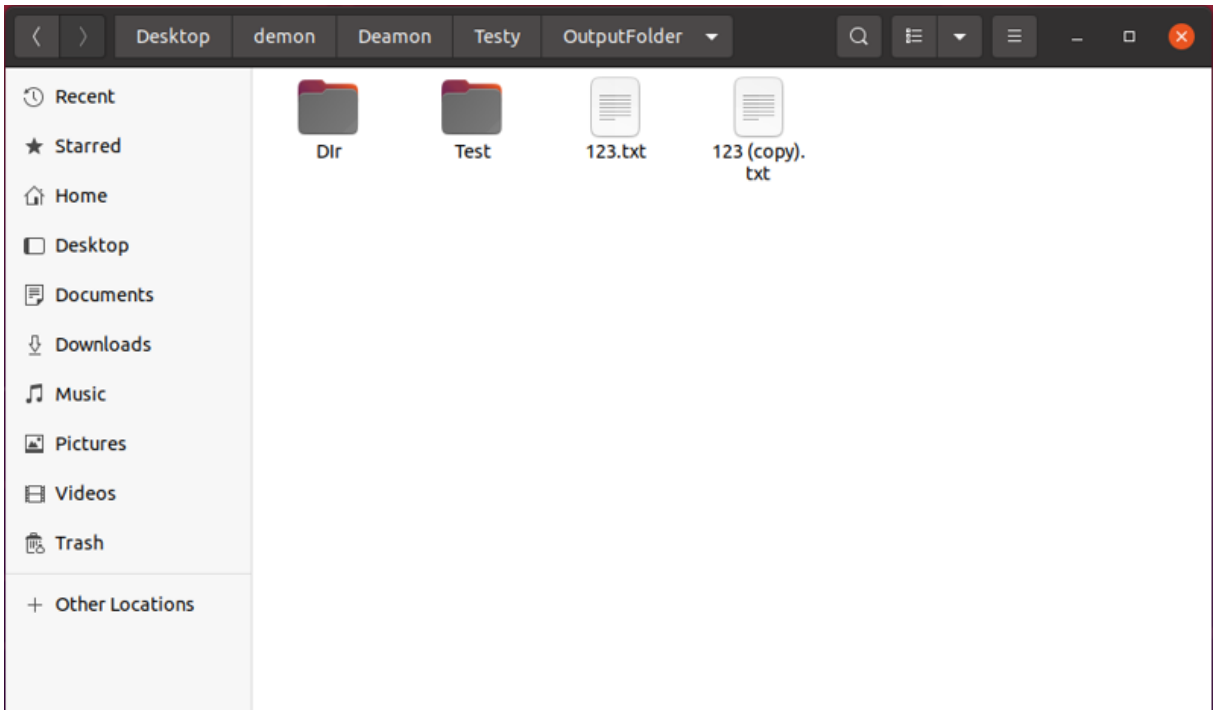
Rys. 15 OutputFolder przed uruchomieniem demona

Folder docelowy stworzony i oczekujący na obudzenie się demona.



Rys. 16 Uruchomienie demona

Na powyższym screenie widać działanie demona. Program uruchomiono z wykorzystaniem własnych parametrów – czas oczekiwania zmieniono na 30 sekund, wielkość pliku do mapowania ustawiono na 1000B, potem wyświetlono logi dla naszego programu, następnie wybudzono demona ręcznie za pomocą sygnału SIGUSR1 i ponownie wyświetlono logi demona.



Rys. 17 OutputFolder po uruchomieniu działania demona

Wszystkie pliki oraz foldery zostały skopiowane poprawnie, z jednakową zawartością i taką samą wagą.

Podział pracy: