Kamil Kośko, Igor Kowalczyk, Michał Kuczyński

Demon synchronizujący dwa podkatalogi

Projekt składa się z trzech komponentów:
pliku nagłówkowego “demonlib.h”
pliku zawierającego wykorzystywanie funkcje “demonlib.c”
pliku głównego “main.c”

Jest to program przyjmujący minimum dwa argumenty (folder źródłowy, oraz docelowy).
Demon śpi (domyślnie 5 minut), usuwa wszelkie elementy z folderu target, których nie ma w source, a później kopiuje elementy z source do target.
Oprócz podstawowego kopiowania program umożliwia:

- Ustawianie czasu snu demona
- Ustawianie rozmiaru pliku, który pozwoli rozróżnić pliki duże od małych, duże pliki są kopiowane przez mapowanie
- Ustawianie rekurencji (kopiowane i usuwane będą też pliki które znajdują się w podkatalogach katalogów źródłowych)
- Wybudzanie demona przy pomocy sygnału SIGUSR1

Demon podczas swojego działania wysyła logi systemowe z informacjami o jego działaniu.

Jak uruchomić demona:
./demon.out -i [ścieżka do source] -o [ścieżka do target] [-t [czas spania]] [-r] [-y [rozmiar dzielący na duże i małe pliki]]
Budzenie za pomocą sygnału SIGUSR1:
kill - SIGUSR1 pid

Pętla przyjmująca argumenty przy wywołaniu demona:

```
while((parameter = getopt(argc, argv, "i:o:t:y:r")) != -1)
{
    switch(parameter)
    {
        case 'i':
        {
            char* check = optarg;
            if(stat(check, &file) == 0)
            {
                if(file.st_mode &S_IFDIR)
                {
                    sourcePath = optarg;
                }
                else
                {
                    syslog(LOG_ERR, "Not a source folder");
                    exit(EXIT_FAILURE);
                }
            }
            break;
        }
        case 'o':
        {
           char* check = optarg;
            if(stat(check, &file) == 0)
            {
                if(file.st_mode &S_IFDIR)
                {
                    targetPath = optarg;
                }
                else
                {
                    syslog(LOG_ERR, "Not a target folder");
                    exit(EXIT_FAILURE);
                }
            }
        }
        case 't':
```

W pętli przyjmowane są argumenty takie jak:

foldery na których demon będzie operować, czas spania demona, rozmiar rozdzielający pliki na małe i duże, oraz rekurencję (czy algorytm ma brać pod uwagę podkatalogi).

Obsługa sygnału SIGUSR1:

```
    if(signal(SIGUSR1, logHandler)==SIG_ERR)
    {
        syslog(LOG_ERR, "Signal Error!");
        exit(EXIT_FAILURE);
    }
```

w przypadku otrzymania sygnału SIGUSR wywoływana jest funkcja logHandler, oraz demon jest “obudzony” tj. wychodzi z funkcji sleep.

Główna pętla demona:

```
while(1)
{
    syslog(LOG_INFO, "Demon has gone to sleep %d seconds ", sleepTimeSeconds;
    sleep(sleepTimeSeconds);
    syslog(LOG_INFO, "Demon awake");
    clearCatalogs(sourcePath, targetPath, targetPath, recurSync);
    compareCatalogs(sourcePath, targetPath, threshold, recurSync);

}
```

Główna pętla, w której demon śpi, oraz wywoływane są dwie najważniejsze funkcje programu, usuwanie, oraz kopiowanie plików.

Usuwanie plików:

```
195     DIR * path, *del;
196     path = opendir(currentFile);
197     while(( file = readdir (path)))
198     {
199         if((file->d_type)==DT_DIR)
200         {
201             if(recurSync)
202             {
203                if( !( strcmp(file->d_name, ".") == 0 || strcmp(file->d_name, "..") == 0))
204                {
205                     char* newPath = pathToFile(currentFile, file->d_name);
206                     clearCatalogs(sourcePath, targetPath, newPath,"1");
207                     free(newPath);
208                     char * pathToDelete = changeCatalogs(newPath, sourcePath);
209                if(!(del=opendir(pathToDelete)))
210                     {
211                         remove(newPath);
212                         syslog(LOG_INFO, "Deleted catalog %s", newPath);
213                     }
214                     else
215                     {
216                         closedir(del);
217                     }
218                     free(pathToDelete);
219                }
220             }
221         }
222         else
223         {
224             char* newPath = pathToFile(currentFile, file->d_name);
225             char* pathToDelete = changeCatalogs(newPath, sourcePath);
226             if(access(pathToDelete,F_OK)== -1 )
227             {
228                 remove(newPath);
229                 syslog(LOG_INFO, "Deleted file %s", pathToDelete);
230             }
231             free(pathToDelete);
```

Algorytm, w zależności czy rekurencja jest włączona porównuje pliki tylko w target, oraz source, lub też w ich podkatalogach. Algorytm ignoruje dowiązania, przechodzi po wszystkich plikach w katalogu target, zamienia ścieżkę z target na teoretyczną w source, jeżeli tego pliku nie ma w katalogu source, usuwa go.

Kopiowanie plików:

```
void compareCatalogs(char* sourcePath, char* targetPath, int threshold, bool recurSync)
{
    DIR* sourceDir = opendir(sourcePath);
    DIR* targetDir;
    struct dirent* sourceStreamDirFile;
    struct dirent* targetStreamDirFile;

    while((sourceStreamDirFile = readdir(sourceDir)))
    {
        if(sourceStreamDirFile->d_type != DT_DIR)
        {
            char* fileNameSource = sourceStreamDirFile->d_name;
            targetDir = opendir(targetPath);
            do
            {
                targetStreamDirFile = readdir(targetDir);
                if(targetStreamDirFile)//znaleziono plik w folderze
                {
                    char* fileNameTarget = targetStreamDirFile->d_name;
                    unsigned char fileType = targetStreamDirFile->d_type;
                    if(!strcmp(fileNameTarget, ".") || !strcmp(fileNameTarget, "..") && fileType == DT_DIR)
                    {
                        continue;
                    }
                    char* sourceFilePath = pathToFile(sourcePath, fileNameSource);
                    char* targetFilePath = pathToFile(targetPath, fileNameSource);

                    if(compareFiles(fileNameSource, fileNameTarget, sourceFilePath, targetFilePath,  threshold))
                    {
                        break;
                    }
                }
```

```
                else//folder jest pusty
                {
                    char* sourceFilePath = pathToFile(sourcePath, fileNameSource);
                    char* targetFilePath = pathToFile(targetPath, fileNameSource);
                    updateFile(sourceFilePath, targetFilePath, threshold);
                    break;
                }
            }while(true);
            closedir(targetDir);
        }
        else if(recurSync == true && sourceStreamDirFile->d_type == DT_DIR)
        {
            char* dirName = sourceStreamDirFile->d_name;
            if(strcmp(dirName, ".") && strcmp(dirName, ".."))
            {
                char* newDirSourcePath = pathToFile(sourcePath, dirName);
                char* newDirTargetPath = pathToFile(targetPath, dirName);
                if(isCatalogExist(dirName,targetPath))
                {
                }
                else
                {
                    if(mkdir(newDirTargetPath, S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH))
                    {
                        syslog(LOG_ERR, "Error while mkdir: %s", newDirTargetPath);
                        exit(EXIT_FAILURE);
                    }
                }
                compareCatalogs(newDirSourcePath, newDirTargetPath, threshold, recurSync);
            }
        }
    }
    closedir(sourceDir);
}
```

Funkcja iteruje po plikach w katalogu source (lub jego podkatalogach), ignoruje dowiązania, sprawdza czy ten plik istnieje w katalogu target i jaką ma datę ostatniej modyfikacji, przy pomocy funkcji compareFiles, jeżeli data jest starsza, lub tego pliku nie ma, jest on kopiowany przy pomocy funkcji updateFile.