Dokumentacja programu Demon Monitorujący.
Autor: Dawid Ugniewski.

1. Lista zaimplementowanych funkcjonalności:

a) Makefile implementujący kompilację programu i usuwanie plików obiektów i wykonywalnych.

b) Skrypt generujący duże pliki (10, 20, 50, 100, 500 i 1000 MiB) do testowania wydajności kopiowania. (**stworz_pliki.sh**)

c) Obsługa wszystkich opcji (długich lub krótkich) i argumentów przy uruchamianiu za pomocą getopt_long z obsługą błędów.

d) Odforkowanie procesu rodzica z obsługą błędów w celu utworzenia demona.

e) Obsługa sygnału SIGUSR1 budzącego demona. Sygnał odbierany jest tylko w momencie, kiedy demon śpi, inaczej jest blokowany. Obsługa polega na wywołaniu „longjmp" do pętli demona.

f) Przy pierwszym uruchomieniu demon tworzy kopię katalogu źródłowego w katalogu docelowym. Następnie odczytuje wszystkie wpisy i dla każdego wpisu źródłowego kopiuje plik do stworzonego katalogu.

g) Kopiowanie plików podzielone jest progiem „dużego pliku" (przyjmowanym jako argument [MB]) na niskopoziomowe read/write i sendfile. W sekcji **3.b)** dokonana jest analiza obu sposobów względem wpływu na czas kopiowania. Za kopiowanie niskopoziomowe odpowiada funkcja **skopiujPlikNiskopoziomowo**, a za kopiowanie przez sendfile **skopiujPlikEfektywnie**. (*dodatkowe)*

h) Po utworzeniu kopii danych demon wchodzi w nieskończoną pętlę i usypia na czas określony w argumencie sleeptime (domyślnie jest to 5 minut)

i) Po przebudzeniu demon odczytuje wpisy z katalogu źródłowego, który monitoruje i ze stworzonego katalogu. Jeśli jakiś plik dodano, usunięto lub zmieniono, demon wykonuje tą zmianę również w stworzonym katalogu i loguje o niej do logu systemowego po czym znowu usypia.

j) Porównywanie treści plików odbywa się na podstawie porównania sum kontrolnych plików funkcją hashującą EVP_sha1 z biblioteki OpenSSL. Za prównywanie odpowida funkcja **porownajPliki.** (*dodatkowe)*

k) Program loguje o wszystkich operacjach dokonywanych na plikach, o otrzymaniu sygnału SIGUSR1 i o akcjach podejmowanych przez demona typu uśpienie/obudzenie do logu systemowego. Wysyłane logi są odpowiedniego rodzaju i priorytetu.

l) Program pracuje z efektywm UID i efektywnym GID takimi jak rzeczywiste UID i rzeczywiste GID użytkownika uruchamiającego program.

m) Logowanie błędów wywołań systemowych lub programu do logu systemowego.

n) Są osobne funkcje do odczytywania tylko plików zwykłych z katalogu jak i do czytania zarówno plików jak i katalogów ale program nie obsługuje synchronizacji podkatalogów. Przy otrzymaniu opcji -R w trakcie uruchamiania informuje o tym i kończy działanie.

o) Funkcja **isDir** odpowiada za sprawdzenie, czy podany ciąg znaków jest katalogiem, funkcje **PlikiIKatalogi** i **tylkoPlikiZwykle** służą jako filtry akceptowania lub odrzucania wpisów przy czytaniu zawartości katalogu przez scandir, funkcje **podajIloscWpisowWKatalogu** i **PodajIloscZwyklychPlikowWKatalogu** zwracają wartości określone ich nazwą. Same wpisy zwracane są odpowiednio przez funkcje **PobierzWpisyZKatalogu** i **PobierzTylkoPlikiZwykleZKatalogu**.

2. Sposób uruchamiania projektu:

a) Do programu dołączony jest **makefile**, którego można użyć w następujący sposób:

- „make” – w celu skompilowania pliku obiektu „monitoruj.o” i pliku wykonywalnego „demon”.

- „make clean” – w celu usunięcia ww. plików.

b) Plik wykonywalny demon można uruchamiać z natępującymi opcjami:

- „-s lub - -source” – jako argument podając katalog źródłowy (wymagana opcja i argument)

- „-t lub - - target” – jako argument podając katalog docelowy, gdzie stworzy się kopia źródłowego (wymagana opcja i argument)

- „-e lub - - sleeptime” – jako argument podając czas spania demona [s] (opcjonalna opcja, wymagany argument)

- „-p lub - - prog” – jako argument podając wielkość [MB] dzielącą rozróżnianie plików na małe i duże (opcjonalna opcja, wymagany argument)

- „-R” – niezaimplementowany tryb synchronizacji podkatalogów podczas skanowania (opcjonalna opcja).

c) Do programu dołączony jest również skrypt **stworz_pliki.sh**, który tworzy duże pliki w katalogu „~/KatalogDoMonitorowania”.

d) W celu użycia skryptu zalecane jest uruchomienie programu z katalogiem źródłowym (opcją -s) jako „~/KatalogDoMonitorowania”, gdzie oprócz plików generownych przez skrypt jest jeszcze kilka mniejszych plików i katalogi.

e) Nie należy używać niezaimplementowanej opcji -R, gdyż zakończy to działanie programu.

Przykładowy sposób uruchomienia demona: „**./demon -s ~/KatalogDoMonitorowania -t ~/Pulpit -e 600 -p 100**”.

3. Sposób testowania projektu:

Projekt testować można pod dwoma kątami:

a) Testowanie czy monitorowanie przebiega poprawnie:

Należy uruchomić demona np. w ww. **przykładowy sposób** i obserwować log systemowy (np. poprzez „tail -f var*/log/syslog”*. Zaobserwować można jakie czynności z plikami podejmuje demon, kiedy usypia, budzi się a kiedy otrzymuje sygnał.

Usunięcie, dodanie lub edytowanie pliku zostanie zaraportowane do logu systemowego niezwłocznie przy najbliższym obudzeniu się (naturalnym lub po otrzymaniu sygnału SIGUSR1). Więc można dodać, usunąć i edytować jakiś plik a następnie obudzić demona poprzez „kill -SIGUSR1 id_demona” i obserwować log. (id_demona jest używane przy każdym wysyłanym przez niego logu, dodatkowo na samym początku działania wypisuje do logu swoje ID)

b) Testowanie wpływu metody kopiowania plików na szybkość kopiowania:

Do tej analizy niezbędne jest posiadanie dużych plików w katalogu monitorowanym (np. tych tworzonych przez skrypt **stworz_pliki.sh**.
Program należy uruchamiać na dwa sposoby. Za każdym razem można użyć ww. **przykładowego sposobu**, jednak należy dobrać następujące argumenty dla progu dzielącego małe i duże pliki:

- „-p 10” – próg dużego pliku zostanie ustawiony na 10MB.

- „-p 1100” – próg dużego pliku zostanie ustawiony na 1,1GB.

W przypadku progu dużego pliku ustawionego na 10MB wszystkie duże pliki z katalogu KatalogDoMonitorowania (10MiB, 20MiB, 30MiB, 50MiB, 100MiB, 500MiB i 1000MiB) będą kopiowane przy użyciu sendfile, natomiast przy ustawieniu progu 1100 wszystkie te pliki znajdą się poniżej i zostaną przekopiowane przy użyciu niskopoziomowych read/write.

Można uruchomić te dwa sposoby po kilka razy i porównać czasy przekopiowania plików (czas kopiowania można określić odczytując różnicę czasu rozpoczęcia i czasu zakończenia kopiowania pliku z logu systemowego).

Wyniki przykładowego testu (pomiar oznacza pomiar czasu kopiowania):

Wielkość pliku [MiB]	Metoda kopiowania	Pomiar 1 [s]	Pomiar 2 [s]	Sredni czas [s]
1000	**sendfile**	**30**	**42**	**36**
1000	read/write	36	44	40
500	**sendfile**	**20**	**19**	**19.5**
500	read/write	27	25	26
100	**sendfile**	**1**	**2**	**1.5**
100	read/write	1	8	4.5
50	**sendfile**	**<0.5**	**<0.5**	**<0.5**
50	read/write	<0.5	<0.5	<0.5
20	**sendfile**	**<0.5**	**<0.5**	**<0.5**
20	read/write	<0.5	<0.5	<0.5
10	**sendfile**	**<0.5**	**<0.5**	**<0.5**
10	read/write	<0.5	<0.5	<0.5

Dodatkowy test:

Wielkość pliku [MiB]	Metoda kopiowania	Pomiar [s]
2000	**sendfile**	**104**
2000	read/write	115
1000	**sendfile**	**48**
1000	read/write	58

Średni czas sendfile dla 1000 MiB : 42s

Średni czas read/write dla 1000 MiB : 46s

Porównując średnie czasy sendfile jest szybszy:
- O ~10% dla plików 2000MiB

- O ~9% dla plików 1000MiB.

- O 25% dla plików 500MiB.

- O ~66% dla plików 100MiB.

Wyniki mogą sugerować, że im większy plik tym sendfile radzi sobie lepiej na tle read/write, natomiast przy bardzo niskich wagach różnica jest niezauważalna. Próby dla plików rzędu 100MiB-500MiB wydają się być mało miarodajne, a granica zauważalnej przewagi sendfile zdaje się przebiegać między 500MiB a 1000MiB.

4. Szczegółowy opis funkcji int skopiujPlikNiskopoziomowo(int fd_zrodlo, int fd_cel) :

Funkcja otrzymuje w wywołaniu deskryptor pliku z którego ma odczytywać dane (**fd_zrodlo**) i deskryptor pliku, do którego ma dane zapisać (f**d_cel**). Deskryptory są otwierane przed wywołaniem tej funkcji, więc z założenia są już otwarte.

Na początku funkcja loguje informację o tym, że zostało wywołane kopiowanie niskopoziomowe (gdyż w programie istnieje jeszcze dedykowane rozwiązanie **sendfile)**. Log wysyłany jest z priorytetem **LOG_INFO**.

Do odczytania danych z **fd_zrodlo** i zapisania do **fd_cel** posłuży **bufor** (tablica znaków o wielkości 8kiB) i dwie zmienne typu **ssize_t** (**bajty_przeczytane** i **bajty_zapisane**) które przechowają informację o ilości odczytanych i zapisanych danych.

Funkcja **read** odczytuje dane z **fd_zrodlo** zaczynając od początku pliku i zapisuje je do **bufora** w ilości 8kiB na jedną iterację. Po przesłaniu danych **read** przypisuje do zmiennej **bajty_przeczytane** ilość bajtów, jaką przeczytało i zapisało w **buforze**. Jeśli ilość zwrócona jest większa od 0 oznacza to, że jeszcze nie napotkała końca pliku. Po zakończonym czytaniu funkcja **write** pobiera dane z **bufora** i zapisuje je do **fd_cel** w ilości takiej, jaką przeczytała ostatnio funkcja **read**.

Po odczycie i zapisie sprawdzana jest ilość **bajtów_przeczytanych** i **bajtów_zapisanych**. Jeśli ilości te nie są takie same lub któraś z funkcji zwróciła wartość **-1** oznacza to, że nastąpił błąd, co jest logowane z priorytetem **LOG_NOTICE**.

Cały kod źródłowy opatrzony jest w liczne komentarze wyjaśniające działanie programu.