Systemy Operacyjne 1
Laboratorium 9
„Operacje na plikach, zajmowanie plików”
(dwa tygodnie)

dr inż. Arkadiusz Chrobot
dr inż. Grzegorz Łukawski

23 grudnia 2022

Wstęp

W tej instrukcji zawarte są informacje na temat niskopoziomowej obsługi plików, która stosowana jest przez programy współbieżne wykonujące operacje na tych samych plikach. Opisano także w niej zagadnienie odwzorowywania plików w pamięci procesu. Rozdział 1 zawiera wprowadzenie do wymienionych wyżej zagadnień. Rozdział 2 jest opisem API używanego w niskopoziomowej i współbieżnej obsłudze plików w systemie Linux. Rozdział 3 zawiera kod źródłowy oraz opis programu stosującego odwzorowanie pliku w pamięci operacyjnej. Instrukcję kończy zestaw zadań do samodzielnego wykonania na zajęciach laboratoryjnych.

1. Niskopoziomowa obsługa plików, dostęp współbieżny do plików

Programista piszący programy dla systemu Linux ma do wyboru dwa rodzaje obsługi plików. Może się posługiwać niskopoziomowymi funkcjami dostarczanymi przez system, lub podprogramami, które oferują biblioteki wejścia-wyjścia większości języków programowania. Pierwsze rozwiązanie pozwala na bezpośredni dostęp do plików (tzn. niebuforowany) przy użyciu deskryptorów plików. Pomimo pozornie małej funkcjonalności, to właśnie ten sposób wykonywania operacji wejścia-wyjścia jest zalecany jeśli z plikiem pracuje kilka współbieżnych procesów lub wątków. Posługiwanie się funkcjami systemowymi nie stanowi jednak samo w sobie rozwiązania problemu współbieżnego dostępu do plików. Istnieją pomocnicze mechanizmy nazywane blokadami pliku (ang. *file lock*), które umożliwiają określonemu procesowi lub wątkowi wykonywanie modyfikacji całości lub części pliku na zasadzie wyłączności lub ochronę pliku odczytywanego współbieżnie przez kilka procesów bądź wątków przed zapisem. Operację zakładania blokady na plik lub jego fragment nazywa się zajmowaniem pliku. Są dwa rodzaje tych operacji: zajmowanie obowiązkowe (ang. *mandatory*) i doradcze (ang. *advisory*). Pierwszy rodzaj został wprowadzony do wersji System V Uniksa i jest włączony do standardu POSIX. Drugi pojawił się w wersji BSD Uniksa. Zajmowanie obowiązkowe pliku (zwane także narzuconym), wymusza na systemie operacyjnym kontrolowanie, czy procesy współbieżne korzystające z pliku nie naruszają nałożonych na niego blokad. Blokady w tym przypadku są związane z plikiem, tzn. po zakończeniu procesu nie są usuwane. Zajmowanie doradcze (zwane także zalecanym) nie wymusza sprawdzania poprawności operacji na pliku przez jądro systemu. O to muszą zadbać procesy odwołujące się do pliku. Blokady te związane są z procesem[1], a więc po jego zakończeniu są usuwane. Linux pozwala na stosowanie obu rodzajów zajmowania, aczkolwiek domyślnie stosowane jest zajmowanie doradcze. Aby móc posługiwać się zajmowaniem obowiązkowym należy zamontować system plików z odpowiednimi flagami i odpowiednio skonfigurować prawa dostępu do plików. Istnieją dwa rodzaje blokad: dzielone (inaczej: do czytania lub współbieżna), które pozwalają kilku procesom czytać współbieżnie z pliku (całego lub części), ale zakazują zapisywania oraz blokady wyłączne (inaczej: do zapisu), które pozwalają tylko jednemu procesowi na zapis do pliku (całego lub części). Proces będący właścicielem blokady może zmienić jej typ. Jeżeli na pliku lub jego obszarze istnieje blokada współbieżna, to inny proces na ten sam plik lub obszar może nałożyć swoją blokadę do czytania, ale nie może nałożyć blokady do zapisu. Jeśli na pliku lub obszarze pliku istnieje blokada wyłączna to żaden proces nie może nałożyć na ten plik (obszar) żadnej innej blokady. Systemy uniksowe, w tym system Linux, pozwalają również odwzorować część lub całość pliku w pamięci operacyjnej, co przyspiesza dostęp do jego zawartości.

2. API niskopoziomowej obsługi plików

Z niskopoziomową obsługą plików, mechanizmem blokad oraz odwzorowywaniem plików w pamięci, związane są następujące funkcje:

`open()` - funkcja ta otwiera plik. Jako argumenty pobiera ścieżkę dostępu do pliku i flagi określające sposób pracy z plikiem. Może ona również pobierać prawa dostępu, jeśli otwierany plik nie istnieje i ma być utworzony. Wartością zwracaną przez funkcję w przypadku powodzenia jest deskryptor pliku (liczba typu `int`), a w przypadku niepowodzenia liczba `-1`.
Szczegóły: `man 2 open`

[1] Dokładniej z relacją proces-plik. W przypadku Linuksa są one związane z obiektem pliku.

`creat()` - funkcja ta tworzy nowy plik i otwiera go do zapisu, czyli dubluje częściowo działanie `open()`. Jako argumenty przyjmuje ona ścieżkę dostępu do pliku oraz liczbę ósemkową określającą prawa dostępu. Zwraca deskryptor utworzonego pliku lub wartość `-1`, w przypadku niepowodzenia.
Szczegóły: `man creat`

`dup()` - funkcja ta tworzy i zwraca kopię deskryptora pliku, który został jej przekazany jako argument wywołania lub wartość `-1` w przypadku niepowodzenia. Nowy deskryptor nie współdzieli ze starym jedynie flagi `close-on-exit`.
Szczegóły: `man dup`

`dup2()` - funkcja ta przyjmuje dwa argumenty wywołania - nowy deskryptor pliku i stary deskryptor. Jeśli nowy deskryptor jest związany z plikiem, to funkcja najpierw go zamyka, a następnie sprawia, że staje się on kopią starego deskryptora pliku. Nowy deskryptor jest także przez nią zwracany w przypadku prawidłowego jej zakończenia, lub zwracana jest wartość `-1` w przypadku niepowodzenia. Tak, jak w przypadku `dup()` nowy deskryptor nie dzieli ze starym jedynie flagi `close-on-exit`.
Szczegóły: `man dup2`

`read()` - funkcja ta odczytuje z pliku określoną ilość informacji. Jako argumenty przyjmuje ona deskryptor pliku, adres zmiennej, w której ma umieścić odczytane informacje, oraz ilość informacji, jaką ma przeczytać, wyrażoną w bajtach. Wartością zwracaną przez funkcję jest natomiast liczba przeczytanych bajtów lub `-1` w przypadku wystąpienia wyjątku. Funkcja `read()` może zostać zastosowana również do odczytu standardowego wejścia, którego deskryptor wynosi zero.
Szczegóły: `man 2 read`

`write()` - ta funkcja zapisuje dane do pliku. Przyjmuje ona takie same argumenty jak `read()` i może być także użyta do zapisu do standardowego wyjścia (deskryptor `1`) lub standardowego wyjścia diagnostycznego (deskryptor `2`). Wartością zwracaną przez funkcję jest liczbę zapisanych bajtów lub `-1` w przypadku wyjątku.
Szczegóły: `man 2 write`

`close()` - ta funkcja zamyka plik związany z przekazanym jej jako argument deskryptorem. W razie niepowodzenia zwraca wartość `-1`, a w przeciwnym przypadku wartość `0`.
Szczegóły: `man close`

`lseek()` - ta funkcja zmienia wartość wskaźnika pliku. Przyjmuje ona trzy argumenty wywołania: deskryptor pliku, liczbę wyrażającą wartość przesunięcia wskaźnika oraz jedną z trzech stałych, które określają względem czego to przesunięcie ma być liczone:

`SEEK_SET` - przesunięcie jest liczone względem początku pliku,

`SEEK_CUR` - przesunięcie jest liczone względem bieżącego położenia wskaźnika pliku,

`SEEK_END` - przesunięcie jest liczone względem końca pliku.

Funkcja zwraca wartość o jaką został zmieniony wskaźnik pliku względem jego początku lub `-1` w przypadku niepowodzenia.
Szczegóły: `man lseek`

`flock()` - funkcja ta zakłada lub usuwa blokadę z pliku. Jest związana z doradczym zajmowaniem. Blokadę przy użyciu tej funkcji można nałożyć tylko i wyłącznie na całość pliku. Funkcja przyjmuje jako argumenty wywołania deskryptor pliku i jedną ze stałych określających rodzaj operacji:

`LOCK_SH` - założenie blokady dzielonej,

`LOCK_EX` - założenie blokady wyłącznej,

`LOCK_UN` - zdjęcie blokady z pliku.

W przypadku powodzenia funkcja zwraca wartość `0`, a w przypadku niepowodzenia `-1`.
Szczegóły: `man 2 flock`

`fcntl()` - działanie tej funkcji polega ogólnie na manipulowaniu deskryptorem pliku. W szczególności pozwala ona zakładać zarówno blokady związane z zajmowaniem obowiązkowym, jaki i doradczym. Domyślnie stosowane jest to pierwsze, ale wymaga ono wykonania montowania systemu plików z odpowiednimi opcjami. Większość systemów uniksowych (w tym Linux) nie włącza tych opcji, zatem `fcntl()` zakłada w ich systemach plików blokady doradcze. Pozwala ona również na zajmowanie ściśle określonych obszarów plików lub całych plików. W przypadku zakładania/zdejmowania blokad ta funkcja przyjmuje trzy argumenty wywołana: deskryptor pliku, jedną z trzech stałych: `F_GETLK`, `F_SETLK`, `F_SETLKW` oraz wskaźnik na strukturę typu `struct flock`. Funkcja zwraca `0` w przypadku powodzenia lub `-1` w przypadku niepowodzenia.
Szczegóły: `man fcntl`

`fsync()` - ta funkcja szereguje bufory ze zmodyfikowanymi danymi oraz metadanymi pliku i zapisuje je na nośnik. Jako argument wywołania przyjmuje deskryptor pliku. Zwraca `0` w przypadku powodzenia lub `-1` w przeciwnym przypadku.
Szczegóły: `man fsync`

`fdatasync()` - ta funkcja przyjmuje takie same argumenty wywołania i działa podobnie jak `fsync()`, ale ograniczona zapis na nośnik wyłącznie dla zmodyfikowanych danych.
Szczegóły: `man fdatasync`

`mmap()` - ta funkcja pozwala odwzorować część lub całość pliku w pamięci operacyjnej. Dzięki temu dostęp do jego zawartości jest szybszy i może być wykonywany w ten sam sposób jak np. dostęp do elementów zwykłej tablicy. Funkcja ta przyjmuje przyjmuje sześć argumentów. Pierwszy to adres początku obszaru pamięci, w którym ma być odwzorowany plik. Ten argument może mieć wartość `NULL`. Drugi to rozmiar odwzorowywanego fragmentu lub całości pliku wyrażony w bajtach. Trzeci argument opisuje sposób ochrony obszaru pamięci, w którym odwzorowana jest treść pliku. Może on przyjmować wartość stałej `PROT_NONE` lub być sumą logiczną następujących stałych: `PROT_EXEC` - zawartość obszaru może być wykonywana, `PROT_READ` - zawartość obszaru może być odczytywana, `PROT_WRITE` - zawartość obszaru może być zapisywana. Czwarty argument to flagi, które mogą przyjąć jedną lub kilka z trzech wartości: `MAP_FIXED` - jeśli pierwszy argument ma wartość różną od `NULL`, to funkcja `mmap()` zwróci błąd jeśli nie uda się jej odwzorować pliku pod tym adresem (musi on być podzielny przez rozmiar strony), `MAP_SHARED` - odwzorowanie może być współdzielone z innymi procesami, `MAP_PRIVATE` - odwzorowanie będzie prywatne, typu „kopiuj przy zapisie”. Piąty argument to deskryptor odwzorowywanego pliku, a szósty to przesunięcie względem początku pliku, od którego jego treść będzie odwzorowana w pamięci operacyjnej. Funkcja zwraca adres pod którym treść pliku jest odwzorowana w pamięci. W przypadku wystąpienia wyjątku ten adres ma wartość `-1`.
Szczegóły: `man mmap`

`msync()` - ta funkcja synchronizuje zawartość odwzorowania treści pliku w pamięci z jego obrazem na nośniku. Przyjmuje ona trzy argumenty wywołania: adres początku obszaru, który ma być zsynchronizowany, jego rozmiar oraz flagi określające sposób synchronizacji. Mogą one mieć jedną lub kilka z trzech wartości: `MS_ASYNC` - funkcja nie czeka na zakończenie zapisu (tj. synchronizacji), `MS_SYNC` - funkcja czeka na zakończenie zapisu (te dwie flagi wzajemnie się wykluczają), `MS_INVALIDATE` - sugeruje unieważnienie zawartości innych odwzorowań. Funkcja zwraca `-1` w razie wystąpienia wyjątku.
Szczegóły: `man msync`

`munmap()` - ta funkcja usuwa odwzorowanie pliku z pamięci operacyjnej, powodując także jego synchronizację z obrazem pliku na nośniku. Przyjmuje ona dwa argumenty, adres początkowy odwzorowania i jego rozmiar wyrażony w bajtach. Zwraca `-1` jeśli wystąpił wyjątek.
Szczegóły: `man munmap`

`fstat()` - ta funkcja pozwala uzyskać dane o pliku. Przyjmuje ona dwa argumenty, deskryptor pliku oraz wskaźnik na zmienną typu `struct stat`, w której zapisuje pozyskane informacje. Zwraca wartość `-1` w przypadku wystąpienia wyjątku.
Szczegóły: `man fstat`

`sendfile()` - ta funkcja kopiuje całość lub fragment pliku do innego pliku z użyciem buforów znajdujących się w przestrzeni adresowej jądra systemu. W Linuksie przyjmuje cztery argumenty. Pierwszy jest deskryptorem pliku docelowego, a drugi źródłowego. Trzeci argument jest wskaźnikiem na zmienną, która zawiera przesunięcie, od którego treść pliku źródłowego będzie kopiowana. Po zakończeniu kopiowania ta zmienna będzie zawierała przesunięcie bajtu znajdującego się tuż za ostatnim skopiowanym bajtem. Jeśli wartość tego argumentu jest różna od `NULL`, to wskaźnik pliku nie jest automatycznie modyfikowany, w przeciwnym przypadku jest. Ostatni argument to liczba bajtów, które mają być przekopiowane. Funkcja zwraca liczbę przekopiowanych bajtów, lub `-1`, jeśli pojawi się wyjątek.
Szczegóły: `man sendfile`

3. Przykład

Listing 1 zawiera kod źródłowy programu, który odwzorowuje część pliku tekstowego w pamięci operacyjnej i wypisuje ją na ekran.

```
1  #include<sys/mman.h>
2  #include<sys/types.h>
3  #include<sys/stat.h>
4  #include<fcntl.h>
5  #include<stdio.h>
6  #include<unistd.h>
7
8  void print_file(char *string)
9  {
10         int i;
11         for(i=0;i<getpagesize();i++)
12                 printf("%c",string[i]);
13         puts("");
14 }
15
16 int main(void)
17 {
18         int descriptor = open("1464.txt.utf-8",O_RDWR,0600);
19         if(descriptor==-1) {
20                 perror("open");
21                 return descriptor;
22         }
23
24         char *string = mmap(NULL,getpagesize(),PROT_READ,MAP_PRIVATE,descriptor,0);
25
26         if(string==(char *)-1) {
27                 perror("mmap");
28                 return -1;
29         }
30
31         print_file(string);
32
33         if(munmap(string,getpagesize())==-1)
34                 perror("munmap");
35
36         if(close(descriptor)==-1)
37                 perror("close");
38
39         return 0;
40 }
```

Listing 1: Przykładowy program odwzorowujący fragment pliku w pamięci operacyjnej.

W krótkim programie z listingu 1 otwierany jest plik do odczytu i zapisu (wiersz nr 18), a następnie jego początkowy fragment o wielkości równej wielkości strony jest odwzorowywany w pamięci operacyj-

nej (wiersz nr 24). Po każdej z powyższych operacji sprawdzane jest, czy nie wystąpił wyjątek. Rozmiar strony pamięci jest ustalany za pomocą wywołania funkcji `getpagesize()`, która go zwraca. Po odwzorowaniu fragmentu pliku w pamięci jest on wypisywany na ekran za pomocą funkcji `print_file()`. Proszę zwrócić uwagę, że ta funkcja odwołuje się do pliku tak, jak do zwykłej tablicy. W wierszu 33 odwzorowanie pliku jest likwidowane, a w wierszu 36 plik jest zamykany.

Zadania

UWAGA: PROGRAMY MUSZĄ BYĆ NAPISANE Z PODZIAŁEM NA FUNKCJE Z PARAMETRAMI ORAZ MUSZĄ SPRAWDZAĆ, CZY WYWOŁYWANE PRZEZ NIE FUNKCJE Z API SYSTEMU OPERACYJNEGO NIE SYGNALIZUJĄ WYJĄTKÓW.

Uwaga: Jeśli do wykonania zadania potrzebne będą gotowe pliki tekstowe, to można je pobrać np. ze strony Projektu Gutenberg: `http://www.gutenberg.org/`

1. Napisz program, który przepisze z istniejącego pliku tekstowego czterdzieści znaków z początku i czterdzieści znaków z końca do nowego pliku.
2. Stwórz plik tekstowy wielkości 512 bajtów, z „dziurą” w środku.
3. Zademonstruj działanie funkcji replikującej deskryptory plików.
4. Zademonstruj działanie funkcji `flock()` na przykładzie, w którym kilka procesów współbieżnych będzie się ubiegało o możliwość zapisu lub odczytu z pliku.
5. Powtórz zadanie czwarte używając `fcntl()` i blokując wybrane fragmenty pliku.
6. Napisz program, który przeczyta i wypisze na ekran treść pliku tekstowego o dowolnej wielkości. Nazwę pliku należy przekazywać jako argument wywołania programu.
7. Napisz odpowiednik programu „cp” do kopiowania plików, bez obsługi katalogów i opcji wykonania.
8. Napisz program, w którym dwa procesy będą zapisywały do jednego pliku, a trzy będą odczytywały. Zastosuj zajmowanie zalecane do synchronizacji pracy procesów.
9. Napisz program, który otworzy do odczytu wskazany plik tekstowy, a następnie wywoła funkcję `fork()`. Proces potomny policzy sumę liczb parzystych znalezionych w otwartym pliku, a macierzysty liczb nieparzystych.
10. Napisz dwa programy: pierwszy będzie generował losowe liczby i zapisywał je porcjami do pliku, a drugi działając współbieżnie z pierwszym będzie je odczytywał i szukał wartości maksymalnej. Do blokowania fragmentów pliku użyj funkcji `fcntl()`.
11. Napisz program, w którym stworzysz cztery procesy. Każdy z nich uzyska kopię deskryptora do pojedynczego pliku, przy czym jeden proces będzie do tego pliku pisał, a pozostałe będą czytały.
12. Napisz program, który będzie zapisywał do pliku informacje tekstowe porcjami po 4 KiB w trzech trybach: z wykorzystaniem funkcji `fsync()`, z użyciem `fdatasync()` i bez stosowania żadnej z tych funkcji. Dokonaj pomiaru czasu zapisu dla każdego z tych przypadków. Do wyznaczenia czasu zapisu użyj funkcji `clock_gettime()` (Szczegóły: `man 2 clock_gettime`).
13. Napisz program, który skopiuje plik w całości używając funkcji `sendfile()`.
14. Napisz program, który odczyta i wyświetli na ekranie całą zawartość pliku tekstowego, korzystając z funkcji `mmap()`.
15. Stwórz dwa procesy niespokrewnione (osobne programy), które będą odwzorowywały ten sam plik tekstowy. Jeden będzie co sekundę odczytywał jego zawartość, a drugi będzie go modyfikował. Pamiętaj, że aby operacja zmiany odwzorowanego pliku się powiodła, to nie wolno do tego pliku dopisywać informacji, można jedynie zmieniać istniejące.
16. Napisz program, który odczyta informacje o wybranym pliku za pomocą funkcji `fstat()`.