# Systemy operacyjne

Projekt programistyczny nr 3

„System plików"

Termin oddawania: 31 stycznia 2022

**Wskazówka:** Programowanie na ostatnią chwilę jest **bardzo złym** pomysłem. Daj sobie trochę czasu na przemyślenie koncepcji!

## Wprowadzenie

Twoim zadaniem jest zaimplementowanie zbioru operacji w pliku `ext2fs.c` na dyskowej strukturze danych systemu plików `ext2`. Dzięki temu otrzymasz funkcjonujące narzędzia:

- «`ext2test`» służące do interaktywnego przeglądania zawartości systemu plików,
- «`ext2list`» drukujące zawartość wszystkich plików w systemie plików.

... z których będzie korzystać skrypt oceniający poprawność Twojego rozwiązania.

**Wskazówka:** Przed przystąpieniem do pracy należy zapoznać się z The Second Extended File System Internal Layout[1].

## Zamieszczanie rozwiązań

Rozwiązania będą zbierane i oceniane przy pomocy systemu GitHub Classroom. Rozwiązania nadsyłane innymi środkami, np. pocztą elektroniczną, nie będą sprawdzane! Prowadzący dostarczy odnośnik do wygenerowania prywatnej wersji repozytorium, w którym należy umieścić rozwiązanie przed upływem terminu oddawania. Rozwiązanie należy umieścić w gałęzi «`master`» swojego repozytorium. W gałęzi «`feedback`» musi się znajdować **niezmodyfikowana** wersja gałęzi «`master`» głównego repozytorium.

Autor zadania zastrzega, że procedura testowania oraz pliki `ext2list.c` i `ext2test.c` mogą ulec zmianie. W związku z tym może zajść potrzeba zsynchronizowania repozytorium rozwiązania studenta z repozytorium głównym zadania. W katalogu swojego repozytorium należy wykonać poniższy szereg poleceń, będąc przygotowanym na rozwiązywanie konfliktów:

```
$ git checkout feedback
$ git pull https://github.com/ii-ask/so-ext2.git
  «tu należy rozwiązać potencjalne konflikty»
$ git push
$ git checkout master
$ git merge feedback
$ git push
```

Prowadzący będzie oceniał zmiany plików w prośbie o połączenie (ang. *pull request*) o nazwie «`Feedback`». Inne prośby będą przez sprawdzającego ignorowane. W zakładce „*Files changed*" mają być widać **wyłącznie** zmiany, które są częścią rozwiązania i zostały przygotowane przez studenta. Oznacza to, że należy się wystrzegać modyfikacji kodu, który został dostarczony.

Zadaniem studenta jest uzupełnienie brakujących fragmentów programu zawartych w wierszach począwszy od dyrektywy «`#ifdef STUDENT`» do najbliższego wystąpienia dyrektywy «`#endif /* !STUDENT */`». Rozwiązanie, w którym zmodyfikowano fragmenty kodu leżące poza wyznaczonymi miejscami, zostanie automatycznie **odrzucone** przez sprawdzarkę!

[1] `https://www.nongnu.org/ext2-doc/ext2.html`

## Samodzielność rozwiązania

Nie można korzystać z żadnych innych źródeł bez zgody wykładowcy. Haniebnym jest by skopiować lub zmodyfikować rozwiązanie studentów z naszej lub innych uczelni, lub wykorzystać inne implementacje dostępne w zasobach sieci Internet, i przedstawić jako własne rozwiązanie.

**UWAGA!** Wykrycie niesamodzielnej pracy będzie skutkować natychmiastowym wydaleniem studenta z kursu, zgłoszeniem przypadku do komisji antyplagiatowej i rozmową z dziekanem dotyczącą kodeksu studiowania.

## Specyfikacja

Dostarczony szkielet robi kilka założeń co do konfiguracji systemu plików `ext2`. Twój kod musi obsługiwać wyłącznie dyskowe struktury danych z pierwszej rewizji systemu `ext2`. Bloki systemu plików mają zawsze rozmiar 1024 bajtów.

Poniżej podano zestaw funkcji będących punktami wejścia w kod systemu plików. Są one odpowiednikami podzbioru operacji oferowanych przez `vnodeops(9)` i `vfsops(9)`, które służą do implementacji pewnych usług jądra oraz wywołań systemowych operujących na plikach.

**`ext2_mount`** Tworzy reprezentację w pamięci operacyjnej systemu plików przechowywanego w zadanym pliku. Wczytuje i weryfikuje superblok oraz tablicę deskryptorów grup bloków. Po wykonaniu tej procedury system plików jest gotowy do działania. Służy do implementacji wywołania systemowego `mount(2)`.

**`ext2_lookup`** Służy do wyszukiwania pliku o podanej nazwie w katalogu zadanym numerem i-węzła. Bazuje na tym algorytm `namei(9)` tłumaczenia ścieżek na i-węzły, który z kolei służy do implementacji wywołania systemowego `open(2)` i wielu innych.

**`ext2_read`** Służy do odczytania fragmentu zawartości pliku o zadanym i-węźle. Musi umieć znaleźć bloki o odpowiednich adresach, wczytać je do pamięci, po czym skopiować ich fragmenty do wskazanego bufora. System plików może korzystać z tej procedury do wczytywania zawartości plików specjalnych (katalogi, dowiązania symboliczne). Służy do implementacji wywołania systemowego `read(2)`.

**`ext2_readdir`** Wczytuje z katalogu, zadanego numerem i-węzła, wpis spod określonego offsetu. Funkcja jest wykorzystywana przez «`ext2_lookup`» do przejrzenia katalogu w poszukiwaniu pliku o zadanej nazwie. Służy do implementacji wywołania systemowego `getdirentries(3)`.

**`ext2_readlink`** Przy pomocy tej funkcji można przeczytać zawartość dowiązania symbolicznego wskazanego przez numer i-węzła. Jądro wykorzystuje je w `namei(9)` do rozwiązywania ścieżek. Służy do implementacji wywołania systemowego `readlink(2)`.

**`ext2_stat`** Tłumaczy metadane zawarte we wskazanym i-węźle na uniwersalny format `vattr(9)` zrozumiały przez podsystem *Virtual File System*. Służy do implementacji wywołania systemowego `stat(2)`, które wypełnia strukturę «`stat`».

Poniżej podano funkcje, które są częścią niskopoziomowego interfejsu dostępu do dyskowych struktur danych systemu plików `ext2`. Powyższe funkcje będą korzystać z tej abstrakcji, żeby ukryć złożoność pewnych akcji.

**`ext2_block_used`** Sprawdza czy blok o podanym adresie jest używany. Musi wyznaczyć grupę bloków do której należy wskazany blok, a następnie pobrać z bufora blok z bitmapą bloków i sprawdzić odpowiedni bit.

**Pamiętaj:** Bloki przechowują wszystkie metadane systemu plików oraz dane plików.

**`ext2_inode_used`** j.w. dla wskazanego numeru i-węzła.

**`ext2_inode_read`** Sprawdza czy dany i-węzeł jest zajęty, wyznacza odpowiednią grupę bloków, pobiera z bufora blok przechowujący dany i-węzeł, a następnie kopiuje i-węzeł we wskazane miejsce.

**Pamiętaj:** Nie wolno Ci czytać i-węzłów, które nie są używane!

**ext2_blkptr_read** Pobiera z bufora blok przechowujący adresy bloków, a następnie wybiera z niego wartość wskaźnika o zadanym indeksie.

**Pamiętaj:** Nie wolno Ci czytać bloków, które nie są używane!

**ext2_blkaddr_read** Jest to najbardziej złożona funkcja w tym zbiorze. Tłumaczy adresy bloków pliku (tj. adresy logiczne) na adresy bloków na dysku (tj. adresy fizyczne). W zależności od rozmiaru pliku oraz numeru bloku pliku będzie trzeba odnaleźć odpowiednik wskaźnik na blok dysku. W tym celu będziesz korzystać ze wskaźników przechowywanych w i-węźle i potencjalnie w blokach pośrednich.

System plików komunikuje się z dyskiem wczytując i zapisując całe bloki. Tj. żeby zmodyfikować jeden bajt w bloku należy: wczytać blok do bufora w pamięci operacyjnej, zmodyfikować pożądany bajt w buforze, a następnie zapisać bufor na dysk. Żeby symulować dostęp do dysku dostarczono interfejs bufora bloków dyskowych, który jest najbardziej niskopoziomowym interfejsem dostępu do danych systemu plików. Każdy bufor przechowuje 1024 bajty. Liczba buforów jest z góry zadana przy pomocy stałej «NBLOCKS».

**blk_alloc** Przydziela blok z listy pustych bloków. Jeśli nie ma wolnych pustych bloków, to zwraca jakiś wypełniony blok, który będzie musiał być wyczyszczony. Wymagane do działania «blk_get».

**blk_get** Pobiera blok z bufora. Może albo pobrać blok przechowujący metadane systemu plików, wtedy wskazany indeks jest adresem fizycznym) albo określony blok pliku, gdzie indeks to adres logiczny.

**Pamiętaj:** Nieużywany blok należy zwrócić do bufora!

**blk_put** Zwraca blok do bufora. Należy wywołać tę funkcję na bloku, którego nie będziemy już przetwarzać w bieżącej operacji.

## Sprawdzanie poprawności

Program «ext2list», korzystający z kodu dostarczonego przez studenta, zostanie uruchomiony na dostarczonym obrazie «debian9-ext2.img». Program wydrukuje listę wszystkich plików w systemie plików wraz z ich właściwościami, dodatkowo dla plików zwykłych wyliczy sumę kontrolną. Wydruk zostanie porównany z wydrukiem referencyjnym, a różnice zostaną wydrukowane przy pomocy narzędzia diff(1).

```
# plik zwykły
path=./sbin/e2fsck ino=110167 mode=100755 nlink=1 uid=0 gid=0 size=344824
 atime=1515801802 mtime=1485910495 ctime=1515801746
 md5=f71589ce5a79ca0736f91d0ac54a4a27

# katalog
path=./var/lib ino=40802 mode=40755 nlink=12 uid=0 gid=0 size=1024
 atime=1515803007 mtime=1515802979 ctime=1515802979

# dowiązanie symboliczne
path=./usr/bin/vi ino=52304 mode=120777 nlink=1 uid=0 gid=0 size=20
 atime=1515802060 mtime=1515801763 ctime=1515801763
 target=/etc/alternatives/vi

# plik urządzenia
path=./dev/tty ino=49777 mode=20666 nlink=1 uid=0 gid=0 size=0
 atime=1515801737 mtime=1515801737 ctime=1515801737
```

## Wskazówki od wykładowcy

Zasadniczo dyskowa struktura danych systemu plików `ext2` jest łatwa do zrozumienia. Jest jednak kilka założeń i przypadków brzegowych, które są nieoczywiste dla studentów, więc powtórzmy je poniżej:

- Adresy bloków oraz numery i-węzłów zaczynają się od jedynki. Wartość zerowa ma specjalne znaczenie!
- Jedną z metod kasowania plików w katalogów jest ustawienie we wpisie katalogu (ang. *directory entry*) numeru i-węzła na zero. Taki nieużytek należy zwyczajnie pominąć.
- Zawartość krótkich dowiązań symbolicznych jest przechowywana bezpośrednio w i-węźle w miejscu na wskaźniki na bloki. Dla dłuższych dowiązań symbolicznych zostaje przydzielony blok danych.
- Dziury w plikach wyrażane są poprzez adresy bloków równe zeru.

## Ocena

W komentarzu w nagłówku pliku «`ext2fs.c`» należy wpisać swoje imię, nazwisko oraz numer indeksu, oświadczając, że jest się jedynym autorem kodu źródłowego. Po wypchnięciu zmian do repozytorium zostaną automatycznie uruchomione testy poprawności rozwiązania przy użyciu GitHub Actions. Rozwiązania niepoprawnie sformatowane lub generujące błędy albo ostrzeżenia w trakcie kompilacji będą odrzucane. Formatowanie kodu będzie sprawdzane programem «`clang-format`».