

Systemy operacyjne

Sprawozdanie - laboratorium 3 „Zaawansowane operacje na plikach”

Andrzej Kołakowski
296586

1) Pozyskiwanie i wyświetlanie metadanych pliku

1. Odpowiedzi na pytania

- Czym różnią się funkcje z rodziny `stat(2)`?

`stat()` - zwraca informacje o pliku wskazywanym przez ścieżkę.

`lstat()` - tak jak `stat()`, z wyjątkiem sytuacji gdy ścieżka jest łączem symbolicznym, wtedy wyświetla informacje o łączu.

`fstat()` - zwraca informacje o pliku na podstawie deskryptora.

- Co reprezentuje flaga `S_IFMT` zdefiniowana dla pola `st_mode` w strukturze `stat`?

Jest maską bitową dla typu pliku. Wykonanie bitowego AND: `sb.st_mode & S_IFMT` oznacza, aby brać pod uwagę tylko bity związane z typem pliku, a pozostałe (związane z uprawnieniami) ignorować.

- Zmienna `sb` jest wypełnioną strukturą typu `struct stat`. Czy można sprawdzić typ pliku (np. czy plik jest urządzeniem blokowym) w następujący sposób?

```
if ((sb.st_mode & S_IFBLK) == S_IFBLK) { /* plik jest urządzeniem
blokowym */ }
```

Nie, należy użyć maski bitowej `S_IFMT`.

Uzasadnienie zaczerpnięte z M.J.Rochkind - Advanced UNIX Programming:

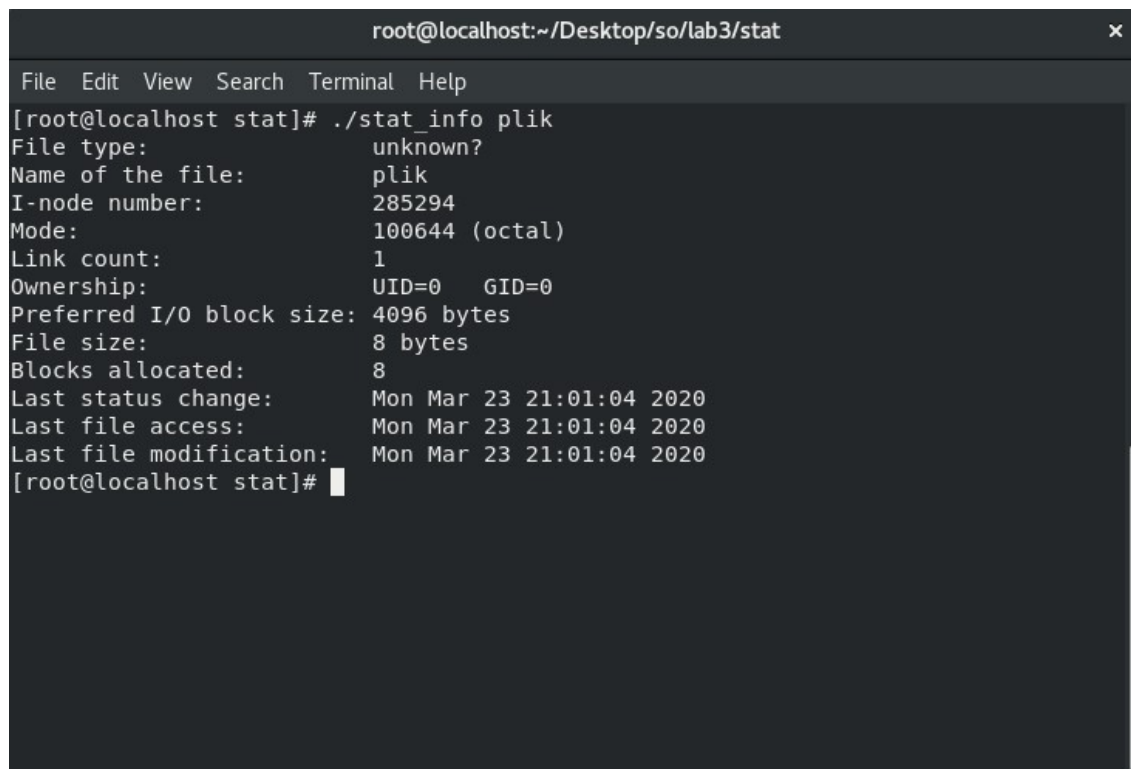
The macro `S_IFMT` defines the bits for the type of file; the others are values of those bits, not bit masks.

2. Program `stat_info.c` - modyfikacja istniejących funkcji

Program `stat_info.c` odczytuje informacje zawarte w `struct stat` dotyczące pliku podanego jako argument i przedstawia je w czytelny, opisowy sposób. Zadanie polegało na przeprowadzeniu szeregu modyfikacji:

- Dodanie do funkcji `print_type` sprawdzenia pozostałych typów pliku
- Wyświetlenie praw dostępu w postaci numerycznej w funkcji `print_perms`
- Poza UID i GID wyświetlenie również nazwy i grupy użytkownika w funkcji `print_owner`
- Wyświetlenie praw dostępu do pliku użytkownika uruchamiającego program w funkcji `print_perms`
- Wyświetlenie rozmiaru pliku w sformatowany sposób w funkcji `print_size`
- Zmodyfikowanie funkcji `print_laststch`, `print_lastacc` i `print_lastmod` tak, aby wyświetlały czas, który minął od daty ostatniej zmiany statusu/dostępu/modyfikacji
- Zmodyfikowanie funkcji `print_name` tak, aby w przypadku linku symbolicznego została wyświetlona zarówno jego nazwa, jak i nazwa pliku na który wskazuje

Początkowe wyjście programu:



```

root@localhost:~/Desktop/so/lab3/stat
File Edit View Search Terminal Help
[root@localhost stat]# ./stat_info plik
File type:                unknown?
Name of the file:         plik
I-node number:            285294
Mode:                     100644 (octal)
Link count:               1
Ownership:                UID=0   GID=0
Preferred I/O block size: 4096 bytes
File size:                8 bytes
Blocks allocated:         8
Last status change:       Mon Mar 23 21:01:04 2020
Last file access:         Mon Mar 23 21:01:04 2020
Last file modification:   Mon Mar 23 21:01:04 2020
[root@localhost stat]#

```

Wyjście po modyfikacjach:

```
root@localhost:~/Desktop/so/lab3/stat
File Edit View Search Terminal Help
[root@localhost stat]# ./stat_info_m plik
File type:          regular file
Name of the file:   plik
I-node number:     285294
Mode:              644 (octal)
You are:           root
Owner is:          root
Your permissions:  read=true write=true execute=false
Link count:        1
Ownership:         root(0)  root(0)
Preferred I/O block size: 4096 bytes
File size:         0.01 KB
Blocks allocated:  8
Last status change: 0 days ago
Last file access:  0 days ago
Last file modification: 0 days ago
[root@localhost stat]#
```

Wszystkie zmiany zostały zawarte w pliku *stat_info.c*

2) Wejście/wyjście asynchroniczne

1. Program z sekcji 1 rozbudowany o funkcję `static void print_content(char *name)`

Powyższa funkcja:

- Pyta użytkownika czy chce wypisać zawartość podanego jako argument pliku.
- Jeżeli tak, to otwiera i przy pomocy funkcji czytania asynchronicznego `aio_read(3)` odczytuje zawartość pliku i wypisuje ją na ekran.

Sposób rozwiązania zaproponowany w instrukcji do laboratorium:

1. Otwieramy plik `open(2)`
2. Inicjalizujemy wczytywanie pierwszej porcji danych `aio_read(3)`
3. Przy pomocy funkcji `aio_error(3)` czekamy do momentu aż odczyt się zakończy
4. Wyświetlamy wczytane dane i wracamy do 2 jeżeli nie osiągnęliśmy końca pliku

W mojej implementacji zamiast `aio_error(3)` wykorzystałem funkcję `aio_suspend(3)`, która zawiesza wątek do czasu wykonania asynchronicznych operacji wejścia/wyjścia. Uważam, że jest to rozwiązanie bardziej eleganckie od porównywania w pętli wartości zwracanej przez `aio_error(3)`.

Rozwiązanie zostało zapisane w pliku *stat_info2.c*

Aby poprawnie skompilować program nie używamy opcji `-Wall` `-pedantic` `-ansi` jednocześnie podając następujące argumenty dla linkera `-lrt`.