# Systemy operacyjne

Laboratorium 2

Mateusz Małek
17 marca 2017

Wersja 0.3

# Laboratorium 2

Funkcje systemowe i biblioteczne do obsługi plików i katalogów

# Wywołania systemowe (syscall) - low level

- Potrzebujemy odwołać się do zasobów (np. dysk, sieć, inne urządzenia)
- Rolą systemu operacyjnego jest kontrolowanie takich operacji
  - Kwestie bezpieczeństwa, “sprawiedliwy” dostęp do zasobów …
- **Konsekwencja (i problem):** nie możemy zatem tego zrobić bezpośrednio
- **Rozwiązanie:** specjalny mechanizm pozwalający programom wykonywanym w trybie użytkownika na żądanie wykonania operacji przez jądro systemu
- Nieco asemblera…
  - (2 semestr) W DOS przerwanie 0x21h, funkcja i argumenty przekazywane przez rejestry
  - (Dawniej) W Linuksie przerwanie 0x80h, funkcja i argumenty przekazywane przez rejestry
  - (Obecnie) Dedykowana instrukcja CPU, funkcja i argumenty przekazywane przez rejestry
- Ale życie jest zbyt krótkie żeby wszystko pisać w asemblerze…
- *Poważniejsze zabawy z syscallami - obierak “Programowanie systemowe”*

# Wywołania systemowe (syscall)

- “Opakowane” dla wygody przez bibliotekę standardową C
  - Nazywa się jak zwykła funkcja
  - Wywołuje się jak zwykła funkcja
  - Przyjmuje argumenty jak zwykła funkcja
  - Zwraca wartość jak zwykła funkcja
  - Ale pod spodem nie jest funkcją, tylko wywołaniem systemowym ;)
- *Dla “nieopakowanych” wywołań (np. nowe funkcje jądra) - funkcja syscall(…)*
- Opisane w sekcji 2 manuala dostępnego z użyciem programu man
  - man 2 chmod, man 2 kill, man 2 open, …
- Można je śledzić z użyciem mechanizmu ptrace
  - Używany w debuggerach, sandboksach …
  - Używany również przez komendę **strace**

# Przykład śledzenia wywołań z użyciem strace

```
[mkwm@temeraire:~] $ strace ls
execve("/usr/bin/ls", ["ls"], [/* 61 vars */]) = 0
brk(NULL)                                = 0x5570a133f000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f43c35cb000
access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=150610, ...}) = 0
mmap(NULL, 150610, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f43c35a6000
close(3)                                 = 0
open("/lib64/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 c\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=153184, ...}) = 0
mmap(NULL, 2253688, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f43c3181000
(...)
```

# Zanim przejdziemy do samych funkcji…

Czasami w manualu obok w pobliżu sygnatury funkcji znajdziemy:

```
Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    snprintf(), vsnprintf():
        _XOPEN_SOURCE >= 500 || _ISOC99_SOURCE ||
            || /* Glibc versions <= 2.19: */ _BSD_SOURCE
```

Oznacza to, że w naszym kodzie (lub opcją -D przy kompilacji) musimy zdefiniować jedną z podanych stałych preprocesora, czasami przypisując im konkretne wartości. W tym przypadku możemy użyć następujących makrodefinicji:

`#define _XOPEN_SOURCE 500` lub `#define _ISOC99_SOURCE`

W manualu może również pojawić się podana wprost linia; np. `#define _GNU_SOURCE`.

# Funkcje systemowe do obsługi plików

- int open(const char *pathname, int flags, mode_t mode) - man 2 open
  - Próbuje otworzyć wskazany plik do odczytu/zapisu/obu - flags musi zawierać O_RDONLY, O_WRONLY albo O_RDWR - a następnie zwraca numer otwartego pliku (numer deskryptora pliku)
  - Liczne flagi dodatkowe (które łączy się ze sobą bitową alternatywą)
    - Utwórz jeśli nie istnieje (O_CREAT)
    - Wymaż zawartość jeśli istnieje (O_TRUNC)
    - Zwróć błąd jeśli plik nie został utworzony w wyniku tego wywołania open (O_CREAT | O_EXCL)
    - Dopisuj na końcu pliku (O_APPEND)
  - Argument mode pojawia się gdy używamy O_CREAT - mówi z jakimi uprawnieniami (“chmod”) utworzyć plik
- int unlink(const char *pathname) - man 2 unlink
  - Usuwa* plik o podanej nazwie
  - * Tak naprawdę usuwa jedną z nazw i-node’a związanego z plikiem; jeśli ostatnia nazwa zostanie usunięta (a najczęściej plik ma tylko jedną) - zostaje usunięty
  - Nazwa funkcji wynika z tego, że technicznie rzecz biorąc każda nazwa pliku to twarde dowiązanie (hard link) do zawartości tego pliku

# Funkcje systemowe do obsługi plików

- ssize_t read(int fd, void *buf, size_t count) - man 2 read
  - Próbuje odczytać wskazaną ilość bajtów z podanego deskryptora i zapisać ją w miejscu wskazanym przez buf, zwraca ilość faktycznie przeczytanych bajtów
  - Jeśli zwraca 0, to dotarliśmy do końca pliku (EOF - End Of File)
- ssize_t write(int fd, const void *buf, size_t count) - man 2 write
  - Próbuje zapisać do deskryptora wskazaną ilość bajtów z podanego obszaru w pamięci, zwraca ilość faktycznie zapisanych bajtów
- int close(int fd) - man 2 close
  - Zamyka podany deskryptor pliku
  - W programach które otwierają wiele plików należy z tej funkcji korzystać w celu oszczędzania zasobów; w prostych programach nie musimy tego robić - zakończenie programu zamyka deskryptory automatycznie
- int fsync(int fd) - man 2 fsync
  - Wymusza natychmiastowe zapisanie przez system operacyjny zmian w pliku na dysku, zamiast kolejkowania ich w buforze zapisu

# Funkcje systemowe do obsługi plików

- off_t lseek(int fd, off_t offset, int whence) - man 2 lseek
  - Służy do przesuwania wskaźnika odczytu/zapisu w obrębie pliku
  - Zwraca położenie wskaźnika po przesunięciu
  - Gdy plik otwarto do zapisu lub odczytu, wskaźnik ustawiany jest na początku pliku; **w trybie dopisywania (append) przed każdym zapisem wskaźnik resetuje się na koniec pliku!**
  - Argument whence służy do określenia względem jakiego punktu podajemy offset dla nowego położenie wskaźnika:
    - SEEK_SET - względem początku pliku (offset = 0 - skok na początek)
    - SEEK_CUR - względem bieżącego położenia wskaźnika
    - SEEK_END - względem końca pliku (jeśli offset > 0, to powodujemy w ten sposób wydłużenie pliku; nadmiarowa przestrzeń to “dziura” wypełniona bajtami \0)
  - Jeśli jako offset podamy 0, możemy odczytać bieżące położenie wskaźnika w pliku lub poznać długość całego pliku (przeskakując jednocześnie na jego koniec)

# Funkcje systemowe do obsługi plików

- int stat(const char *pathname, struct stat *buf)
  - Zapisuje w buf strukturę opisującą plik pathname lub cel dowiązania symbolicznego znajdującego się pod podaną ścieżką (“podąża” za dowiązaniami)
- int fstat(int fd, struct stat *buf)
  - Robi to co stat, ale w odniesieniu do już otwartego pliku o podanym numerze deskryptora
- int lstat(const char *pathname, struct stat *buf)
  - Robi to co stat, ale nie podąża za dowiązaniami symbolicznymi (zwraca informacje o samym dowiązaniu)
- Wszystkie powyższe funkcje są opisane w man 2 stat
- Struktura stat zawiera wiele informacji…

# Struktura stat

```
struct stat {
  dev_t     st_dev;         /* ID of device containing file */
  ino_t     st_ino;         /* inode number */
  mode_t    st_mode;        /* file type and mode */
  nlink_t   st_nlink;       /* number of hard links */
  uid_t     st_uid;         /* user ID of owner */
  gid_t     st_gid;         /* group ID of owner */
  dev_t     st_rdev;        /* device ID (if special file) */
  off_t     st_size;        /* total size, in bytes */
  blksize_t st_blksize;     /* blocksize for filesystem I/O */
  blkcnt_t  st_blocks;      /* number of 512B blocks allocated */

  /* Since Linux 2.6, the kernel supports nanosecond
     precision for the following timestamp fields.
     For the details before Linux 2.6, see NOTES. */

  struct timespec st_atim;  /* time of last access */
  struct timespec st_mtim;  /* time of last modification */
  struct timespec st_ctim;  /* time of last status change */

#define st_atime st_atim.tv_sec      /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};
```

# Funkcje biblioteczne do obsługi plików

- FILE *fopen(const char *path, const char *mode) - man 3 fopen
  - Otwiera podaną ścieżkę path w trybie wskazanym przez mode (np. "r" - odczyt istniejącego pliku, "w+" - odczyt i zapis + utworzenie/wyczyszczenie pliku w momencie otwarcia, "a" - dopisywanie na końcu pliku)
  - Zwraca uchwyt do otwartego pliku
- size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream) - man 3 fread
  - Odczytuje z uchwytu stream *nmemb * size* bajtów do pamięci wskazywanej przez ptr i zwraca ilość odczytanych pełnych rekordów (pojedynczy ma rozmiar *size*, chcemy przeczytać ich *nmemb*)
- size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream) - man 3 fwrite
  - Zapisuje *nmemb * size* bajtów z obszaru wskazywanego przez ptr do uchwytu stream i zwraca ilość zapisanych pełnych rekordów (pojedynczy ma rozmiar *size*, chcemy zapisach ich *nmemb*)
- int fflush(FILE *stream) - man 3 fflush
  - Utrwala zmiany wykonane przez nas w pliku (nie mamy gwarancji że fwrite zmodyfikowało plik od razu!)
  - Czyści bufory odczytu skojarzone ze strumieniem (na wypadek gdyby inny proces zmodyfikował fragmenty otwartego przez nas pliku, które biblioteka odczytała sobie “na zapas”)
- int fclose(FILE *stream) - man 3 fclose
  - Zamyka uchwyt do pliku i wywołuje “po cichu” fflush (czyli gwarantuje utrwalenie zmian w pliku)

# Funkcje biblioteczne do obsługi plików

- int fseek(FILE *stream, long offset, int whence) - man 3 fseek
  - Funkcja analogiczna do funkcji systemowej lseek - różnica to podawanie uchwytu pliku zamiast numeru deskryptora i zwracanie 0 zamiast pozycji w pliku (gdy nie było błędu)
- long ftell(FILE *stream) - man 3 ftell
  - Zwrócenie bieżącej pozycji w pliku; odpowiada zwróceniu wyniku seek(fd, 0, SEEK_CUR)
- void rewind(FILE *stream) - man 3 rewind
  - Przesunięcie na początek pliku; równoważne wywołaniu seek(fd, 0, SEEK_SET)
- int fgetpos(FILE *stream, fpos_t *pos) - man 3 fgetpos
  - Równoważne przypisaniu *pos = seek(fd, 0, SEEK_SET)
- int fsetpos(FILE *stream, const fpos_t *pos) - man 3 fsetpos
  - Równoważne wywołaniu seek(fd, *pos, SEEK_SET)

# Funkcje biblioteczne do obsługi plików

- FILE *fdopen(int fd, const char *mode) - man 3 fdopen
  - Pozwala uzyskać uchwyt do pliku dla funkcji bibliotecznych na podstawie numeru deskryptora pliku używanego w funkcjach systemowych
  - Argument mode musi być zgodny z tym którego użyto przy wywoływaniu open
- int fileno(FILE *stream) - man 3 ferror
  - Operacja odwrotna do fdopen - pozwala otrzymać numer deskryptora pliku który ukrywa się pod uchwytem strumienia
- int feof(FILE *stream) - man 3 ferror
  - Pozwala stwierdzić czy zerowa ilość odczytanych rekordów w fread jest wynikiem dotarcia do końca pliku
- int ferror(FILE *stream) - man 3 ferror
  - Pozwala stwierdzić czy zerowa ilość odczytanych rekordów w fread jest wynikiem wystąpienia błędu w trakcie odczytu

# Funkcje systemowe do obsługi katalogów

- int mkdir(const char *pathname, mode_t mode) - man 2 mkdir
  - Tworzy katalog pod podaną ścieżką z podanymi uprawnieniami
- int rmdir(const char *pathname) - man 2 rmdir
  - Usuwa katalog o podanej ścieżce
  - Katalog musi być pusty!
- Katalog również ma cechy pliku, można go otworzyć funkcją open i uzyskać do niego deskryptor oraz potraktować go (niektórymi) funkcjami operującymi na deskryptorach (np. fstat) …
  - … ale pisanie przez read i write raczej nie zadziała ;)

# Funkcje biblioteczne do obsługi katalogów

- DIR *opendir(const char *name) - man 3 opendir
  - Otwiera podany katalog name do przeszukiwań i zwraca uchwyt do niego
- struct dirent *readdir(DIR *dirp) - man 3 readdir
  - Zwraca wskaźnik do struktury dirent opisującej kolejny element znajdujący się w danym katalogu
  - Kolejne wywołania readdir mogą nadpisać zawartość struktury z poprzednich wywołań!
  - Struktura mogła zostać zaalokowana w sposób statyczny i nie należy próbować robić na niej free
- int closedir(DIR *dirp) - man 3 closedir
  - Zamyka otwarty uchwyt katalogu
- void rewinddir(DIR *dirp) - man 3 rewinddir
  - Wraca na początek listy elementów katalogu o podanym uchwycie
- long telldir(DIR *dirp) - man 3 telldir
  - Zwraca bieżącą pozycję w liście elementów katalogu o podanym uchwycie
- void seekdir(DIR *dirp, long loc) - man 3 seekdir
  - Przesuwa bieżące położenie w liście elementów katalogu do podanego miejsca (klucza)

# Struktura dirent

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* Null-terminated filename */
};
```

# Funkcje biblioteczne do obsługi katalogów

- int nftw(const char *dirpath, int (*fn) (const char *fpath, const struct stat *sb, int typeflag, struct FTW *ftwbuf), int nopenfd, int flags) - man 3 nftw
  - Rekurencyjnie przeszukuje drzewo katalogów z korzeniem w dirpath i wywołuje funkcję fn dla każdego pliku/katalogu (używa nie więcej niż nopenfd deskryptorów jednocześnie)
  - Na pierwszy rzut oka - straszna sygnatura, ale wystarczy zdefiniować funkcję:
    `int fun(const char *fpath, const struct stat *sb, int tflag, struct FTW *ftwbuf)`
    i wywołać całość jako nftw(dirpath, fun, nopenfd, flags)
  - **fpath** - ścieżka do pliku w drzwie, **sb** - wskaźnik na strukturę stat bieżącego elementu, **tflag** - rodzaj wpisu (np. FTW_F - plik, FTW_D - katalog, FTW_SL - dowiązanie symb.), **ftwbuf** - wskaźnik do struktury mówiącej o poziomie zagłębienia w przeszukiwaniach drzewa
  - Callback powinien zwracać 0 (chyba że chcemy przerwać przeszukiwanie drzewa)
  - W manualu jest przykład gotowego programu korzystającego z tej funkcji
- Funkcja ftw zachowuje się jak nftw z flags = 0, a w sygnaturze callbacków nie ma argumentu ftwbuf

# Ryglowanie plików

- **Problem:** co zrobić, jeśli dwa procesy chcą zmodyfikować ten sam plik lub wykonują operacje które zakładają że plik się nie zmieni (w całości lub jego fragment)?
- **Rozwiązanie:** wprowadzenie tymczasowych ograniczeń dostępu do pliku dla działających procesów
- Mechanizm takich ograniczeń nazywamy… na różne sposoby:
  - rygle
  - blokady
  - file locks
- Istnieje kilka podziałów blokad które możemy nałożyć na pliki

# Ryglowanie plików

- Podział ze względu na sposób działania
  - Mandatory/obowiązujące - po założeniu ich respektowanie wymusza system operacyjny
    - W Linuksie się z nich nie korzysta, są opcjonalne i na drodze do usunięcia z jądra
  - Advisory/doradcze - próbujemy samodzielnie uzyskać blokadę zanim zaczniemy wykonywać operację (jeśli inne procesy tego nie będą robiły, nasza blokada jest bezużyteczna)
- Podział ze względu na blokowany zakres
  - Na całym pliku - blokadą objęty jest cały plik
  - Na zakresie bajtów - blokadą objęte są konkretne bajty w ramach pliku
- Podział ze względu na możliwe operacje i dostęp innych procesów
  - Exclusive/wyłączne/do zapisu - tylko jeden proces może skutecznie uzyskać taką blokadę i jedynie ten któremu się udało może w danym momencie czytać/pisać w danym pliku
  - Shared/dzielone/do odczytu - wiele procesów może jednocześnie założyć taką blokadę, a wszystkie takie procesy mogą tylko i wyłącznie czytać z pliku

# Ryglowanie całych plików

- int flock(int fd, int operation) - man 2 flock
- Operacją może być:
  - LOCK_SH - nałożenie na plik blokady dzielonej
  - LOCK_EX - nałożenie na plik blokady wyłącznej
  - LOCK_UN - zwolnienie blokady
- Funkcja flock zakłada blokady zalecane
- Dawno, dawno temu ryglowanie całego pliku było emulowane przez zablokowanie jego zwartości od pierwszego do ostatniego bajtu; współcześnie korzysta z dedykowanego wywołania systemowego
- Zaryglować cały plik można również z wykorzystaniem polecenia flock (man 1 flock) w konsoli (wymaga podania jako argumentu programu na czas wykonania którego blokada ma być założona)

# Ryglowanie zakresów bajtów

- int lockf(int fd, int cmd, off_t len) - man 3 lockf
  - Blokuje zakresy względem bieżącej pozycji w pliku (lseek)
  - W Linuksie stanowi wyłącznie “opakowanie” dla blokad tworzonych przez fcntl, więc nie będziemy używać ;) Co innego na jakimś Uniksie…
- int fcntl(int fd, int cmd, ... /* arg */ ) - man 2 fcntl
  - Dużo więcej niż tylko blokady, ale nas interesują w tym momencie tylko blokady ;)
  - Interesują nas trzy komendy (cmd) - we wszystkich przypadkach trzecim argumentem funkcji będzie wskaźnik do struktury flock
    - F_SETLK - próbuje zdjąć lub założyć blokadę i jeśli nie jest to możliwe z powodu innych blokad, natychmiast zwraca -1 i ustawia errno na EACCES lub EAGAIN
    - F_SETLKW - podobne do powyższego, ale wisi i czeka na “okazję” zamiast zwracać -1
    - F_GETLK - przekazujemy strukturę opisującą blokadę, którą (potencjalnie) chcielibyśmy założyć, a w odpowiedzi nasza struktura zostaje wypełniona informacją o konfliktującej blokadzie (PID procesu, zakres bajtów) lub informacją że założenie blokady byłoby możliwe (ustawienie F_UNLCK w polu l_type struktury)

# Struktura flock

```
struct flock {
    ...
    short l_type;    /* Type of lock: F_RDLCK,
                        F_WRLCK, F_UNLCK */
    short l_whence;  /* How to interpret l_start:
                        SEEK_SET, SEEK_CUR, SEEK_END */
    off_t l_start;   /* Starting offset for lock */
    off_t l_len;     /* Number of bytes to lock */
    pid_t l_pid;     /* PID of process blocking our lock
                        (set by F_GETLK and F_OFD_GETLK) */
    ...
};
```

- l_type
  - F_RDLCK - blokada dzielona
  - F_WRLCK - blokada wyłączna
  - F_UNLCK - zwolnienie/brak blokady
- l_whence - znaczenie identyczne jak whence w lseek (miejsce względem którego podajemy początek zakresu)
- l_start - offset początku względem wybranego punktu odniesienia
- l_len - ilość blokowanych bajtów
- l_pid - PID procesu który uniemożliwia nam założenie blokady

# Ryglowanie plików

- Komenda lslocks umożliwia podejrzenie aktualnych blokad dla plików i zakresów bajtów (kolumna M oznacza blokady obowiązkowe):

```
[mkwm@temeraire:~] $ lslocks
COMMAND           PID  TYPE   SIZE MODE  M      START        END PATH
flock            5738 FLOCK     0B WRITE 0          0          0 /home/mkwm/.cache/lotto.db
gnome-shell      1297 POSIX     0B WRITE 0          0          0 /
gnome-shell      1297 POSIX     0B WRITE 0          0          0 /
thunderbird      2533 POSIX   512K READ  0 1073741826 1073742335 /home/mkwm/.thunderbird/xyz.default/cookies.sqlite
thunderbird      2533 POSIX    32K READ  0        128        128 /home/mkwm/.thunderbird/xyz.default/cookies.sqlite-shm
(...)
```

- W Linuksie rygle obowiązkowe (o ile są aktywne w jądrze systemu), wymagają ustawienia uprawnień pliku na setgid (g+s) przy jednoczesnym braku prawa wykonania dla grupy (g-x) ORAZ zamontowania systemu plików z opcją mand; blokady ustawiane przez fcntl są dla takiego pliku wymuszane

Dziękuję za uwagę

# Propozycja na resztę laboratorium

1. Napisać kod który wczytuje losowe dane z /dev/urandom i zapisuje je do pliku (w dowolny sposób)
2. Zmierzyć łączny czas wczytywania danych z pliku niewielkimi fragmentami
   a. Z użyciem read (funkcje systemowe)
   b. Z użyciem fread (funkcje biblioteczne)
3. Zmierzyć łączny czas zapisywania danych do pliku niewielkimi fragmentami
   a. Z użyciem write (funkcje systemowe)
   b. Z użyciem fwrite (funkcje biblioteczne)
4. Które funkcje były szybsze? Jak myślisz - dlaczego? Czy masz pomysł jak uzyskać odwrotny wynik?