Show title & tags

Lista 5

tags: SE

Zadanie 1

Co robi `read` i `write` jeśli bufor rury jest pełny?

- **read**
  Próby czytania z pustej rury są blokowane. Jeśli koniec do zapisu rury jest zamknięty to po odczytaniu całej zawartości rury otrzymamy EOF.
- **write**
  Jeśli proces do rury mniej niż *PIPE_BUF* to czekamy aż w rurze zrobi się miejsce na pełny zapis, wpp. piszemy tyle ile jest miejsca w rurze i blokujemy.

Jakie gwarancje daje operacja `write` na rurze, do której piszе wiele procesów (wiersze < *PIPE_BUF*)?

Mamy gwarancję, że write wykona się atomowo, tj. przesyłane rura bajty nie wymieszają się z bajtami innych procesów.

Czemu wszystkie procesy potoku `ps -ef | grep sh | wc -l` zakończą się bez interwencji powłoki, jeśli co najmniej jeden z nich umrze?

Przed zakończeniem działa proces przyczy zamyka swój deskryptor pliku, tak aby przy następnej próbie odczytu proces czytający natrafił na EOL. Jeśli proces czytający zakończy się wcześniej to do procesu piszącego wysyłany jest sygnał `SIGPIPE`. Próba pisanie do rury z zamkniętym wyjściem kończy się błędem `EPIPE`.

Kiedy zwracany jest *short count*?

- **read**
  Jeśli w buforze jest mniej bajtów niż zamierzano przeczytać i wejście do rury jest zamknięte.
- **write**
  Jeśli write zostanie zatrzymany sygnałem to po powrocie z obsługi sygnału zwróci tylko odczytane do tej pory bajty.

Jak można połączyć rodzica i dziecko rurą, która została utworzona po uruchomieniu dziecka?

Możemy w tym celu skorzystać z nazwanej (posiadającej nazwę w systemie plików) rury.
FIFO tworzy się za pomocą wywołania `mkfifo`.
Zapis i odczyt odbywa się tak jak ze zwykłego pliku, tj. przy użyciu *open* z odpowiednimi flagami *O_WRONLY* i *O_RDONLY*.

Zadanie 2

- **Urządzenia znakowe** - urządzenia, do których zapis i odczyt odbywa się bez buforowania tj. bajt po bajcie (np. konsole).
- **Urządzenia blokowe** - odczyt i zapis jest wykonywany blokami danych, wymaga się również, aby urządzenia tego typu umożliwiały swobodny dostęp (np. dyski i partycje).

https://www.ibm.com/docs/en/zos/2.1.0?topic=functions-ioctl-control-device
ioctl(fd, request, *argp) - wywołanie systemowe do komunikacji z urządzeniami.

- fd – file descriptor
- request – kod żądania dla danego urządzenia zawiera
- *argp – wskaźnik na strukturę z ustawieniami dla urządzenia

```
\*
* ioctl's have the command encoded in the lower word, and the size of
* any in or out parameters in the upper word. The high 3 bits of the
* upper word are used to encode the in/out status of the parameter.
*
*       31 29 28                17 16                      8 7                   0
*     +---------------------------------------------------------------------------+
*     | I/O | Parameter Length     | Command Group | Command                    |
*     +---------------------------------------------------------------------------+
*/
```

https://embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux/

```
#define EJECT_DISC    _IOW ('d', 112, int)          // eject removable disk
#define KEYTYPE       _IOR ('k', 9, int);           // get keyboard type
#define SIOCGIFCONF   _IOR ('i', 30, struct ifconf)  // get ifnet list
```

DIOCEJECT – wysunięcie usuwalnego nośnika

KIOCTYPE - uzyskanie typu klawiatury

SIOCGIFCONF - uzyskanie listy struktur do manipulowania zasobami sieci (kompatybilność tylko z IPv6) (https://www.freebsd.org/cgi/man.cgi?query=rtnetl&sektion=9)

Krytyka

Zdaniem autora `ioctl` rozrasta się bardzo szybko, jest nieprzenaszalna i słabo udokumentowana. Jest to skutkiem unikowej filozofii wedlug, której każdy plik to "worek bajtów". `The problem with the Unix approach is that every program that writes the file has to know about it. Thus, for example, if we want the file to carry type information inside it, every tool that touches it has to take care to either preserve the type field unaltered or interpret and then rewrite it. While this would be theoretically possible to arrange, in practice it would be far too fragile.`

zadanie 3

nieużytki – po usunięciu pliku miejsce, w którym się on znajdował staje się nieużytkiem. Jest to niewykorzystana pamięć w reprezentacji katalogu zachowująca się jak padding dla poprzedzającego ją pliku.
kompaktowanie – operacja usuwania nieużytków wykonywana, gdy nieużytków jest dość dużo by opłacało się reorganizować katalog w celu odzyskania inaczej marnującego się miejsca.

Dodawanie i usuwanie

Obie te operacje wymagają potencjalnego przejrzenia (w *pesymistycznym przypadku*) całego katalogu, aby zmiękkować kolizi tej operacji wykorzystywane są cache.
Gdy dodajemy plik do katalogu sprawdzamy czy jest miejsce i jeśli go nie ma to wykonujemy operacje kompaktowania w celu jego uzyskania.
Usuwając plik najpierw odszukujemy go w katalogu po czym *zmieniamy wartość entry size jego* poprzednika tak, aby obejmował teraz "usunięty" plik.

zadanie 4

flagi ls

- a – pokazuje ukryte pliki oraz `.` i `..`
- l – long list format (pliki są wypisywane wierszami i zawierają informacje o użytkownikach, prawach dostępu i dacie utworzenia)
- i – dopisuje indeks (nr. inode) każdego pliku

Pokaż jak algorytm trawersuje ścieżkę bezwzględną

```
stat / | grep Inode

ls -lia / | grep -w usr

ls -lia /usr | grep -w bin

ls -lia /usr/bin | grep -w cc

ls -lia /usr/bin

stat -L /usr/bin/cc
```

Od jakiego i-węzła zaczynamy?

- ext4 – 2
- btrfs – 2567 :)

Skąd wiemy, gdzie znajduje się i-ty bajt pliku?

Czemu nie możemy tworzyć dowiązań do plików znajdujących się w obrębie innych systemów plików?

In /proc/version foo

zadanie 5

```
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;     /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;      /* Number of 512B blocks allocated */
    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 */
};

struct linux_dirent {
    unsigned long  d_ino;     /* Inode number */
    unsigned long  d_off;     /* Offset to next linux_dirent */
    unsigned short d_reclen;  /* Length of this linux_dirent */
    char           d_name[];  /* Filename (null-terminated) */
                              /* length is actually (d_reclen - 2 -
                                 offsetof(struct linux_dirent, d_name)) */
    /*
    char           pad;       // Zero padding byte
    char           d_type;    // File type (only since Linux
                              // 2.6.4); offset is (d_reclen - 1)
    */
}

int Getdents(int fd, struct linux_dirent *dirp, unsigned count) {
  int rc = syscall(__NR_getdents, fd, dirp, count);
  if (rc < 0)
    unix_error("Getdents error");
  return rc;
}
```

- **fstatat**(dirfd, pathname, statbuf,) – pozwyskuje strukturę **stat** z deskryptora fd
  - dirfd – deskryptor katalogu
  - pathname – ścieżka do pliku, jeśli jest względna to jest rozwijana od katalogu dirfd
  - statbuf – zmienna, której przekazujemy przez argument struktu **stat**
  - flags – interesuje nas tylko flaga AT_SYMLINK_NOFOLLOW, która zapobiega dereferencji dowiązań symbolicznych.
- **major/major** – urządzenie składa się z dwóch części: major ID identyfikujące klasę urządzenia oraz minor ID, identyfikujące instancję urządzenia danej klasy. ID urządzenia jest reprezentowane przy użyciu typu **dev_t**.
- **readlink(path[4], name, "buf, size)** – czyta do bufora size znaków
- **stick(t) (T)** – informuje, że pliki tego katalogu mogą być usuwane tylko przez właściciela i użytkownika z prawami do zapisu (np. /tj. /tmp).
- **set-gid(s) S)** – informuje, że plik należy do do grupy swojego katalogu, a nie do użytkownika, który go utworzył.
- **set-uid(s) S)** – dla plików oznacza to, że proces będzie miał prawa użytkownika który jest właścicielem pliku wykonywalnego, a nie użytkownika ten plik uruchamiającego.

```
static void print_mode(mode_t m) {
  char t;

  if (S_ISDIR(m))
    t = 'd';
  else if (S_ISCHR(m))
    t = 'c';
  else if (S_ISBLK(m))
    t = 'b';
  else if (S_ISREG(m))
    t = '-';
  else if (S_ISFIFO(m))
    t = 'f';
  else if (S_ISLNK(m))
    t = 'l';
  else if (S_ISSOCK(m))
    t = 's';
  else
    t = '?';

  char ur = (m & S_IRUSR) ? 'r' : '-'; // Read by owner
  char uw = (m & S_IWUSR) ? 'w' : '-'; // Write by owner
  char ux = (m & S_IXUSR) ? 'x' : '-'; // Execute by owner
  char gr = (m & S_IRGRP) ? 'r' : '-'; // Read by group
  char gw = (m & S_IWGRP) ? 'w' : '-'; // Write by group
  char gx = (m & S_IXGRP) ? 'x' : '-'; // Execute by group
  char or = (m & S_IROTH) ? 'r' : '-'; // Read by others
  char ow = (m & S_IWOTH) ? 'w' : '-'; // Write by others
  char ox = (m & S_IXOTH) ? 'x' : '-'; // Execute by others

  /* TODO: Fix code to report set-uid/set-gid/sticky bit as "ls" does. */
  gx = (m & S_ISUID) ? ((gx == 'x' ? 's' : 'S')) : gx;
  ox = (m & S_ISGID) ? ((ox == 'x' ? 's' : 'S')) : ox;
  ox = (m & S_ISVTX) ? ((ox == 'x' ? 't' : 'T')) : ox;

  printf("%c%c%c%c%c%c%c%c%c%c", t, ur, uw, ux, gr, gw, gx, or, ow, ox);
}
```

```
static void file_info(int dirfd, const char *name) {
  struct stat sb[1];

  /* TODO: Read file metadata. */
  fstatat(dirfd, name, sb, AT_SYMLINK_NOFOLLOW);

  print_mode(sb->st_mode);
  printf("%4ld", sb->st_nlink);
  print_uid(sb->st_uid);
  print_gid(sb->st_gid);

  /* TODO: For devices: print major/minor pair; for other files: size. */
  if (S_ISCHR(sb->st_mode) || S_ISBLK(sb->st_mode))
    printf(" %3u,%3u", major(sb->st_rdev), minor(sb->st_rdev));
  else
    printf(" %11ld", sb->st_size);

  char *now = ctime(&sb->st_mtime);
  now[strlen(now) - 1] = '\0';
  printf(" %24s", now);

  printf(" %s", name);

  if (S_ISLNK(sb->st_mode)) {
  /* TODO: Read where symlink points to and print "-> destination" string. */
    char symlink[PATH_MAX];
    ReadLink(dirfd, name, symlink, PATH_MAX);
    printf(" -> %s", symlink);
  }

  putchar('\n');
}
```

```
/* TODO: Iterate over directory entries and call file_info on them. */
for (int i = 0; i < n; i += d->d_reclen) {
  d = (struct linux_dirent *)(buf + i);
  file_info(dirfd, d->d_name);
}
```

zadanie 6

zadanie 7

```
python gen-nums.py 30 | ./mergesort
```

```
static void Sort(int parent_fd) {
  int nelem = ReadNum(parent_fd);

  if (nelem < 2) {
    WriteNum(parent_fd, ReadNum(parent_fd));
    Close(parent_fd);
    return;
  }

  socketpair_t left = MakeSocketPair();
  /* TODO: Spawn left child. */
  if (fork()) {
    Close(left.parent_fd);
  } else {
    Close(parent_fd); // dziadek nie jest nam potrzebny
    Close(left.child_fd);
    Sort(left.parent_fd);
    return;
  }

  socketpair_t right = MakeSocketPair();
  /* TODO: Spawn right child. */
  if (fork()) {
    Close(right.parent_fd);
  } else {
    Close(parent_fd);
    Close(left.child_fd); // nie ma potrzeby, aby prawe dziecko miało deskryptor do
    Close(right.child_fd);
    Sort(right.parent_fd);
    return;
  }

  /* TODO: Send elements to children and merge returned values afterwards. */
  int lower_half = nelem / 2;
  SendElem(parent_fd, left.child_fd, lower_half);
  SendElem(parent_fd, right.child_fd, nelem - lower_half);
  Merge(left.child_fd, right.child_fd, parent_fd);

  Close(parent_fd);
  Close(left.child_fd);
  Close(right.child_fd);

  /* Wait for both children. */
  Wait(NULL);
  Wait(NULL);
}
```