Laborator 11

Operații I/O avansate - Linux

Sisteme de Operare

12-19 Mai 2010

Multiplexare

- mai multe canale, un singur fir de execuție
  - canale = set de descriptori
  - evenimente IN/OUT
- apeluri stateless
  - înregistrare interes cuplata cu așteptare
  - select, poll
- apeluri statefull
  - înregistrare interes separată de așteptare
  - epoll

Mecanisme

- select
  - readfds, writefds, exceptfds
  - simplu, ineficient :)
- poll
  - pollfd (fd, events, revents)
  - simplu, la fel de ineficient
- epoll
  - struct epoll_event (events, data)
  - level-triggered vs edge-triggered
  - simplu, eficient

Generalizarea multiplexării

- eventfd
  - notificări pentru evenimente
- signalfd
  - notificări pentru primire de semnale
- timerfd
  - notificări pentru timere

Operații asincrone

- POSIX AIO
  - -lrt, dacă este suportat
  - aio_read, aio_write, aio_suspend,...
- kernel AIO
  - -laio
  - struct iocb
  - AIO context
  - io_setup, io_submit, io_destroy, io_getevents,...

IO Optimization

- zero-copy
  - splice
- vectored IO
  - struct iovec
  - readv, writev

Întrebări

- Este posibil ca un apel write să blocheze deși un apel anterior select indică faptul că se poate scrie?
- Cum putem fi siguri că un apel write nu blochează procesul curent?
- Începând cu versiunea de kernel 2.6.23 apelul open poate primi ca flag O_CLOEXEC. Până la această versiune O_CLOEXEC putea fi setat doar ulterior folosind apelul fcntl. De ce credeți că s-a făcut schimbarea?
- Cum depinde performanța unei aplicații folosind select/poll/epoll în funcție de numărul descriptorilor de fișier urmăriți?
- Care este diferența între un apel writev cu 5 elemente în vectorul de intrare și 5 apeluri ale funcției write?