

SO Cheat Sheet

Operatii I/O avansate - Linux

Multiplexare IO

select

```
int select(int nfd, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)
```

nfd valoarea celui mai mare file descriptor plus 1
readfds file descriptorii urmăriți pentru citire
writefds file descriptorii urmăriți pentru scriere
exceptfd file descriptorii urmăriți pentru excepții
timeout timpul maxim după care select se întoarce. NULL semnifică blocarea indefinit
întoarce numărul total de file descriptori urmăriți; 0 dacă timeout-ul a expirat sau -1 în caz de eroare

poll

```
int poll(struct pollfd *fds, nfds_t nfd, int timeout)
```

fds conține un descriptor de fișier, evenimentele urmărite pe acest descriptor și parametrul de ieșire care ne spune dacă a apărut un eveniment pe acel descriptor
nfd numărul structurilor fds
timeout timpul maxim după care poll se întoarce. -1 semnifică blocarea indefinit
întoarce numărul de structuri pentru care au apărut evenimente; 0 dacă timeout-ul a expirat sau -1 în caz de eroare

epoll

```
int epoll_create(int size)
```

size hint pentru kernel asupra numărului de descriptori ce vor fi urmăriți
întoarce un file descriptor sau -1 în caz de eroare

```
int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event)
```

epfd file descriptor obținut în urma unui apel `epoll_create`
op operația efectuată asupra `epfd`. Poate fi una dintre: `EPOLL_CTL_ADD`, `EPOLL_CTL_DEL`, `EPOLL_CTL_MOD`
fd file descriptor pentru care se face operația
event structura care descrie evenimentul urmărit
întoarce 0 pentru succes; -1 în caz de eroare

```
int epoll_wait(int epfd, struct epoll_event *event, int max_events, int timeout)
```

epfd file descriptor obținut în urma unui apel `epoll_create`
events parametru de ieșire în care se vor pune evenimentele disponibile; trebuie să fie prealocat
max_events numărul maxim de eventimente întoarse
timeout timpul maxim după care funcția se întoarce; -1 pentru așteptare la infinit
întoarce numărul de file descriptori disponibili pentru I/O sau -1 în caz de eroare

Generalizarea Multiplexării

```
int eventfd(unsigned int initval, int flags)
```

initval valoarea inițială a contorului intern
flags flaguri pentru a schimba comportamentul lui `eventfd`; poate fi lăsat 0
întoarce file descriptor `eventfd` sau -1 în caz de eroare

```
int signalfd(int fd, const sigset_t *mask, int flags)
```

fd -1 pentru a crea un nou descriptor `signalfd`, sau un descriptor deja existent pentru modificarea măștii
mask masca de semnale pe care apelantul dorește să le accepte via descriptorul de fișier
flags flaguri pentru a schimba comportamentul lui `signalfd`; poate fi lăsat 0
întoarce file descriptor sau -1 în caz de eroare

Operații asincrone

```
void io_prep_pread(struct iocb *iocb, int fd, void *buf, size_t count, long long offset)
```

iocb structura `iocb` care va fi inițializată
fd file descriptorul pe care se va face operația
count cât se dorește să fie scris
offset offsetul din fișier de unde să aibă loc operația
întoarce nimic

```
void io_prep_pwrite(struct iocb *iocb, int fd, void *buf, size_t count, long long offset)
```

iocb structura `iocb` care va fi inițializată
fd file descriptorul pe care se va face operația
count cât se dorește să fie citit
offset offsetul din fișier de unde să aibă loc operația
întoarce nimic

```
int io_setup(unsigned int nr_events, io_context_t *ctx)
```

nr_events numărul de evenimente care pot fi primite în contextul curent
ctx parametru de ieșire în care va fi salvat noul context `io`
întoarce 0 pentru succes sau -1 în caz de eroare

```
int io_destroy(io_context_t *ctx)
```

ctx context AIO deja existent
întoarce 0 pentru succes sau -1 în caz de eroare

```
int io_submit(io_context_t ctx, long nr, struct iocb *ios[])
```

ctx context create anterior
nr numărul de elemente din vectorul `ios`
ios vector de pointeri la structurile `iocb` în care se află operațiile care se doresc a fi submise
întoarce numărul de structuri `iocb` submise (poate fi și 0); în caz de eroare întoarce un număr negativ care desemnează eroarea

```
int io_getevent(io_context_t ctx, long min_nr, long nr, struct io_event *events, struct timespec *timeout)
```

ctx context AIO deja existent
min_nr numărul minim de evenimente care trebuie întoarse
nr numărul maxim de evenimente care trebuie întoarse
events vector de evenimente cu evenimentele întoarse
timeout specifică cât să aștepte operația; NULL înseamnă că operația se va întoarce pentru `min_nr` evenimente dacă operația are succes
întoarce numărul de evenimente terminate până la timeout (poate fi și 0) sau un număr negativ reprezentând codul de eroare

```
int io_cancel(io_context_t ctx, struct iocb *iocb, struct io_event *evt)
```

ctx context create anterior
iocb structura `iocb` corespunzând operației care se dorește a fi anulate
result dacă există deja un rezultat, va fi întors în aceasta variabilă
întoarce 0 în caz de succes; în caz de eroare întoarce un număr negativ care desemnează eroarea

Vectored IO

```
ssize_t readv(int fd, const struct iovec *iov, int iovcnt)
```

fd file descriptor pe care se face operația
iov vector cu structuri reprezentând bufferele din care se va citi
iovcnt numărul de elemente ale vectorului `iov`
întoarce numărul de octeți citiți sau -1 în caz de eroare

```
ssize_t writev(int fd, const struct iovec *iov, int iovcnt)
```

fd file descriptor pe care se face operația
iov vector cu structuri reprezentând bufferele din care se va scrie
iovcnt numărul de elemente ale vectorului `iov`
întoarce numărul de octeți scriși sau -1 în caz de eroare