Operaţii I/O avansate - Linux

Multiplexarea I/O

select

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

`nfds`	valoarea celui mai mare file descriptor plus 1
`readfs`	file descriptori urmăriţi pentru citire
`writefds`	file descriptori urmăriţi pentru scriere
`exceptfds`	file descriptori urmăriţi pentru excepţii
`timeout`	timpul maxim după care select întoarce. O valoare NULL semnifică blocarea indefinita
întoarce	numărul total de file descriptori urmăriţi sau 0 dacă timeout-ul a expirat sau -1 în caz de eroare

poll

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

`fds`	conţine un descriptor de fişier, evenimentele urmărite pe acest descriptor şi parametru de ieşire care ne spune dacă a apărut un eveniment pe acel descriptor
`nfds`	numarul structurilor fds
`timeout`	timpul maxim după care poll întoarce. O valoare -1 semnifică blocarea indefinita
întoarce	numărul de structuri pentru care au apărut evenimente sau 0 dacă timeout-ul a expirat sau -1 in caz de eroare

epoll

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

`size`	hint pentru kernel asupa numărului de descriptori care vor fi urmăriţi
întoarce	un file descriptor sau -1 în caz de eroare

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

`epfd`	descriptor de fişier obţinut în urma unui apel epoll_create
`op`	operaţia de efectuat asupra lui epfd. Poate fi: EPOLL_CTL_ADD, EPOLL_CTL_DEL, EPOLL_CTL_MOD
`fd`	descriptor de fişier pentru care se va face operaţia op
`events`	structură care descrie evenimentele urmărite
întoarce	0 în caz de success, -1 în caz de eroare

```
int epoll_wait(int epfd, struct epoll_event *events, int
maxevents, int timeout);
```

`epfd`	descriptor de fişier obţinut în urma unui apel epoll_create
`events`	parametru de ieşire în care se vor pune evenimentele disponibile
`maxevents`	numărul maxim de evenimente întoarse de funcţie
`timeout`	timpul maxim după care epoll_wait întoarce, -1 pentru aşteptare nedefinită
întoarce	numărul de descriptori de fişier disponibili pentru I/O sau -1 in caz de eroare

Generalizarea multiplexării

eventfd

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

`initval`	valoarea iniţială a contorului intern
`flags`	flag pentru a schimba comportamenul lui eventfd
întoarce	file descriptor eventfd, sau -1 in caz de eroare

signalfd

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

`fd`	-1 pentru crearea unui nou signalfd
`mask`	mască de semnale pe care apelantul doreşte să le accepte via descriptorul de fişier
`flags`	flag pentru a schimba comportamenul lui signalfd
întoarce	file descriptor signalfd, sau -1 in caz de eroare

Operaţii asincrone

```
int io_setup(unsigned nr_events, aio_context_t *ctxp);
```

`nr_events`	numar de evenimente care pot fi primite in contextul curent
`ctxp`	context AIO deja existent
întoarce	0 in caz de succes, -1 în caz de eroare

```
int io_destroy(aio_context_t ctx);
```

`ctx`	contextul creat anterior
întoarce	0 in caz de succes, -1 în caz de eroare

```
int io_submit(aio_context_t ctx_id, long nr, struct iocb
**iocbpp);
```

`ctx_id`	contextul creat anterior
`iocbpp`	operaţii AIO trimise pentru procesare în contxtul ctx_id
întoarce	0 in caz de succes, -1 în caz de eroare

```
int io_getevents(aio_context_t ctx_id, long min_nr, long nr,
struct io_event *events, struct timespec *timeout);
```

`ctx_id`	contextul creat anterior
`min_nr`	numărul minim de operaţii asincrone aşteptate
`nr`	numărul maxim de operaţii asincrone aşteptate
`events`	structura completată ca urmare a încheierii unor operaţii asincrone
`timeout`	timpul maxim de aşteptare pentru terminarea unor operaţii asincrone
întoarce	0 in caz de succes, -1 în caz de eroare

```
int io_cancel(aio_context_t ctx_id, struct iocb *iocb,
struct io_event *result);
```

`ctx_id`	contextul creat anterior
`iocb`	operaţia care se doreşte a fi anulată
`result`	rezultatul operaţiei de anulare
întoarce	0 in caz de succes, -1 în caz de eroare

Vectored IO

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

`fd`	descriptor fişier
`iov`	vector de citire
`iovcnt`	numărul de elemente care vor fi citite
întoarce	numărul de octeţi citiţi în caz de succes sau -1 în caz de eroare

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

`fd`	descriptor fişier
`iov`	vector de scriere
`iovcnt`	numărul de elemente care vor fi scrise
întoarce	numărul de octeţi scrişi în caz de succes sau -1 în caz de eroare

Zero Copy

```
long splice(int fd_in, loff_t *off_in, int fd_out, loff_t
*off_out, size_t len, unsigned int flags);
```

`fd_in`	file descriptor pentru fişierul de intrare
`off_in`	offset în fişierul de intrare
`fd_out`	file descriptor pentru fişierul de ieşire
`off_out`	offset pentru fişierul de ieşire
`len`	numărul de octeţi implicaţi în copiere
`flags`	flag care poate modifica comportamentul apelului splice
întoarce	numărul de octeţi copiaţi din/în pipe sau -1 în caz de eroare