Kreiranje novog procesa

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork();
```

U slučaju uspeha u roditeljskom procesu vraća PID deteta a u procesu detete vraća 0. U slučaju greške vraća -1.

PID tekuceg procesa

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid();
```

PID roditeljskog procesa

```
#include <sys/types.h>
#include <unistd.h>
pid_t getpid();
```

Suspendovanje izvršavanja procesa

```
#include <sys/types.h>
#include <unistd.h>
unsigned int sleep(unsigned int seconds);
```

Izvršavanje procesa se suspenduje zadati broj sekundi. Nakon isteka specificiranog vremena proces normalno nastavlja svoje izvršenje od prve naredne linije koda.

Završetak procesa

```
#include <stdlib.h>
void exit(int status);
```

Završava se izvršavanje procesa koji je pozvao funkciju: zatvaraju se sve otvorene datoteke oslobađa se zauzeta memorija obaveštava se roditeljski proces i prosleđuje mu se exit kod procesi deca za roditeljski proces dobijaju proces init (PID = 1) Nikad ne dolazi do greške prilikom poziva funkcije exit i funkcija ne vraća vrednost. Argument status definiše exit kod procesa (status kojim se završio). Uobičajeno je da 0 označava da se proces normalno završio a negativna vrednost da je došlo do greške. Ne postoji standardna lista statusa koji se mogu primenjivati. Status procesa se može dobiti korišdenjem wait funkcije.

Status procesa

```
#include <sys/types.h>
#include <unistd.h>
pid_t wait ( int *status-ptr )
pid_t waitpid ( pid_t pid, int *status-ptr )
```

Proces koji je pozvao funkciju se zaustavlja dok se ne završi proces dete. Funkcija wait zaustavlja proces dok se ne završi bilo koji proces dete, a funkcija waitpid dok se ne završi proces dete čiji je PID specificiran. Promenljiva status-ptr prihvata exit kod sa kojim se završio proces dete.

Učitavanje nove izvršne datoteke

```
#include <sys/types.h>
#include <unistd.h>
int execl(const char *path, const char *arg, ...);
int execv(const char *path, char *const argv[]);
int execlp(const char *file, const char *arg, ...);
int execvp(const char *file, char *const argv[]);
int execle(const char *path, const char * arg, ..., char
*const envp[]);
int execve(const char *filename, char *const argv[], char
*const envp[]);
```

Funkcija execl očekuje listu argumenata koja se završava sa NULL. Funkcija execv očekuje niz (vektor) argumenata. Prvi argument odgovara argv[0] i mora da bude ime programa koji se poziva, poslednji element niza mora d aima vrednost NULL. Funkcije execlp i execvp omogudavaju pretraživanje izvršnih datoteka u PATH putanji (ukoliko ime izvršne datoteke ne sadrži / a u suprotnom se ponašaju kao standardne execl i execv funkcije). Funkcije execle i execve omogućavaju specificiranje vrednosti environment promenljivih.

Kreiranje niti

```
#include <pthread.h>
int pthread_create ( pthread_t *threadhandle,
pthread_attr_t *attribute, void *(*start_routine)(void*),
void *arg ) ;
```

Argumenti: handle novokreirane niti ukoliko se funkcija uspešno izvrši argumenti za kreiranje niti (mogu imati NULL vrednost) pokazivač na funkciju koja sadrži programski kod koji de niti zvršavati. Funkcija mora da vraća void * i da ima jedan ulazni argument koji je void *. pokazivač na ulazni argument niti

---

Čekanje niti

```
#include <pthread.h>
int pthread_join ( pthread_t threadhandle, void
**returnvalue )
```

Kreiranje mutexa

```
#include <pthread.h>
int pthread_mutex_init ( pthread_mutex_t * mutex,
pthread_mutexattr_t * attr );
```

Prvi argument je pokazivač na promenljivu tipa mutex koja je prethodno deklarisana. Drugi argument definiše atribute mutex-a koji se kreira. NULL vrednost ovog argumenta obezbeđuje korišćenje podrazumevanih vrednosti.

Brisanje mutexa

```
#include <pthread.h>
int pthread_mutex_destroy ( pthread_mutex_t * mutex);
```

Argument je pokazivač na objekat tipa mutex koji želimo da obrišemo.

Zaključavanje mutex-a

```
#include <pthread.h>
int pthread_mutex_lock ( pthread_mutex_t * mutex);
```

Otključavanje mutex-a

```
#include <pthread.h>
int pthread_mutex_lock ( pthread_mutex_t * mutex);
```

Kreiranje uslovne promenljive

```
#include <pthread.h>
int pthread_cond_init ( pthread_cond_t * condition,
pthread_attr_t   * attr );
```

Prvi argument je pokazivač na uslovnu promenljivu koja je prethodno deklarisana. Drugi argument definiše atribute uslovne promenljive koja se kreira. NULL vrednost ovog argumenta obezbeđuje korišćenje podrazumevanih vrednosti.

Brisanje uslovne promenljive

```
#include <pthread.h>
int pthread_cond_destroy ( pthread_cond_t * condition );
```

Argument je pokazivač na uslovnu promenljivu koji želimo da obrišemo.

Blokiranje niti na uslovnoj promenljivoj

```
#include <pthread.h>
int pthread_cond_wait ( pthread_cond_t * condition,
pthread_mutex_t * mutex );
```

Sistemski poziv blokira izvršavanje niti na određenoj uslovnoj promenljivoj. Nit se blokira na uslovnoj promenljivoj sve dok neka druga nit ne signalizira da je uslov koji se čeka ispunjen. Funkcija se poziva samo ukoliko je objekat tipa mutex zaključan. Mutex se automatski otključava tokom čekanja niti.

Signalizacija uslovne promenljive

```
#include <pthread.h>
int pthread_cond_signal ( pthread_cond_t * condition );
int pthread_cond_broadcast ( pthread_cond_t * condition
);
```

Sistemski poziv signalizira (budi) nit koja čeka na uslovnoj promenljivoj. Funkcija se poziva u okviru kritične sekcije odnosno mutex mora biti zaključan. Ukoliko više niti čeka na istu uslovnu promenljivu onda se koristi broadcast funkcija.

Kreiranje POSIX semafora

```
#include <semaphore.h>
int sem_init ( sem_t * sem, int pshared, unsigned int value
);
```

Prvi argument je pokazivač na promenljivu tipa POSIX semafor koja je prethodno deklarisana. Drugi argument je uvek 0 (zato što POSIX semafore koristimo samo za sinhronizaciju niti) Treći argument predstavlja vrednost na koju se kreirani semafor inicijalizuje.

Brisanje POSIX semafora

```
#include <semaphore.h>
int sem_destroy ( sem_t * sem );
```

Pi V funkcije POSIX semafora

```
#include <semaphore.h>
int sem_wait ( sem_t * sem );
int sem_post ( sem_t * sem );
```

Kreiranje System V semafora

#include <sys/sem.h>
int semget (key_t key, int nsems, int flag);

Prvi argument key predstavlja jedinstveni identifikator System V semafora na nivou čitavog sistema. Mora biti poznat svim procesima koji žele da koriste određeni semafor. Drugi argument nsems specificira broj celobrojnih vrednosti koje semafor sadrži.

Operacije nad System V semaforima

#include <sys/sem.h>
int semop (int semid, struct sembuf * semops, int nsops);

Prvi argument semid predstavlja identifikator (referencu) System V semafora koji je dobijen pozivom funkcije semget. Drugi argument semops predstavlja niz operacija koje treba izvršiti nad celobrojnim vrednostima System V semafora. Svaka operacija je zadata kao sembuf struktura. Treći argument nsops predstavlja broj elemenata niza sa operacijama.

Struktura sembuf je deklarisana u zaglavlju
<sys/sem.h>
i ima slededi izgled:

```
struct sembuf {
        ushort sem_num;
        short sem_op;
        short sem_flg;
};
```

Polje sem_num specificira indeks celobrojne vrednosti na koju se operacija odnosi. Polje sem_op definiše operaciju koja se obavlja. Mogude vrednosti su: > 0 – pozitivne vrednosti se dodaju odgovarajudoj celobrojnoj vrednosti semafora odnosno ekvivalent je V operacija < 0 – negativne vrednosti se oduzimaju od odgovarajude celobrojne vrednosti semafora odnosno ekvivalent je P operacija. Nijedna celobrojna vrednost ne može biti negativna. 0 – proces koji je izvršio sistemski poziv se blokira dok odgovarajuća celobrojna vrednost ne dobije vrednost 0. Polje sem_flag definiše način na koji se operacija obavlja. Najčešde ima vrednost NULL (kada se prihvata podrazumevani način izvršavanja operacije). Vrednost IPC_NOWAIT sprečava blokiranje procesa koji je izvršio sistemski poziv. Sistemski poziv vraća 0 ukoliko su sve operacije uspešno izvršene odnosno -1 ako je došlo do greške prilikom izvršavanja neke od operacija.

Kontrola System V semafora

#include <sys/sem.h>
int semctl (int semid, int semnum, int cmd, union semun arg);

Prvi argument semid predstavlja identifikator (referencu) System V semafora koji je dobijen pozivom funkcije semget. Drugi argument semnum predstavlja indeks celobrojne vrednosti na koju se operacija odnosi. Tredi argument cmd definiše operaciju koju treba izvršiti nad semaforom. predstavlja broj elemenata niza sa operacijama. Četvrti argument arg omogućava definisanje parametara neophodnih za operaciju i definiše se kao unija semun.

Unija semun je deklarisana u zaglavlju
<sys/sem.h>
i ima slededi izgled:

```
union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
        struct seminfo * __buf;
        void * __pad;
};
```

Polje val se koristi za zadavanje vrednosti prilikom inicijalizacije celobrojne vrednosti semafora. Neke od mogućih operacija koje se mogu izvršiti nad System V semaforom: SETVAL – definiše vrednost odgovarajuće celobrojne vrednosti System V semafora IPC_RMID – brisanje System V semafora iz sistema.

Kreiranje datavoda

#include <unistd.h>
int pipe (int p[2]);

p[0] – kraj datavoda koji se koristi za čitanje p[1] – kraj datavoda koji se koristi za pisanje

---

Zatvaranje datavoda

#include <unistd.h>
int close (int fd);

Kada se u svim procesima zatvore svi deskriptori koji ukazuju na datavod datavod se automatski uništava.

Citanje i pisanje iz datavoda

#include <unistd.h>
ssize_t read(int fd, void * buff, size_t count);
ssize_t write(int fd, void * buff, size_t count);

Prvi argument predstavlja deskriptor odgovarajućeg kraja datavoda. Drugi argument predstavlja pokazivač na bafer iz koga se podaci upisuju u datavod ili u koji se podaci upisuju iz datavoda. Treći argument predstavlja veličinu bafera u bajtovima. Povratna vrednost funkcija predstavlja broj pročitanih bajtova odnosno broj upisanih bajtova u datavod.

Preusmeravanje UI operacija

#include <unistd.h>
int dup(int oldfd);
int dup2(int oldfd, int newfd);

Sistemski poziv dup nalazi najmanji slobodan deskriptor datoteke i postavlja ga da ukazuje na istu datoteku na koju ukazuje i deskriptor oldfd. Sistemski poziv dup2 zatvara deskriptor newfd, ukoliko je otvoren, i modifikuje ga tako da ukazuje na istu datoteku na koju ukazuje i deskriptor oldfd.

Kreiranje imenovanog datavoda

#include <unistd.h>
int mkfifo(const char * pathname, mode_t mode);

Kreiranje signala

#include <signal.h>
#include <sys/types.h>
int kill (pid_t pid, int sig);

Prvi argument pid definiše procese kojima se signal šalje: pid > 0 – signal se šalje procesu sa zadatim identifikatorom pid == 0 – signal se šalje svim procesima koji pripadaju istoj grupi kao i proces koji šalje signal pid < -1 – signal se šalje svim procesima koji pripadaju grupi –pid pid == -1 – signal se šalje svim procesima kojima tekući proces može da pošalje signal isključujući njega samog i proces init(1)

Drugi argument sig definiše signal koji se šalje: SIGHUP – prekinuta komunikacija terminalom SIGINT – procesu se šalje signal Ctrl + C SIGQUIT – procesu se šalje signal Ctrl + \ SIGKILL – signal koji prekida proces i koji se ne može blokirati SIGBUS/SIGSEGV – greška na magistrali odnosno greška pri segmentaciji SIGPIPE – write opoeracija nad datavodom iz koga nijedan proces ne čita podatke SIGALRM – signal alarma SIGTERM – signal za terminaciju procesa SIGSTOP – signal koji suspenduje izvršavanje procesa SIGTSTP- izvršavanje procesa u pozadini (Ctrl + Z) SIGCONT – proces nastavlja izvršavanje nakon suspendovanja SIGCHLD – proces dete je završilo sa izvršavanjem SIGUSR1/SIGUSR2 – korisnički definisani signali

---

Obrada signala

#include <signal.h>
void (*signal(int signum, void(*handler)(int)))(int);

Prvi argument signum predstavlja identifikator signala za koji se definiše način obrade. Drugi argument handler definiše način obrade: SIG_DFL – podrazumevana obrada signala SIG_IGN – signal se ignoriše pokazivač na funkciju koja se poziva po pristizanju signala Sistemski poziv vraća pokazivač na funkciju koja je prethodno bila zadužena za obradu signala u slučaju uspeha odnosno SIG_ERR u slučaju greške.

Suspendovanje izvrsenja procesa

#include <unistd.h>
int pause();

Slanje SIGALRM signala

#include <unistd.h>
int alarm(unsigned int seconds);

Podrazumevana obrada za signal SIGALRM je prekid izvršavanja procesa.

Kreiranje reda poruka

#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
int msgget(key_t key, int msgflg);

Prvi argument key predstavlja jedinstveni identifikator reda poruka na nivou čitavog sistema. Mora biti poznat svim procesima koji žele da koriste određeni red poruka.

Vrednost drugog argumenta msgflg se definiše kao rezultat OR operacije nad različitim vrednostima i određuje : prava pristupa redu poruka (koristidemo vrednost 0666 koja svim korisnicima dodeljuje sve privilegije nad red poruka) mod kreiranja reda poruka. Najčešća vrednost je IPC_CREAT – sistemski poziv kreira red poruka ukoliko on već ne postoji u sistemu.

Slanje i primanje poruka

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
int msgsnd(int msgid, int msgbuf * msgp, size_t msgsz, int
msgflg);
int msgrcv(int msgid, int msgbuf * msgp, size_t msgsz, int
msgtyp, int msgflg);
```

Prvi argument msgid predstavlja identifikator (referencu) reda poruka koji je dobijen pozivom funkcije msgget. Drugi argument msgp predstavlja pokazivač na bafer koji sadrži poruku koja se šalje ili u koji de biti smeštena poruka koja se prima. Treći argument msgsz predstavlja veličinu bafera poruke u bajtovima.

Argument msgflg definiše mod slanja/prijema poruke a vrednost 0 označava prihvatanje podrazumevanog moda rada. Argument msgtyp kod funkcije za prijem je nenegativan ceo broj koji definiše tipove poruka koje se čitaju iz reda: msgtyp == 0 – čitaju se sve poruke bez obzira na tip msgtyp > 0 – čitaju se samo poruke specificiranog tipa

---

Poruka

```
struct msgbuf {
        long mtype; /* tip poruke, mora biti > 0 */
        char mtext[1]; /* tekst poruke */
};
```

Polje mtext je niz (ili nek adruga struktura) koji definiše sadržaj poruke. Dozvoljene su poruke nulte veličine koje ne sadrže polje mtext. Polje mtype mora je pozitivan ceo broj koji omogućava procesu koji prima poruke da selektuje samo poruke od interesa.

Kontrola reda poruka

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
int msgctl (int msqid, int cmd, struct msgid_ds buf);
```

Prvi argument msgmid predstavlja identifikator (referencu) reda poruka koji je dobijen pozivom funkcije msget. Drugi argument cmd definiše operaciju koju treba izvršiti nad semaforom. Za brisanje reda poruka se koristi operacija IPC_RMID.

Dinamicko zauzimanje i oslobadjanje memorije

```
#include <stdlib.h>
char * malloc (unsigned int size);
void free (char * ptr);
```

Rezervisanje memorije z anis elemenata i promena velicine prethodno rezervisanog segmenta

```
#include <stdlib.h>
char * calloc (unsigned int elem, unsigned int elsize);
char * realloc (char * ptr, unsigned int size);
```

Rad sa sadrzajem memorijskih lokacija

```
#include <memory.h>
void *memcpy(void *dest, const void *src, size_t n);
void *memccpy(void *dest, const void *src, int c, size_t n);
void *memchr(const void *s, int c, size_t n);
int memcmp(const void *s1, const void *s2, size_t n);
void *memset(void *s, int c, size_t n);
```

Kreiranje deljene memorije

```
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int flg);
```

Prvi argument key predstavlja jedinstveni identifikator segmenta deljene memorije na nivou čitavog sistema. Mora biti poznat svim procesima koji žele da koriste određeni segment deljene memorije.

Vrednost drugog argumenta size definiše veličinu segmenta deljene memorije koji se kreira. Vrednost tredeg argumenta flag se definiše kao rezultat OR operacije nad različitim vrednostima i određuje : prava pristupa segmentu deljene memorije (koristićemo vrednost 0666 koja svim korisnicima dodeljuje sve privilegije nad deljenom memorijom) mod kreiranja segmenta deljene memorije. Najčešća vrednost je:
IPC_CREAT – sistemski poziv kreira segment deljene memorije ukoliko on ved ne postoji u sistemu. Sistemski poziv vrada celobrojni identifikator (referencu) segmenta deljene memorije a u slučaju greške vrada (-1). Dobijeni identifikator (referenca) je važeći samo kod procesa koji je izvršio sistemski poziv i kod njegove dece.

Mapiranje segmenta deljene memorije

#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
void * shmat(int shm_id, char * ptr, int flag);

Prvi argument shm_id predstavlja identifikator (referencu) segmenta deljene memorije koji je dobijen pozivom funkcije shmget. Drugi argument ptr u našem slučaju uvek ima vrednost NULL. Treći atgument definiše način korišdenja segmenta deljene memorije: SHM_RDONLY – deljenu memoriju je mogude samo čitati 0 – moguda je i i zmena sadržaja deljene memorijeu bajtovima. U slučaju uspeha sistemski poziv vrača pokazivač na početak adresnog prostora u koji je mapirana deljena memorija. U slučaju greške sistemski poziv vrada NULL.

Uklanjanje segmenta deljene memorije

#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
void * shmdt(const void * shmaddr);

Prvi argument shmaddr predstavlja pokazivač na početak adresnog prostora u koji je mapiran segment deljene memorije (adresa koju vrada funkcija shmat).

Kontrola reda poruka

#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/msg.h>
int shmctl (int shmid, int cmd, struct shmid_ds buf);

Prvi argument shmid predstavlja identifikator (referencu) segmenta deljene memorije koji je dobijen pozivom funkcije shmget. Drugi argument cmd definiše operaciju koju treba izvršiti nad segmentom deljene memorije. Za brisanje deljene memorije se koristi operacija IPC_RMID.

Otvaranje datoteke

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open (char * filename, int flags [, mode_t mode]);

Prvi argument funkcije filename je znakovni niz koji sadrži puno ime datoteke zadato kao apsolutna ili relativna putanja. Drugi argument flags se dobija kao razultat OR operacije nad slededim konstantama: O_RDONLY – datoteka se otvara samo za čitanje. O_WRONLY – datoteka se otvara samo za upis. O_RDWR – datoteka se otvara za upis i za čitanje. O_APPEND – pozicionira pokazivač datoteke na kraj. Datoteka je spremna za upis na kraj. O_CREAT – ukoliko datoteka sa zadatim imenom ne postoji, kreira se nova. O_TRUNC – ukoliko datoteka sa zadatim imenom postoji njen sadržaj se briše (datoteka postaje dužine nula) Treći argument mode je opcioni i koristi se prilikom kreiranja nove datoteke za definisanje privilegija. Ukoliko se izostavi koriste se podrazumevane privilegije koje zavise od toga koji je korisnik pokrenuo proces.

Zatvaranje datoteke

#include <unistd.h>
int * close (int fd);

Arguemnt fd predstavlja deskriptor datoteke koja se zatvara.

Čitanje i pisanje iz datoteka

#include <unistd.h>
ssize_t read(int fd, void * buff, size_t count);
ssize_t write(int fd, void * buff, size_t count);

Prvi argument predstavlja deskriptor odgovarajude datoteke. Drugi argument predstavlja pokazivač na bafer iz koga se podaci upisuju u datoteku ili u koji se podaci upisuju iz datoteke. Tredi argument predstavlja broj bajtova koji se upisuju u datoteku ili se čitaju iz datoteke. Povratna vrednost funkcija predstavlja broj pročitanih bajtova odnosno broj upisanih bajtova u datoteku. U slučaju greške funkcija vrada -1.

Pozicioniranje pokazivaca datoteke

#include <unistd.h>
#include <sys/types.h>
off_t lseek(int fd, off_t offset, int mode);

Prvi argument fd predstavlja deskriptor odgovarajude datoteke. Drugi argument offset definiše novu poziciju unutar datoteke a tumači se na osnovu vrednosti tredeg argumenta. Tredi argument mode definiše kakos e određuje nova pozicija: SEEK_SET – offset je definisan u odnosu na početak datoteke. SEEK_CUR – offset je definisan u odnosu na trenutnu poziciju pokazivača datoteke. SEEK_END – offset je definisan u odnosu na kraj datoteke. Povratna vrednost funkcija predstavlja novu poziciju u datoteci U slučaju greške funkcija vrada -1.

Informacije o datoteci

```
#include <unistd.h>
#include <sys/types.h>
int stat(char * name, struct stat * buff);
```

Prvi argument name predstavlja ime datoteke koje je zadato kao apsolutna ili relativna putanja. Drugi argument buffer predstavlja pokazivač na strukturu koja de prihvatiti informacije o datoteci.

Struktura *stat*

```
#include <sys/stat.h>
struct stat {
        dev_t st_dev; /* broj uređaja */
        ino_t st_ino; /* broj inode strukture */
        mode_t st_mode; /* dozvole pristupa datoteci */
        nlink_t st_nlink; /* broj linkova ka ovoj datoteci
*/
        uid_t st_uid; /* identifikacija korisnika */
        gid_t st_gid; /* identifikacija grupe */
        dev_t st_rdev; /* tip uređaja */
        off_t st_size; /* veličina datoteke u bajtovima*/
        blksize_t st_blksize; /* veličina blokova za U/I
datotečnog sistema */
        blkcnt_t st_blocks; /* broj dodeljenih blokova */
        time_t st_atime; /* vreme poslednjeg pristupa
datoteci */
        time_t st_mtime; /* vreme poslednje
modifikacije datoteke */
        time_t st_ctime; /* vreme poslednje promene
statusa */
};
```

Makro naredbe za odredjivanje tipa datoteke

Makro naredbe koje za argument uzimaju atribut st_mode iz strukture stat a vradaju TRUE ili FALSE u zavisnosti od tipa datoteke:

S_ISDIR – direktorijum
S_ISREG – regularna datoteka
S_ISCHR – znakovna specijalna datoteka
S_ISBLK – blok specijalna datoteka
S_ISFIFO – imenovani datavod (pipe) ili
FIFO S_ISLNK – link
S_ISSOCK – socket

Otvaranje direktorijuma

```
#include <dirent.h>
DIR * opendir(const char * pathname);
```

Prvi argument pathname predstavlja ime direktorijuma koje je zadato kao apsolutna ili relativna putanja.

Citanje sadrzaja direktorijuma

```
#include <dirent.h>
struct dirent * readdir(DIR * dp);
```

Prvi argument dp predstavlja pokazivač na otvoreni direktorijum. U slučaju uspeha sistemski poziv vrada pokazivač na dirent strukturu koja sadrži informacije o stavki direktorijuma. Pokazivač dp se automatski inkrementira da pokazuje na sledeću stavku u direktorijumu. Kako bi obradili sve stavke u nekom direktorijumu treba sukcesivno pozivati funkciju readdir dok funkcija ne vrati NULL vrednost. Funkcija vraća NULL vrednost kada pokazivač dp dodje do kraja direktorijuma.

Stavka direktorijuma

```
struct dirent {
        ino_t d_ino;
        char d_name[NAME_MAX+1];
        /* ime stavke koja se nalazi u direktorijumu,
može biti poddirektorijum, datoteka ili veza (link) */ };
```

Resetovanje direktorijuma

```
#include <dirent.h>
void rewinddir(DIR * dp);
```

Zatvaranje direktorijuma

```
#include <dirent.h>
void close(DIR * dp);
```

Kreiranje direktorijuma

```
#include <sys/stat.h>
int mkddir(const chat * pathname, mode_t mode);
```

Brisanje direktorijuma

```
#include <sys/stat.h>
int rmdir(const chat * pathname);
```