Forking&IPC

Fork

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);

#include <sys/types.h>
#include <sys/wait.h>

pid_t wait(int *status);

#include <unistd.h>

int execlp(const char *file, const char *arg, ... /*, (char *)0 */);
```

Pipe

```
#include <unistd.h>

int pipe(int *fildes);
```

File

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>

ssize_t read(int d, void *buf, size_t nbytes);
ssize_t write(int d, const void *buf, size_t nbytes);

#include <unistd.h>

int close(int d);

#include <fcntl.h>

int open(const char *path, int flags, ...);
```

Flags:

O_RDONLY	open for reading only
O_WRONLY	open for writing only
O_RDWR	open for reading and writing
O_NONBLOCK	do not block on open
O_APPEND	append on each write
O_TRUNC	truncate size to 0

```
O_EXCL          error if create and file exists
O_CREAT         create file if it does not exist
```

Ako je primenjena O_CREAT opcija zahteva se davanje prava pristupa, a može se upotrebiti i tip "**mode_t**".

int **open**(*const char *path*, *int flags*, *mode_t mode*)

Prava pristupa:

```
#define S_IRWXU 0000700    /* RWX mask for owner */
#define S_IRUSR 0000400    /* R for owner */
#define S_IWUSR 0000200    /* W for owner */
#define S_IXUSR 0000100    /* X for owner */
#define S_IRWXG 0000070    /* RWX mask for group */
#define S_IRGRP 0000040    /* R for group */
#define S_IWGRP 0000020    /* W for group */
#define S_IXGRP 0000010    /* X for group */

#define S_IRWXO 0000007    /* RWX mask for other */
#define S_IROTH 0000004    /* R for other */
#define S_IWOTH 0000002    /* W for other */
#define S_IXOTH 0000001    /* X for other */

#define S_ISUID 0004000    /* set user id on execution */
#define S_ISGID 0002000    /* set group id on execution */
#define S_ISTXT 0001000    /* sticky bit */
```

Tip fajla:

```
#define S_IFIFO  0010000               /* named pipe (fifo) */
#define S_IFCHR  0020000               /* character special */
#define S_IFDIR  0040000               /* directory */
#define S_IFBLK  0060000               /* block special */
#define S_IFREG  0100000               /* regular */
#define S_IFLNK  0120000               /* symbolic link */
#define S_IFSOCK 0140000               /* socket */
```

#include <unistd.h>

int **dup**(*int oldd*);

int **dup2**(*int oldd*, *int newd*);

#include <unistd.h>

int **mknod**(*const char *path*, *mode_t mode*, *dev_t dev*);

Signals

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

Sockets

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

Domeni:

PF_UNIX
PF_INET

Tipovi:

SOCK_STREAM
SOCK_DGRAM

```
struct sockaddr {
        unsigned char   sa_len;         /* total length */
        sa_family_t     sa_family;      /* address family */
        char            sa_data[14];    /* actually longer; address value */
};

struct sockaddr_un {
        unsigned char   sun_len;        /* sockaddr len including null */
        sa_family_t     sun_family;     /* AF_UNIX */
        char    sun_path[104];          /* path name (gag) */
};

#include <sys/types.h>
#include <sys/socket.h>

int connect(int s, const struct sockaddr *name, socklen_t namelen);
```

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int s, const void *msg, size_t len, int flags);
ssize_t recv(int s, void *buf, size_t len, int flags);

Flags:
MSG_DONTWAIT    do not block

#include <sys/types.h>
#include <sys/socket.h>

int socketpair(int d, int type, int protocol, int *sv);

#include <sys/types.h>
#include <sys/socket.h>

int bind(int s, const struct sockaddr *addr, socklen_t addrlen);
int listen(int s, int backlog);
int accept(int s, struct sockaddr * restrict addr, socklen_t * restrict addrlen);

struct addrinfo {
    int              ai_flags;     // AI_PASSIVE, AI_CANONNAME, etc.
    int              ai_family;    // AF_INET, AF_INET6, AF_UNSPEC
    int              ai_socktype;  // SOCK_STREAM, SOCK_DGRAM
    int              ai_protocol;  // use 0 for "any"
    size_t           ai_addrlen;   // size of ai_addr in bytes
    struct sockaddr *ai_addr;      // struct sockaddr_in or _in6
    char            *ai_canonname; // full canonical hostname
    struct addrinfo *ai_next;      // linked list, next node
};

struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};

struct in_addr {
    uint32_t s_addr; // that's a 32-bit int (4 bytes)
};
INET_ADDRSTRLEN
INET6_ADDRSTRLEN
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

const char * inet_ntop(int af, const void * restrict src, char * restrict dst, socklen_t size);
int inet_pton(int af, const char * restrict src, void * restrict dst);

#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *hostname, const char *servname, const struct addrinfo *hints, struct addrinfo **res);

void freeaddrinfo(struct addrinfo *ai);

struct addrinfo {
    int ai_flags;             /* input flags */
    int ai_family;            /* protocol family for socket: PF_UNSPEC */
    int ai_socktype;          /* socket type: SOCK_STREAM, SOCK_DGRAM, SOCK_RAW */
    int ai_protocol;          /* protocol for socket */
    socklen_t ai_addrlen;     /* length of socket-address */
    struct sockaddr *ai_addr; /* socket-address for socket */
    char *ai_canonname;       /* canonical name for service location */
    struct addrinfo *ai_next; /* pointer to next in list */
};
```

Java Sockets

```
import java.io.*;
import java.net.*;

Socket(String host, int port);

OutputStream getOutputStream();
InputStream getInputStream();
void close();

OutputStream();
void close() throws IOException;
void flush() throws IOException;
void write(byte[] b) throws IOException;
void write(byte[] b, int off, int len) throws IOException;

BufferedOutputStream(OutputStream out);
void write(byte[] b, int off, int len) throws IOException;
void write(int b) throws IOException;
void close() throws IOException;
void flush() throws IOException;

PrintWriter (OutputStream out);
PrintWriter(OutputStream out, boolean autoFlush);
void close() throws IOException;
void print(String s);
void println(String x);
void println(boolean x);
void println(char x);
void println(char[] x);
void println(double x);
void println(float x);
void println(int x);
void println(long x);
void flush() throws IOException;

InputStream();
void close();
int read(byte[] b) throws IOException;

InputStreamReader(InputStream in);

BufferedReader(Reader in);
String readLine();
```