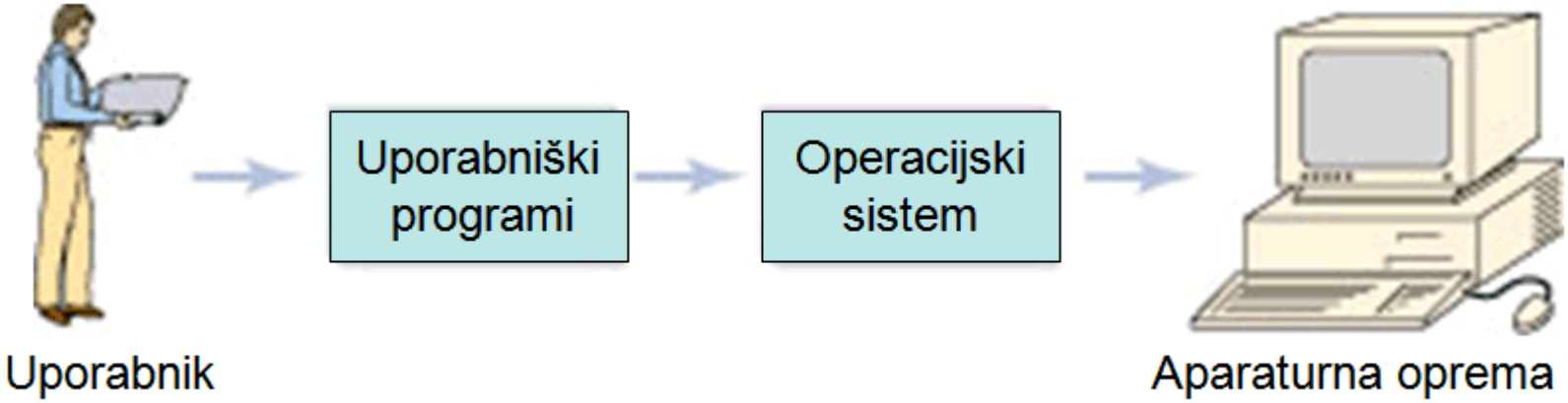
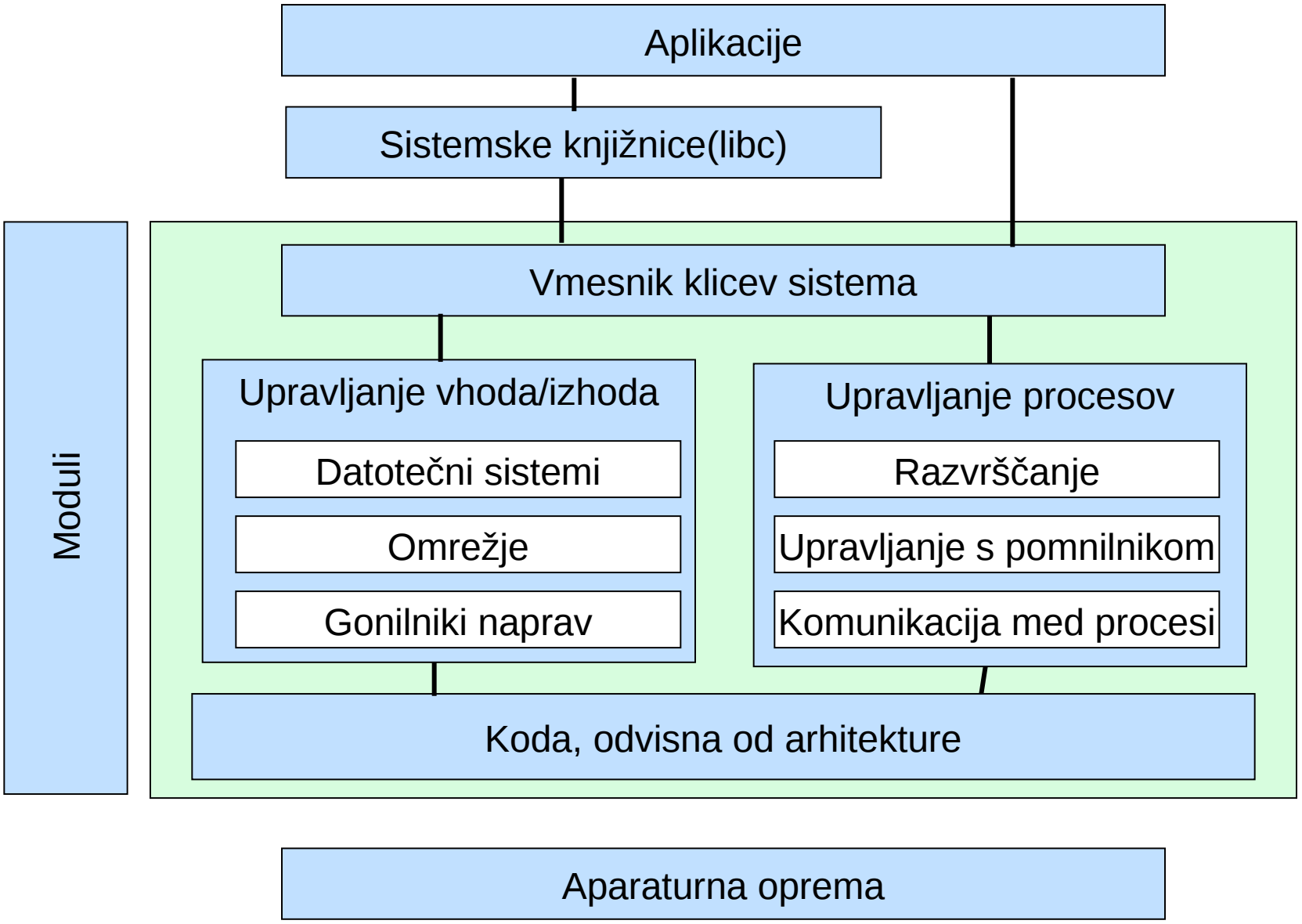
# Sistemsko programiranje

# Operacijski sistem računalnika

- Zbirka programov, ki krmilijo osnovne operacije računalniške aparaturne opreme
- Cilj – računalniške operacije naj bodo za uporabnika transparentne



# Komponente sistema LINUX



# Sistemski klici za delo z datotekami

| System call | Description |
|---|---|
| fd = creat(name, mode) | One way to create a new file |
| fd = open(file, how, ...) | Open a file for reading, writing or both |
| s = close(fd) | Close an open file |
| n = read(fd, buffer, nbytes) | Read data from a file into a buffer |
| n = write(fd, buffer, nbytes) | Write data from a buffer into a file |
| position = lseek(fd, offset, whence) | Move the file pointer |
| s = stat(name, &buf) | Get a file's status information |
| s = fstat(fd, &buf) | Get a file's status information |
| s = pipe(&fd[0]) | Create a pipe |
| s = fcntl(fd, cmd, ...) | File locking and other operations |

- **s** je koda napake
- **fd** je opisnik datoteke
- **position** je položaj v datoteki

# Delo z datotekami

**Delček kode, ki ponazarja tipično zaporedje dogodkov:**

```
int fd; /*File descriptor */
...
fd = open (fileName, ...); /* Open file, return file desciptor */
    if (fd == -1) { /* Set some error condition */ }
...
read (fd, ...); /* Read from file */
...
write (fd, ...); /* Write to file */
...
lseek (fd, ...); /* Seek within file */
close (fd); /* Close the file, freeing file descriptor */
```

# Sistemski klici za zaščito datotek

| System call | Description |
|---|---|
| s = chmod(path, mode) | Change a file's protection mode |
| s = access(path, mode) | Check access using the real UID and GID |
| uid = getuid( ) | Get the real UID |
| uid = geteuid( ) | Get the effective UID |
| gid = getgid( ) | Get the real GID |
| gid = getegid( ) | Get the effective GID |
| s = chown(path, owner, group) | Change owner and group |
| s = setuid(uid) | Set the UID |
| s = setgid(gid) | Set the GID |

- **s** je koda napake
- **uid** in **gid** sta identifikatorja uporabnika in skupine

# Sistemski klici za upravljanje z direktoriji

| System call | Description |
|---|---|
| s = mkdir(path, mode) | Create a new directory |
| s = rmdir(path) | Remove a directory |
| s = link(oldpath, newpath) | Create a link to an existing file |
| s = unlink(path) | Unlink a file |
| s = chdir(path) | Change the working directory |
| dir = opendir(path) | Open a directory for reading |
| s = closedir(dir) | Close a directory |
| dirent = readdir(dir) | Read one directory entry |
| rewinddir(dir) | Rewind a directory so it can be reread |

- **s** je koda napake
- **dir** identificira direktorij
- **dirent** je vhod direktorija

# Nekaj sistemskih klicev za različne naloge

**Miscellaneous**

| Call | Description |
|---|---|
| s = chdir(dirname) | Change the working directory |
| s = chmod(name, mode) | Change a file's protection bits |
| s = kill(pid, signal) | Send a signal to a process |
| seconds = time(&seconds) | Get the elapsed time since Jan. 1, 1970 |

# Primer: c simulacija linux ukaza "cd"

```
#include<stdio.h>
#include<unistd.h>

   main(int argc,char **argv)    {
      if (argc < 2) {
         printf("Usage: %s   <pathname> \n",argv[0]);
         exit(1);
    } if (chdir(argv[1]) != 0) {
        printf(``Error in chdir n"); exit(1);
    }
}
```

# Upravljanje programskih procesov

**int fork()**

tvorba novega procesa (otroka), ki je (ob rojstvu) enak svojemu očetu, od katerega se razlikuje le po svoji, specifični procesni številki (PID). Ker od trenutka rojstva oba procesa (oče in otrok) živita ločeno, lahko spoznata, kdo je kdo (oče ali otrok) po tem, da očetu vrne klic fork vrednost PID procesa-otroka, otroku pa vrne vrednost 0.

**getpid()** vrne PID kličočega procesa,

**getppid()** vrne PID njegovega očeta.

**int wait( int * status)**

Status je kazalec na celoštevilčno vrednost, v katero shrani UNIX vrednost, ki jo vrača ob svojem koncu proces-otrok.

**void exit (int status)**

Pomen spremenljivke status smo že pojasnili pri opisu klica wait().

**int execv(char * fileName, *argv[ ])**

*fileName* je ime izvršljive (programske) datoteke, ki naj se transformira v proces. V polju *arg* pa so argumenti, ki jih temu procesu posredujemo.

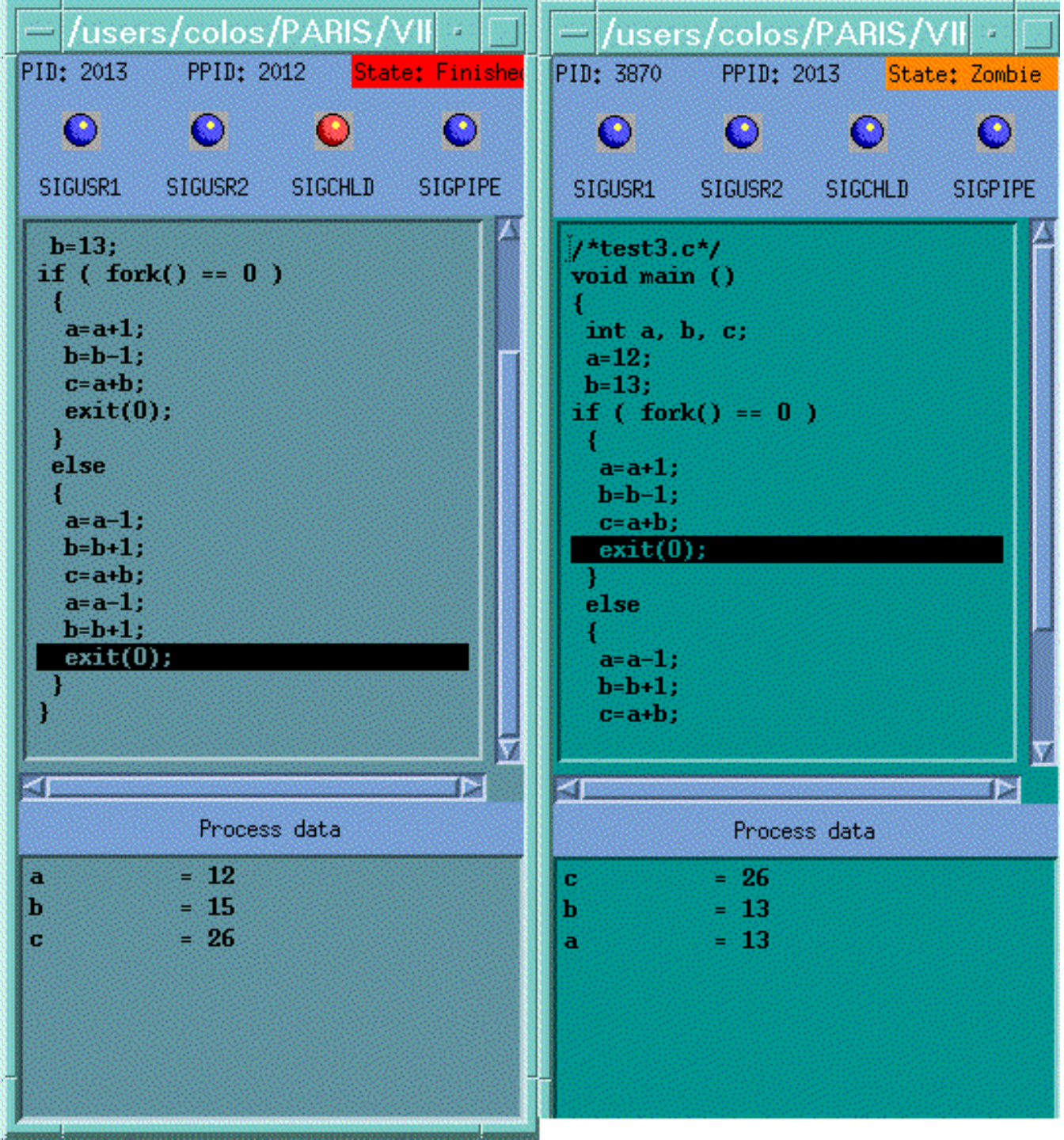
# Sistemski klici za upravljanje procesov

| System call | Description |
|---|---|
| pid = fork( ) | Create a child process identical to the parent |
| pid = waitpid(pid, &statloc, opts) | Wait for a child to terminate |
| s = execve(name, argv, envp) | Replace a process' core image |
| exit(status) | Terminate process execution and return status |
| s = sigaction(sig, &act, &oldact) | Define action to take on signals |
| s = sigreturn(&context) | Return from a signal |
| s = sigprocmask(how, &set, &old) | Examine or change the signal mask |
| s = sigpending(set) | Get the set of blocked signals |
| s = sigsuspend(sigmask) | Replace the signal mask and suspend the process |
| s = kill(pid, sig) | Send a signal to a process |
| residual = alarm(seconds) | Set the alarm clock |
| s = pause( ) | Suspend the caller until the next signal |

**s** je koda napake

**pid** je identifikator procesa

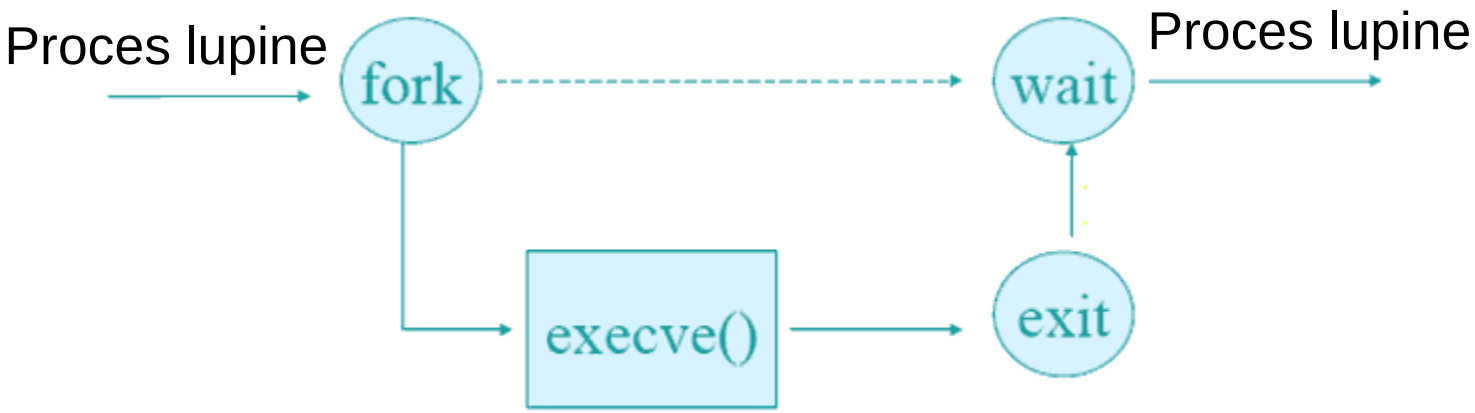
**residual** je preostali čas od zadnjega alarma




Animirana demonstracija

# Kako deluje lupina LINUX

- Interpreter ukazne vrstice
- Vgrajeni ukazi
- Zunanji ukazi




# Poenostavljena koda lupine

```
while (TRUE) {                                  /* repeat forever /*/
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, params);          /* read input line from keyboard */

        pid = fork( );                          /* fork off a child process */
        if (pid < 0) {
                printf("Unable to fork0);       /* error condition */
                continue;                       /* repeat the loop */
        }

        if (pid != 0) {
                waitpid (–1, &status, 0);       /* parent waits for child */
        } else {
                execve(command, params, 0);     /* child does the work */
        }
}
```

# Procesni signali

**signal (signame,SIG_IGN);**
**signal (SIGINT, funkcija)**

| | |
|---|---|
| SIGHUP | Ta signal je posredovan procesom, katerih terminal je bil izklopljen. |
| SIGINT | Prekinitveni zahtevek s tastature terminala |
| SIGILL | Nelegalna instrukcija |
| SIGFPE | *" Floating point"* napaka: delitev z 0, prekoračitev in podobno |
| SIGKILL | *Kill*. Ta signal lahko ignoriramo, ujamemo ali blokiramo. |
| SIGSYS | Napačen argument v sistemskem klicu |
| SIGPIPE | Pisanje v cev, ki je nihče ne bere |
| SIGALRM | Signal " alarmne ure" |

# Kontrola časa

**localtime()** Vrne lokalni čas.

```
longint t;
 ......
time(t);
printf(" time:\%s" ,asctime( localtime (t)));
```

**sleep( int secs)**
Proces bo zaspal za dano število sekund

**alarm(int secs),**
Čez koliko sekund dvignemo alarmni signal *SIGALRM*.

S klicem ***alarm(0),*** torej z argumentom z vrednostjo 0, nastop alarma preprečimo.

# Primer uporabe funkcije time in localtime

```
 #include <time.h>
#include <stdio.h>
#define SIZE 256

    int main (void)   {
      char buffer[SIZE];
      time_t curtime;
      struct tm *loctime;

      /* Get the current time. */
      curtime = time (NULL);

      /* Convert it to local time representation. */
      loctime = localtime (&curtime);

      /* Print out the date and time in the standard format. */
      fputs (asctime (loctime), stdout);
      /* Print it out in a nice format. */
      strftime (buffer, SIZE, "Today is %A, %B %d.\n", loctime);
      fputs (buffer, stdout);
      strftime (buffer, SIZE, "The time is %I:%M %p.\n", loctime);
      fputs (buffer, stdout);
      return 0;
    }
```

Kakšen bo izpis:

```
Wed Jul 31 13:02:36 1991
Today is Wednesday, July
31. The time is 01:02 PM.
```

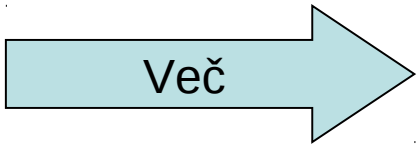
# Podatki o uporabnikih

**getlogin()** Vrne vstopno (login) ime uporabnika

**getuid()** Vrne identiteto (UID) uporabnika procesa

**setuid()** Spreminjanje identitete (UID) uporabnika

**getuid()** Vrne ident skupine (grupe) uporabnikov



# Primer getlogin()

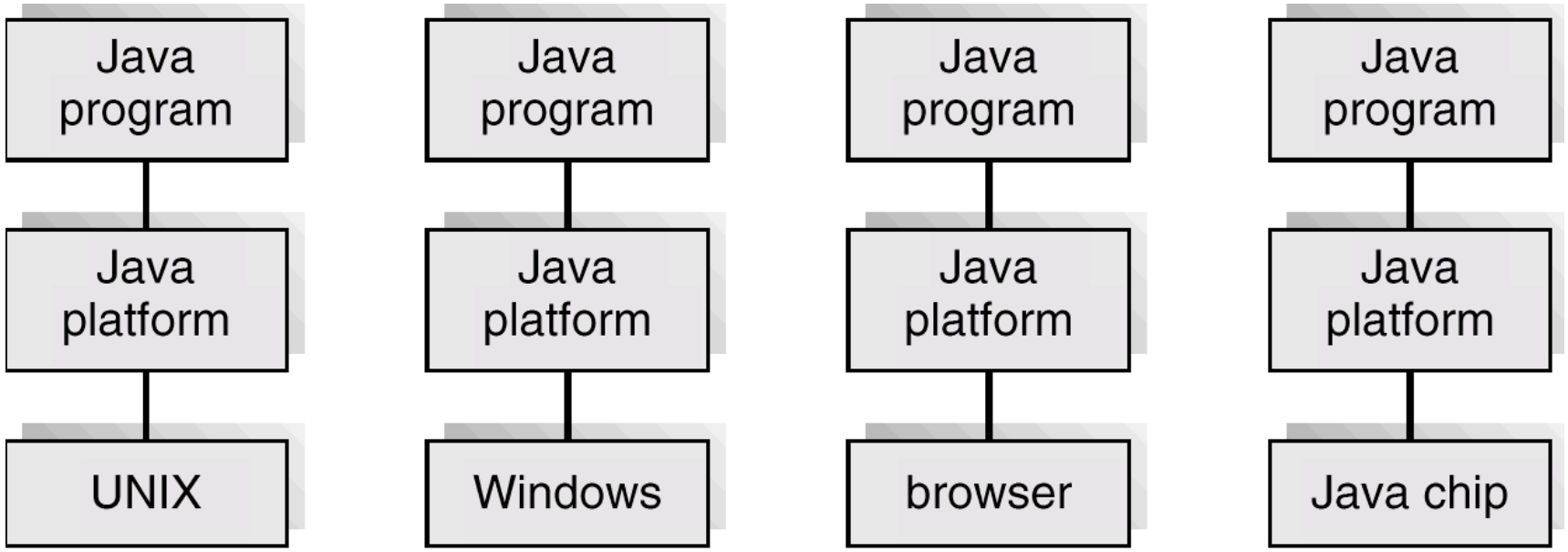
```
/*  implementing who am i using system calls  */

#include<stdio.h>
#include<utmp.h>
int main()  {
        char *s,*c;
        struct utmp *u;
        int i;
        c=getlogin();
        setutent();
        u=getutent();
        while(u!=NULL)  {
           if(u->ut_type==7 && strcmp(u->ut_user,c)==0) {
                printf("%-12s",u->ut_user);
                printf("%-9s",u->ut_line);
                s=ctime(&u->ut_time);
                for(i=4;i<16;i++)
                printf("%c",s[i]);
                printf("(%s",u->ut_host);
                printf(") ");
           }
        u=getutent();
        }
}
```

# Primerjava sist.klicev UNIX in Win32

| UNIX | Win32 | Description |
|---|---|---|
| fork | CreateProcess | Create a new process |
| waitpid | WaitForSingleObject | Can wait for a process to exit |
| execve | (none) | CreateProcess = fork + execve |
| exit | ExitProcess | Terminate execution |
| open | CreateFile | Create a file or open an existing file |
| close | CloseHandle | Close a file |
| read | ReadFile | Read data from a file |
| write | WriteFile | Write data to a file |
| lseek | SetFilePointer | Move the file pointer |
| stat | GetFileAttributesEx | Get various file attributes |
| mkdir | CreateDirectory | Create a new directory |
| rmdir | RemoveDirectory | Remove an empty directory |
| link | (none) | Win32 does not support links |
| unlink | DeleteFile | Destroy an existing file |
| mount | (none) | Win32 does not support mount |
| umount | (none) | Win32 does not support mount |
| chdir | SetCurrentDirectory | Change the current working directory |
| chmod | (none) | Win32 does not support security (although NT does) |
| kill | (none) | Win32 does not support signals |
| time | GetLocalTime | Get the current time |

# Java in delo s sistemskimi paketi



# Java 1.4 Paketi

- java.applet
- java.awt (*)
- java.beans (*)
- java.io
- java.lang (*)
- java.math
- java.net
- java.nio (*)
- java.rmi (*)
- java.security (*)
- java.sql
- java.text
- java.util (*)
- javax.accessibility
- javax.crypto (*)
- javax.imageio (*)
- javax.naming (*)
- javax.net (*)
- javax.print (*)
- javax.rmi (*)
- javax.security (*)
- javax.sound (*)
- javax.sql
- javax.swing (*)
- javax.transaction (*)
- javax.xml (*)
- org.ietf.jgss
- org.omg.CORBA (*)
- org.omg.CosNaming (*)
- org.omg.Dynamic (*)
- org.omg.IOP (*)
- org.omg.Messaging
- org.omg.PortableInterceptor (*)
- org.omg.PortableServer (*)
- org.omg.SendingContext
- org.omg.stub.java.rmi
- org.w3c.dom
- org.xml (*)

# Najbolj važni (core) javanski paketi

- **java.lang**
  - Osnovni razredi za načrtovanje programskega jezika Java. Implicitno ga rabijo vsi drugi paketi.
- **java.util**
  - uslužnostni razredi, delo s časom, internaciolnalizacija,..
- **java.io**
  - Sistemski vhod ni in izhodni tokovi, serializacija datotečnega sistema.
- **java.math**
  - Razredi za izvajanje natančnih (BigInteger) celoštevilčnih in decimalnih aritmetičnih operacij
- **java.sql**
  - API za dostop do podatkov, hranjenih v relacijski podatkovni bazi in njihovo obdelavo
- **java.text**
  - razredi in vmesniki za rokovanje z besedili, datumi, števili in obvestili na način, neodvisen od naravnega jezika