Operacijski sistemi

Procesi – API

Jurij Mihelič, FRI, Uni LJ

Vsebina

- Windows
- Unix/Linux

Windows

- Windows procesni API
  - CreateProcess()
  - ExitProcess()
  - TerminateProcess(proces, status)
  - GetExitCodeProcess(proces, status)
  - WaitForSingleObject()

Windows

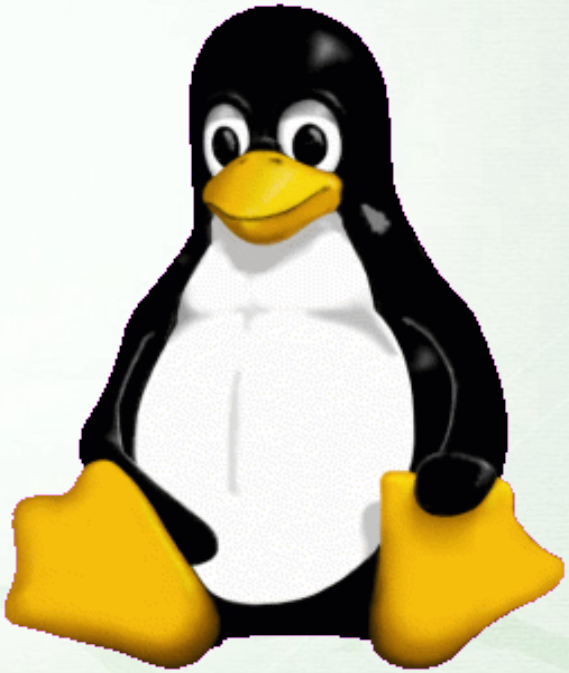
- Stvaritev procesa: CreateProcess(…)
  - CreateProcess(
    - ime programa, NULL … uporabi ukaz
    - ukazna vrstica, argv[1]
    - atributi procesa, NULL … brez dedovanja
    - atributi niti, NULL … brez dedovanja
    - dedovanje ročajev, FALSE .. brez dedovanja
    - zastavice, 0 … brez zastavic
    - okolje, NULL … okolje starša
    - trenutni imenik NULL … imenik starša
    - zagonske informacije, &si … STARTUPINFO
    - procesne informacije &pi … PROCESS_INFORMATION
  -)

Windows

- Končanje procesa
  - ExitProcess(status)
  - TerminateProcess(proces, status)
  - GetExitCodeProcess(proces, status)
- Čakanje procesa
  - WaitForSingleObject(handle, miliseconds)

Unix-podobni sistemi

- Procesni API na Unix-podobnih sistemih
  - info o procesu
  - ustvarjanje procesov
  - končanje procesov
  - čakanje procesov
  - …



Unix-podobni sistemi

- Info o procesu
  - PID procesa: int getpid()
  - PPID procesa: int getppid()
  - UID procesa: int getuid()
  - GID procesa: int getgid()

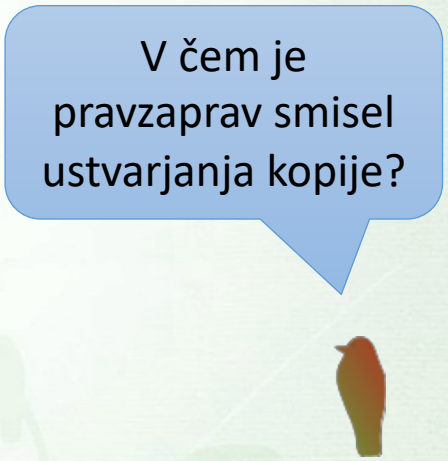
Unix-podobni sistemi

- Ostalo
  - Spanje: int sleep(unsigned int seconds)
    - lahko se zbudi prej, če prejme signal
  - Časi procesa: clock_t times(struct tms * buf)

```
struct tms {
    clock_t tms_utime;  /* user time */
    clock_t tms_stime;  /* system time */
    clock_t tms_cutime; /* user time of children */
    clock_t tms_cstime; /*  system time of children */
};
```

Unix-podobni sistemi

- Stvaritev procesa – fork
  - kopiranje trenutnega procesa
    - starš ustvari nov proces – otroka
  - otrok je **kopija** starša
    - otrok ima svoj nov deskriptor procesa
      - večina podatkov se kopira
      - različni PID, PPID
      - iste odprte datoteke
      - različne ključavnice
    - enak naslovni prostor
      - ista koda (in rokovalniki signalov))
      - enaki podatki, sklad, kopica (vendar kopija)



Unix-podobni sistemi

- Stvaritev procesa – fork
  - sistemski klic: int fork()
    - v primeru neuspeha vrne -1
    - sicer pa se iz funkcije vrneta dva procesa
      - proces starš, kateremu vrne PID otroka
      - proces otrok, kateremu vrne 0

```
int pid = fork();
if (pid > 0) {
        // STARŠ
} else if (pid == 0) {
        // OTROK
} else { // pid < 0
        // NEUSPEH
}
```

Unix-podobni sistemi

- Končanje procesa – exit
  - sistemski klic: exit(int status)
    - proces se zaključi s podanim izhodnim statusom
    - jedro sprosti vire končanega procesa
      - pomnilnik, zapre datoteke itd.
    - otroke končanega procesa posvoji proces **init**
      - posvojenim otrokom pravimo tudi **sirote** (orphan)
    - funkcija, iz katere se nikoli ne vrnemo

Unix-podobni sistemi

- Končanje procesa – exit
  - izhodni status končanega procesa
    - 8 bitna vrednost
    - 0 pomeni uspešen zaključek programa
    - 1 … 127 koda napake, neuspešen zaključek
    - 128 … 255 zaključek zaradi signala
      - številka signala = status - 127

Unix-podobni sistemi

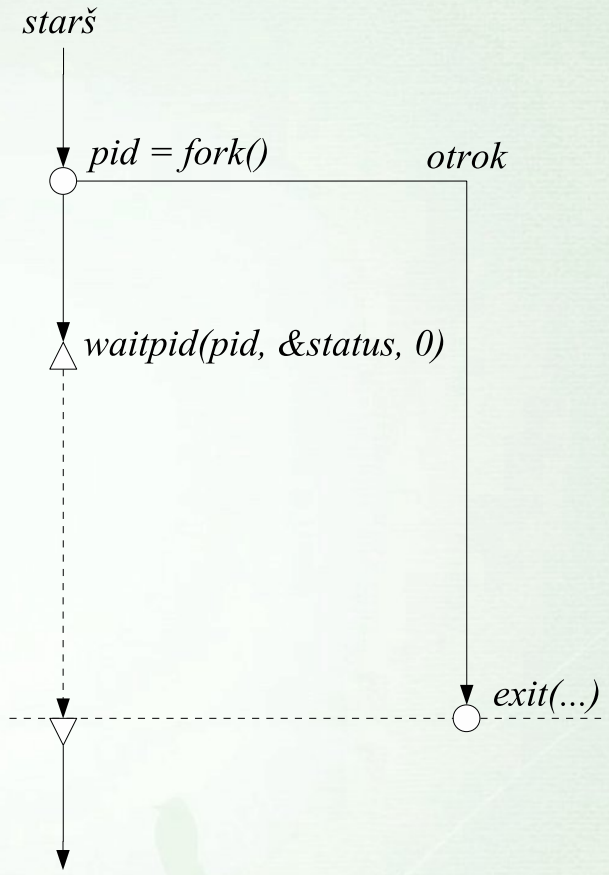
- Končanje procesa – exit
  - izhodni statusa končanega procesa
    - izhodni status se shrani v deskriptorju procesa dokler ga ne prevzame **otrokov starš**
    - dokler status ni prevzet je otrok **zombi**
      - proces je končan, ne zaseda več virov
      - razen njegov deskriptor procesa še ni sproščen
      - zombijev ne želimo imeti veliko v sistemu
    - prevzem statusa končanega otroka je potrebno posebej sprogramirati
    - kako starš ve, da se je otrok končal?
      - staršu jedro pošlje signal SIGCHLD

Unix-podobni sistemi

- **Prevzem** izhodnega status otroka – wait
  - in **čakanje** na dokončanje, če še ni končan
  - čakanje na določenega otroka
    - int waitpid(pid, &status, opcije)
    - glej tudi wait4()
  - čakanje na poljubnega otroka
    - int wait(&status)
      - enako kot waitpid(-1, &status, 0)
    - glej tudi wait3()
  - izhodni status (skrit) v spremenljivki status
    - uporabljaj makroje

Unix-podobni sistemi

- Časovni diagram procesov
  - fork
    - stvaritev otroka
    - otrok dobi vse podatke od starša
  - exit
    - oddaja statusa staršu
  - wait
    - prejem statusa



Unix-podobni sistemi

- Zagon programa – exec
  - program je shranjen v izvršljivi datoteki
    - Linux: ELF – executable and linkable format
  - nadomestitev trenutnega procesa
    - nov naslovni prostor
      - koda, sklad, kopica, podatki
    - trenutni deskriptor procesa se ponastavi
      - ohrani se PID, PPID
      - ohranijo se odprte datoteke
      - ohranijo se trenutni in korenski imenik, ipd.
  - preko klica exec lahko podamo tudi
    - argumente
    - okoljske spremenljivke

Unix-podobni sistemi

- Zagon programa – exec
  - argumenti programa oz. procesa
    - seznam nizov vključno z imenom programa
    - int main(int argc, char **argv)
  - okoljske spremenljivke
    - seznam imen spremenljivk in njihovih vrednosti
    - int main(int argc, char** argv, char **envp)
  - prenos informacije je enosmeren
    - od starega procesa, ki kliče exec
    - do novega procesa, ki je nadomestil starega
    - spreminjanje argumentov in okoljskih spremenljivk se ne odraža na ostalih procesih, tudi na staršu ne

Unix-podobni sistemi

- Zagon programa – exec
  - družina funkcij: exec[lv][pPe]?(…)
    - podamo pot do izvršljive datoteke oz. ukaza
    - pripone
      - l – argumenti ukaza so podani kot argumenti funkcije
      - v – argumenti ukaza so podani posebej v tabeli
      - p – uporaba $PATH pri iskanju izvršljive datoteke
      - P – podamo tudi iskalno pot
      - e – okoljske spremenljivke so podane v tabeli

funkcija
execl(exe, arg0, …)
execlp(exe, arg0, …)
execle(exe, arg0, …, 0, envp)
execv(exe, args)
execvp(exe, args)
execvP(exe, path, args)
execve(exe, args, envp)

Unix-podobni sistemi

- Zagon programa – exec
  - primeri

```
execl("/bin/ls", "ls", "-alp", "/home/jure", NULL);

char* args[] = { "ls", "-alp", "/home/jure", NULL };
execvp("ls", args);

char* envp[] = { "FRI=cool","OS=even cooler", NULL}
execve("ls", args, envp)

execvp(argv[1], &argv[1]);

execlp("ls", "proggy");
```

Unix-podobni sistemi

- Prednosti Unix pristopa – fork & exec
  - preprosto ustvarjanje procesa
  - ločitev ustvarjanja in nalaganja procesa
    - fork() & exec()
  - možnost izvajanja kode po fork() in pred exec()
    - ustvarimo otroka – fork
    - nastavimo vse potrebno
      - odprte datoteke, preusmeritve itd.
    - naložimo nov (otroški) program – exec
    - pomembno za izvedbo lupine

Unix-podobni sistemi

- Slabosti Unix pristopa – fork & exec
  - kopiranje procesov je neučinkovito
    - potrebno je kopirati celoten naslovni prostor
  - optimizacija: vfork()
    - zakaj bi kopirali, če bo takoj sledil exec()
    - vfork() je različica fork(), kjer se pričakuje, da se bo takoj zatem izvedel še exec()
  - optimizacija: COW (copy-on-write)
    - leno kopiranje
    - pomnilniškega prostora ne kopiramo takoj, ampak šele po potrebi, ko pride do prvega pisanja (v staršu ali otroku)

Glej tudi: https://www.microsoft.com/en-us/research/uploads/prod/2019/04/fork-hotos19.pdf