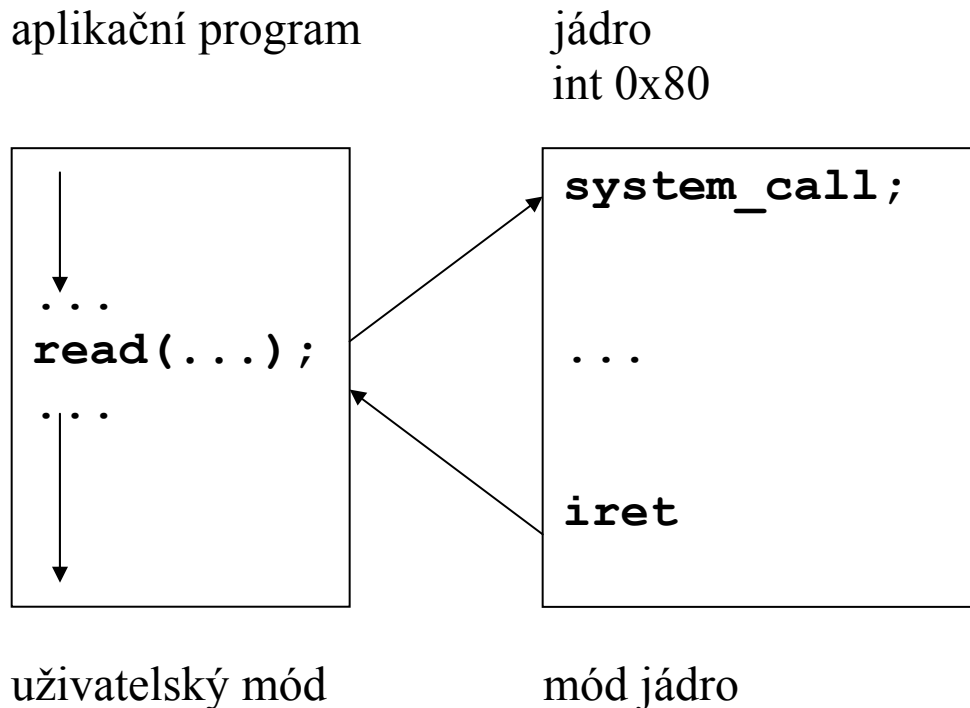


Proces a jádro

co je jádro?

je jádro proces?



vykonávaný program – proces = posloupnost instrukcí aplikačního (uživatelského) programu a jádra

jádro – speciální program zavedený do hlavní paměti při startu systému přímo vykonávaný HW

virtuální adresový prostor procesu
adresový prostor procesu (uživatelský)
systémový prostor (jádra)

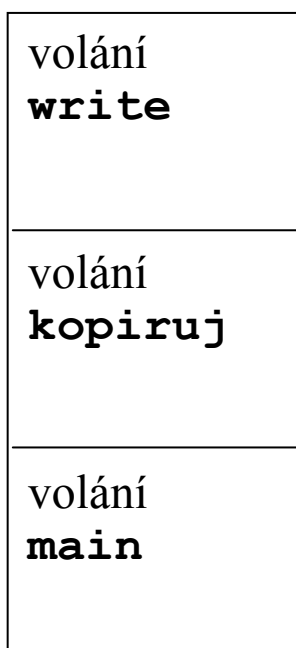
proces v uživatelském módu má přístup ke svému adresovému prostoru, k systémovému prostoru voláním `system_call()`

jádro má přístup k adresovému prostoru procesů

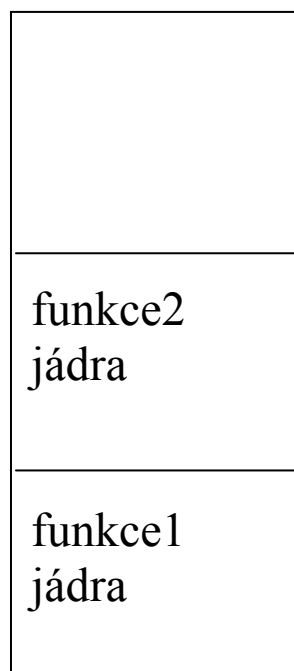
proces používá dva zásobníky

- uživatelský
- jádra

uživatelský
zásobník

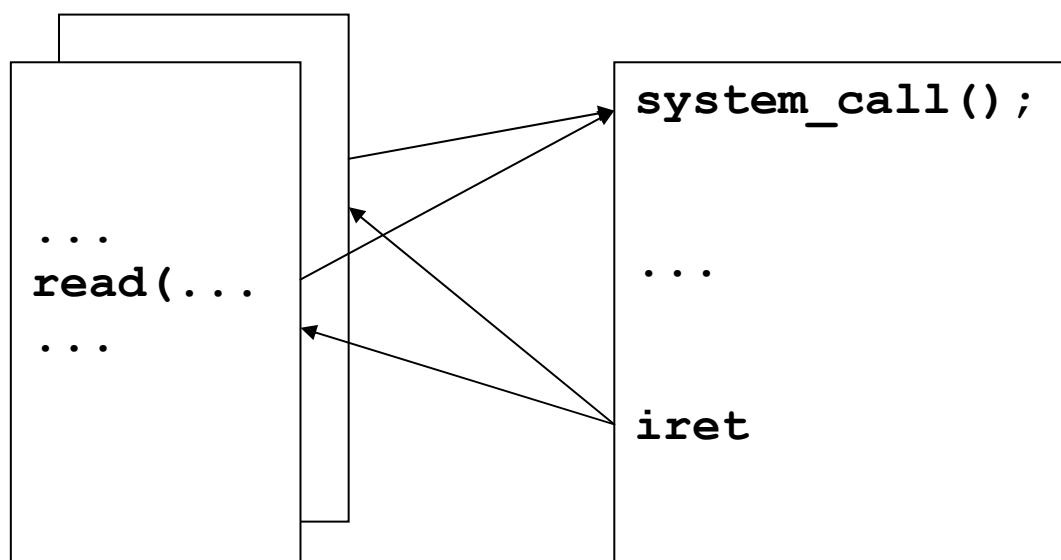


zásobník
jádra



↑
růst
zásobníku

více procesů



jádro je reentrantní

každý proces má svůj zásobník jádra, často v adresovém prostoru procesu – chráněný, spravovaný jádrem

každý proces má položku v tabulce procesů **proc** záznam a u (*user*) oblast

u oblast – údaje potřebné když je proces vykonávaný

- tabulku deskriptorů souborů otevřených souborů
- okamžitý adresář
- kořenový adresář
- často zásobník jádra procesu

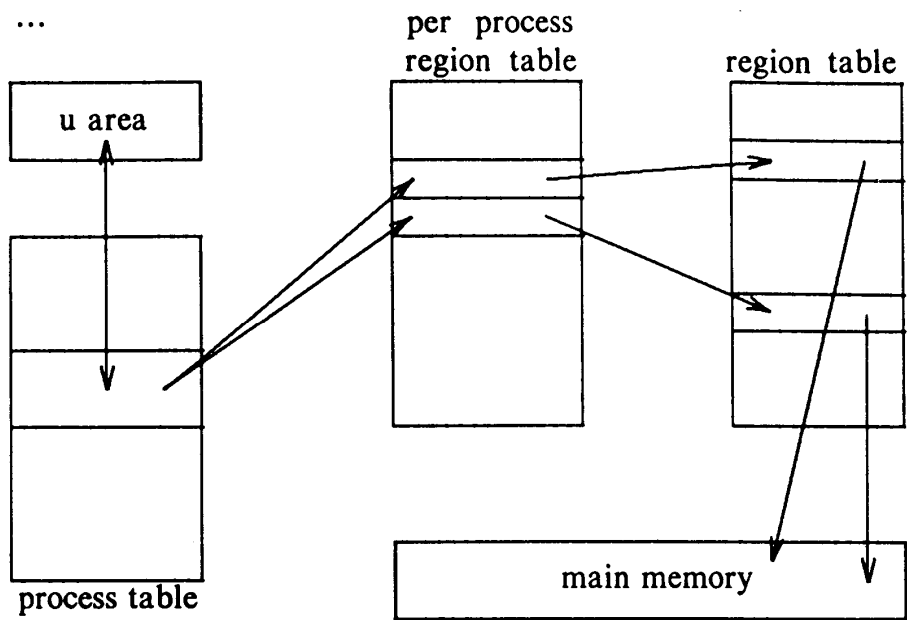
...

v adresovém prostoru procesu – chráněná, spravovaná jádrem

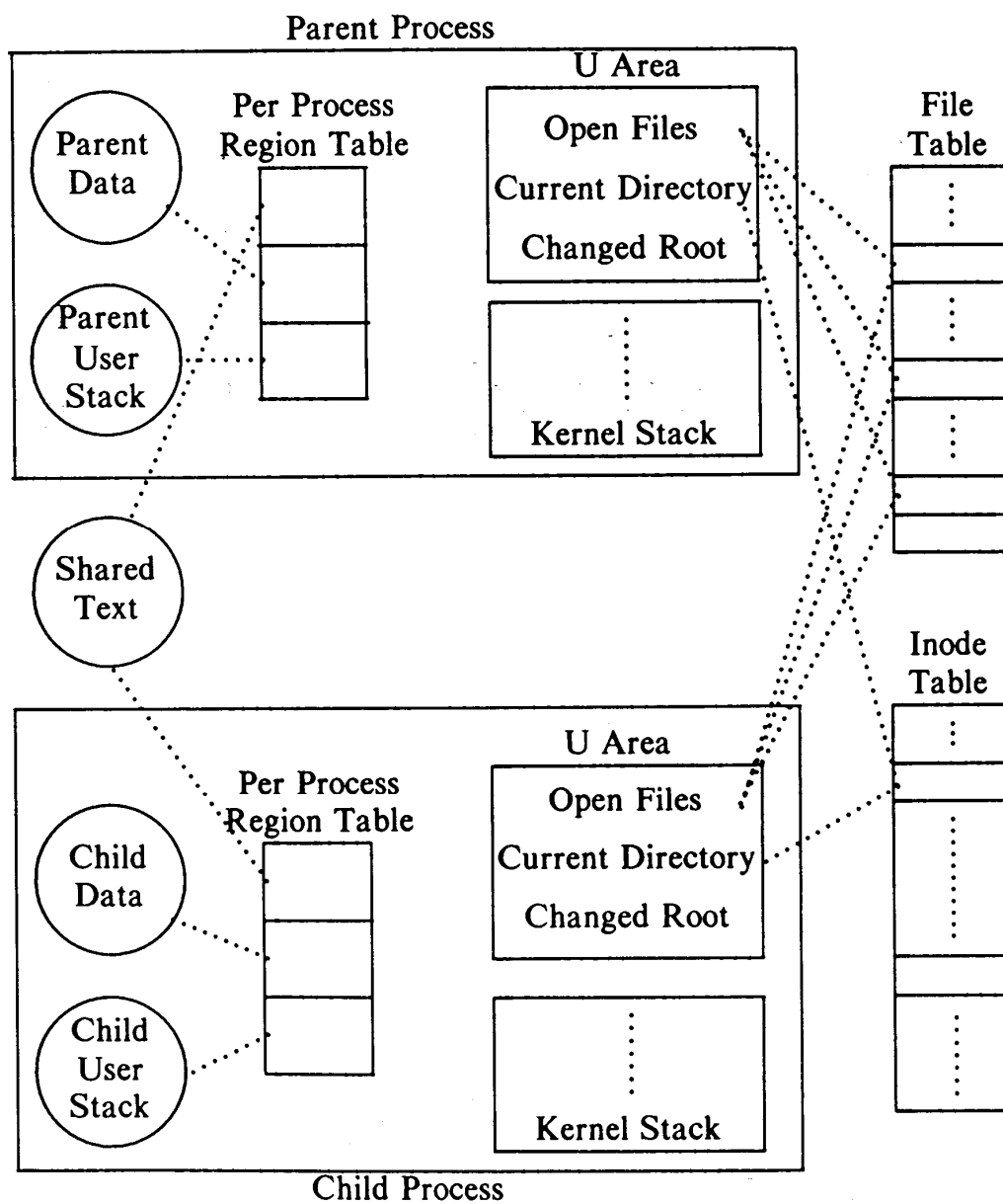
proc záznam

- tabulka oblastí (*region*) procesu
- stav procesu

...



vykonání **fork**



Fork Creating a New Process Context

Kontext procesu - jeho opis (stav)

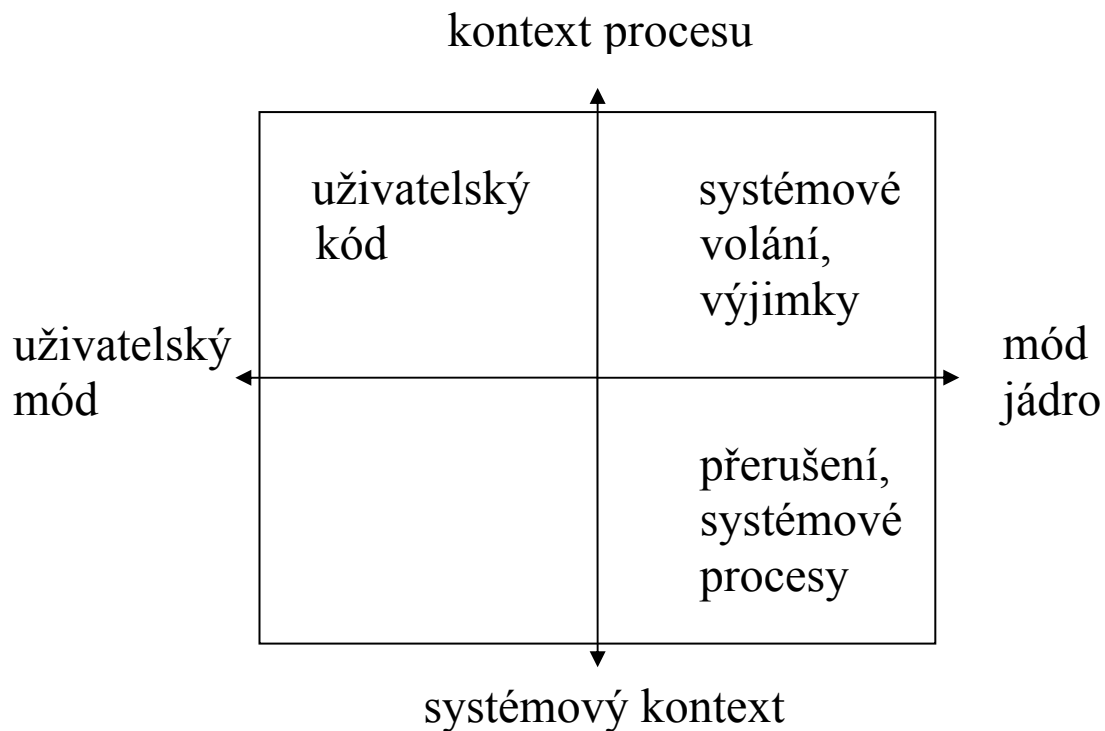
- uživatelský adresový prostor
 - text (vykonatelný kód)
 - data
 - uživatelský zásobník
- řídicí informace
 - u oblast
 - **proc** záznam
 - zásobník jádra
- registry (HW kontext)
 - počítadlo instrukcí
 - ukazatel zásobníku
 - stavové slovo procesoru
 - mód
 - úroveň přerušovací priority
 - přetečení
 - ...
 - registry pro správu paměti
 - registry jednotky pohyblivé čárky

jádro

- vykonává služby
- zpracovává výjimky (pokus dělit nulou, přetečení uživatelského zásobníku, ...)
- zpracovává přerušení od periferních zařízení
- vykonává systémové procesy (správa paměti, přepočítávání priorit procesů)

jádro pracuje

- v kontextu procesu
- v systémovém kontextu



uživatelský kód – přístup jenom k (uživatelskému) adresovému prostoru procesu

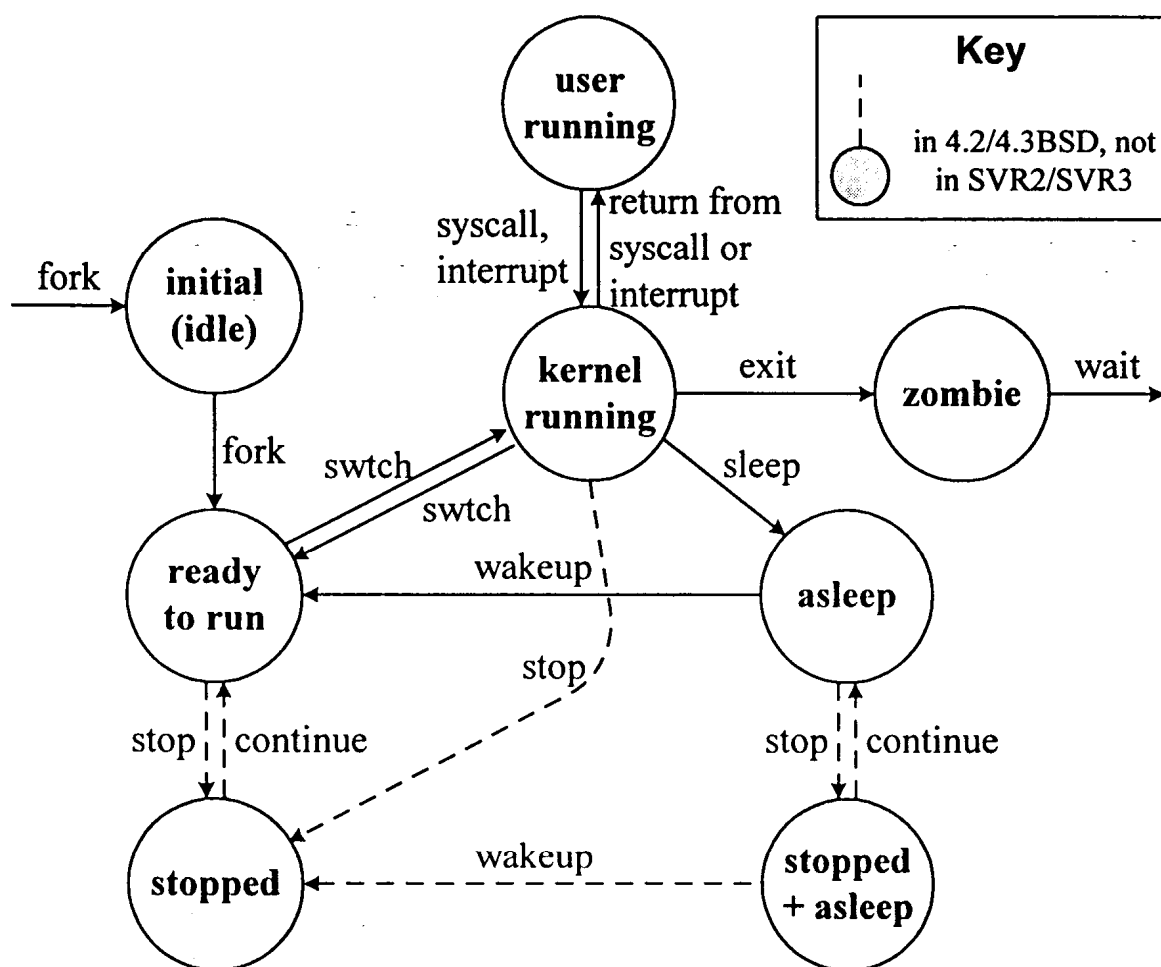
systémové volání, výjimky – přístup k uživatelskému i systémovému adresovému prostoru

přerušení, systémové procesy – přístup jenom k systémovému adresovému prostoru

Stavy procesu

v každém okamžiku se proces nachází v nějakém definovaném stavu

přechody mezi jednotlivými stavy znázorňuje diagram stavových přechodů



Process states and state transitions.

[Vahalia]

začáteční (*initial/idle*) – **fork** začal vytváření procesu

připraven na vykonání (*ready to run*) – proces čeká až bude naplánován na přidělení procesoru

běžící v jádře (*kernel running*) – byl naplánován, vykoná se přepnutí kontextu, procedura jádra **swtch ()** uloží HW kontext do registrů

běžící uživatelsky (*user running*) – může přejít do stavu běžící v jádře v důsledku volání služby jádra nebo přerušení, po skončení obsluhy se vrátí

spící (*asleep*) – při vykonávání systémového volání se může stát, že je nutno čekat na nějakou událost nebo prostředek, proces (v jádře) zavolá proceduru **sleep ()**, když událost nastane, jádro vzbudí proces a proces se stane **připraven na vykonání** a po naplánování pokračuje obsluha systémového volání ve stavu **běžící v jádře**

připraven na vykonání se může stát je-li běžící po uplynutí přiděleného časového kvanta, vykoná se preempce běžícího procesu a to ve stavu **běžící uživatelsky** nebo při návratu do něj, jádro je nepreemptivní

Linux 2.6, Solaris – preemptivní

Main Entry: **pre·emp·tion** 🗣️

Pronunciation: – 'em (p) –sh&n

Function: *noun*

Etymology: Medieval Latin *praeemption-*, *praeemptio* previous purchase, from *praeemere* to buy before, from Latin *prae-* pre- + *emere* to buy -- more at [**REDEEM**](#)

Date: 1602

1 a : the right of purchasing before others; *especially* : one given by the government to the actual settler upon a tract of public land **b** : the purchase of something under this right

2 : a prior seizure or appropriation : a taking possession before others

[m-w.com]

přerušení se může vyskytnout i ve stavu **běžící v jádře**, kdy po skončení obsluhy proces zůstane ve stavu **běžící v jádře**

proces končí voláním **exit** anebo v důsledku signálu, přechází do stavu **mátoha** (zombie), dokud rodič nevykoná **wait**

do stavu **zastaven (stopped)** nebo **spící a zastaven** přejde proces po stop signálech

- SIGSTOP zastav proces
- SIGTSTP CTRL-Z
- SIGTTIN tty čtení procesu v pozadí
- SIGTTOU tty psaní procesu v pozadí

signál SIGCONT převede proces do stavu **připraven na vykonání** nebo do stavu **spící**

+ **Linux**

Oprávnění uživatele (*user credentials*)

každý uživatel má v systému identifikován číslem,
identifikátor uživatele (user identifier) UID a obdobně
identifikátor skupiny (group identifier) GID

identifikátory ovlivňují vlastnictví vytvářených souborů,
používají se na kontrolu přístupových práv a kontrolu zasílání
signálů jiným procesům

procesy dědí oprávnění

privilegovaný uživatel *superuser*
UID 0, GID 1

nově, rozdělení privilegií (Linux zatím v jádře)

IEEE Std 1003.1-2001

každý proces má

reálný UID (*real UID*)

efektivní UID (*effective UID*)

uložený nastavovací UID (*saved set UID*)

reálný UID identifikuje reálného uživatele a ovlivňuje právo
posílat signály

efektivní UID ovlivňuje vlastnictví souborů a přístup k
souborům

proces změny efektivní UID

- vykoná-li **exec** programu s nastaveným bitem SUID
- systémovým voláním **setuid**

bit SUID, nastavení UID (set UID), je jeden z bitů „přístupových“ práv k souboru

- je-li nastaven, efektivní UID a uložený nastavený UID se nastaví na ID vlastníka souboru programu

setuid(uid)

- je-li okamžitý efektivní UID procesu privilegovaný uživatel, potom reálný UID, efektivní UID a uložený nastavovací UID se nastaví na **uid**
- jinak **setuid(uid)** nastaví efektivní UID na hodnotu **uid**, je-li **uid** rovno reálnému UID nebo uloženému nastavovacímu UID, reálný UID a uložený nastavovací UID se nezmění

| | setuid(e) | |
|-------------------------|------------------|----------------|
| | euid==0 | euid!=0 |
| reálný UID | e | nezměněn |
| efektivní UID | e | e |
| uložený nastavovací UID | e | nezměněn |

- následující program po přeložení vlastní „on“, UID == 800, má nastavený bit SUID, právo vykonávat mají všichni
- uživatel „on“ vlastní soubor „on“ s právem čtení jenom pro vlastníka
- uživatel „já“ , UID == 500 vlastní soubor „ja“ s právem čtení jenom pro vlastníka

```
main()  
{  
    int uid, euid, fdja, fdon;  
  
    uid = getuid();    /*reálný*/  
    euid = geteuid(); /*efektivní*/  
    printf("uid %d euid %d\n",  
           uid,euid) ;  
  
    fdja = open("ja",O_RDONLY) ;  
    fdon = open("on",O_RDONLY) ;  
    printf("fdja %d fdon %d\n",  
           fdja, fdon) ;  
  
    setuid(uid) ;  
    printf("uid %d euid %d\n",  
           uid,euid) ;  
  
    fdja = open("ja",O_RDONLY) ;  
    fdon = open("on",O_RDONLY) ;  
    printf("fdja %d fdon %d\n",  
           fdja, fdon) ;  
  
    setuid(euid) ;  
    printf("uid %d euid %d\n",  
           uid,euid) ;  
}
```

vykonání uživatelem „ja“:

```
uid 500 euid 800
fdja -1 fdon 3
uid 500 euid 500
fdja 4 fdon -1
uid 500 euid 800
```

vykonání uživatelem „on“:

```
uid 800 euid 800
fdja -1 fdon 3
uid 800 euid 800
fdja -1 fdon 4
uid 800 euid 800
```

Je-li `euid == 0` změní se reálný UID, efektivní UID i uložený nastavovací UID.

- **login** po úspěšném přihlášení uživatele nastaví reálný UID, efektivní UID i uložený nastavovací UID na UID uživatele a vykoná **exec shell**

Chce-li uživatel změnit své heslo, nemůže tak učinit přímo. Vykoná program **passwd**, který má nastavený bit SUID, a který vlastní superuser.

Obdobně s GID

Rodina volání:

getegid() , *geteuid()* , *getgid()* , *getuid()* , *setegid()* ,
seteuid(), *setgid()* , *setregid()* , *setreuid()* , *setuid()*

u oblast a **proc** záznam

datové struktury obsahující řídicí informace pro procesy

často tabulka procesů s položkami **proc** záznam má pevnou velikost v adresovém prostoru jádra

nověji pole ukazatelů, na dynamicky vytvářené **proc** záznamy, ale pole pevné velikosti, SVR4

u oblast je mapovaná do uživatelského adresového procesu →
je viditelná, když je proces běžící, často mapovaná na
pevnou virtuální adresu, proměnná **u** v jádře

proc záznam procesu je přímo přístupný i když proces není
běžící

u oblast

- ukazatel na **proc** záznam
- reálný a efektivní UID a GID
- argumenty a návratové hodnoty systémových volání
- ošetření signálů
- tabulka deskriptorů otevřených souborů
- okamžitý adresář a řídicí terminál
- využití CPU a kvóty
- v mnoha implementacích zásobník jádra

proc záznam

- identifikaci PID
- umístění u oblasti
- stav
- ukazatele na vytvoření seznamů všech procesů, čekajících procesů, ...
- událost, na kterou proces čeká
- informace pro plánování
- pole neošetřených signálů
- informace pro správu paměti
- propojení na PID v rozptýlené (hash) tabulce
- informace pro hierarchii procesů

Linux

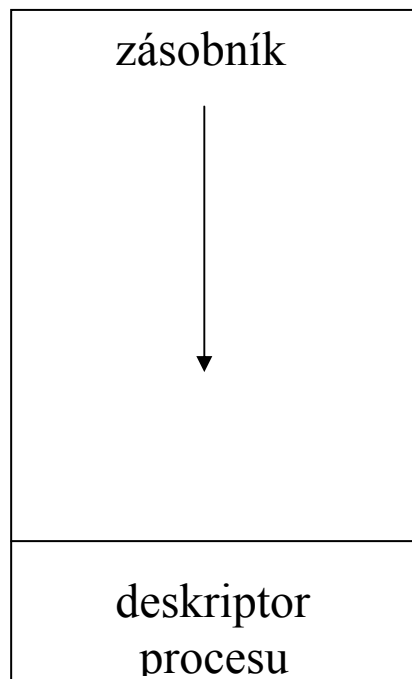
jedna údajová struktura – deskriptor procesu, **task_struct**

položky obsahují ukazatele, např. na informace o prostředcích

| | |
|----------------------|--|
| tty_struct | terminál sdružený s procesem |
| fs_struct | okamžitý adresář |
| files_struct | ukazatele na deskriptory souborů |
| mm_struct | ukazatele na deskriptory oblastí paměti |
| signal_struct | přijaté signály |

deskriptor procesu a zásobník jádra v jedné oblasti paměti o velikosti 8KB

```
union task_union {  
    struct task_struct task;  
    unsigned long stack[2048];  
};
```



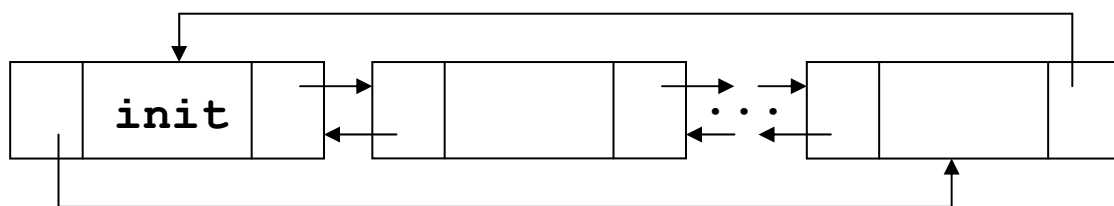
po přepnutí z uživatelského módu do módu jádro, ukazatel zásobníku ukazuje na vrchol maskováním nejnižších 13 bitů ($8\text{KB} = 2^{13}$) získáme adresu deskriptoru procesu

```
movl $0xffffe000, %ecx  
andl %esp, %ecx  
movl %ecx, p
```

výhodné pro multiprocesorové systémy

pro efektivní hledání, např. všech připravených procesů, deskriptory procesů jsou kruhově obousměrně spojeny – kruhový obousměrný spojový seznam ukazatelé jsou položky **task_struct**

seznam procesů (*process list*)



procesy mohou být ve více seznamech
seznam připravených (*runqueue*)
seznamy čekajících na nějakou událost (*wait queues*)

seznamy umožní nalézt všechny nebo všechny procesy s nějakou vlastností

jádro někdy musí určit deskriptor procesu z jeho PID
zaslání signálu procesu systémovým voláním

kill(pid, sig)

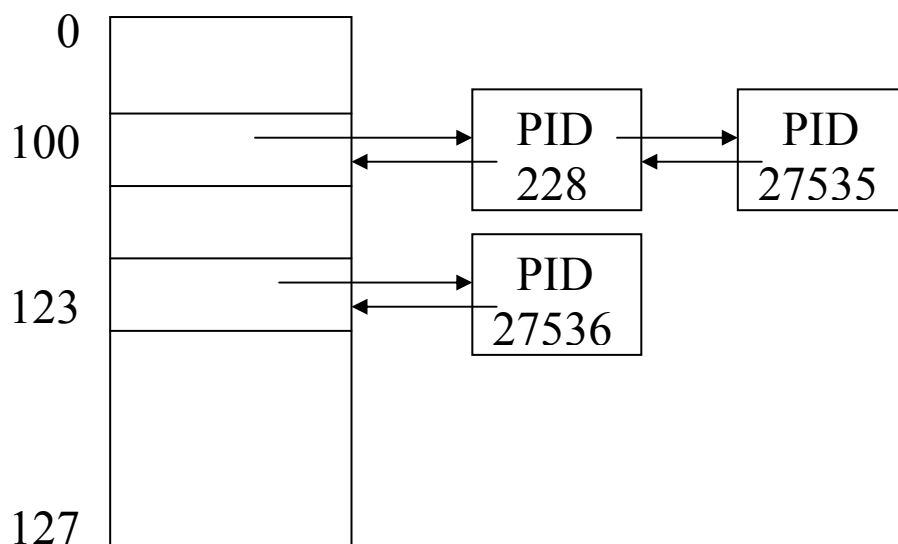
řešení

- projít seznam všech procesů a zjišťovat shodu **pid** a PID v deskriptoru procesu ?
- vytvořit pole, kterého prvek s indexem **pid** obsahuje ukazatel na deskriptor procesu ? (2^{16} , 2^{32})

rozptýlená tabulka (*hash table*)

- položka obsahuje ukazatel na deskriptor procesů se zadaným PID
- v případě kolize jsou odpovídající deskriptory zřetězeny
- ukazatele jsou součástí

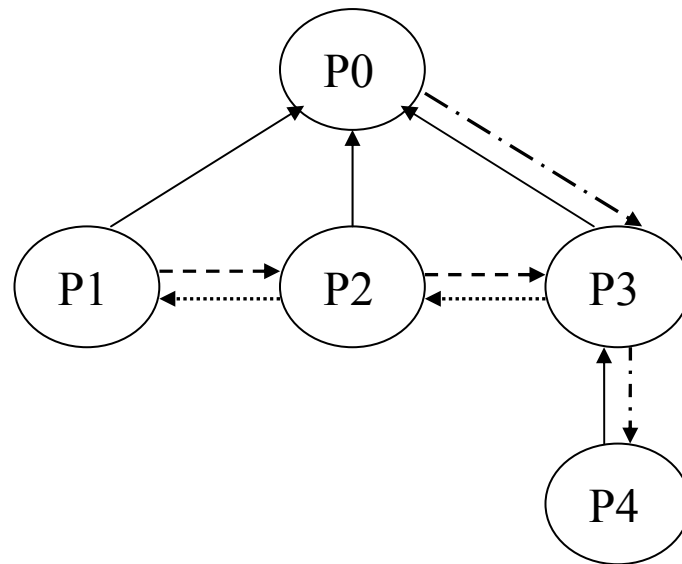
```
#define pid_hashfn(x) \
    (((x) >> 8 ^ (x)) & (PIDHASH_SZ-1))
```



hierarchie procesů – vztahy rodič/potomek se vytváří položkami v deskriptoru procesů

| | |
|---------|---------------------------------|
| p_opptr | originální rodič |
| p_pptr | rodič, rozdíl např. po ptrace() |
| p_cptr | nejmladší potomek |
| p_ysptr | mladší sourozenec |
| p_osptr | starší sourozenec |

proces P0 vytvořil postupně procesy P1, P2, P3 a proces P3 vytvořil proces P4



p_pptr —————>
p_ysptr - - - - ->
p_osptr ·········>
p_cptr - · - · - ·>