PROCESY A VLÁKNA

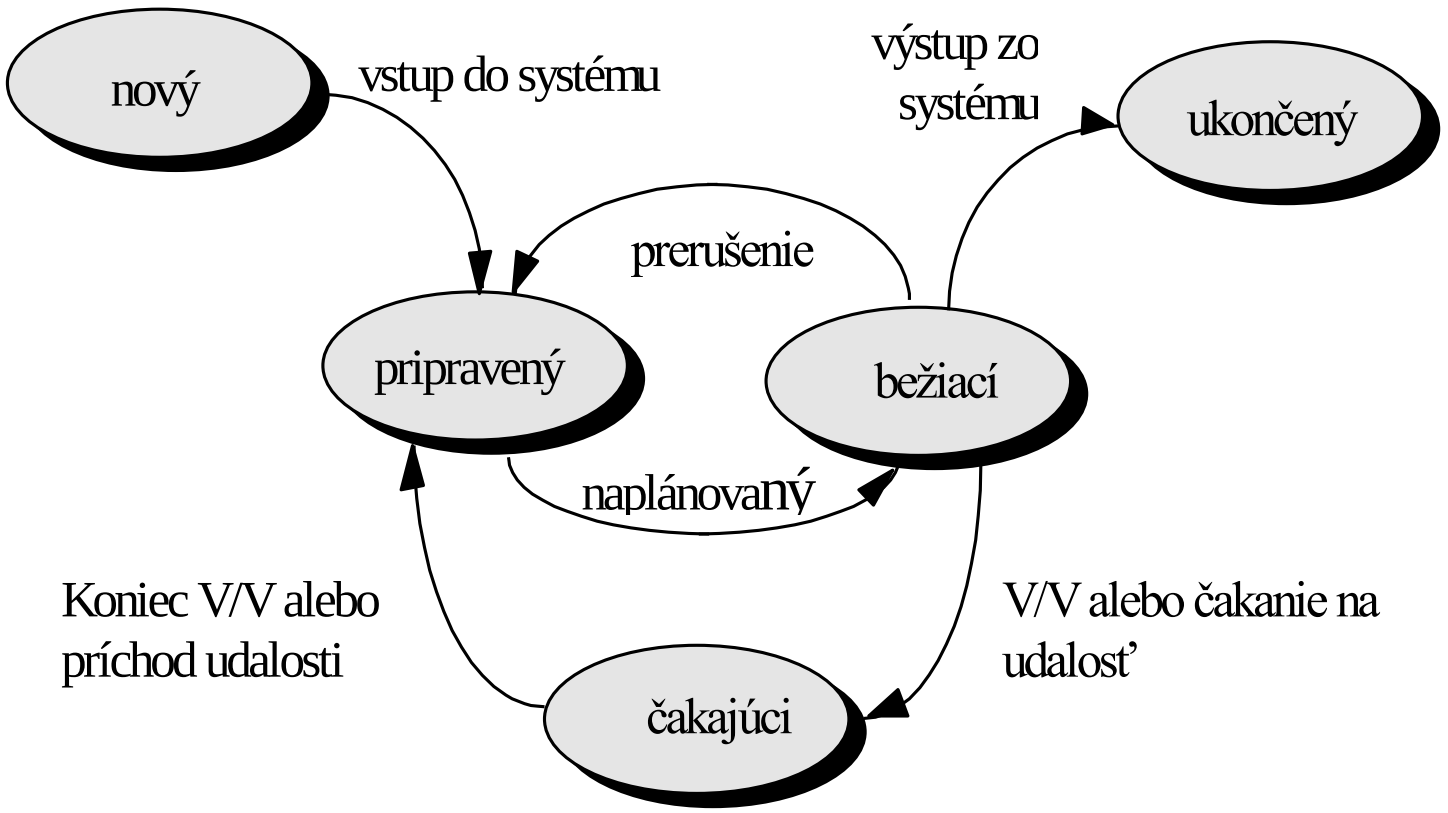
Definícia procesu

- Neformálne, ***proces*** je bežiaci program, vrátane všetkých informácií potrebné pre jeho opätovnému spusteniu od inštrukcie, kde bol prerušený (s tými istými prostriedkami a s tou istou prioritou)
- Formálne prvky procesu
  - Stav (status)
  - Riadiace informácie
  - Pridelené prostriedky
  - Vykonateľný súbor

Stav procesu

- Môže byť:
  - **Nový** - proces je práve vytvorený.
  - **Bežiaci** - vykonávajú sa inštrukcie programu.
  - **Čakajúci** - proces čaká na nejakú udalosť, napr. dokončenie V/V operácie alebo na signál.
  - **Pripravený** - proces čaká na pridelenie procesora
  - **Ukončený** - proces dokončil svoje vykonanie – normálne alebo spôsobil výnimku

Stavový diagram



Udalosti počas behu procesu

- **Interné** - vznikajú v rámci procesu a zapríčinia zmenu stavu procesu:
  - systémové volanie - skok do jadra
  - chyba - zlá inštrukcia, porušenie oprávnenia prístupu do pamäte atď.,
  - zlyhanie stránky (page fault) .
- **Externé** - udalosti, ktoré proces neriadi, sú to vonkajšie udalosti a obyčajne ich oznamujú prerušenia, ktoré obsluhuje operačný systém:
  - vstup z terminálu (znak),
  - ukončenie diskovej operácie,
  - prerušenie od časovača.

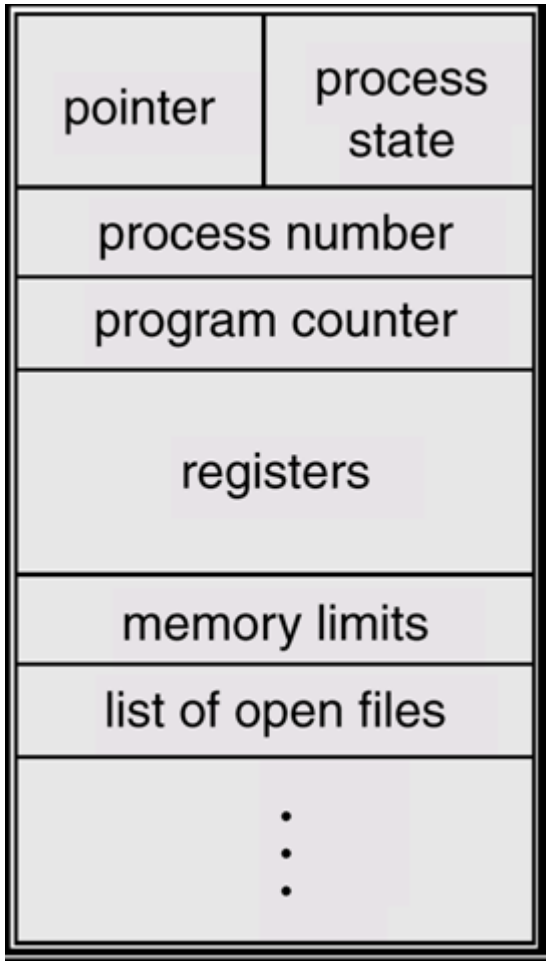
Riadiaci blok procesu (PCB)1

- PCB obsahuje informácií o procese, z ktorých najdôležitejšie sú:
  - **Stav procesu** - nový, pripravený, bežiaci, čakajúci atď.
  - **Hodnota počítadla inštrukcií** - indikuje adresu inštrukcie, ktorá bude vykonaná ako nasledujúca
  - **Registre CPU** - akumulátory, index registre, ukazovatele zásobníkov, univerzálne registre, informácie o podmienených kódov a iné.

Riadiaci blok procesu (PCB)$_2$

- **Informácie pre plánovanie procesu** - priorita procesu, ukazovatele na fronty pre plánovanie a iné.
- **Informácie pre správu pamäte** - hodnoty limitných a bázových registrov, tabuľku stránok alebo segmentov, podľa použitej techniky správy pamäte.
- **V/V informácie** - obsahujú zoznam V/V zariadení, ktoré sú pridelené procesu, zoznam otvorených súborov atď.

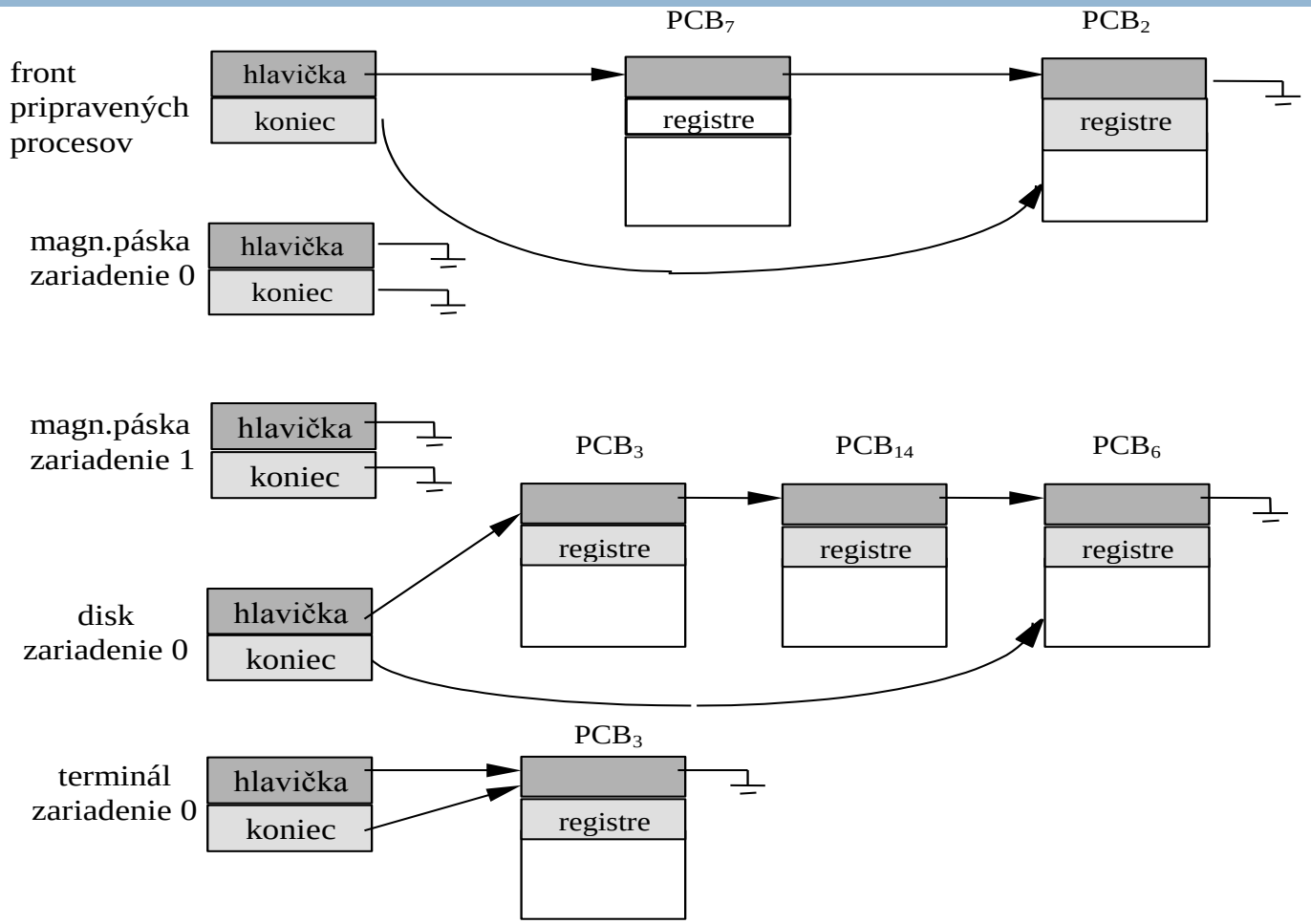
Riadiaci blok procesu (PCB)$_3$



Plánovanie procesov

- Má za úlohu určiť ktorému procesu bude pridelený procesor pri prepnutí.
  - V multiprogramovom OS - bežiacim procesom sa striedavo prideľuje CPU
  - Swapovanie v time-sharingových systémoch
    - keď časové kvantum procesu vyprší
  - Prístup k sekundárnej pamäti a V/V
    - môže spôsobiť vznik (spustenie) nového procesu

Fronty



Front pripravených procesov a fronty periférnych zariadení

Obsluha frontov

- Cez ukazovatele, ktoré vytvárajú zreťazený zoznam PCB procesov
- Front pre CPU alebo pre zariadenia
- Interpretácia: prvý proces používa zariadenie, ostatní čakajú
- PCB môže byť vo viacerých frontoch
- Ukončené procesy sa odstránia zo všetkých frontov

Fronty, reprezentujúce plánovanie procesov

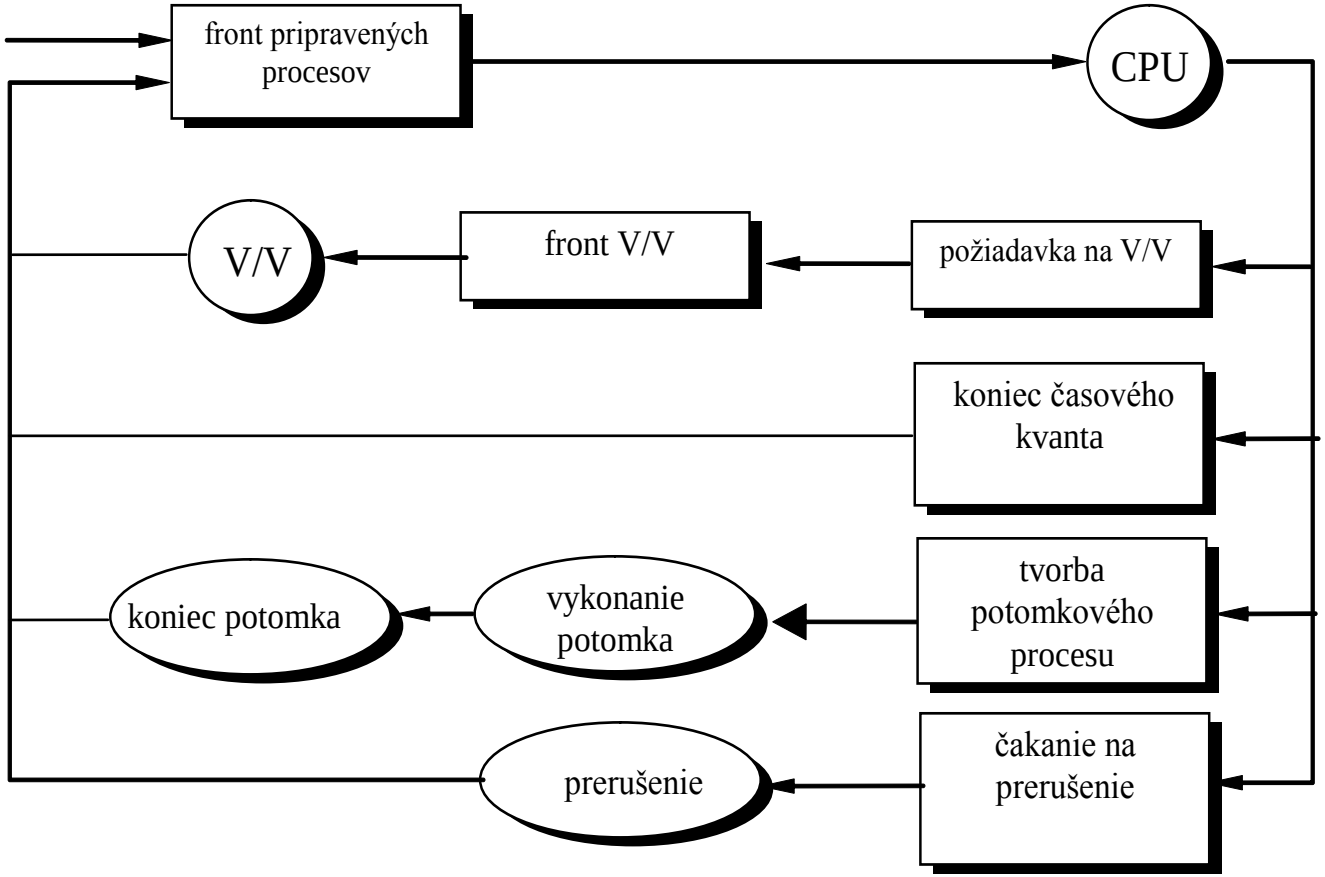


Diagram frontov, ktoré reprezentujú plánovanie procesov

Pohyb medzi frontami

Keď vznikne nový proces môže nastať jedna z týchto situácií:

- proces **požiada o V/V operáciu** a bude umiestnený do frontu príslušného zariadenia,
- proces **vytvorí nový podproces** a čaká na jeho dokončenie,
- procesu môže byť odobratý procesor ako výsledok **prerušenia** a proces je umiestnený do frontu pripravených procesov,
- **skončí časové kvantum** pridelené procesu.

Plánovač (Scheduler)

Rozdelený na 2 alebo viac častí

Dlhodobý plánovač

- vyberá z procesov pripravených na disku,
- ukladá ich do pamäte, vytvára z nich procesy do frontu pripravených procesov

Krátkodobý plánovač

- vyberá z pripravených procesov v pamäti a prideľuje CPU jednému z nich
- pracuje s pripravenými procesmi
- vyvoláva sa približne každých 100 ms

Dlhodobý plánovač

- Pridáva procesy do frontu pripravených procesov.
- Pracuje v dlhších intervaloch – **minúty**
- Snaží sa vytvárať **mix úloh**, viazaných na I/O a úloh, vyžadujúcich prevažne CPU;
  - veľa úloh, viazaných na I/O môže viesť k dlhšiemu čakaniu a multiprogramovanie potom nemá veľký vplyv
- **Zodpovedný za dobrú priechodnosť systému**

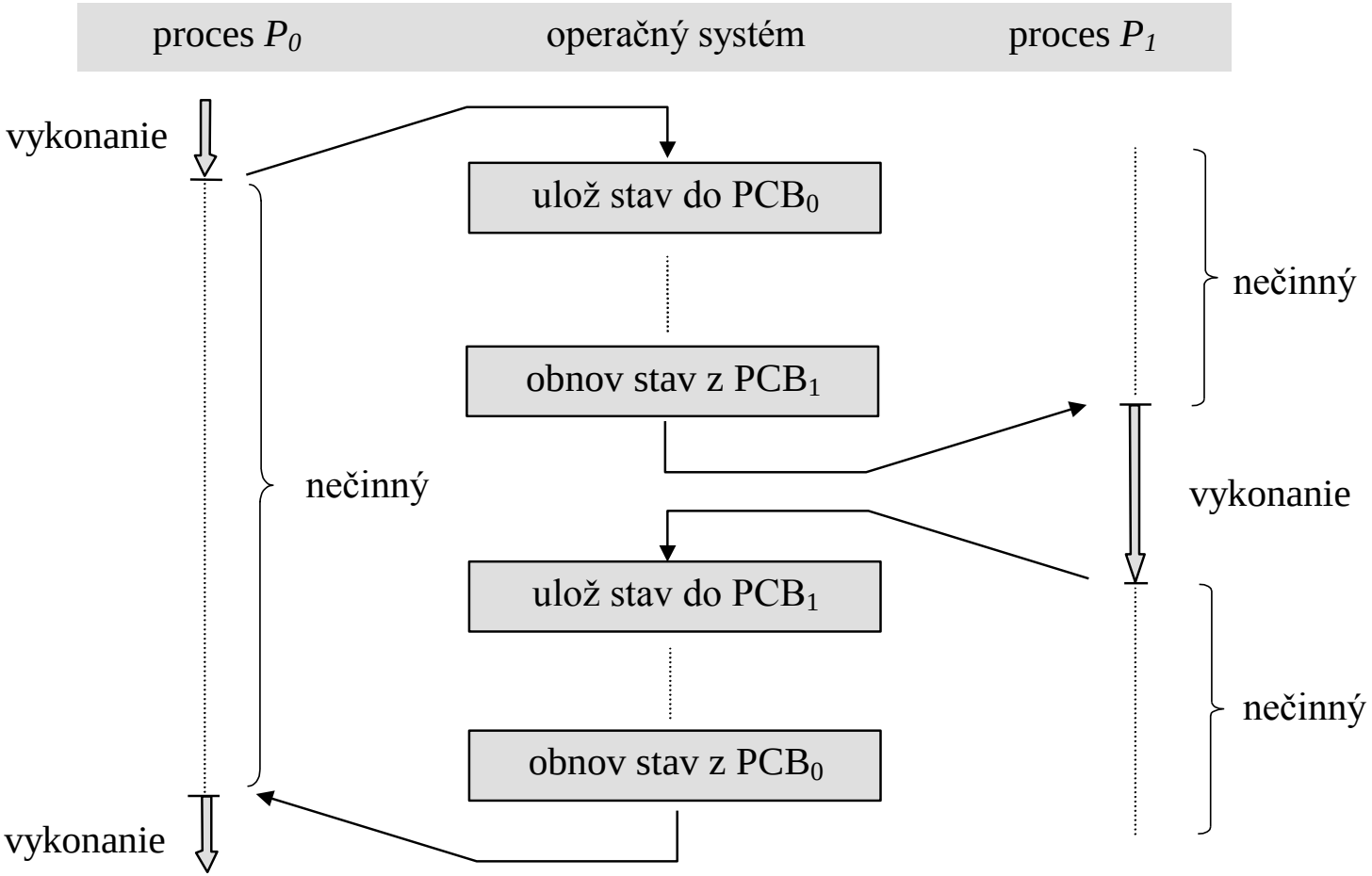
Krátkodobý plánovač

- **Presúva úlohy do/z frontu pripravených procesov** a poskytuje im CPU alebo iné prostriedky
- Veľmi dôležitý v time-sharingových systémoch kvôli **doby odozvy**
- Spolupracuje s časovačom (timer) a plánovacou politikou, **aby rozhodol kedy sa prepne proces**

Výmena bežiaceho procesu

- Nazýva sa ***prepínanie kontextu*** – na nasledujúcom obrázku
  - Zaberá veľa času, **je to réžia**, pretože sa nevykonáva práca, požadovaná procesom
  - Zaberie od 1 mikrosekundy do 1 milisekundy
    - „drahý“
    - uchovanie a obnovenie mapy pamäte procesu
    - vyprázdnenie cache pamäte

Prepínanie kontextu procesov



Strednodobý plánovač

- Existuje v niektorých systémoch
- Snaží sa zabrániť prehnane vysokej úrovne multiprogramovania (*thrashing*)

  Napr. *ak dlhodobý plánovač podcení potreby pamäte procesov, ktoré sú vo fronte pripravených procesov*
- Plánuje využitie RAM pamäte
- Zasahuje a *swapuje (odsunuje)* niektoré procesy mimo slučky krátkodobého plánovača a tak sa na chvíľu zabráni preplneniu pamäte a CPU

Činnosti OS pre procesy

- Obsluhuje *systémové volania*
- To znamená volania pre:
  - **tvorbu** procesu
  - **ukončenie** procesu
  - **komunikácie** medzi procesmi

Tvorba

- "Rodičovský" proces požaduje vytvorenie "potomkového" procesu; v Linuxe – ***fork()***
- Vykonanie môže byť súbežné alebo nie
  - "potomkový" proces môže, ale nemusí zdediť pamäť rodiča
  - "potomkový" proces sa pridáva do frontu pripravených procesov
- "Rodičovský" proces - "potomkový" proces komunikujú cez *identifikátory* (*PID*)
- Keď "potomkový" proces sa ukončí:
  - jeho PID a výstup môžu byť vrátený rodičovi - ako návratová hodnota systémového volania

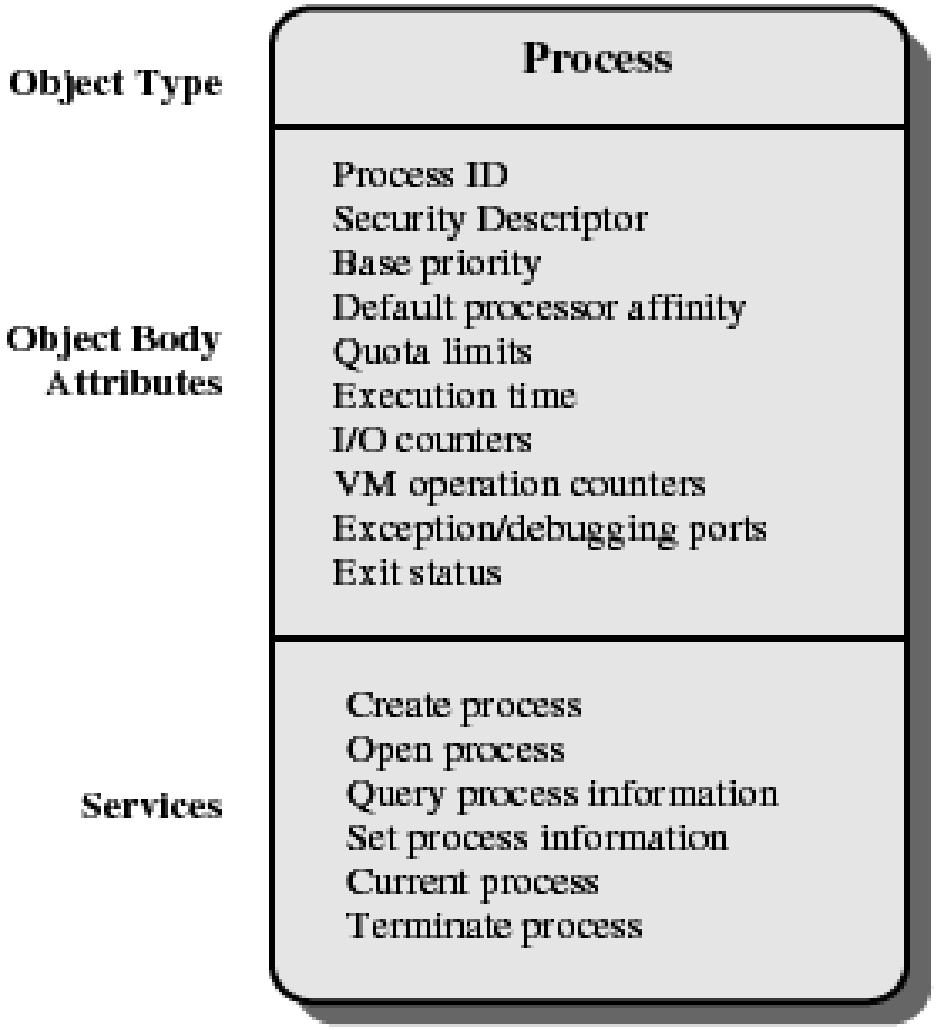
Ukončenie

- Rodič môže ukončiť proces, ak ten prekročí pridelené prostriedky, stane sa zbytočným alebo rodič už končí svoje vykonanie kvôli inému problému
  - *exit, abort, kill*
- „Potomkový“ proces končí (skoro vždy) pri ukončení rodiča
- Všetky prostriedky ukončeného procesu sa dealokujú a vracajú OS

Procesy v Windows

- Implementované ako objekty
- Proces môže mať jedno alebo viac vlákien
- Procesy a vlákna majú zabudované synchronizačné schopnosti

Objekt procesu v Windows

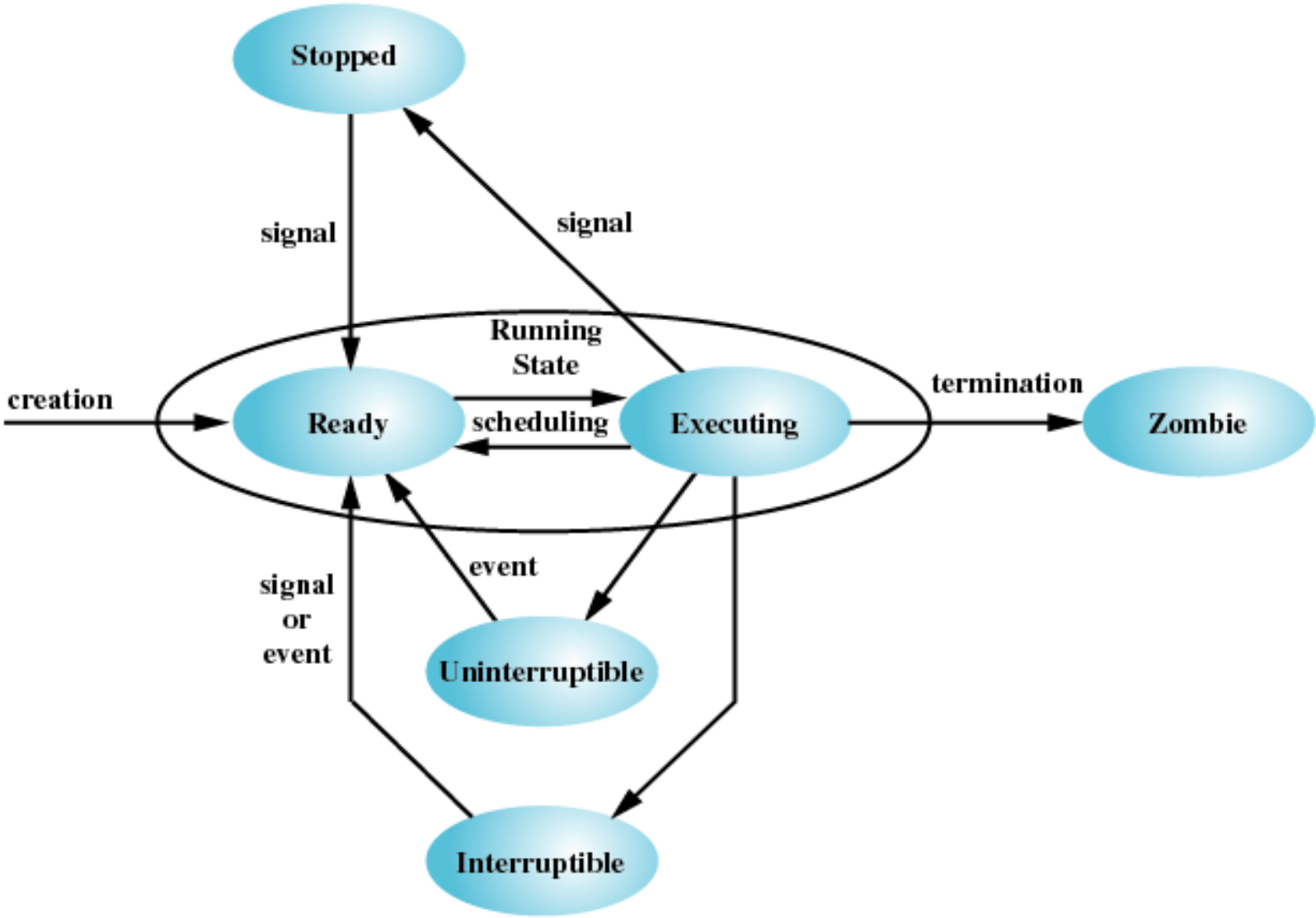


(a) Process object

Vlákna (threads)[1]

- Vlákno je podobné procesu – nasleduje obrázok
  - niekedy nazývané odľahčený proces LWP
- Vlákno podobne ako proces
  - má stavy
    - pripravený, bežiaci, zablokovaný...
    - registre, pamäť, kód
  - má svoj riadiací blok

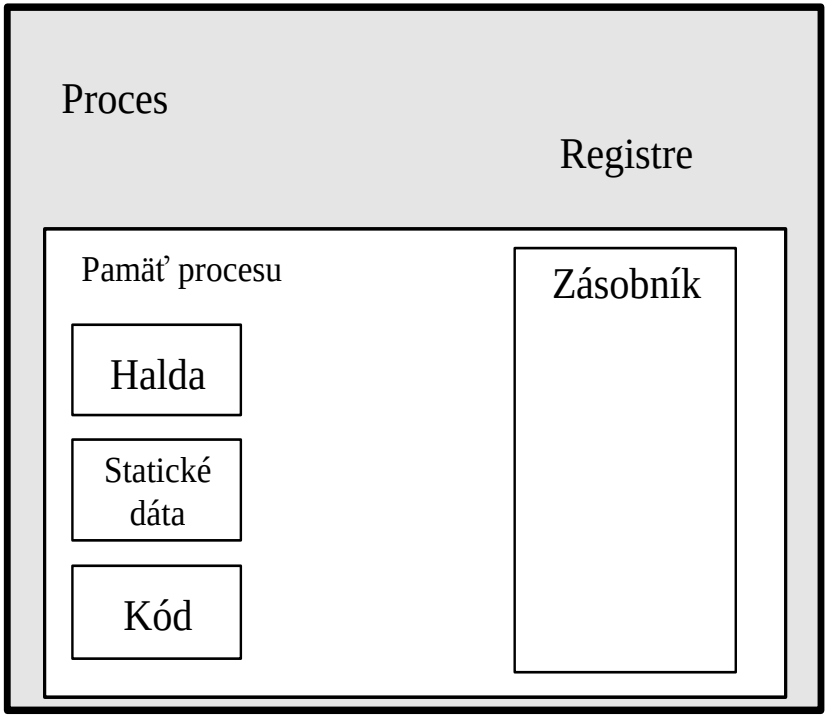
Príklad: stavový diagram procesov (vlákien) v Linuxe



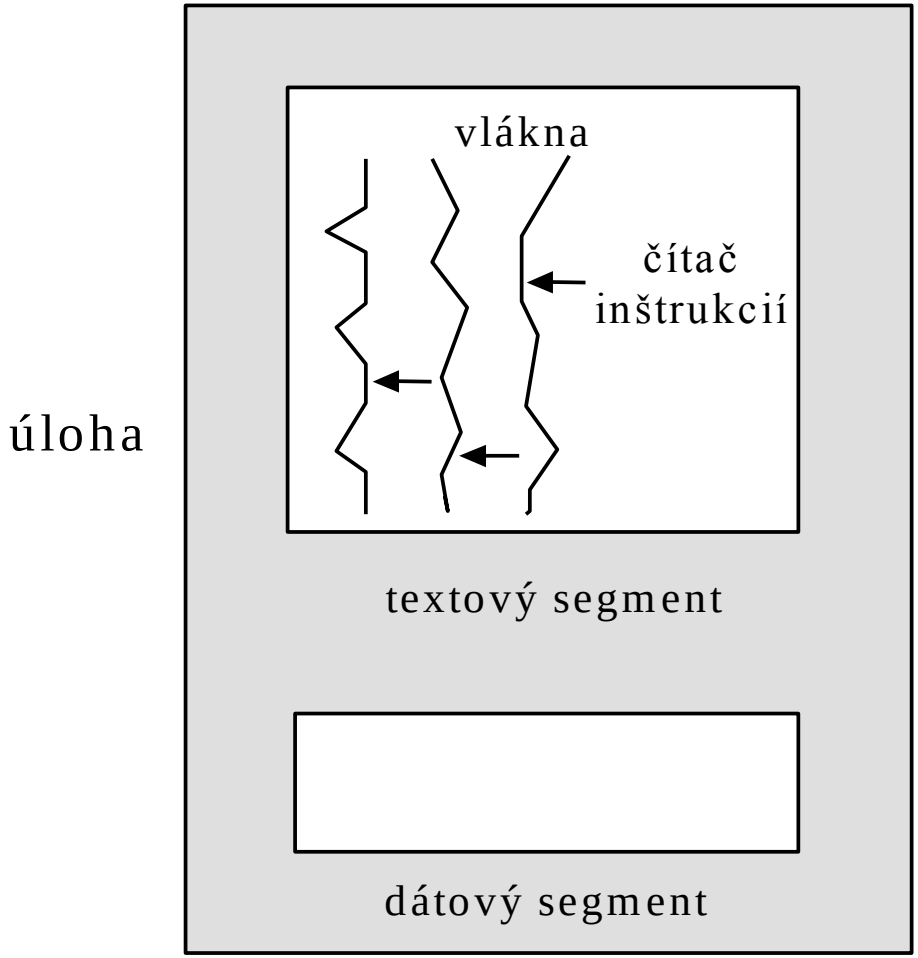
Vlákna (threads)$_2$

- Vlákna sú jednoduchšie a efektívnejšie
  - zdieľajú pamäť a súbory, **NIE ZÁSOBNÍKY**
    - na čo slúži zásobník?
  - rýchlejšie sa vytvárajú
  - prepínanie medzi vláknami je rýchlejšie
- Proces vlastní pamäť a súbory
- **Multithreading** – systém umožňuje procesu vytvoriť viac vlákien

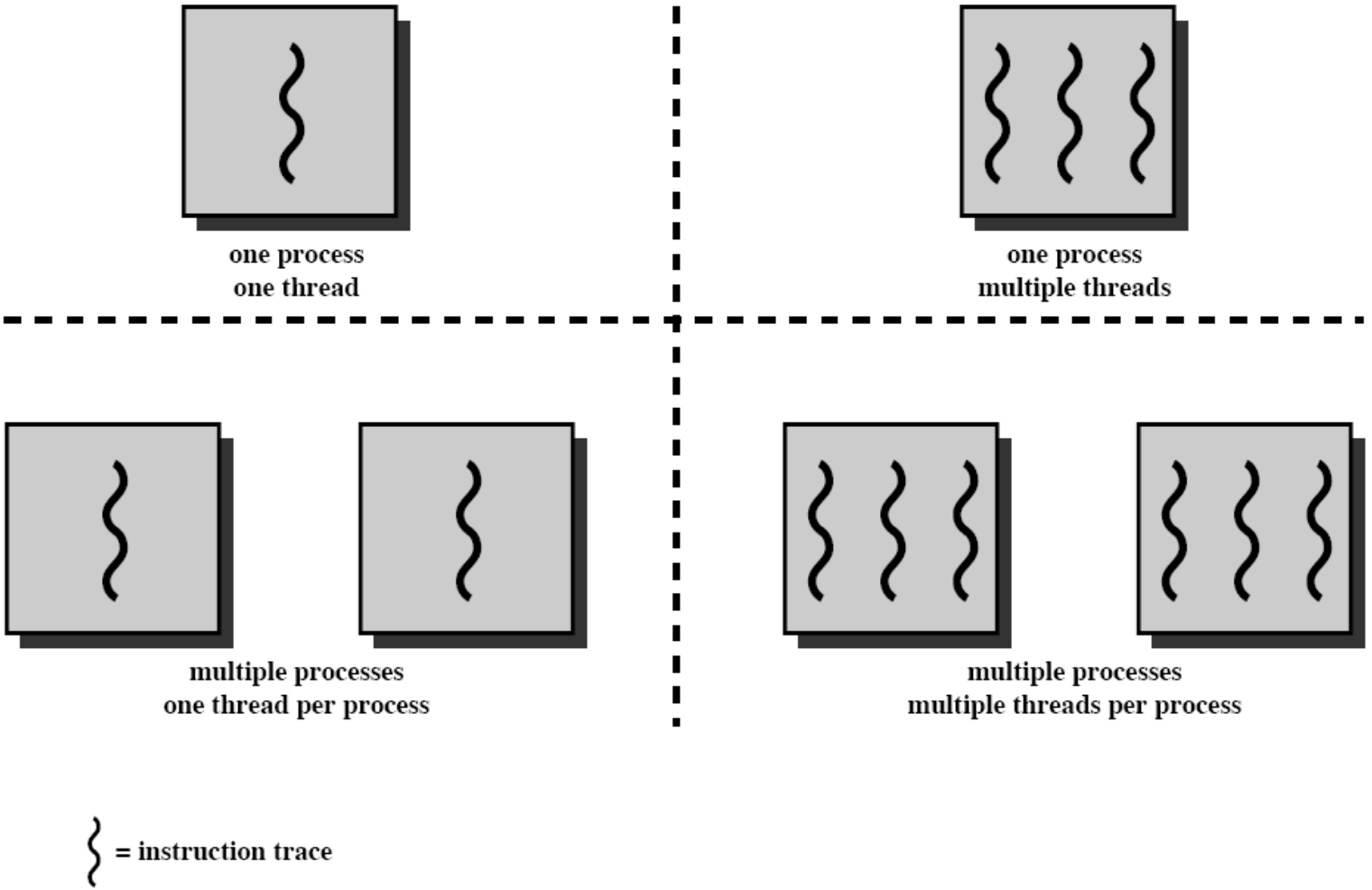
Štruktúra klasického procesu



Vlákna v rámci úlohy (procesu)



Vlákna v rámci úlohy[1]



V čom sú jednoduchšie

- **Vlákna zdieľajú pamäť**
  - pre používateľa
    - komunikácie sú jednoduchšie
    - komunikácie sú efektívnejšie
    - ochrana nemusí byť
  - pre OS
    - prepínanie je efektívnejšie
      - mapovanie pamäte zostáva nezmenené
      - nie vždy sa musí vyprázdniť cache pamäť

Typy vlákien v závislosti od implementácie

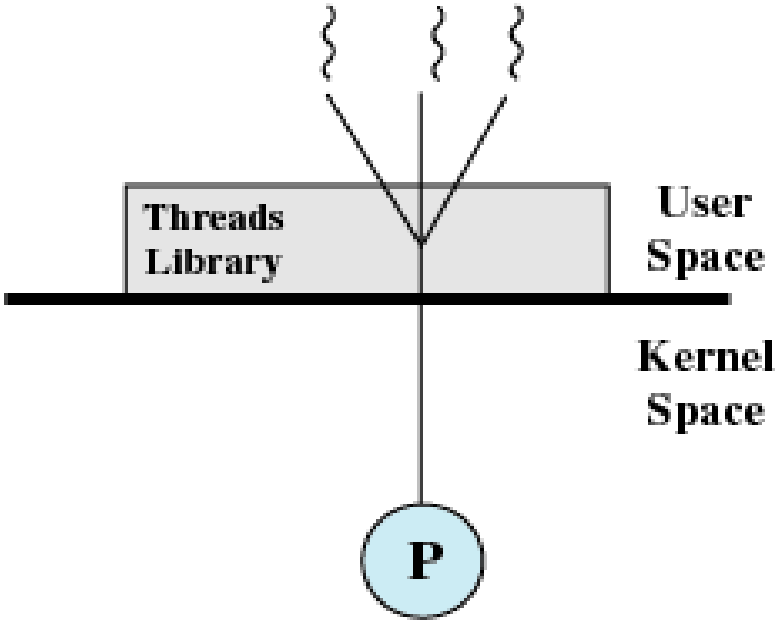
- **Na úrovni používateľa**
  - OS nevie o nich
  - implementované pomocou knižníc
  - plánovanie v rámci úlohy
- **Vlákna na úrovni jadra**
  - OS ich pozná – **jednotka pre plánovanie**
  - OS plánuje ich vykonanie

Vlákna na úrovni používateľa

- Operácie sú rýchlejšie
  - prepínanie kontextu
  - komunikácie
  - riadenie
- Nie je možná podpora viacprocesorových systémov
- Operácie na základe lokálnych kritérií
  - môžu byť menej efektívne
  - napr. priority pri plánovaní

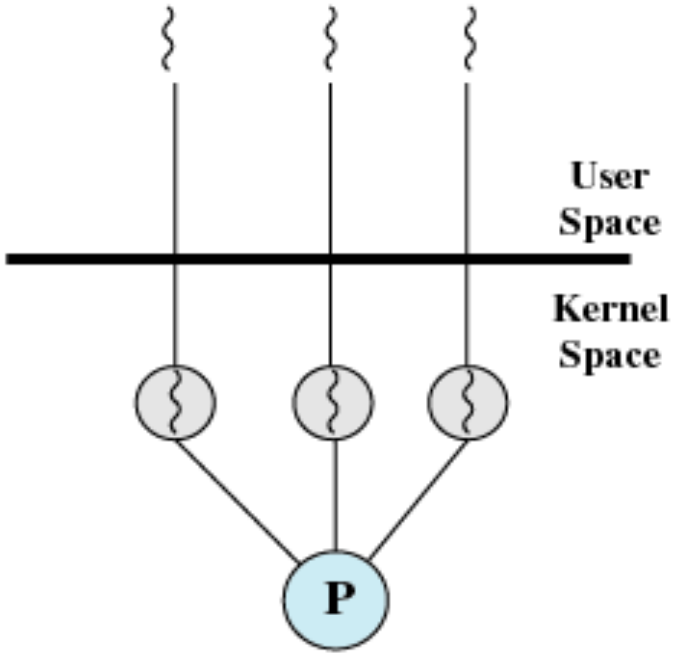
Vlákna na úrovni používateľa

- run-time podpora, obyčajne implementované knižnicou
- vykonáva sa v užívateľskom režime



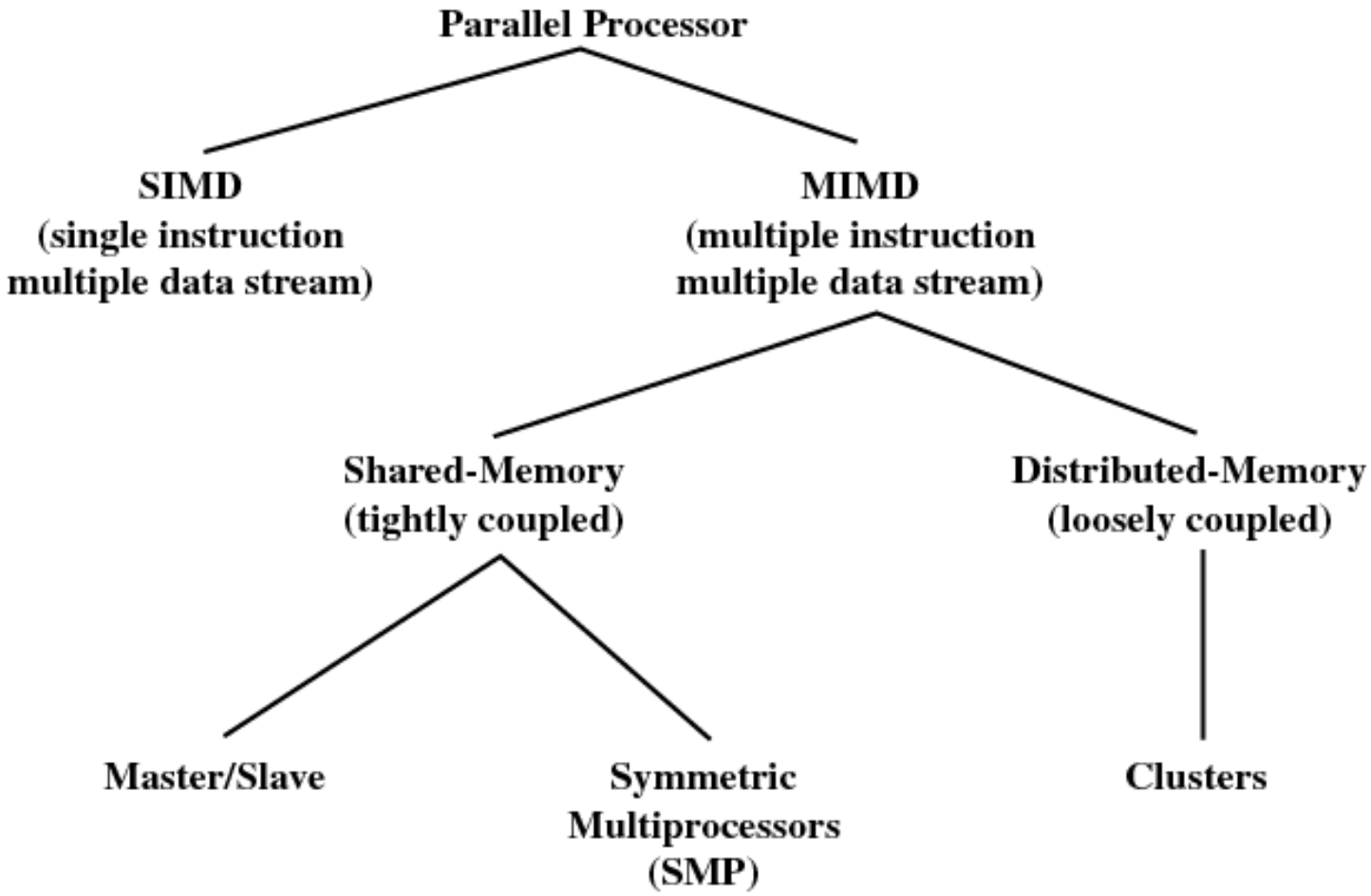
Vlákna na úrovni jadra

- Operácie sú pomalšie
  - musia isť cez OS
- Pre prepínanie medzi vláknami – potreba prepínať medzi režimami user-kernel
- Jednoduchšia podpora viacjadrových systémov



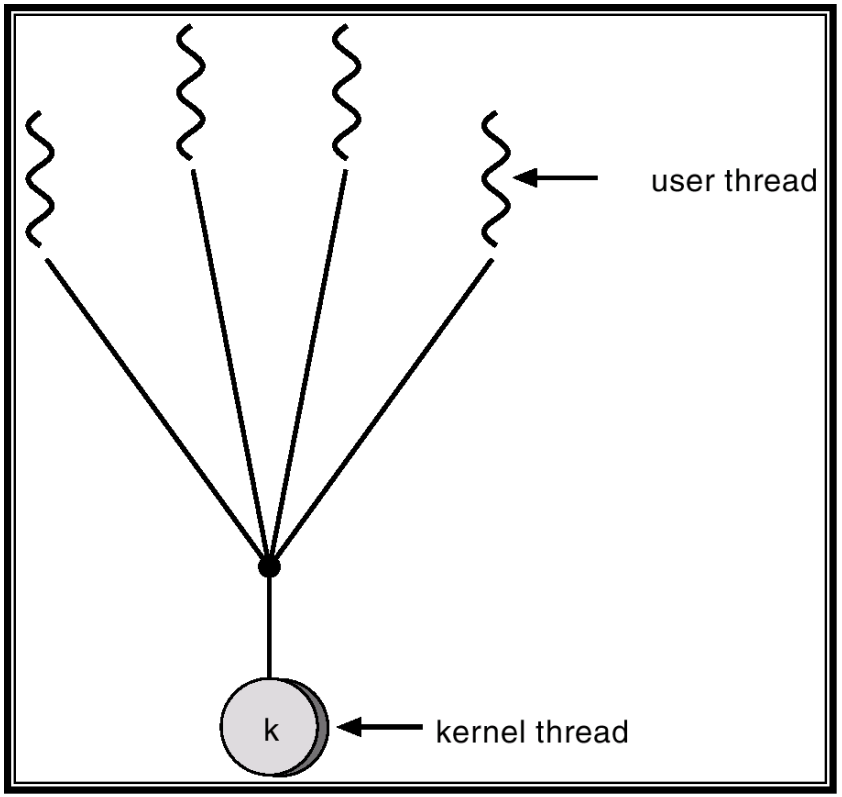
(b) Pure kernel-level

Architektúry paralelných procesorov



Modely vykonávania vlákien[1]

- **N:1**

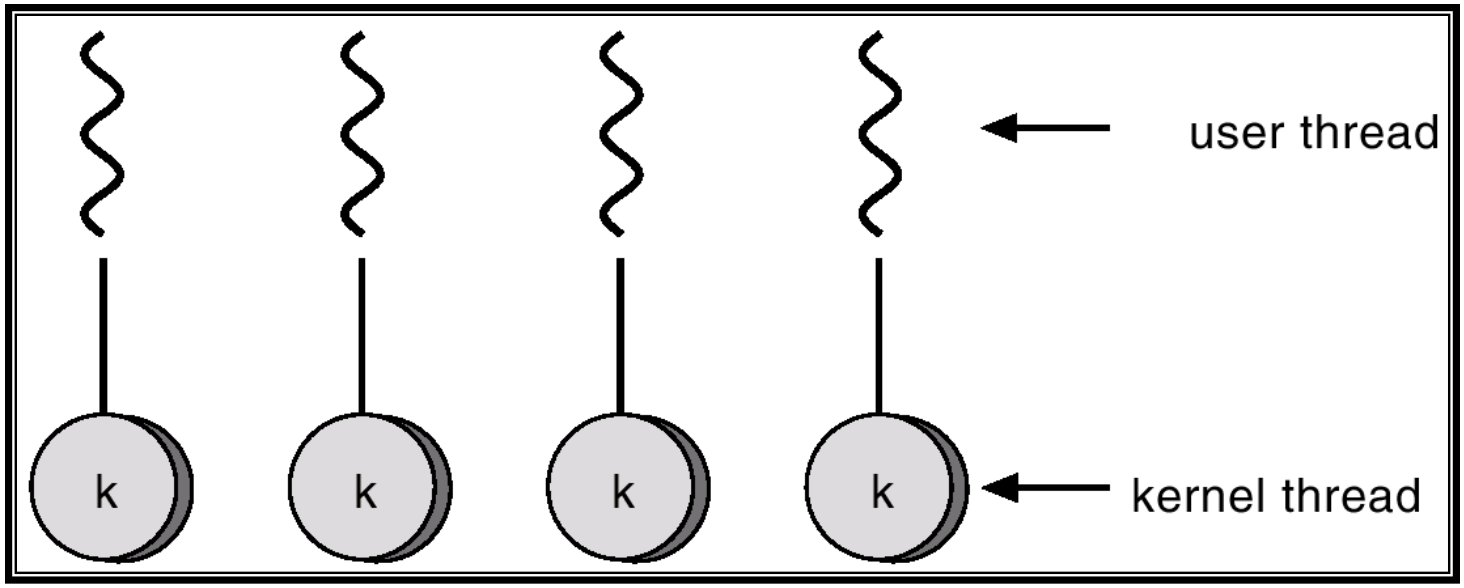


Model, použitý v Solarise a v prvých verziach Javy, malo efektívny, malo používaný v súčasnosti

Bežný spôsob mapovania medzi „user“ a „kernel“ vláknami, používa sa v systémoch, ktoré nepodporujú „kernel“ vlákna

Modely vykonávania vlákien2

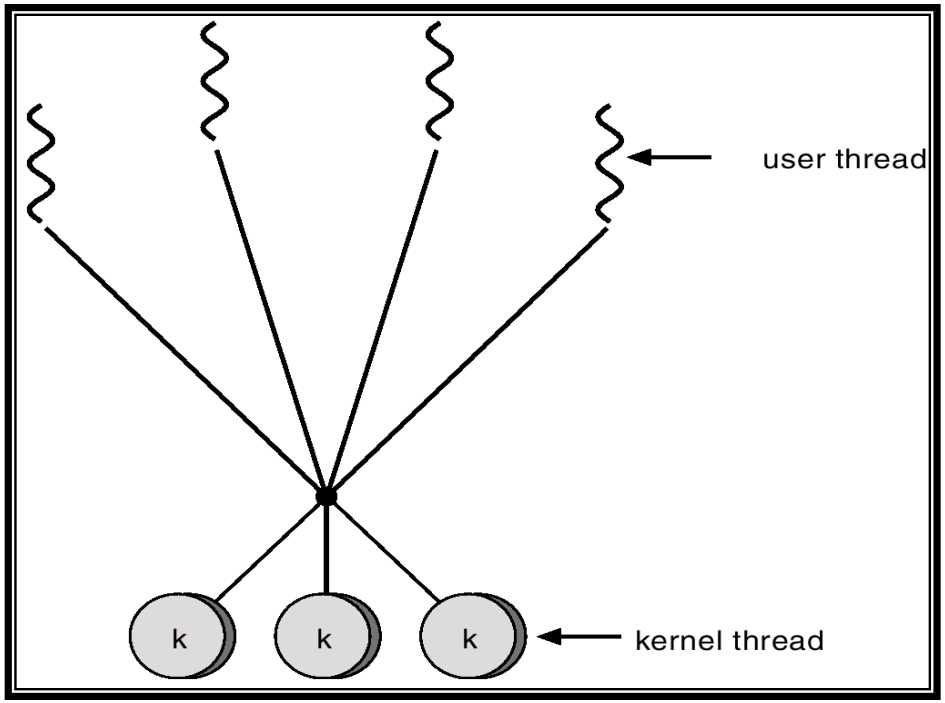
- 1:1



Použitý v Windows 95/98/NT/2000, OS/2, Unix

Modely vykonávania vlakien3

- **N:N**



Použitý v Solaris 2, Windows NT/2000 pomocou balíku *ThreadFiber*

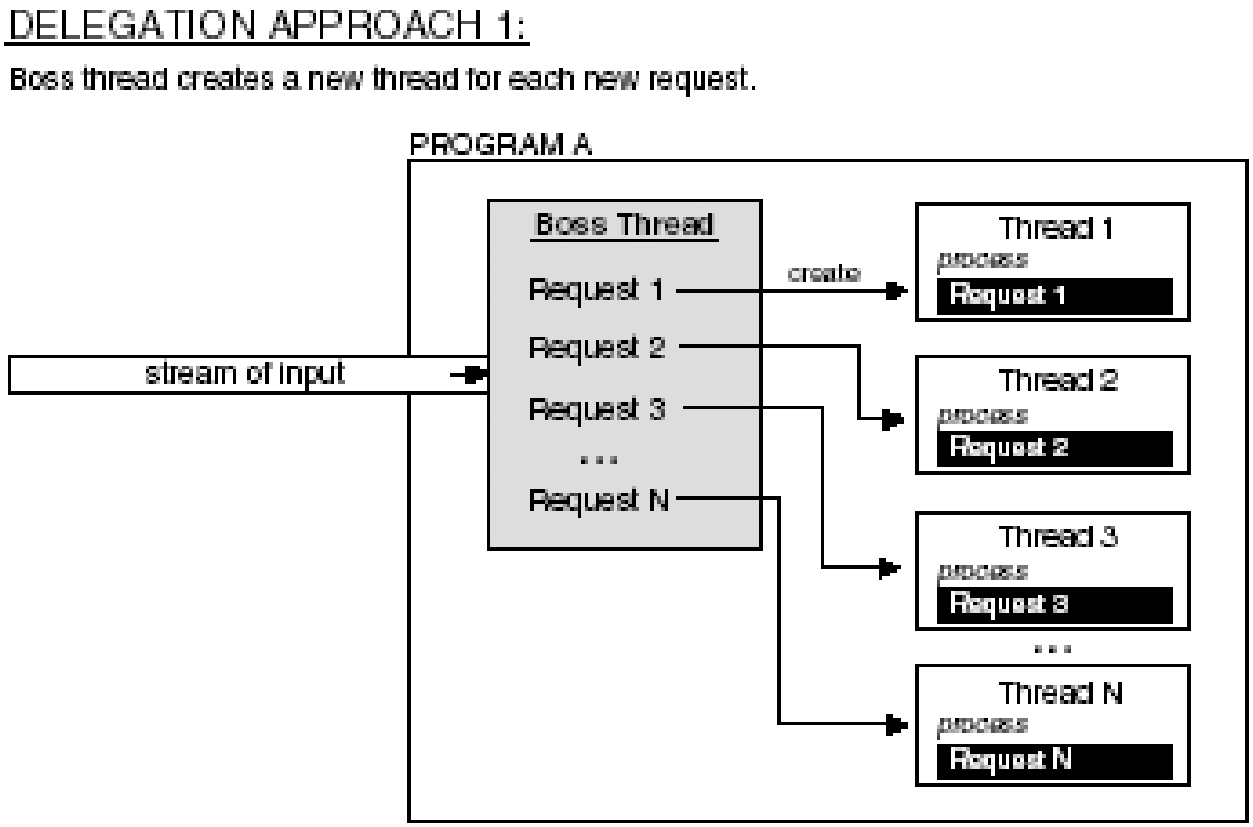
Modely vykonávania vlakien4

- **1:N**

- Používa sa v niektorých distribuovaných systémoch, vlákna môžu migrovať medzi jednotlivými strojmi a adresnými priestormi

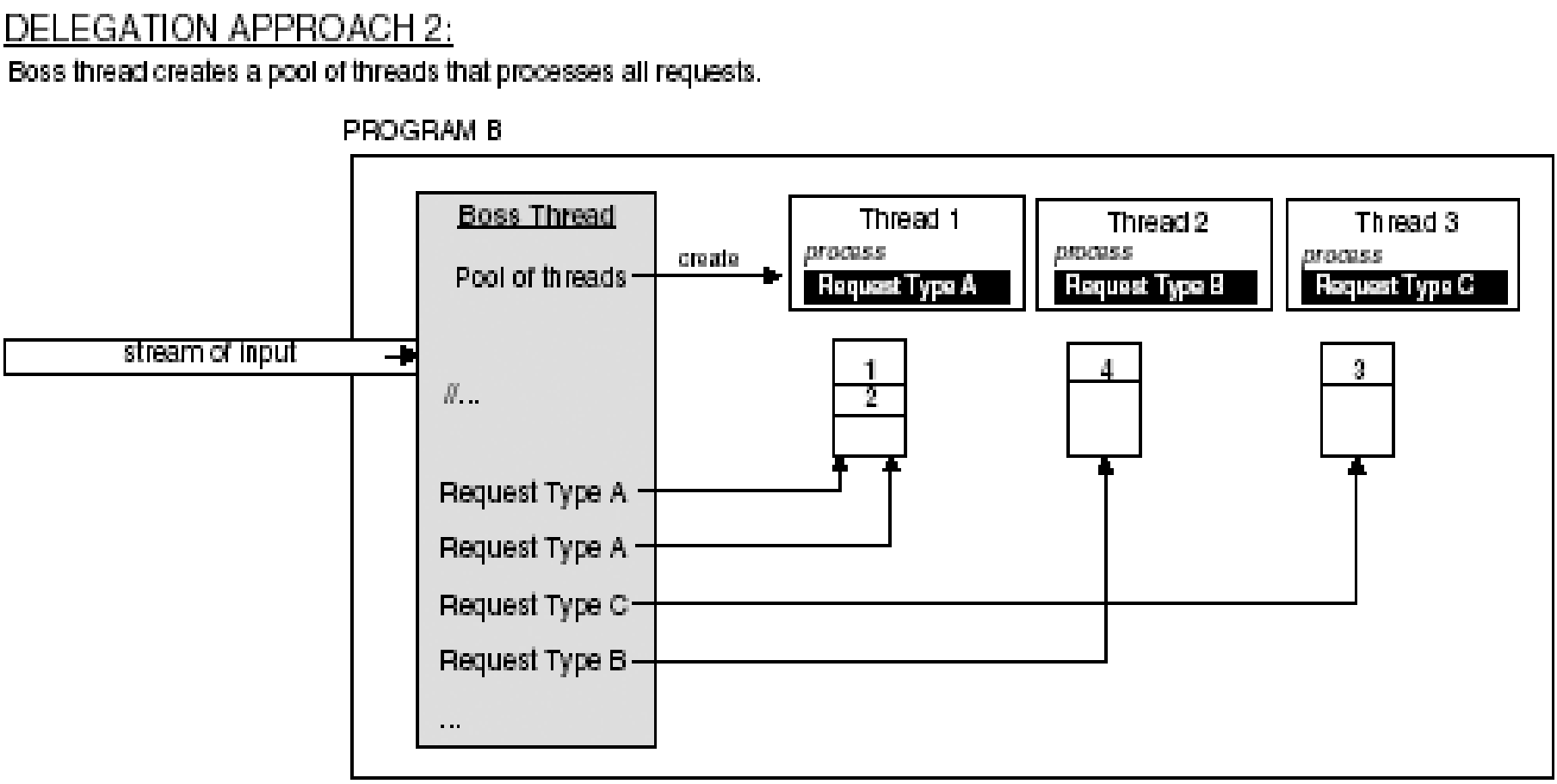
Modelové situácie použitia vlákien

a) Model „pán - otrok“ – „pán“ vytvára vlákno pre každú novú požiadavku



Modelové situácie použitia vlákien

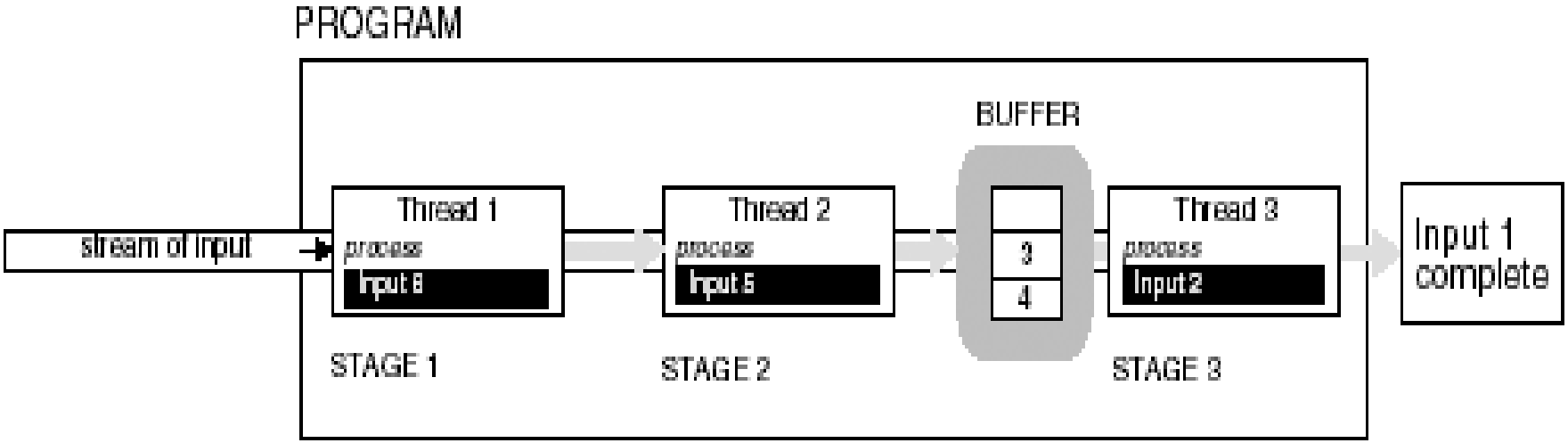
b) Model „člen skupiny“ – „vytvorí sa zásobník vlákien, ktoré spracovávajú požiadavky



Modelové situácie použitia vlákien

c) Postupný model

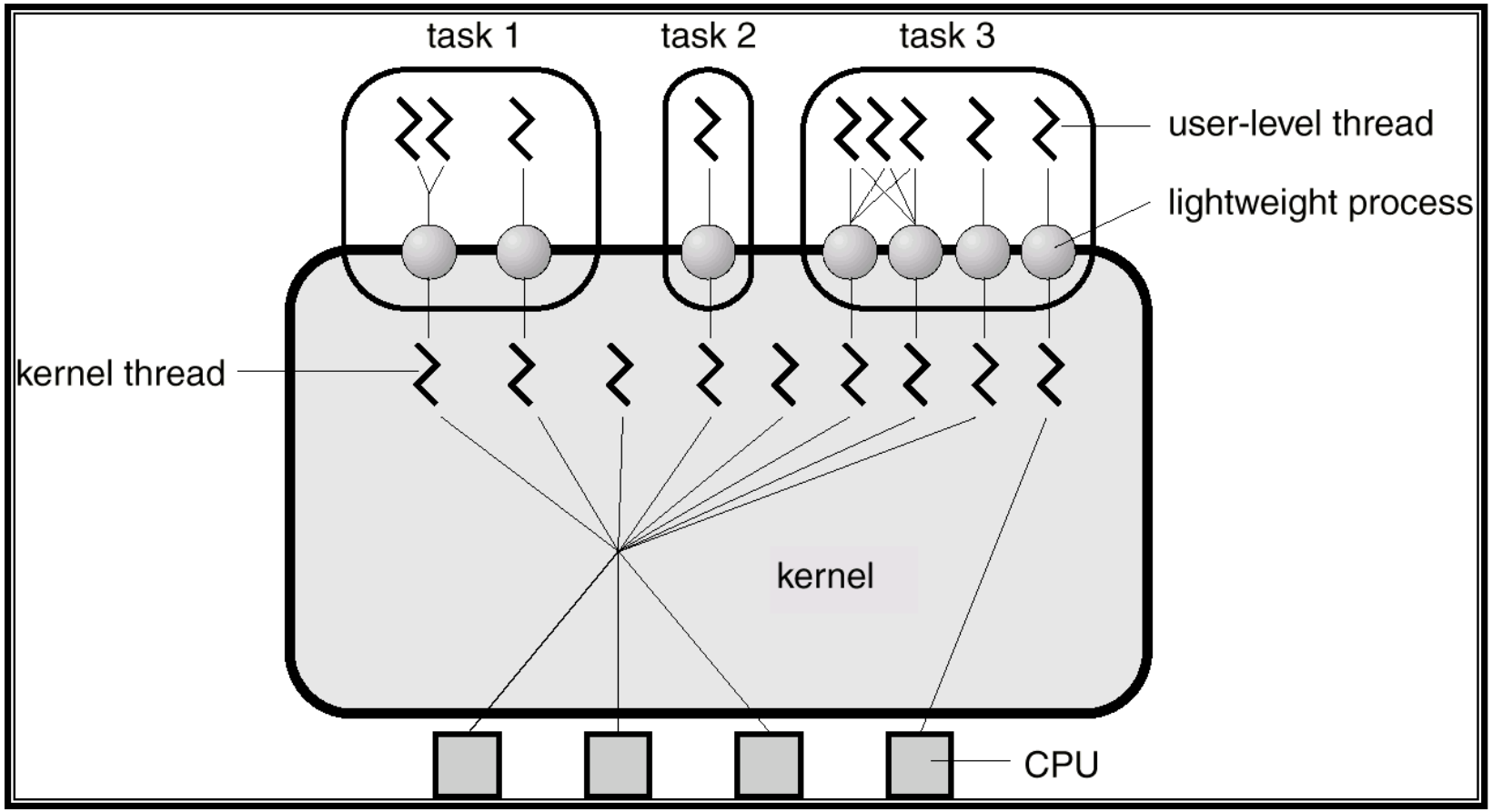
Príklad spracovania požiadavky v troch krokoch: výstup z prvej fázy ide na vstup druhej, výstup z druhej ide na vstup tretej a výstup z tretej je výstup procesu



OS Solaris[1]

- 5 úrovni
  - Procesory (CPU)
  - Vlákna v jadre
  - Úlohy
  - LWP – (light-weight) odľahčené procesy
  - Vlákna na užívateľskej úrovni
    - viz obrázok

Vlákna v OS Solaris2



OS Solaris

- Plánuje vykonanie vlákien jadra
  - spúšťa ich na procesoroch
  - vlákno jadra môže byť priradene pevne ku CPU
  - Úloha pozostáva z jedného alebo viacero LWP
  - LWP procesy v úlohe môžu
    - použiť systémové volanie
    - blokovať počas čakania na I/O operácie
- LWP je priradený ku vláknu jadra
- Sú vlákna jadra bez LWP

Model procesov v Unix-e

- Proces je základný stavebný prvok v UNIX-e , platí pre vlákna a pre odľahčené procesy
- V každej verzii tento model je podporovaný a rozvíjaný
- Pre každý proces systém udržuje dátovú štruktúru ***task_struct*** - približne 1KB

- V počítačoch s architektúrou Intel *task_struct* má presne 960 bajtov.
- *task_struct* sa nachádza na spodku zásobníka jadra (kernel stack)

Usporiadanie zásobníka jadra x86

----- 0xXXXX0000

(spodok zásobníka a adresa *task _struct*)

TASK_STRUCT

----- 0xXXXX03C0

(posledný bajt zo zásobníka použiteľný ako zásobník jadra)

KERNEL_STACK

----- 0xXXXX2000

(vrchol zásobníka, prvý bajt použiteľný ako zásobník jadra)

Procesy a vlákna v Linux-e

- Procesy a vlákna v Linuxe sú skoro rovnaké!
- Vlákna zdieľajú adresný priestor (AP) a prepnutie kontextu vlákna = skok na inú adresu v tom istom AP
- Na nižšej úrovni interfejsu sa pre tvorbu vlákna používa systémové volanie ***clone()***, na vyššej úrovni - ***pthread_create()***. (Pozri *man 2 clone*)
- **clone** () dovoľuje „potomkovému“ procesu zdieľať časti kontextu vykonávania s „rodičom“ ako napr. adresný priestor, tabuľku popisovačov súborov a tabuľku obsluhy signálov.
- **clone** () slúži na implementáciu vlákien

Príklad tvorby vlákna pomocou knižnice ***pthread (POSIX Threads)***

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>

static void *thread_func(void *vptr_args) {
    int i;

    for (i = 0; i < 20; i++) {
        fputs("  b\n", stderr);
        sleep(1);   // „uspatie“ vlákna na 1 sek.
    }
    return NULL;  }

int main(void) {
    int i;   pthread_t thread;

    if (pthread_create(&thread, NULL,
                       thread_func, NULL) != 0)
    {
        return EXIT_FAILURE;
    }

    for (i = 0; i < 20; i++) {
        fputs("a\n", stdout);
        sleep(1);
    }

    if (pthread_join(thread, NULL) != 0)
    {
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

Procesy a vlákna v Linux-e

- Ak „potomkový“ proces bol **vytvorený pomocou fork() – vykonáva kód rodiča od miesta, kde bol zavolaný fork()**
- Ak „potomkový“ proces bol **vytvorený pomocou clone() – vykonáva funkciu, zadanú pri volaní.**
- V Linuxe prepínanie kontextu medzi procesmi je rýchle, medzi vláknami – ešte rýchlejšie
- Vzťah proces-vlákno v Linuxe je 1:1 do verzie 2.x
- V ďalších verziách je to M:1
- Evolúcia vláknových knižníc: Linux Threads,

  NGPT, NPTL (native posix thread library) .

Vlákna v OS Linux[1]

- Štruktúra, udržovaná pre každý proces/vlákno
  - Stav (executing, ready, zombie, ...)
  - Informácia pre plánovanie
  - ID procesu, používateľa, skupiny
  - Informácia pre IPC
  - Ukazovatele na „rodiča“, „súrodencov“, „potomkov“
  - Ukazovatele otvorených súborov
  - Informácia pre virtuálnu pamäť
  - Kontext príslušných procesorov
- Vlákna sú implementované ako procesy, ktoré zdieľajú súbory, virtuálnu pamäť, signály atď.

Vlákna v OS Linux2

- Podporuje user-space thread knižnice od verzie 1.0.9.
- Od verzie 1.3.56, Linux podporuje kernel-space multithreading.
- Od verzie 2.0 jadro Linux-u podporuje SMP
- Od verzie 2.1.x, pamäťový priestor bol navrhnutý znova, takže teraz môže pristupovať k užívateľskej pamäte rýchlejšie.

Triedy pre plánovanie (Scheduling Classes)

- V Linux 2.2.*x* sú 3 skupiny procesov, ako vidno z ich definícií pre plánovač (z linux/include/linux/sched.h):

```
#define SCHED_OTHER  0        /* conventional, time-shared process */
#define SCHED_FIFO 1          /* real-time process */
#define SCHED_RR 2            /* real-time process */
#define SCHED_DEADLINE   3    /* since Linux 3.14, RT Earliest deadline first */
SCHED_OTHER                   /* default */
```

- Úlohy bežiace s prioritou **SCHED_FIFO** nikdy nebudú prerušené. Procesor opustia ***len*** pre čakanie na synchronizáciu medzi jadrami alebo pre explicitný *sleep()* alebo pre explicitnú požiadavku na preplánovanie.
- Úlohy bežiace s prioritou SCHED_RR sú v reálnom čase (RT) a procesor opustia len ak iná RT úloha čaká vo fronte pripravených úloh.
- Len *root* môže zmeniť triedu úlohy cez systémové volanie **sched_setscheduler()**.