EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Operační systémy

Správa procesů

Strategický projekt UTB ve Zlíně, reg. č. CZ.02.2.69/0.0/0.0/16_015/0002204

Martin Sysel
Fakulta aplikované informatiky
Univerzita Tomáše Bati ve Zlíně

Úvod do správy procesů

- Počítače vykonávají operace souběžně
  - např. kompilace programu, tisk souboru, zobrazení webové stránky, přehrávání hudby a přijímání e-mailu
- Pojmy: Job – Task – Program – Proces – Vlákno
  - Úloha (Job) – celková práce, skládá se z jednotlivých úkolů.
  - Úkol (Task) – často úkol = proces. Úkol popisuje **co** se dělá, proces **jak**.
  - Program – posloupnost instrukcí (přeložený zápis algoritmu).
  - Proces – instance běžícího programu, obsahuje jedno nebo více vláken.
  - Vlákno (Thread) – bod běhu, jednotka plánování a provádění

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013; Tanenbaum, 2015)

Analogie

- Analogie s pekařem
  - Program – recept
  - Úloha (job) – pečení (vše co souvisí s pečením pečiva)
  - Úkol (task) – upéct chleba
  - Proces – pečení chleba (akce)
  - Vlákno – předehřátí trouby, vážení a míchání ingrediencí, …
  - Data – ingredience
  - Procesor - pekař

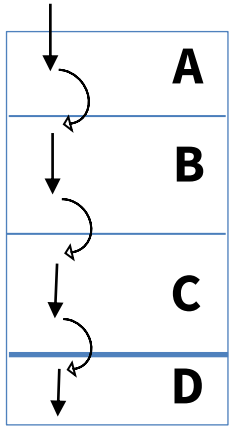
Analogie k přerušení

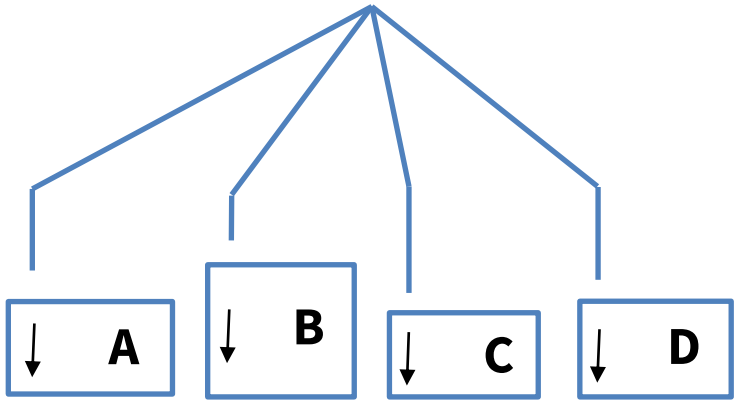
- Pekařův syn si zlomí ruku
- První pomoc pak odpovídá obsluze přerušení

(Tanenbaum, 2015)

Paralelismus X Pseudoparalelismus

- Procesy umožňují systému vykonávat paralelní operace
  - Jedno jádro CPU dokáže pracovat s jedním vláknem
    - Nebereme v úvahu HT (Intel), SMT (AMD)
  - Paralelismus X Pseudoparalelismus



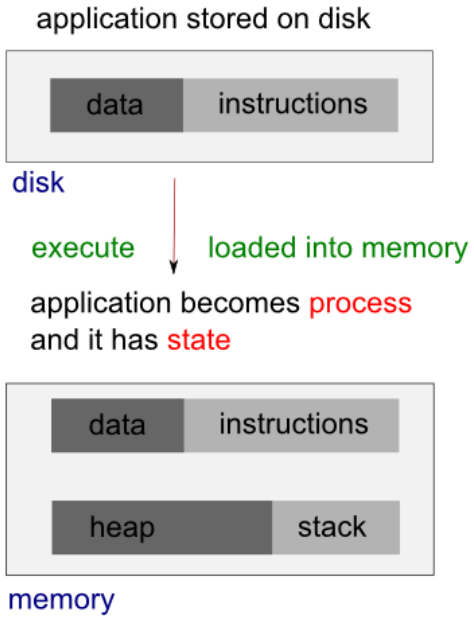


Multitasking

- Iluze běhu více procesů současně
  - Implementováno pomocí sdílení času (Time-Sharing Systems - TSS)
- Jádro OS velmi rychlé střídá procesy
  - Procesy prochází jednotlivými stavy
  - Každému procesu je přiděleno časové kvantum na procesoru
- Zvýšená režie a problémy se zabezpečením
  - např. ochrana paměti

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013; Štěpán, 2014; Tanenbaum, 2015; Lažanský, 2018)

Definice procesu

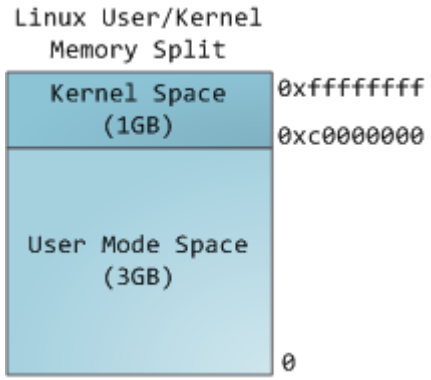


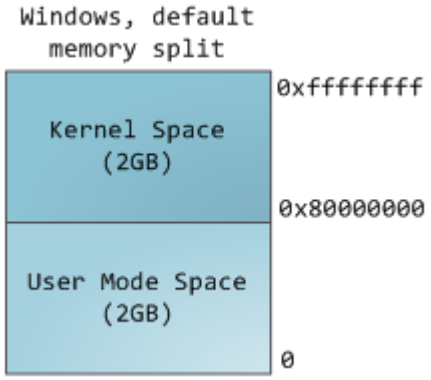
(source: BogoToBogo)

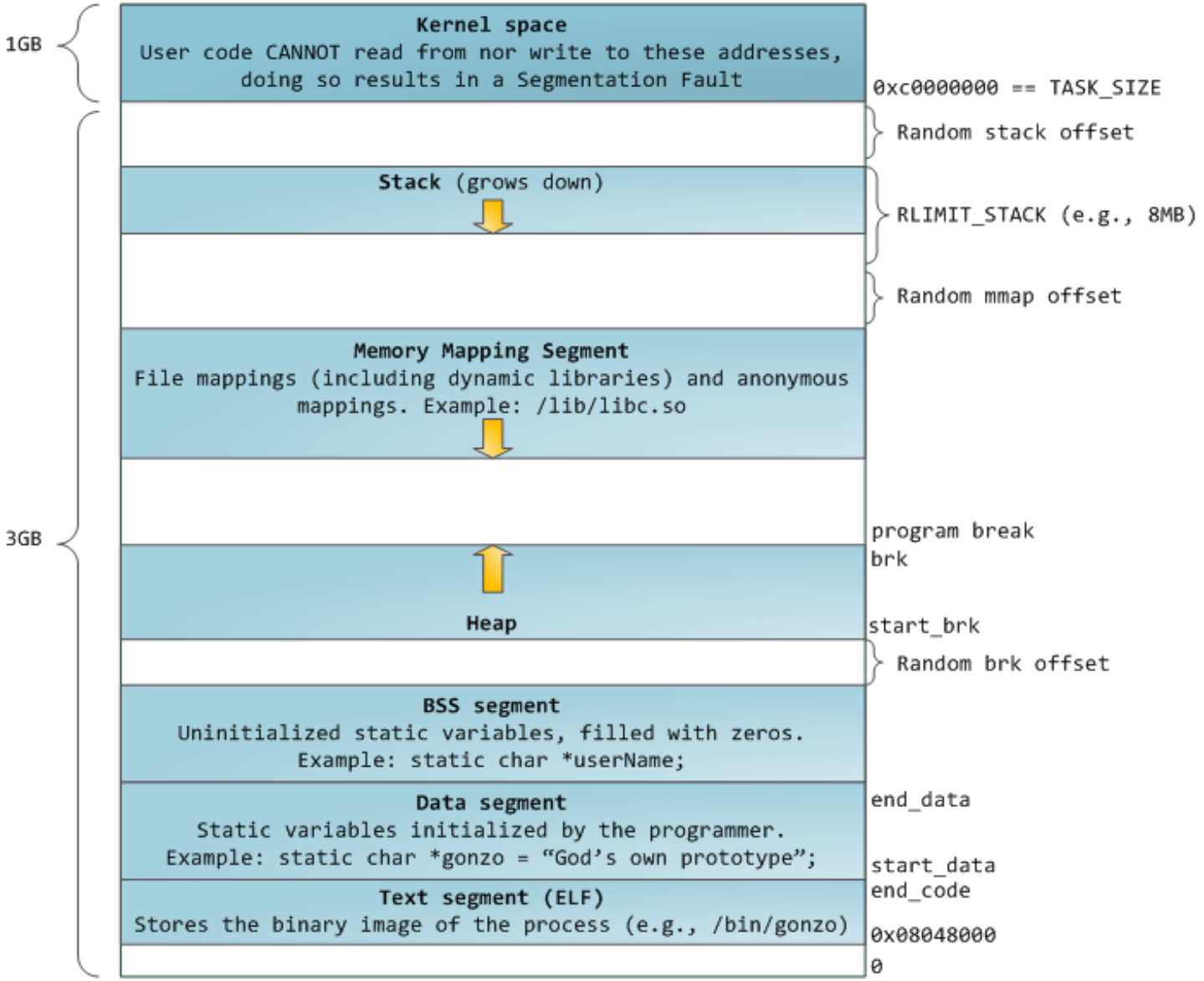
- Instance běžícího programu (běžící program)
  - Proces má svůj vlastní adresní prostor
    - Textová část – kód vykonávaný procesorem
    - Datová část – proměnné a dynamicky alokovaná paměť
      - Data segment, BSS segment, halda (Heap)
    - Zásobník (Stack) – instrukce a lokální proměnné pro volání
  - Obsahy registrů procesoru
    - čítač instrukcí, ukazatel zásobníku, uživatelské registry, FPU registry, …
  - Otevřené soubory, I/O zařízení, …
  - Obsahuje jedno nebo více vláken

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013; Štěpán, 2014; Tanenbaum, 2015; Lažanský, 2018)

Proces v paměti







(Duarte, 2009)

Proces

- Proces je identifikovatelný – PID (Process Identification) - číslo
- Jednoznačně jde určit jeho stav, zdroje které vlastní
- Podléhá plánování
- V OS podporujícím vlákna je chápán jako kontejner vláken
- Správa procesů
  - OS vykonává operace s procesy jako např. vytvoření, rušení, odložení, obnovení nebo vzbuzení procesu

Silberschatz, Galvin & Gagne, 2013; Štěpán, 2014; Lažanský, 2018)

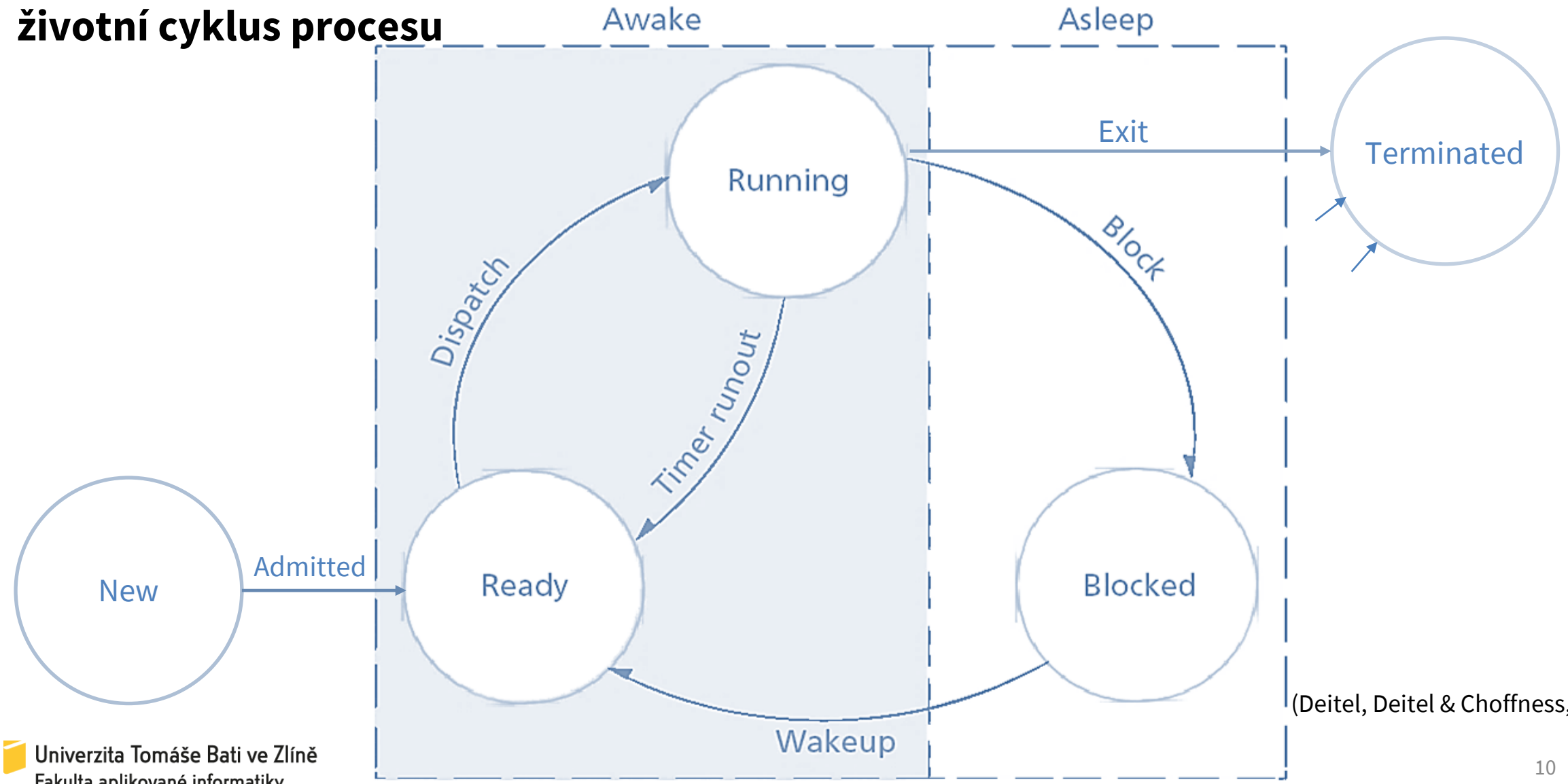
Stavy procesu

životní cyklus procesu

- procesy prochází jednotlivými stavy
  - New (nový)
    - Proces je právě vytvářen
  - Running state (stav běžící)
    - Proces je právě vykonáván procesorem
  - Ready state (stav připravený)
    - Proces by mohl být vykonáván procesorem, kdyby byl právě volný.
  - Blocked state (stav blokovaný), někdy označován jako Waiting (čekající)
    - Proces čeká až nastane nějaká událost po které by mohl pokračovat
  - Terminated (stav ukončený)
    - Proces byl ukončen. Stále vlastní některé systémové prostředky
- OS udržuje aktualizovaný seznam připravených a seznam blokovaných procesů

Stavy procesu

životní cyklus procesu



(Deitel, Deitel & Choffness, 2004)

Přechody mezi stavy

- Přechod ze stavu připravený do stavu běžící
  - Prvnímu procesu ve frontě připravených procesů je přiděleno CPU
    - Proces je odeslán ke zpracování (Dispatch)
- Přechod ze stavu běžící do stavu připravený
  - Běh procesu byl přerušen (přerušení, např. vypršelo časové kvantum)
    - Proces by pokračoval kdyby nebyl přerušen
- Přechod ze stavu běžící do stavu blokovaný
  - Proces na něco čeká, nemůže pokračovat.
- Přechod ze stavu blokovaný do stavu připravený
  - Nastala událost na kterou se čekalo a proces může pokračovat

(Deitel, Deitel & Choffness, 2004)

Fronty a seznamy procesů

- Fronta připravených procesů
  - Množina procesů připravených k běhu, čekají na procesor
    - Seřazeny podle strategie plánování (např. priorita)
- Fronta na dokončení I/O operace
  - Každé zařízení má samostatnou frontu
- Seznam odložených procesů
  - Množina procesů čekající na přidělení místa v hlavní paměti (FAP)
- Fronty související se semafory
  - Množina procesů čekajících na synchronizační události
- Fronta na přidělení paměti
  - Procesy potřebující zvětšit svůj adresní prostor
- …

- Proces přechází do fronty dle svého stavu (potřeb)

(Lažanský, 2014)

Správa procesů

- OS poskytuje základní služby procesům
  - vytvoření (create) procesu
  - ukončení (destroy) procesu
  - spuštění (dispatch)
  - blokování (block)
  - vzbuzení (wake up)
  - změna priority procesu
  - komunikace mezi procesy (IPC), synchronizace procesů
  - odložení (suspend) procesu
  - obnovení (resume) procesu

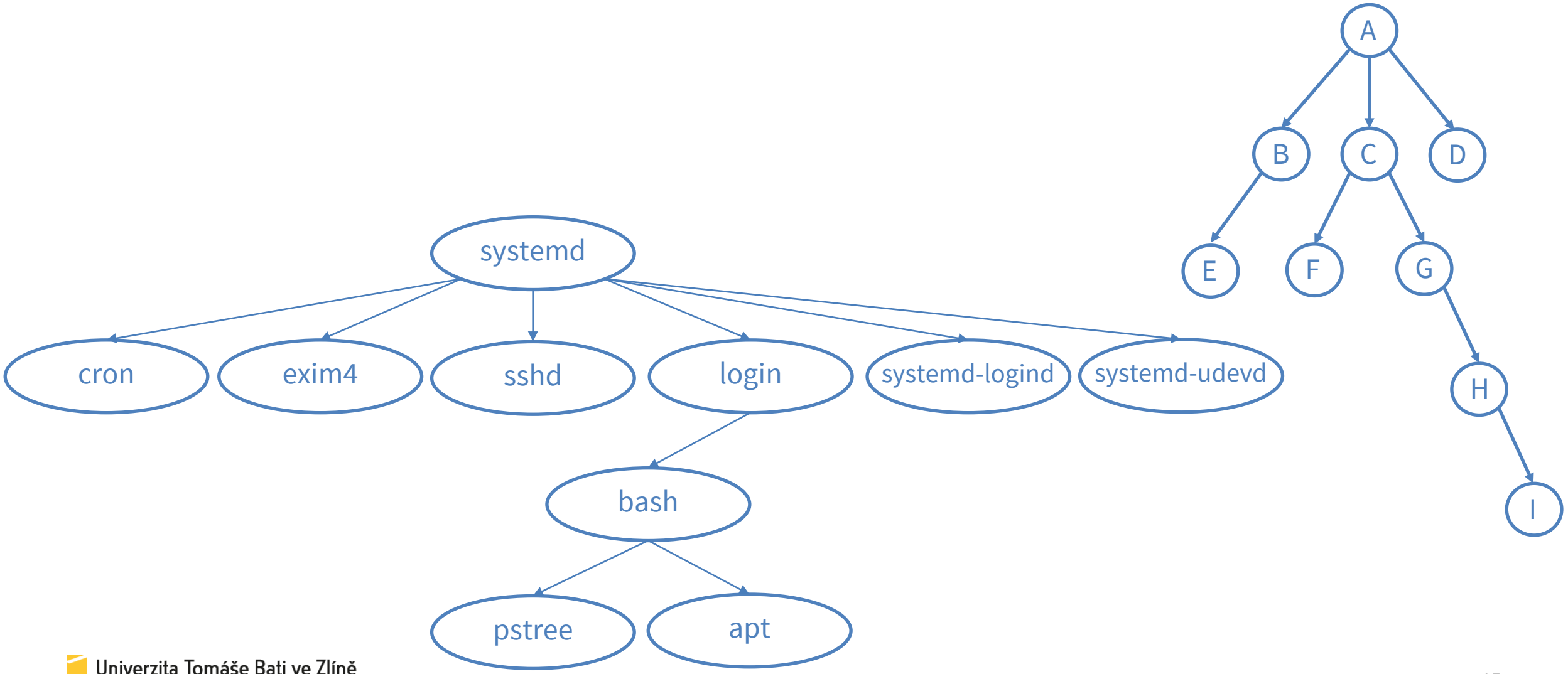
(Deitel, Deitel & Choffness, 2004)

Vytvoření procesu

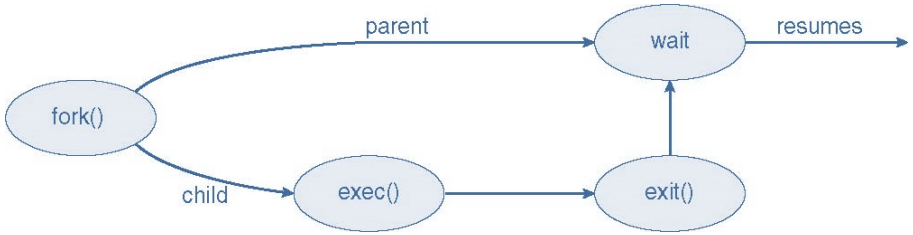
- Proces může vytvořit nový proces
  - Systémové volání createprocess
  - vytvářející proces se nazývá rodičovský
  - vytvořený proces je jeho potomek (podproces)
  - Každý podproces má právě jeden rodičovský proces
    - může ale vytvořit několik svých potomků (podprocesů).
- Takové vytváření procesů vede k hierarchické struktuře

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013)

Hierarchická stromová struktura procesů



Vytvoření procesu v Linuxu



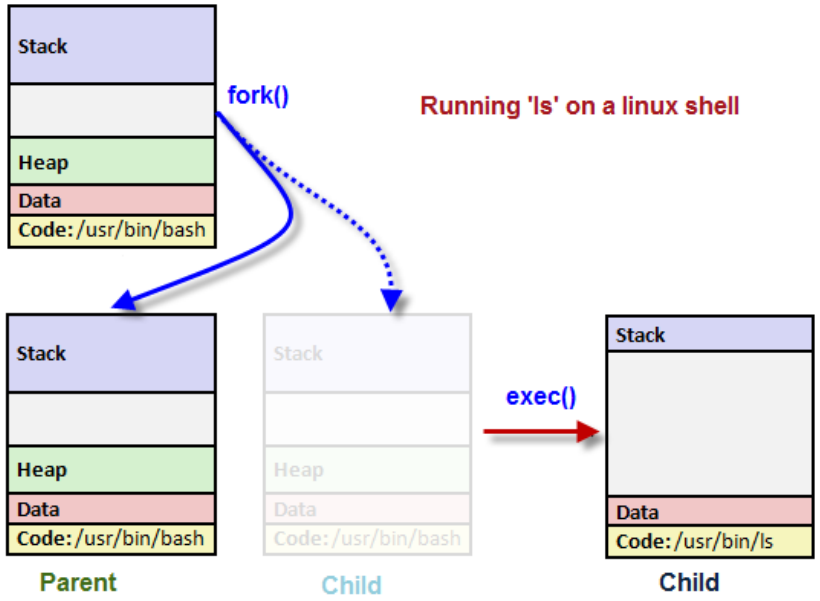
(Silberschatz, Galvin & Gagne, 2013)

- Procesy jsou tvořeny systémovým voláním fork()
  - Někdy se to nazývá forknutí procesu.
- Potomek vytvořený pomocí fork je kopií rodičovského procesu
  - Vyjma PID, které má vlastní
- Po fork pokračují oba procesy ve vykonávání
  - Potomek pokračuje ve vykonávání stejného programu jako rodič
    - Z místa, kam se navrátí po systémovém volání fork
      - Což je málokdy užitečné
- Potomek může spustit jiný program pomocí funkcí exec()
  - Spuštění nového programu zapříčiní zapomenutí všeho o předchozím procesu
  - Rodič může čekat na dokončení potomka, volání *wait* nebo *waitpid*

(http://www.gnu.org/software/libc/manual/html_node/Process-Creation-Concepts.html)

Vytvoření procesu v Linuxu, *fork()*

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main()
{
    // Fork returns process id
    pid_t child_pid, wpid;
    int status = 0;

    printf("Parent PID = %d\n", getpid());

    //child process because return value zero  (value -1 means fork failed)
    if ((child_pid = fork()) == 0)
    {
        printf("Hello from child, PID = %d\n", getpid());
        exit(0);
    }

    //parent process
    printf("Hello from parent, PID = %d\n", getpid());
    wpid = wait(&status);
    printf("Exit status of %d was %d\n", (int)wpid, status);

    return 0;
}
```



(Source: BogoToBogo)

Výstup:

Parent PID = 99
Hello from parent, PID = 99
Hello from child, PID = 100
Exit status of 100 was 0

Vytvoření procesu ve Windows

- Nový proces se vytváří funkcí CreateProcess
  - Potomek běží nezávisle na rodiči
  - Vzájemný vztah je odkazován jako *parent-child relationship*
- Jestliže je funkce úspěšná, tak vrací strukturu
  - PROCESS_INFORMATION
  - Obsahuje popisovače (handles) a identifikátory pro nový proces a jeho primární vlákno

(https://docs.microsoft.com/en-us/windows/desktop/procthread/creating-processes)

Vytvoření procesu ve Windows

```c
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

void _tmain( int argc, TCHAR *argv[] )
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );

    if( argc != 2 )
    {
        printf("Usage: %s [cmdline]\n", argv[0]);
        return;
    }

    // Start the child process.
    if( !CreateProcess( NULL,   // No module name (use command line)
        argv[1],        // Command line
        NULL,           // Process handle not inheritable
        NULL,           // Thread handle not inheritable
        FALSE,          // Set handle inheritance to FALSE
        0,              // No creation flags
        NULL,           // Use parent's environment block
        NULL,           // Use parent's starting directory
        &si,            // Pointer to STARTUPINFO structure
        &pi )           // Pointer to PROCESS_INFORMATION structure
    )
    {
        printf( "CreateProcess failed (%d).\n", GetLastError() );
        return;
    }

    // Wait until child process exits.
    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles.
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
}
```

- Struktura PROCESS_INFORMATION

```c
typedef struct _PROCESS_INFORMATION {
  HANDLE hProcess;
  HANDLE hThread;
  DWORD  dwProcessId;
  DWORD  dwThreadId;
} PROCESS_INFORMATION, *PPROCESS_INFORMATION, *LPPROCESS_INFORMATION;
```

- Popisovače procesu/vlákna se používají k určení procesu ve všech funkcích, které provádějí operace s objektem procesu.
- ID se používají k identifikaci procesu/vlákna. Hodnota je platná od vytvoření do uzavření popisovačů.

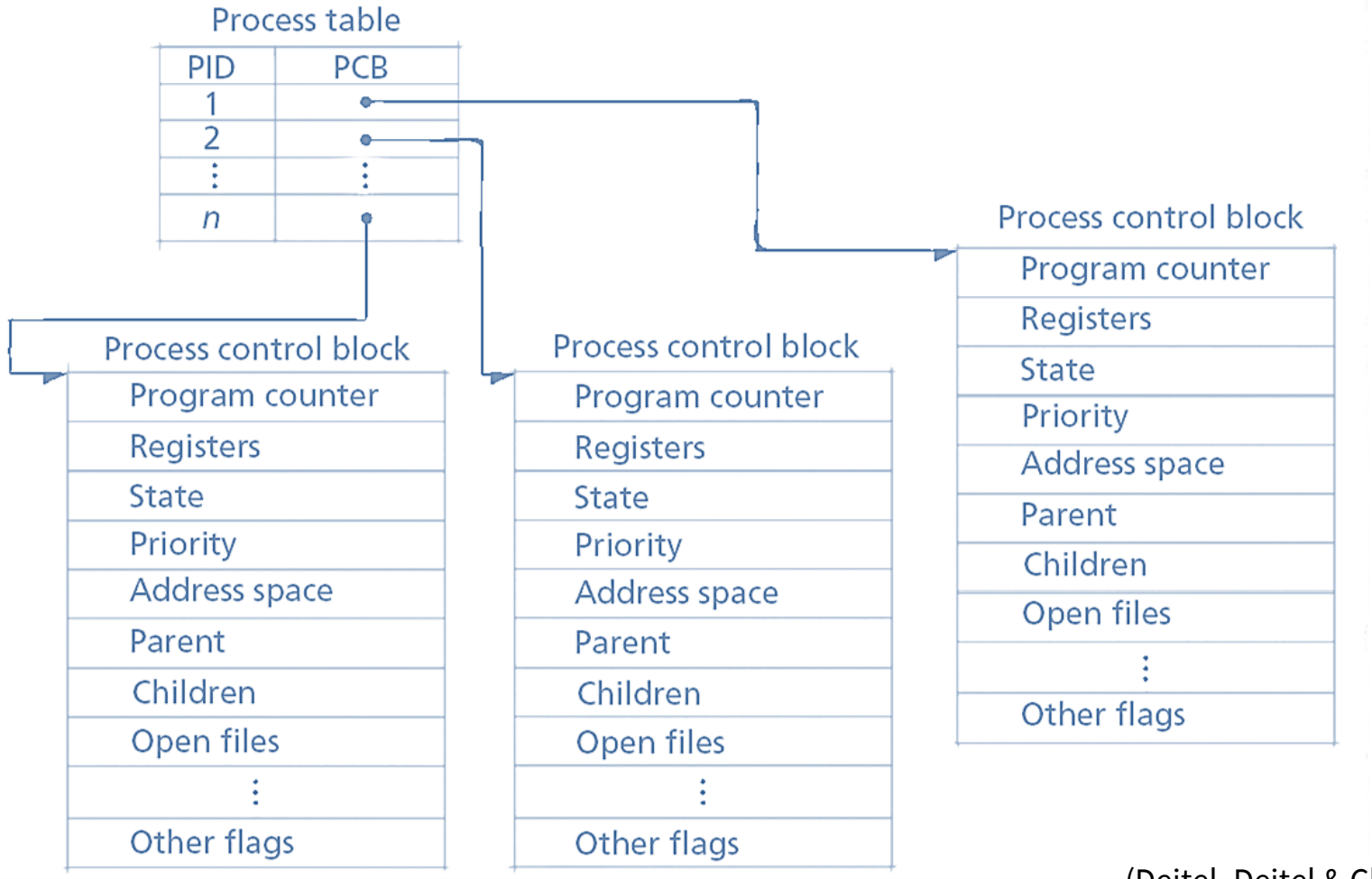
(https://docs.microsoft.com/en-us/windows/desktop/procthread/creating-processes;
https://docs.microsoft.com/cs-cz/windows/win32/api/processthreadsapi/ns-processthreadsapi-process_information)

Implementace procesů

- OS spravuje tabulku procesů nazvanou (*Process table)*
  - Obsahuje jeden záznam pro každý proces
    - Jako identifikátor se používá *PID*
  - Tento záznam odkazuje na *Process Control Block* (**PCB**)
    - Deskriptor procesu
    - Obsahuje informace potřebné ke správě procesu
    - Jedná se o datovou strukturu
      - Obsah se v různých OS liší

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015; Silberschatz, Galvin & Gagne, 2013)

Tabulka procesů a deskriptory procesů



(Deitel, Deitel & Choffness, 2004)

Deskriptor procesu (Process Control Block)

- Typicky obsahuje informace o:
  - Správě procesů
  - Správě procesoru
  - Správě paměti
  - Správu souborů
  - Správu I/O
    - Např. ukazatele na alokované zdroje
  - Statistické informace
  - …

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015; Silberschatz, Galvin & Gagne, 2013)

Obsah PCB

- **Informace pro správu procesu**
  - CPU registry
    - včetně programového čítače, stavového slova programu PSW, příznaky, stack pointer
  - stav procesu
  - ukazatel na další proces ve frontě
  - ukazatel na seznam procesů-potomků
  - ukazatel na rodičovský proces
  - skupina procesů
  - skutečné číslo uživatele (real user id)
  - skutečné číslo skupiny (real group id)
  - čas do příštího alarmu
  - ukazatel do fronty zpráv
  - příznaky nevyřízených signálů
  - číslo procesu
  - různé další příznaky

(Deitel, Deitel & Choffness, 2004)

Obsah PCB

- **Informace pro správu souborů**
  - aktuální adresář
  - kořenový adresář
  - maska práv souborů
  - popisovače souborů
  - efektivní číslo uživatele (effective user id)
  - efektivní číslo skupiny (effective group id)
- **Účtovací informace**
  - čas startu procesu
  - spotřebovaný čas procesoru
  - čas procesoru spotřebovaný procesy-potomky
  - počet přečetných a zapsaných diskových bloků
  - počet vytisknutých stránek na tiskárně

(Deitel, Deitel & Choffness, 2004)

Obsah PCB

- **Informace pro správu procesoru**
  - priorita procesu
  - časové kvantum
  - využití předešlých časových kvant
- **Informace pro správu paměti**
  - ukazatel na segment programu
  - ukazatele na další segmenty paměti
  - tabulky stránek
  - informace pro ochranu paměti
  - efektivní číslo uživatele (effective user id)
  - efektivní číslo skupiny (effective group id)
  - kód ukončení úlohy
  - stav signálů

(Deitel, Deitel & Choffness, 2004)

Střídání procesů

- Multitasking
  - Souběžné vykonávání více úkolů (procesů) v určitém časovém období
    - Nevyžaduje se nutně paralelní provádění
      - Multitasking je možný i s jedním procesorem
    - Dochází k velmi rychlému přepínání procesů
      1. Přeruší se běh procesu
      2. Uloží se jeho stav
      3. Načte se uložený stav jiného procesu
  - **Preemptivní multitasking** používá časovač, který zajistí běh po určitý interval (časové kvantum).
  - **Kooperativní multitasking** umožňuje, aby byl každý proces dokončen
    - Každý proces se vzdává času CPU pro jiný proces
      - problém: Proces může spotřebovat celý čas CPU sám za sebe

Multitasking

- Moderní OS používají **Preemptivní multitasking**
  - **Preempce** označuje přerušení probíhajícího procesu
- Přerušení (Interrupt)
  - Moderní architektury jsou řízeny přerušením (Interrupt driven)
  - Když nastane přerušení, tak je přerušen aktuálně běžící proces tak, aby ho bylo možné později obnovit
    - Ukládá se jeho stav - kontext

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015; Silberschatz, Galvin & Gagne, 2013; Štěpán, 2018)

Přepínání kontextu (Context Switching)

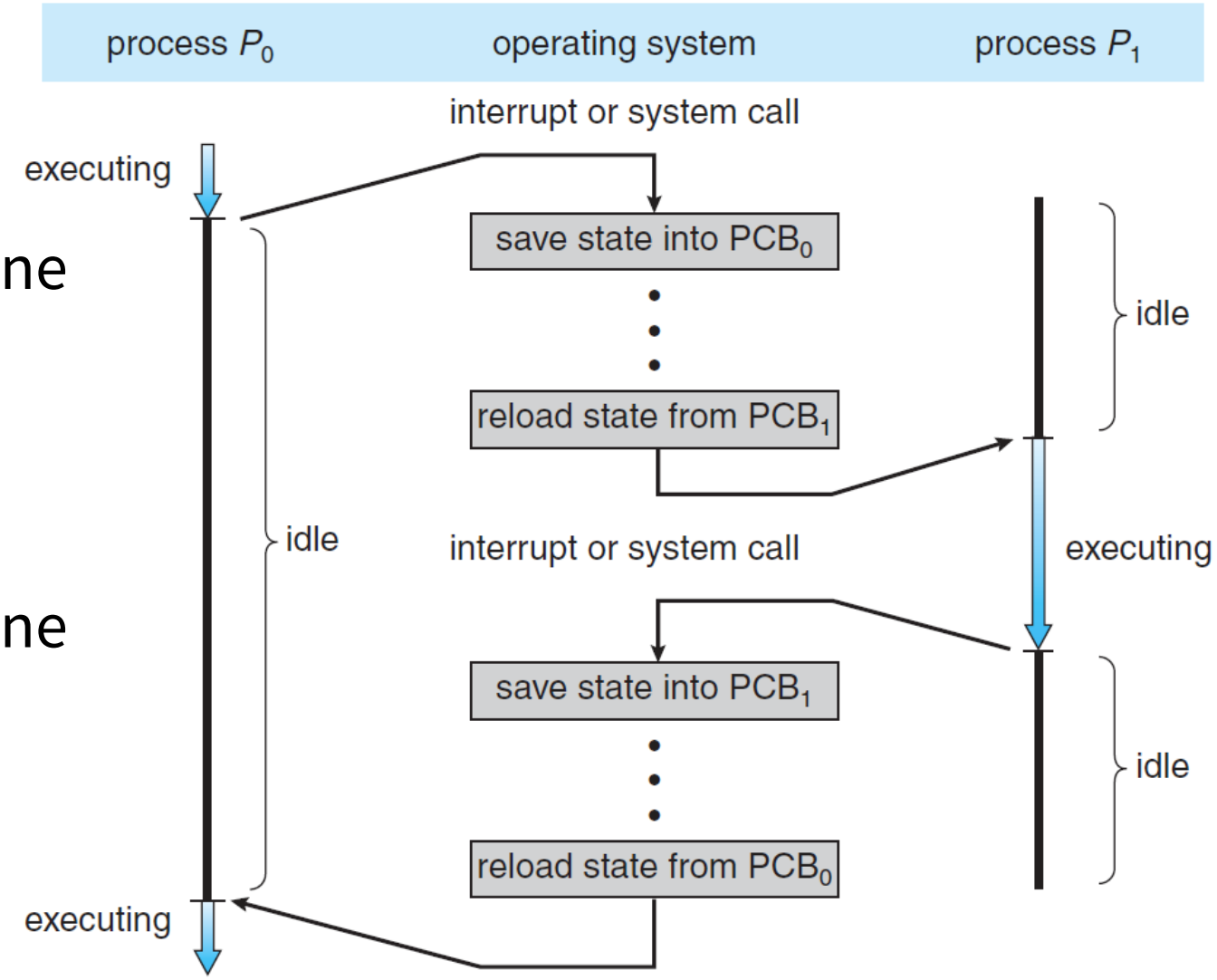
- Přepínání kontextu se skládá ze dvou částí
  - Uložení kontextu původně běžícího procesu (Context save)
  - Načtení kontextu nového procesu (Context load)
- Kontext se ukládá do Process Control Block
- Každé přepnutí kontextu je režie (žádná užitečná práce)
- Hardwarová podpora
  - měla by být poměrně rychlá, ale běžný systém ji nepoužívá
  - Hardwarové přepínání kontextu neukládá všechny registry (FPU) a naopak ukládá nepotřebné registry (segmentové registry)
- Softwarové přepínání kontextu softwaru může být selektivní
  - není o moc pomalejší
- Přepínání kontextu je pro procesy transparentní (neví o tom)

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015; Silberschatz, Galvin & Gagne, 2013)

Přepnutí kontextu

- Proces P_0 běží na CPU
- Po přerušení se kernel rozhodne spustit proces P_1
  - Uloží kontext do PCB_0
  - Načte kontext z PCB_1
- Proces P_1 běží na CPU
- Po přerušení se kernel rozhodne spustit proces P_0
  - Uloží kontext do PCB_1
  - Načte kontext z PCB_0
- Proces P_0 běží na CPU

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015)



(Silberschatz, Galvin & Gagne, 2013)

Přerušení (Interrupts)

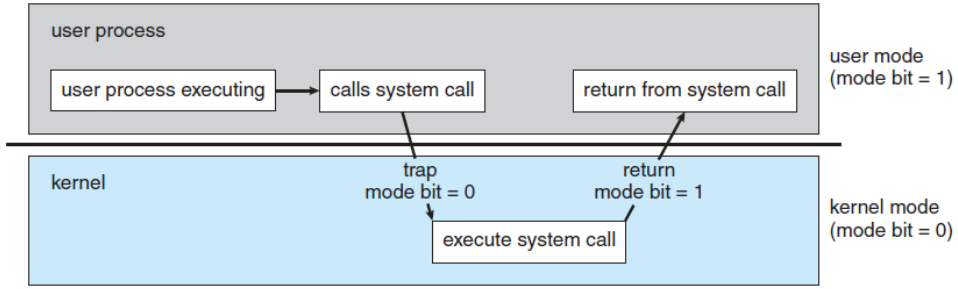
- Moderní OS je řízen přerušením (interrupt driven)
  - Důvodem jsou nižší režie
- HW přerušení (IRQ – Interrupt Request)
  - Kdykoliv může být zaslán signál do CPU
    - hardwarová přerušení jsou asynchronní (I/O, časovač, interprocesor)
- SW přerušení (Výjimka – Exception, Trap)
  - Výjimka - Může být spuštěna provedením instrukce, která způsobila chybu
    - např. dělení nulou, nelegální přístup do paměti, pokus o provedení neplatného kódu,…
  - Trap - Může být spuštěna provedením speciální operace zvané systémové volání

Pooling

Alternativní přístup.

Procesor opakovaně kontroluje stav všech zařízení. Zvyšuje se režijní zátěž spolu s rostoucí složitostí celého systému.

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013)



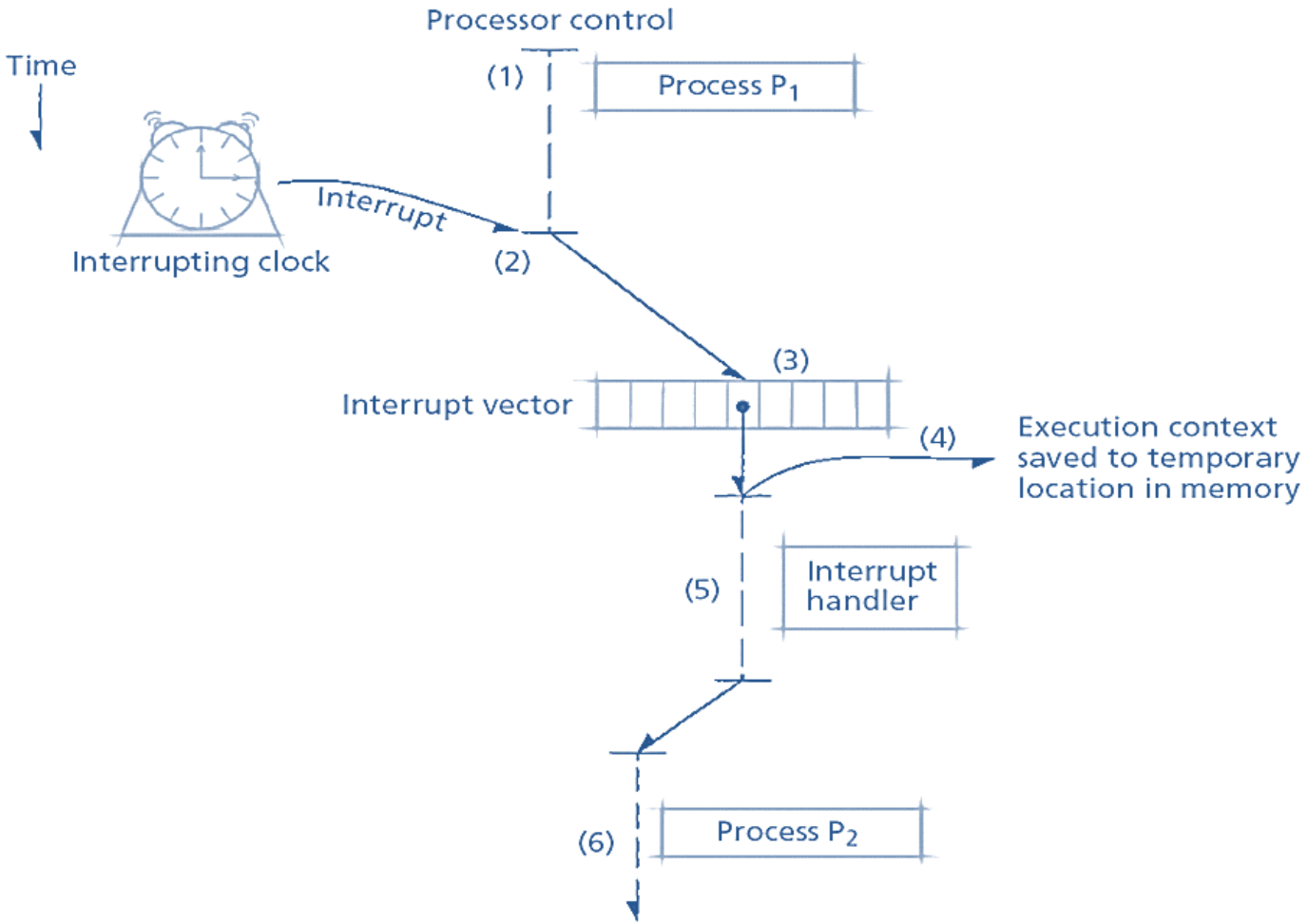
(Silberschatz, Galvin & Gagne, 2013)

Zpracování přerušení

- Po obdržení přerušení procesor dokončí provedení aktuální instrukce a poté aktuální proces pozastaví.
  - Kontext je uložen do PCB.
- Procesor poté provede jednu z funkcí jádra pro zpracování přerušení
  - Každý typ přerušení má jedinečnou hodnotu používanou jako index vektoru přerušení (*interrupt vector*), což je pole ukazatelů k obsluze přerušení (*interrupt handler*).
- Obslužný program přerušení určuje, jak bude systém reagovat
- Po dokončení obsluhy přerušení se obnoví přerušený proces (nebo nějaký jiný proces). Kontext tohoto procesu se načte.

(Deitel, Deitel & Choffness, 2004)

Zpracování přerušení



(Deitel, Deitel & Choffness, 2004)

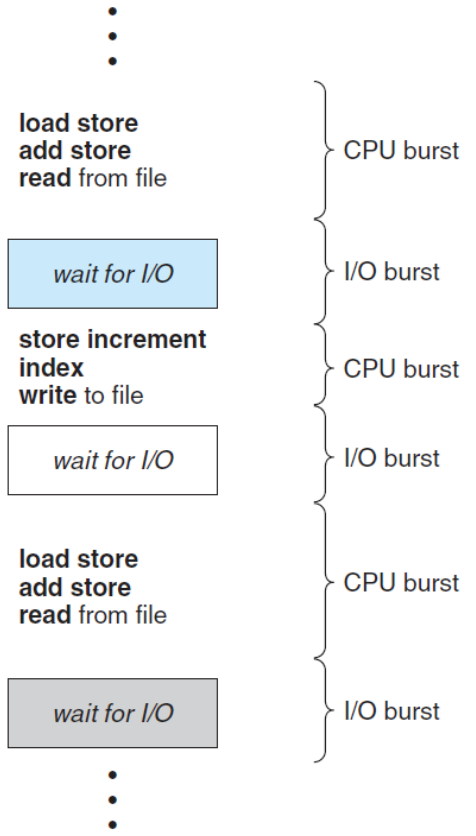
Plánovače (Schedulers)

- Dlouhodobý plánovač (Plánovač úloh - Job scheduler)
  - Vybírá, který proces má být přenesen do fronty připravených procesů
  - Dlouhodobý plánovač řídí stupeň multiprogramování
  - Je spouštěn zřídka
- Krátkodobý plánovač (plánovač CPU)
  - Vybírá, který proces by měl bude spuštěn jako další a přidělí mu CPU
  - Někdy jediný plánovač v systému
  - Tento plánovač je vyvoláván často (milisekundy)
- Střednědobý plánovač
  - Stará se o odkládání procesu z paměti
    - ukládá na disk, načte zpět z disku - swaping, stránkování (paging)

(Silberschatz, Galvin & Gagne, 2013; Deitel, Deitel & Choffness, 2004)

Typy procesů a plánování

- I/O vázaný proces
  - Tráví více času zpracováním I/O než výpočty
  - Mnoho krátkých využití CPU
- Proces vázaný na CPU
  - Tráví více času výpočtem než I/O
  - Využívá plně přidělené časové kvantum



(Silberschatz, Galvin & Gagne, 2013)

- Dobrý dlouhodobý plánovač usiluje o dobrý mix procesů
  - Rovnoměrné využití I/O i CPU

(Silberschatz, Galvin & Gagne, 2013; Deitel, Deitel & Choffness, 2004)

Cíle a kritéria plánovacích algoritmů

- Všechny systémy
  - Spravedlnost - dává každému procesu spravedlivý podíl CPU, zamezení vyhladovění (Starvation).
  - Vymáhání politiky – prosazení priorit. Zajištění toho, že uvedená politika je prováděna.
  - Rovnováha - udržování rovnoměrné zátěže všech částí systému.
- Dávkové systémy
  - Propustnost - maximalizujte počet úloh za jednotku času (např. za hodinu).
  - Doba obratu - minimalizujte čas mezi odesláním a ukončením (doba potřebná k provedení).
  - Využití procesoru – maximalizace využití CPU.
- Interaktivní systémy
  - Doba odezvy - rychlá reakce na požadavky. Doba od požadavku do první reakce.
  - Proporcionalita – použitelnost, splnění očekávání uživatelů.
- Systémy v reálném čase
  - Dodržování termínů – Dosažení meze, zaručení dokončení do dané meze. Deadline.
  - Předvídatelnost – očekávané chování systému (i při zátěži).

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015)

Cíle a kritéria plánovacích algoritmů

- uživatelské hledisko
  - doba zpracování
  - doba odpovědi
  - dosažení meze
  - Předvídatelnost
- systémové hledisko
  - Propustnost
  - Využití procesoru
  - Spravedlnost
  - prosazení priorit
  - Vyvážení I/O

- Požadavky uživatelů jdou často proti požadavkům systému

(Deitel, Deitel & Choffness, 2004; Tanenbaum, 2015)

Plánovací algoritmy

- First Come, First Serve
  - Round Robin (preemptivní FCFS algoritmus)
- Shortest Job First
  - Shortest Remaining Time Next (preemptivní SJF algoritmus)
- Prioritní plánování
- Multilevel Queue plánování (víceúrovňové)
- Multilevel Feedback Queue plánování (víceúrovňové zpětnovazební)
- . . .

Poznámka: Základní jednotka plánování je vlákno (Thread). (viz dále)

(Deitel, Deitel & Choffness, 2004; Silberschatz, Galvin & Gagne, 2013)

First Come, First Served

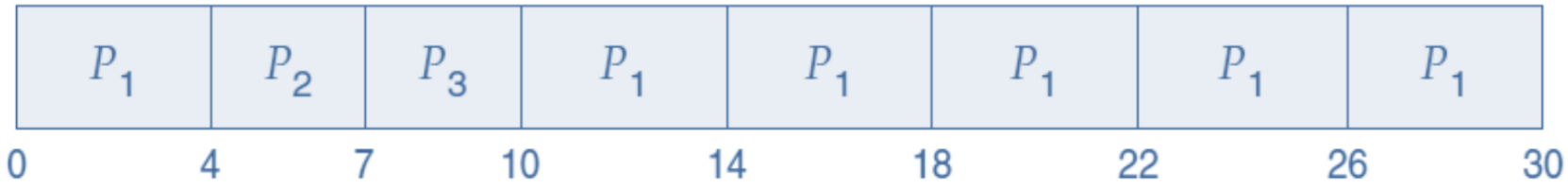
- Úkoly se provádějí podle zásady „Kdo dřív přijde, ten dřív mele“.
- Snadno pochopitelné a implementovatelné.
- Špatný výkon, protože průměrná doba čekání je vysoká.
  - Samostatně se prakticky nepoužívá

Průměrná doba čekání : (0+24+27) / 3 = 17 ms

(Silberschatz, Galvin & Gagne, 2013; Deitel, Deitel & Choffness, 2004)

Round Robin

- Každému procesu je poskytnuta fixní doba pro provedení
  - Časové kvantum (10 – 100 ms)
- Jakmile proces vyčerpá dané časové kvantum, tak je přerušen
  - Procesor je přidělen jinému procesu.
- Přepínání kontextu (Context Switching) se používá k uložení stavů procesů.

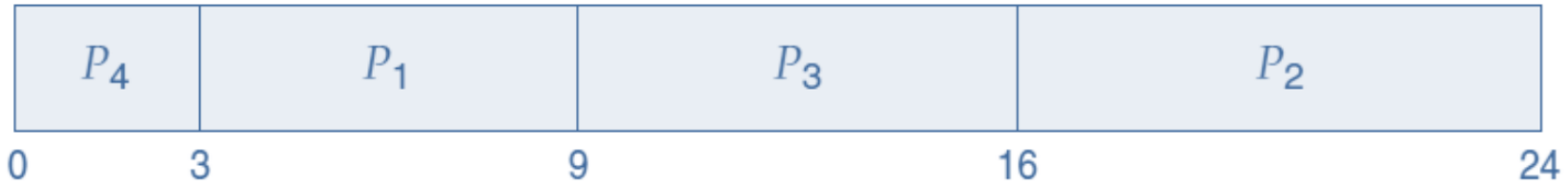


*P*1 čeká 6 ms (10 - 4), *P*2 čeká 4 ms, a *P*3 čeká 7 ms.
Průměrná doba čekání je 17/3 = 5.66 ms.

(Silberschatz, Galvin & Gagne, 2013; Deitel, Deitel & Choffness, 2004)

Shortest Job First

- Nejlepší přístup k minimalizaci čekací doby.
- Optimální (pokud je kritériem Průměrná doba čekání).
  - Nelze implementovat
  - Procesor by musel předem vědět, jak dlouho bude proces trvat.



Průměrná doba čekání: (3+16+9+0) / 4 = 7 ms

(Silberschatz, Galvin & Gagne, 2013; Deitel, Deitel & Choffness, 2004)

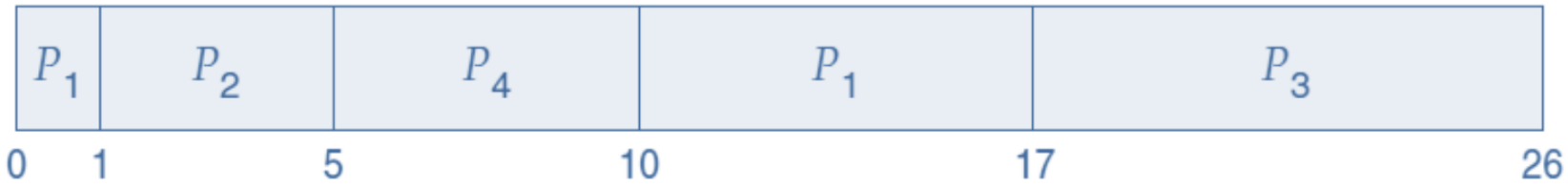
Shortest Remaining Time Next

- Nejkratší zbývající čas.
  - CPU dostane proces, který potřebuje nejméně času k dokončení.
  - Jestliže existuje proces jehož čas k dokončení je kratší, tak dojde k přerušení běžícího procesu.
- Pokus o přiblížení se k optimálnímu plánování (SJF)
  - Realizovatelné na základě předchozího použití CPU
  - Očekáváme, že příští použití CPU bude stejně dlouhé jako předchozí

Shortest Remaining Time Next

Zbývající čas pro proces *P*1 (7 ms) je větší než čas vyžadovaný pro proces *P*2 (4 ms), proces *P*1 is přerušen a proces *P*2 je spuštěn.

Proces	Čas příchodu	Využití CPU
P1	0	8
P2	1	4
P3	2	9
P4	3	5



Průměrná doba čekání
[(10 − 1) + (1 − 1) + (17 − 2) + (5 − 3)]/4 = 26/4 = 6.5 ms

(Silberschatz, Galvin & Gagne, 2013)

Prioritní plánování

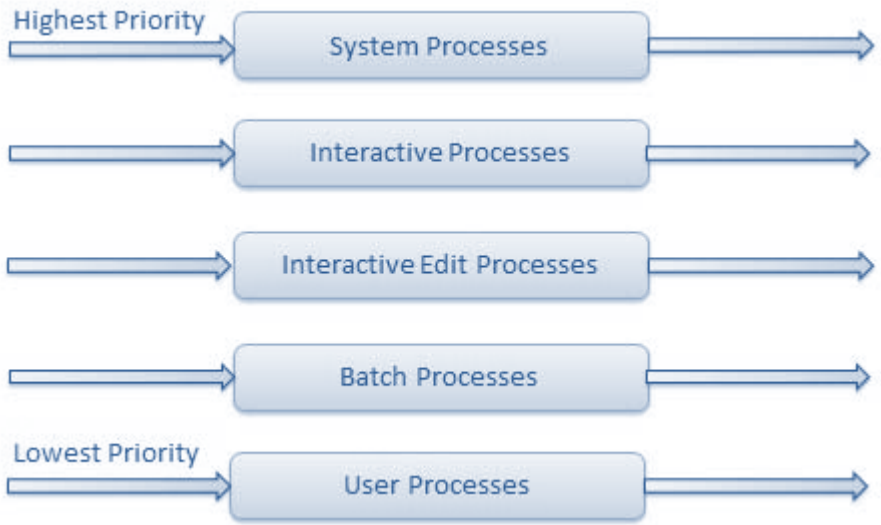
- Každému procesu je přiřazena priorita. Celé číslo.
- Procesor je přidělen procesu s nejvyšší prioritou.
  - Procesy s stejnou prioritou používají jiný algoritmus
  - Nejvyšší prioritě odpovídá nejnižší prioritní číslo (Unix, Linux, …)
    - Ve Windows je to obráceně
- Problém s plánováním priorit je blokování (hladovění - *Starvation*).
  - Připravený proces ke spuštění stále čeká na CPU, protože existují procesy s vyšší prioritou.
  - Problém je vyřešen postupným zvyšováním priority (stárnutí - *Aging*)

(Silberschatz, Galvin & Gagne, 2013; Deitel, Deitel & Choffness, 2004)

Multilevel Queue plánování

- U procesů se udržuje více front.
- Procesy jsou rozděleny do různých skupin.
- Každá fronta může mít své vlastní algoritmy plánování.
- Každé frontě je přiřazena priorita.

(Deitel, Deitel & Choffness, 2004)



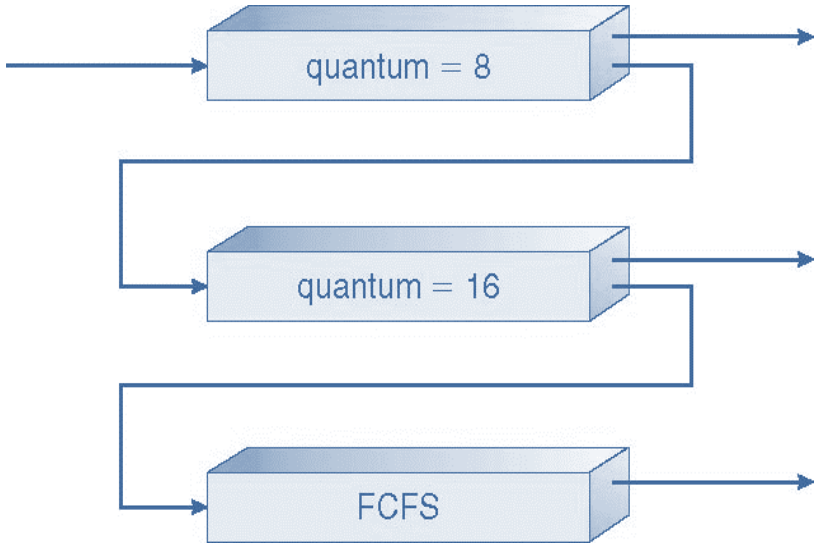
(Silberschatz, Galvin & Gagne, 2013)

Multilevel Feedback Queue plánování

- Proces se může pohybovat mezi různými frontami.
- Myšlenka: oddělené procesy s různými charakteristikami cyklu CPU.
  - Pokud nějaký proces CPU příliš používá, bude přesunut do fronty s nižší prioritou.
  - Proces, který dlouho čekal ve frontě s nižší prioritou, lze přesunout do fronty s vyšší prioritou.
  - To umožňuje vyšší prioritu pro interaktivní a I/O procesy.

Multilevel Feedback Queue plánování

- Příklad plánování
  - Proces vstupuje do fronty Q_0 a získává CPU a přidělené časové kvantum 8 milisekund
    - Pokud nedokončí práci do 8 milisekund, bude přesunut do fronty Q_1
  - Proces ve frontě Q_1 získá CPU na 16 dalších milisekund
    - Pokud se stále nedokončí, je přerušen a přesunut do fronty Q_2
  - Fronta Q_2 je plánována algoritmem FCFS



(Silberschatz, Galvin & Gagne, 2013)

Plánování v Linuxu

- Tradiční plánování v UNIXu - Před jádrem verze 2.5
  - Multilevel Feedback Queue pomocí Round Robin v každé z prioritních front.
  - Priorita na základě typu procesu a využitého časového kvanta.
    - Přepočítáno jednou za sekundu

- $P_j(i) = B_j + ½\ CPU_j(i) + nice_j$ Nižší hodnota je rovna vyšší prioritě (rozsah -20 to 19)
  - $CPU_j(i) = ½ CPU_j(i - 1)$

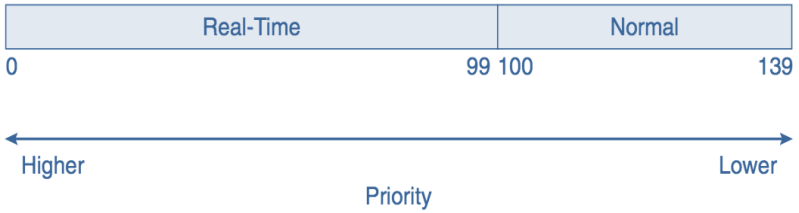
  - $CPU_j(i)$ - Využití procesoru procesem ***j*** v intervalu ***i***.
  - $P_j(i)$ - Priorita procesu ***j*** na začátku intervalu ***i***.
  - B_j - Základní priorita procesu ***j***.
  - $nice_j$ - Uživatelem nastavitelný faktor procesu ***j***. Ohleduplnost.

Plánování v Linuxu

Kernel Version 2.5 and 2.6.22 – O(1) plánovač

- Víceúrovňové zpětnovazební plánování front (preemptivní, založené na prioritách)
- Každý proces přiřazeno časové kvantum, každý procesor má vlastní fronty (runqueue).
  - Úloha je spustitelná, dokud zbývá časová kvantum
    - Navrací se do fronty připravených - aktivní stav.
  - Pokud nezbývá časové kvantum, proces se přesune do stavu expired.
  - Pokud již neexistují žádné další aktivní úkoly, jsou aktivní a expired runque zaměněny.

  - Každá fronta má 140 úrovní priority s pointery na seznamy všech procesů s danou prioritou.
    - Existuje-li více než jeden proces dané priority, použije se plánovač:
      - Vlákna jádra a úlohy RT používají FCFS nebo Round Robin. (100 statických úrovní priority - rozsah v reálném čase)
      - Uživatelské úkoly používají O(1) algoritmus (viz výše). Dynamická priorita v rozsahu od -20 do +19 (viz manuálové stránky pro *nice* a *renice*)



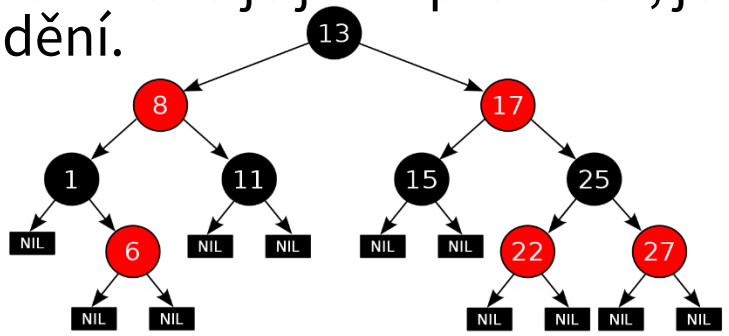
(Silberschatz, Galvin & Gagne, 2013)

Plánování v Linuxu

Kernel Version 2.6.23 +

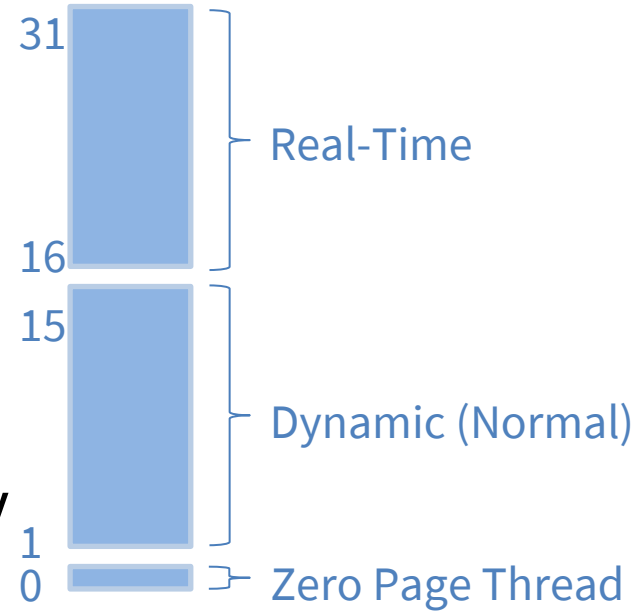
- Plánovač O(1) je nahrazen „Zcela férovým plánovačem“ (CFS)
  - Completely Fair Scheduler
- Místo front používá jednu strukturu (červeno-černý strom)
  - Udržuje procesy uspořádané podle spotřebovaného času - „čas provádění“ .
  - Uzly jsou indexovány „časem provádění“ na procesoru v nanosekundách (vruntime).
  - Pro každý nový proces se počítá „maximální doba provádění“.
    - doba, kterou by proces očekával, že bude běžet na „ideálním procesoru“ na základě rovnoměrného sdílení času
  - Procesům se vypočte časové kvantum na základě priorit.
- K provedení se odešle proces s nejnižším časem provádění (nejvíce vlevo).
  - Když proces dosáhne své maximální doby provádění nebo je jinak přerušen, je znovu vložen do stromu s aktualizovaným časem provádění.

(Lažanský, 2014; Vojnar, 2011)



červeno-černý strom(source: Wikipedia.org)

Plánování ve Windows



Thread priority levels

- Víceúrovňové zpětnovazební plánování front
  - preemptivní, prioritní, na základě interaktivity
- Úrovně priority vláken, fronta pro každou úroveň priority
  - Úrovně od 0 do 31 (nejvyšší)
  - Priorita 0 je vlákno správy paměti pro nulování volných stránek.
- První vlákno s nejvyšší prioritou se spustí první.
  - Pokud neexistuje žádné vlákno ke spuštění, spustí se vlákno idle (nečinnost).
- Priorita se dědí od rodiče, obvykle na úrovni priority 8.
- Úrovně priorit vláken jsou přiřazeny ze dvou různých perspektiv:
  - z pohledu rozhraní Win API a z pohledu jádra.
    - Win API nejprve organizuje procesy podle třídy priority a poté přiřazuje relativní prioritu jednotlivým vláknům v rámci těchto procesů.

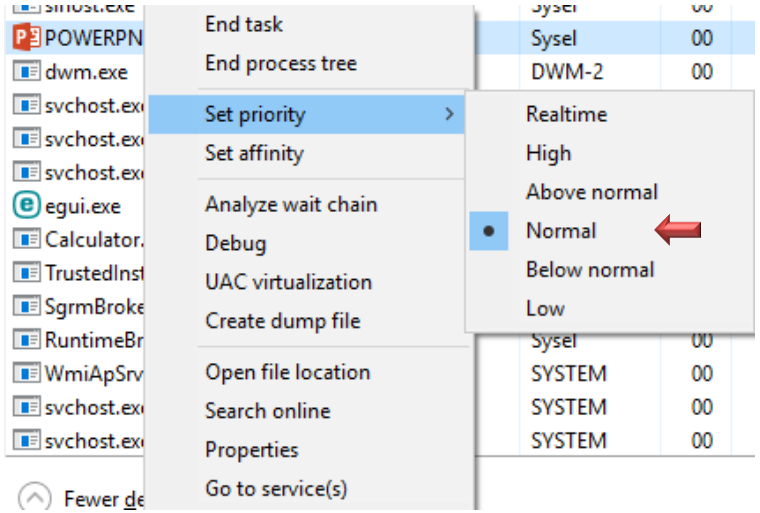
(YOSIFOVICH, P., IONESCU, A., RUSSINOVICH, Mark E. and David A. SOLOMON, 2017; Silberschatz, Galvin & Gagne, 2013)

Plánování ve Windows

- Základní priorita je dána nastavenou kombinací plánovací třídy a plánovací úrovně v rámci třídy.
- Systém může základní prioritu běžných procesů dynamicky zvyšovat či snižovat (Priority boost):
  - Zvyšuje prioritu procesů spojených s oknem, které se dostane na popředí.
  - Zvyšuje prioritu procesů spojených s oknem, do kterého přichází vstupní zprávy (myš, časovač, klávesnice, ...).
  - Zvyšuje prioritu procesů, které jsou uvolněny z čekání (např. na I/O operaci).
  - Zvýšená priorita se snižuje po každém vyčerpání kvanta o jednu úroveň až do dosažení základní priority.

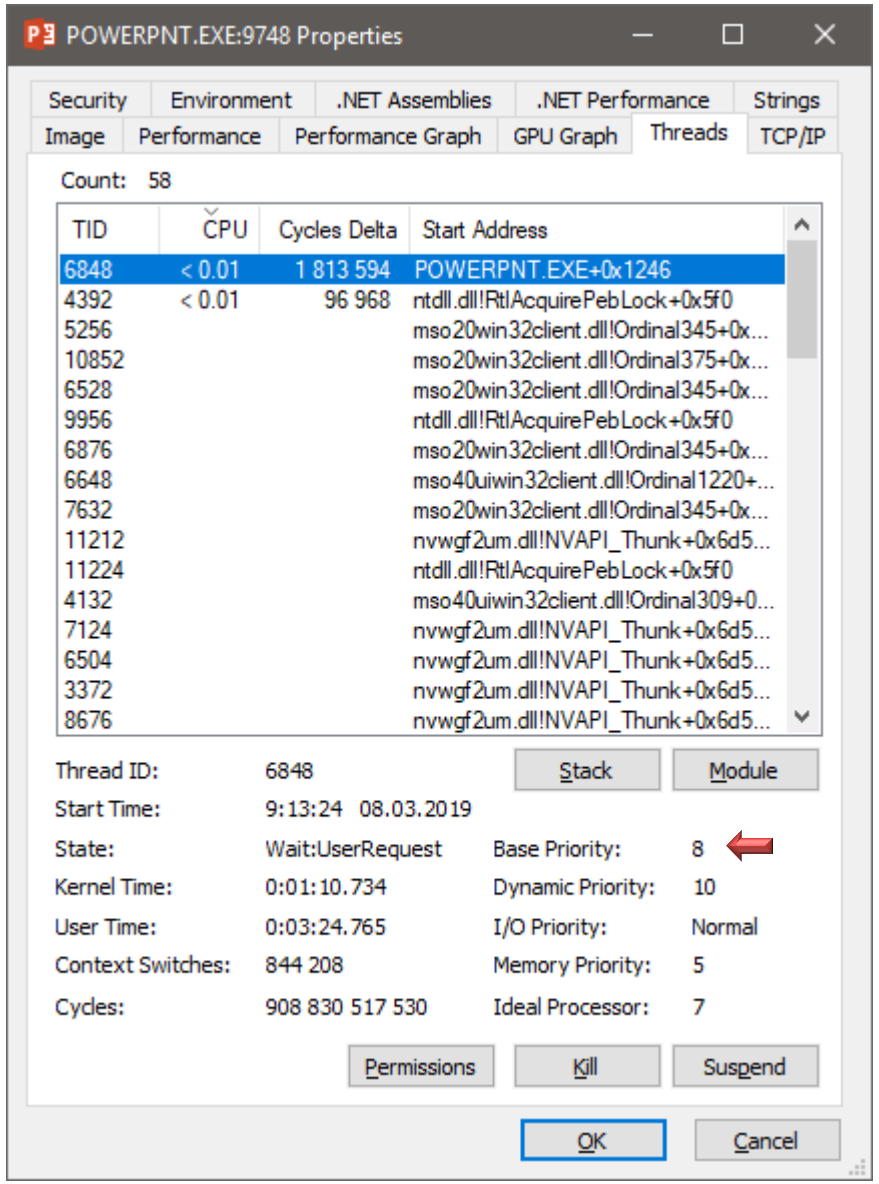
(YOSIFOVICH, P., IONESCU, A., RUSSINOVICH, Mark E. and David A. SOLOMON, 2017; Vojnar, 2011)

Windows Priority



Task Manager: Třídy Priorit (v záložce podrobnosti)

Priority Class Relative Priority	Real-Time	High	Above-Normal	Normal	Below-Normal	Idle
Time Critical (+Saturation)	31	15	15	15	15	15
Highest (+2)	26	15	12	10	8	6
Above Normal (+1)	25	14	11	9	7	5
Normal (0)	24	13	10	8	6	4
Below Normal (-1)	23	12	9	7	5	3
Lowest (-2)	22	11	8	6	4	2
Idle (-Saturation)	16	1	1	1	1	1



Process Explorer: Properties

Mapování jaderných priorit na WinAPI

(YOSIFOVICH, P., IONESCU, A., RUSSINOVICH, Mark E. and David A. SOLOMON, 2017; Silberschatz, Galvin & Gagne, 2013)

Problém inverze priorit

- Nízko prioritní proces si naalokuje nějaký zdroj, více prioritní procesy ho předbíhají a nemůže dokončit práci s tímto zdrojem.
- Časem tento zdroj mohou potřebovat více prioritní procesy
  - jsou nutně zablokovány a musí čekat na nízko prioritní proces.
- Pokud v systému jsou v tomto okamžiku další středně prioritní procesy, které nepotřebují daný zdroj, pak poběží a budou dále předbíhat nízko prioritní proces.
- Tímto způsobem uvedené středně a nízko prioritní procesy získávají efektivně vyšší prioritu než zablokované vysoko prioritní.
- Problém pokud se jedná o kritické systémové procesy
  - Může to zvyšovat odezvu systému

(Vojnar, 2011)

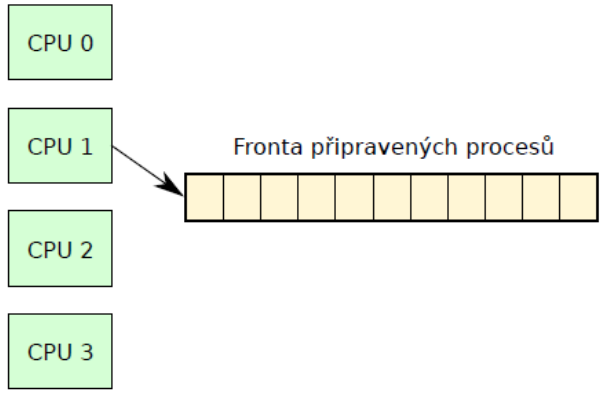
Plánování s více procesory (jádry)

- Architektura „master/slave“
  - Jeden CPU vyhrazen pro běh klíčových funkcí jádra
    - Odpovídá za plánování využití dalších CPU pro ostatní vlákna.
    - Slave žádá o služby mastera. Master může být úzkým místem.
- Symetrický multiprocesing (SMP)
  - Procesory jsou si rovny. Vlákna JOS mohou běžet kdekoliv.
  - SMP vyžaduje podporu vláken v jádře
    - Procesy musí být děleny na vlákna, aby byl SMP účinný (zvýšení účinnosti)
    - Používá většina moderních OS

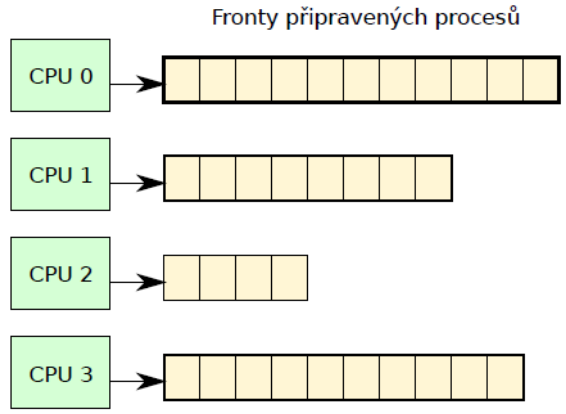
(Lažanský, 2014; Štěpán, 2018)

Symetrický multiprocesing (SMP) - plánování

- Jedna společná fronta pro všechny CPU
  - Plánovač na každém procesoru si vybere vlákno z fronty
  - Vlákna nemusí nutně pokračovat na stejném CPU
    - Problém s využití cache
- Každý procesor má svou frontu
  - Migrace je udržuje přibližně stejně dlouhé
  - Dynamické vyvažování (Load Balancing) využití jednotlivých procesorů



(Lažanský, 2014; Štěpán, 2018)



(Štěpán, 2018)

Ukončení procesu - Process Termination

- Životní cyklus procesu končí, když je jeho ukončení nahlášeno rodičovskému procesu
  - v tom okamžiku jsou již uvolněny všechny prostředky procesu, včetně jeho Process ID a záznamu v tabulce procesů.
  - Zombie (Linux)
- Důvody k ukončení procesu
  - Normální ukončení procesu (např. po dokončení požadovaného úkolu) *Dobrovolně*
  - Ukončení s chybou (např. špatné vstupní parametry při spuštění) *Dobrovolně*
  - Ukončení s kritickou chybou (např. nelegální provedení instrukce, odkaz na neexistující paměť, dělení nulou) *Nedobrovolné*
  - Ukončení jiným procesem (kill, taskkill) *Nedobrovolné*
    - Proces musí mít dostatečná práva

(Deitel, Deitel & Choffness, 2004)

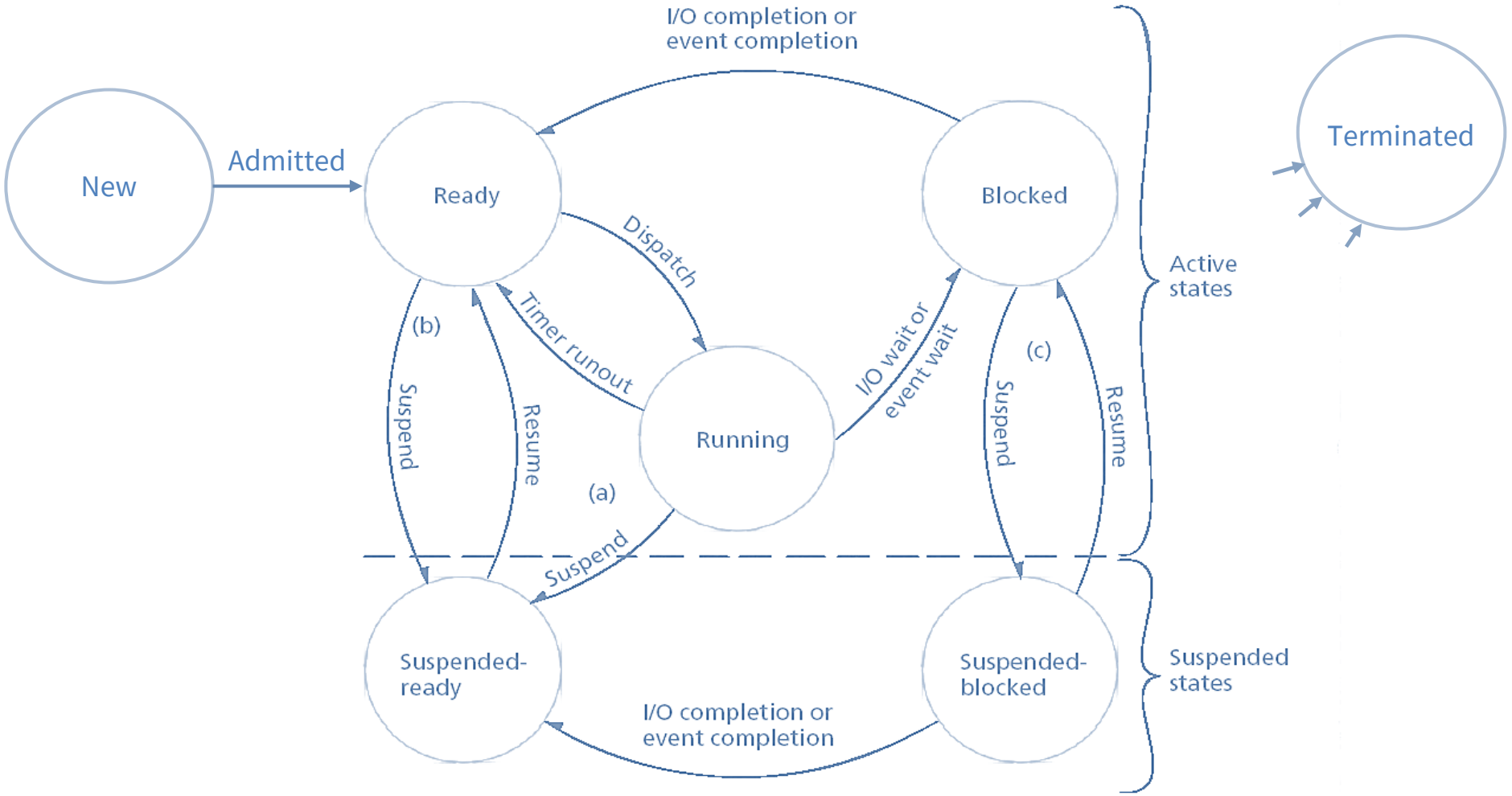
Stavy procesu: Suspend a Resume

- Odložení procesu (Pozastavení - **Suspend**)
  - Odstranění procesu z fronty na CPU, aniž by byl ukončen.
  - Nejčastější důvod: nedostatek systémových zdrojů (paměť)
    - Užitečné také pro detekci bezpečnostních hrozeb a pro účely ladění softwaru.
  - Odložení je akce s vysokou prioritou.
  - Odložení může být zahájeno pozastavením procesu nebo jiným procesem.
  - Odložený proces musí být obnoven (**Resume**) jiným procesem.

- Dva pozastavené stavy:
  - *Odložený - připravený*
  - *Odložený – blokovaný*

(Deitel, Deitel & Choffness, 2004)

Životní cyklus procesů s odkládáním



(Deitel, Deitel & Choffness, 2004)

Použitá a doporučená literatura

- DEITEL H. M., DEITEL P. J. & CHOFFNES D. R.: *Operating systems*. 3rd ed., Pearson/Prentice Hall, 2004. ISBN 0131246968.
- TANENBAUM A. S.: *Modern operating systems*. 4th ed. Boston: Pearson, 2015. ISBN 0-13-359162-x.
- SILBERSCHATZ A., GALVIN P. B. & GAGNE G.: *Operating system concepts*. 9th ed. Hoboken, NJ: Wiley, 2013. ISBN 978-1-118-06333-0.
- STALLINGS W.: *Operating Systems: Internals and Design Principles.* 8th ed., Pearson Education Limited, 2014.

Použitá a doporučená literatura

- Štěpán Petr. *Operační systémy*. Přednášky FEL ČVUT v Praze, 2018.
- Lažanský Jiří. *Operační systémy a databáze*. Přednášky FELK ČVUT v Praze, 2014.
- Staudek Jan. *Operační systémy.* Přednášky FI MUNI, 2013.
- Vojnar Tomáš. *Operační systémy*. Přednášky FIT VUT v Brně, 2011.
- Klimeš Cyril. *Principy výstavby počítačů a operačních systémů.* On-line: https://publi.cz/books/11/Cover.html

Použitá a doporučená literatura

- BogoToBogo: Linux Processes and Signals. [on-line]. Dostupné z: https://www.bogotobogo.com/Linux/linux_process_and_signals.php
- DUARTE, Gustavo. Anatomy of a Program in Memory. *Many But Finite: Tech and science for curious people.* [online]. 2009, Jan 27th, 2009 [cit. 2018-02-28]. Dostupné z: https://manybutfinite.com/post/anatomy-of-a-program-in-memory/
- YOSIFOVICH, P., IONESCU, A., RUSSINOVICH, Mark E. and David A. SOLOMON. *Windows internals: System architecture, processes, threads, memory management and more*. 7th ed. Part 1, Redmond, Wash.: Microsoft Press, 2017. ISBN 978-0-7356-8418-8.