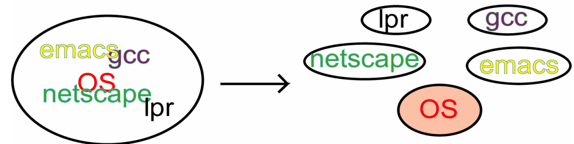
Procesy

- Všechen běžící software v systému je organizován jako množina sekvenčně běžících procesů.
- **(Sekvenční) proces**
  - Abstrakce běžícího programu.
  - Sekvence výpočetních kroků **závisí** pouze na **výsledcích předchozích kroků** a na **vstupních datech**.
- **Paralelní sekvenční procesy** (paralelní program)
  - Množina sekvenčních procesů běžících „současně“.
  - Procesy mohou běžet na jednoprocesorovém systému (**pseudo parallelism**) nebo na multiprocesorovém systému (**real parallelism**).
- **Výsledek** paralelního deterministického programu by **neměl záviset na rychlosti provádění** jednotlivých procesů.

Proč procesy? Jednoduchost

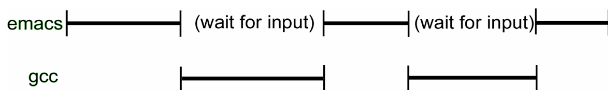
- **Systém provádí spoustu různých věcí**



- **Jak to zjednodušit?**
  - Z každé jednotlivé věci udělat izolovaný proces.
  - OS se zabývá v jednom okamžiku pouze jednou věcí.
  - Univerzální trik pro správu složitých problémů: **dekompozice problému**.

Proč procesy? Rychlost

- **V/V paralelismus**
  - Zatímco jeden proces čeká na dokončení V/V operace jiný proces může používat CPU.
  - Překrývání zpracování: dělá z 1 CPU více CPU.
  - Reálný paralelismus.

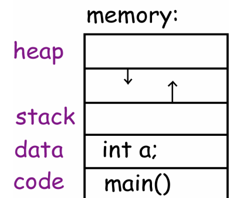


Program vs. Proces

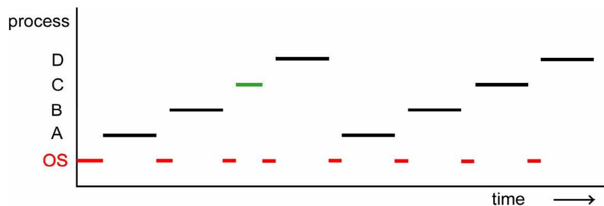
- **Program**
  - obsahují seznam instrukcí a data,
  - je pasivní (uložený v souboru).
- **Proces**
  - abstrakce **běžícího programu**, obsahuje **aktuální hodnoty** registrů a proměnných,....
- Např. spustíme dvakrát netscape:
  - stejný program,
  - ale rozdílné procesy.

file:

```
int a;
void main () {
  printf("Hello");
};
```



Přepínání kontextu

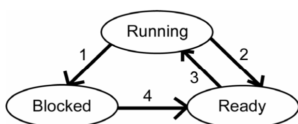


- **Pseudo paralelismus:** procesy běží (pseudo) paralelně na jedno procesorovém systému díky **přepínání kontextu**, procesor střídavě provádí kód jednotlivých procesů (**multiprogramming, timesharing, multiprocessing**).
- **Skutečný hardwarový paralelismus:** každý proces běží na svém procesoru (multiprocesorový systém).

Příklad: přepínání kontextu

- Hodnota časového kvanta je kompromisem mezi **potřebami uživatele** (malý čas odezvy) a **potřebami systému** (efektivní přepínání kontextu, přepínání kontextu trvá zhruba 2-5 ms).
- Časové kvantum v OS je zhruba 10-100 ms (parametr jádra **time slice**).

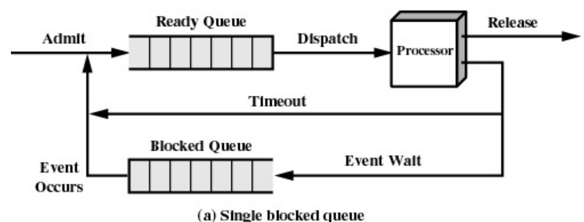
Stavy procesu



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

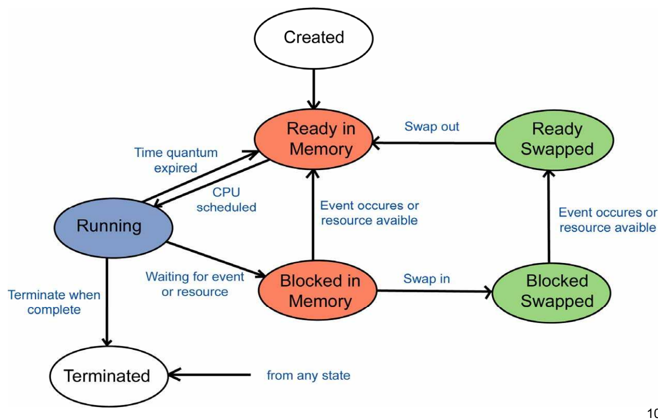
- **Základní stavy procesu**
  - **Running**: v tomto okamžiku právě používá CPU.
  - **Ready**: připraven použít CPU, dočasně je proces zastaven a čeká až mu bude přiřazeno CPU.
  - **Blocked**: neschopný použít CPU v tomto okamžiku, čeká na nějakou externí událost (např. načtení dat z disku,…).

Implementace pomocí front



(a) Single blocked queue

Stavy procesu (2)



Vytvoření procesu

- Nový proces se vytvoří když exitující proces zavolá příslušné **systémové volání** (např. `fork()` and `exec()` v Unix, nebo `CreateProcess()` v MS Windows).
- **Vytvoření**
  - OS inicializuje v jádře datové struktury spojené s novým procesem.
  - OS nahraje kód a data programu z disku do paměti a vytvoří prázdný systémový zásobník pro daný proces.
- **Klonování**
  - OS zastaví aktuální proces a uloží jeho stav.
  - OS inicializuje v jádře datové struktury spojené s novým procesem.
  - OS udělá kopii aktuálního kódu, dat, zásobníku a stavu procesu.

Příklad: vytvoření procesu v Unixu

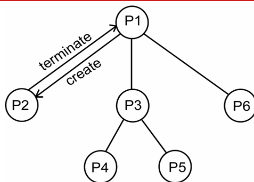
1. Systémové volání `fork()` vytvoří přesnou kopii volajícího procesu (rodič i potomek mají stejný obsah paměti, stejné nastavení prostředí, stejné otevřené soubory,…
2. Systémové volání `exec()` nahraje do paměti volajícího procesu nový program.

```
    ...
pid=fork();
if(pid != 0) {
        /* parent */
        wait(pid); /* wait for child to finish */
else
        /* child process */
        exec("ls");  /* exec does not return */
}       ...
```


Ukončení procesu

- **Normal exit** (dobrovolné)
  - Když proces dokončí svou práci, použije systémové volání, aby řekl OS, že končí (např. `exit()` v Unixu nebo `ExitProcess()` v MS Windows).
- **Error exit** (dobrovolné)
  - Například když proces zjistí fatální chybu (např. žádný vstupní soubor,...).
- **Fatal error** (nedobrovolné)
  - Chyba způsobená procesem, často např. díky chybě v programu. OS proces násilně ukončí.
- **Ukončení jiným procesem** (nedobrovolné)
  - Proces použije systémové volání, aby řekl OS o ukončení nějakého jiného procesu (např. `kill()` v Unixu nebo `TerminateProcess()` v MS Windows).

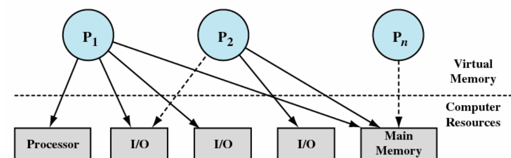
Hierarchie procesů



- V některých systémech, když proces vytvoří další proces, rodičovský proces a potomek jsou jistým způsobem svázány (např. Unix: **vztah parent process – children process**).
- Potomek může **zdědit** některé rysy od svého rodiče (např. kód procesu, globální data,…).
- Na druhé straně, každý nový proces má **svůj vlastní zásobník, reakce na signály, lokální data**.

Procesy a prostředky

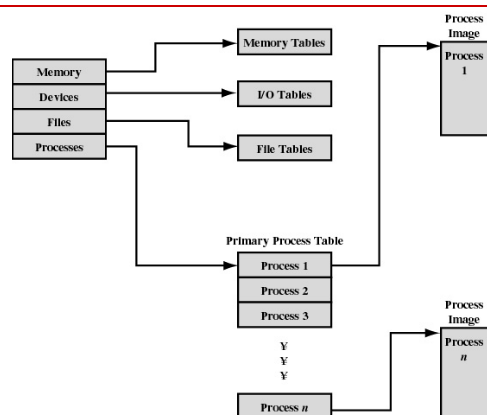
- OS spravuje procesy a systémové prostředky.
- **Informaci** o každé spravovanou entitě si udržuje v **tabulce**.



Systémové řídící struktury

- **Tabulky paměti** (memory table)
  - informace o fyzické a virtuální paměti
- **Tabulky V/V** (I/O table)
  - informace V/V zařízeních (alokace, stav,…)
- **Tabulky souborů** (file table)
  - informace o otevřených souborech
- **Tabulky procesů** (process table)
  - informace o existujících procesech

Systémové tabulky (2)



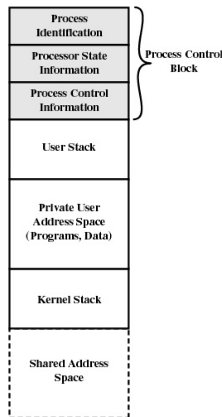
Implementace procesu

- OS zpravuje tabulku (pole struktur), která se nazývá **tabulka procesů**, s jednou položkou pro jeden proces, nazývanou **process control block (PCB)**.
- **PCB** obsahuje **informaci o procesu**, která musí být uložena, když se proces přesouvá ze stavu *running* do stavu *ready* nebo *blocked*, **tak aby mohl být restartován později**.
- Například:
  - V Unixu, maximální velikost tabulky procesů je definována parametrem jádra `nproc`.
  - PCB má v Unix přibližně 35 položek.

Položky PCB

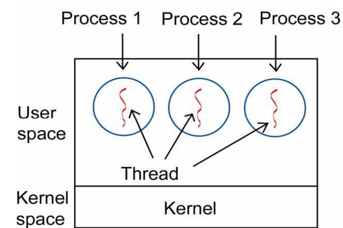
- **Informace pro identifikaci procesu** (process identification)
  - jméno procesu, číslo procesu (PID), rodičovský proces (PPID), vlastník procesu (UID, EUID), skupiny do kterých proces patří, …
- **Stavové Informace procesoru** (processor state information)
  - hodnoty viditelných registrů CPU,
  - hodnoty řídících a stavových registrů CPU (program counter, program status word (PSW), status information, …)
  - ukazatelé na zásobníky, …
- **Informace pro správu procesu** (process control information)
  - stav procesu,
  - priorita
  - informace nutné pro plánování
  - informace o událostech, na které proces čeká,
  - informace pro meziprocesovou komunikaci,
  - informace pro správu paměti (ukazatel na tabulku stránek,…)
  - ukazatelé na kód, data a zásobník programu, ...
  - alokované a používané prostředky,…

Položky PCB (2)



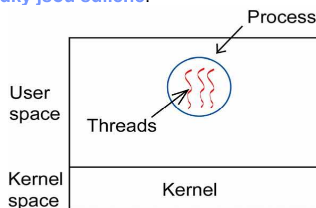
Proces vs. vlákno

- **Proces model**
  - Každý proces **alokuje příslušné prostředky** (adresové prostor obsahující kód, data a zásobník procesu, otevřené soubory, potomky, reakce na signály, ...)
  - **Jedno vlákno výpočtu**.



Proces vs. vlákno (2)

- **Vláknový model**
  - Odděluje alokaci prostředků a samotný výpočet.
  - **Proces** slouží k **alokaci společných prostředků**.
  - **Vlákna** jsou **jednotky plánované pro spuštění** na CPU.
  - **Vlákno** má svůj vlastní **program counter** (pro uchování informace o výpočtu), **registry** (pro uchování aktuálních hodnot), **zásobník** (který obsahuje historii výpočtu), **lokální proměnném**, ale **ostatní prostředky jsou sdílené**.

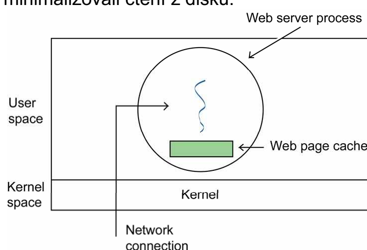


Vláknový model

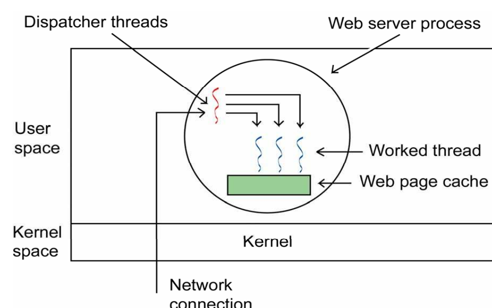
- Jednotlivá vlákna v daném procesu **nejsou nezávislá** tak jako jednotlivé procesy.
- Všechny vlákna v procesu **sdílí** stejný adresový prostor, stejné otevřené soubory, potomky, reakce na signály, …
- **Multithreading**
  - Procesy se spouští s jedním vláknem.
  - Toto vlákno může **vytvářet další vlákna** pomocí knihovní funkce (např. *thread_create(name_of_function)*).
  - Když chce vlákno skončit, může se opět **ukončit** pomocí knihovní funkce (např. *thread_exit()*).

Příklad: jednovláknový Web Server

- **Klient**
  - pošle požadavek na konkrétní web. stránku
- **Server**
  - ověří zda klient může přistupovat k dané stránce
  - načte stránku a pošle obsah stránky klientovi
- **Často používané stránky** zůstávají uloženy v **hlavní paměti**, abychom minimalizovali čtení z disku.



Příklad: vícevláknový Web Server



Příklad: vícevláknový Web Server (2)

- **Dispatcher thread:** čte příchozí požadavky, zkoumá požadavek, vybere nevyužité pracovní vlákno a předá mu tento požadavek.
- **Worked thread:** načte požadovanou stránku z hlavní paměti nebo disku a pošle ji klirntovi.

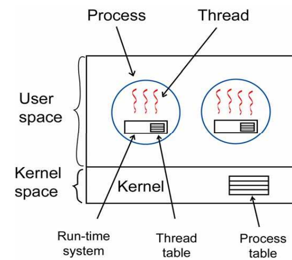
Dispatcher thread

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

Worked thread

```
while(TRUE) {
  wait_for_work(&buf);
  look_for_page_in_cache(&buf,&page);
  if (page_not_in_cache(&page))
    read_page_from_disk(&buf,&page);
  return_page(&page);
}
```


Implementace vláken v uživatelském prostoru

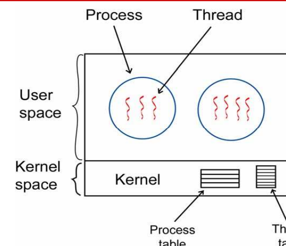


- **Run-time system**: množina funkcí, která spravuje vlákna.
- Vlákna jsou implementována pouze v uživatelském prostoru.
- Jádro o vláknech nemá žádné informace.

Implementace vláken v uživatelském prostoru (2)

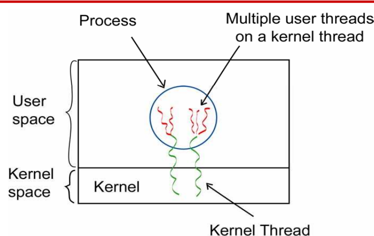
- **Výhody**
  - Vlákna mohou být implementována v OS, které nepodporuje vlákna.
  - Rychlé plánování vláken.
  - Každý proces může mít svůj vlastní plánovací algoritmus.
- **Nevýhody**
  - Jak budou implementována blokující systémová volání? (změna systémových volání na neblokující nebo požití systémového volání `select`)
  - Co se stane když dojde k výpadku stránky?
  - Žádný clock interrupt uvnitř procesu. (jedno vlákno může okupovat CPU během celého časového kvanta procesu)

Implementace vláken v prostoru jádra



- Jádro má tabulku vláken, která obsahuje informace o všech vláknech v systému.
- **Výhody**
  - Žádný problém s blokujícími systémovými voláními.
- **Nevýhody**
  - Vytváření, ukončování a plánování vláken je pomalejší.

Hybridní implementace vláken



- Jádro se stará pouze o **kernel-level threads** a plánuje je.
- Některé kernel-level threads mohou mít user-level threads.
- **User-level threads** jsou vytvářená, ukončovaná a plánovaná uvnitř procesu.
- Např. Solaris, Linux, MS Windows

Hybridní implementace vláken (2)

- Anderson et. al. 1992.
- **Kombinuje** výhody **uživatelských vláken** (dobrá výkonnost) s výhodami **kernel vláken** (jednoduchá implementace).
- **Princip**:
  - Jádro přidělí určitý počet **virtuálních procesorů** každému procesu.
  - (Uživatelský) **run-time system** může **alokovat** vlákna na virtuální procesor, **požádat o další** nebo **vrátit** virtuální procesory jádru.

Hybridní implementace vláken (3)

- **Problem**: blokující systémová volání.
- **Řešení**:
  - Když jádro ví, že **bude vlákno zablokováno**, jádro aktivuje **run-time system a dá mu o tom vědět** (tzv. „upcall“).
  - Aktivovaný run-time systém **může naplánovat další** ze svých vláknen.
  - Když původní vlákno je opět ve stavu „ready“, jádro provede znova upcall, aby to oznámilo run-time systému.

Vlákna ve Windows XP

- **Job**: množina procesů, které sdílejí kvóty a limity (maximální počet procesů, čas CPU, paměť, …).
- **Process**: jednotka, která alokuje zdroje. Každý proces má aspoň jedno vlákno (thread).
- **Thread**: jednotka plánována jádrem.
- **Fiber**: vlákno spravované celé v uživatelském prostoru.