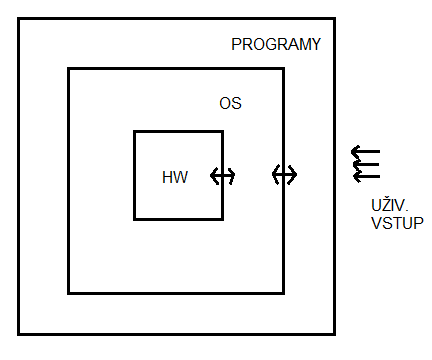
**OS a plánování procesů**

**Charakteristika OS**

* jedná se o program, respektive kolekci programů, které tvoří spojující vrstvu mezi hardwarem (může být i virtualizován) a uživatelskými programy
* je základním programovým vybavením PC, který se zavádí do operační paměti při startu PC a zůstává v činnosti až do jeho vypnutí
* zajišťuje spuštění procesů a jejich správu, správu paměti, správa souborových systémů, obsluhu virtuální a vnější paměti, příkazového procesoru, uživatelského rozhraní



**Typy jader OS**

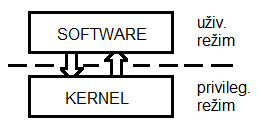
Jádro

* největší a nejzákladnější část OS – core, kernel
* je zavedeno jako první při startu OS a bězí celou dobu jeho činnosti
* nemělo by nikdy ztratit kontrolu nad PC -> privilegovaný režim (nutná podpora procesoru)
* může provádět libovolně operace s HW
* navazuje přímo na HW (skutečný i virtualizovaný)
* pro uživatele a jeho aplikace je zcela zapouzdřené
* zajišťuje základní správu prostředků a tvorbu prostředí pro vyšší vrstvy OS a uživatelské aplikace (přepínaní kontextu)
  + zavádění stránek či práce s virtuální pamětí (nastavení HW)
* systém i uživatelské programy mohou explicitně žádat jádro o služby prostřednictvím systémových volání (sys\_call)
* využívá speciálních instrukcí, které způsobují kontrolovaný přístup do režimu jádra
* 2 typy volání jádra:
  + Kernel interface – přímé volání
  + Library interface – nepřímě volání
    - Voláme funkce ze systémových knihoven jež mohou, ale nemusí vést volání služeb jádra (fopen)
* Služby jádra dostupné v unixových systémech – open, close, read, kill, fork

Typy jader OS

1. Monolitické jádro

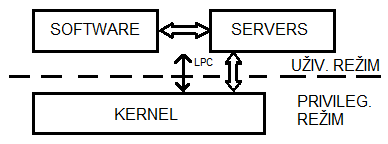
* Veškerý kód běží v privilegovaném režimu a je těsně provázán za účelem vysoké efektivity
* Veškerý kód =
  + všechny subsystémy implementující služby jako např. správa paměti, plánování, meziprocesní komunikace, souborové systémy nebo síťová komunikace
  + chyba v 1 subsystému může ovlivnit další subsystémy a nebo až shodit celé jádro -> operační systém



* dynamické nahrávání modulu
  + modul je možno přidat až za běhu a není nutný restart systému
  + např: obsluha USB, síťové protokoly
  + moduly jsou zavedeny do adresového prostoru jádra, kde se s ním propojí
  + při zavádění může dojít ke zpoždění -> řešení = zavádět při startu systému
* představitelé: MsDos, Windows 95/8/ME, Mac OS do 8.6 -> nemodulární
  + - FreeBSD, Linux -> modulární

1. Mikrojádro

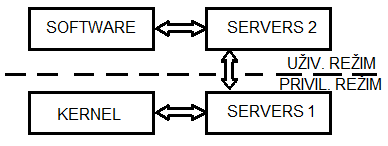
* Velmi malé, minimalizuje rozsah a nabízí jednoduché rozhraní s jednoduchými abstrakcemi a malým počtem služeb pro nejzákladnější správu procesoru, vstupní a výstupní zařízení, paměti a meziprocesní komunikace



* Ostatní služby jako např. plánování nebo správa souborového systému případně ovladače jsou implementovány mimo jádro do tzv. serverů -> neběží v implementovaném režimu, ale v uživatelském
* Výhody:
  + Jednodušší programování systému díky rozdělení na log. Celky
  + Flexibilita (možnost více současně běžících implementací různých služeb)
  + Možnost pozastavit nebo odpojit jednotlivé servery
  + Zabezpečení – jelikož servery neběží v privilegovaném režimu tak pokud v nich bude chyba (útok na ně) tak by nemělo dojít k ovládnutí nebo pádu PC
* Nevýhody:
  + IPC (inter process communication) – velká režie díky zasílaní IPC zpráv
* Představitelé: minix, symbionOS

1. Hybridní jádro

* Kombinace předchozích dvou jader



* Mikrojádra, která jsou rozšířená o kód jež by mohl být implementován v podobě serveru, ale za účelem menší režie je těsně provázán s mikrojádrem a běží v jeho režimu (souborový systém, síťové protokoly)
* Nedokáže za běhu samozavádět moduly tak jako monolitické jádro
* Představitelé: veškeré komerční OS – MacOS, Windows 7,8,10

1. EXO jádro

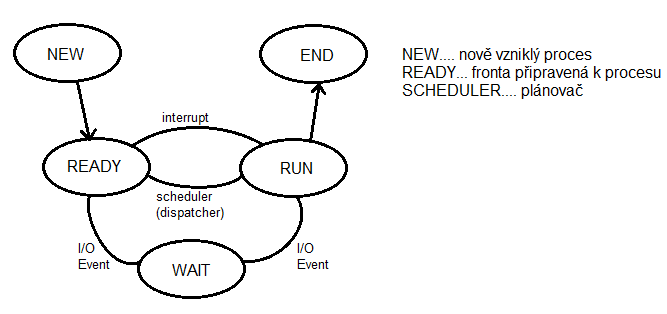
* Experimentální jádra, poskytující velmi nízké rozhraní zaměřené hlavně na bezpečné sdílení prostředků
* Jsou ještě menší než mikrojádra
* Představitelé – Aegis, Nemesis

1. Nano jádro

**Proces vs Vlákno**

Proces

* Běžící program



*SCHEDULER- plánuje jak budou procesy využity (dispatcher provádí)*

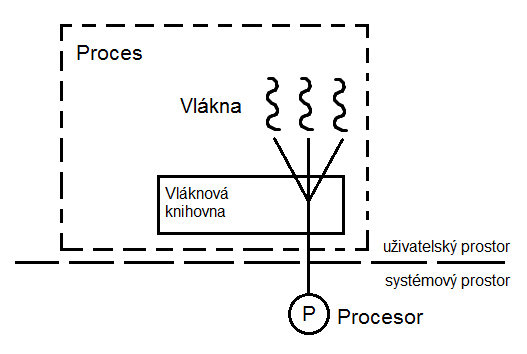
*INTERRUPT – vnitřní časovač, který po určité době odebere procesor procesu*

* Umístěn v operační paměti v podobě strojových instrukcí
* proces je definován:
  + indentifikátorem (PID)
  + programem, kterým je řízen
  + obsahem registru (čítač instrukcí), adresa zásobníku
  + daty (proměnné, konstanty, atd…)
  + využití dalších zdrojů procesu a vazbou na jeho objekty
* procesů v OS leží spousty a je nutné je spracovat
  + process managment
  + přepínání kontextu – dispatcher
  + plánovač – scheduler – plánuje, v jakém pořadí budou procesy využity
  + správa paměti – memory managment – přiděluje procesu určité časti OP

Vlákna

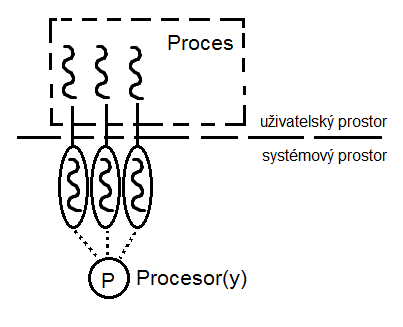
* odlehčený proces – thread -> nemůže existovat samo -> patří do procesu
* vlákna podléhají plánování
* životní cyklus podobný jako u procesu
* snižuje se režie OS na přepínání kontextu
* vlákna na rozdíl od procesu mezi sebou sdílí paměť
* mají stejná práva
* OS, který nepodporuje vlákna má potom 1 proces = 1 vlákno
* Vlákno je možné použít jak na uživ. úrovni tak na úrovni jádra OS

Uživatelská úroveň



* OS o vláknech neví, jsou plně v režii programátora
* Je k tomu potřeba knihovna -> k obsluze vláken
* NEVÝHODY
  + Procesorový čas je přidělovaný procesoru, nikoliv vláknu
  + Pokud vlákno volá službu OS tak zablokuje všechna ostatní vlákna v daném procesu
* VÝHODY
  + Rychlejší přepínaní vláken
  + Uživatelský proces má plnou kontrolu nad vlákny

Úroveň jádra



* OS ví o vláknech a má je plně pod kontrolou
* Je možno využít více procesorů -> běh více vláken najednou
* Procesorový čas je přidělovaný jednotlivým vláknům -> nefér mezi procesy, budou mít jiný počet vláken -> tzn. jeden proces bude mít více času na zpracování než druhý
* Volání služby neblokuje ostatní vlákna
* Zástupci: Windows NT (New Technology) a Linux
* Použití: vykreslování grafiky, zpracování vstupních dat
* VÝHODY
  + Urychlení výpočtu, efektivní využití systému
  + Jednodušší komunikace a sdělení než mezi procesy
  + Umožňují paralelní běh, což je využití více jader nebo procesorů v systému
  + Lepší a přehlednější struktualizace programu
* NEVÝHODY
  + Náročnější přepínání vláken
  + Složité sledování toku programu
  + Synchronizace vláken (aby nedošlo k tomu, že 2 a více vláken mění 1 proměnnou)
  + Omezení počtu vytvořených vláken – pro efektivní využití systému

PCB vs TCB

PCB

* Process control block
* Dynamická datová struktura obsahující nejnutnější informace týkající se daného procesu
* Každý proces má svůj řídící prvek
* Umístění= ve stejné časti jako jádro (privilegovaný režim)
  + Z důvodu ochrany => bez přístupu uživatelů a jiných procesů

|  |  |
| --- | --- |
| UKAZATEL | STAV PROCESU |
| ČÍSLO PROCESU | |
| PC | |
| REGISTRY | |
| LIMIT PAMĚTI | |
| SEZNAM OT. SOUBORŮ | |
| Řídící blok procesu | |

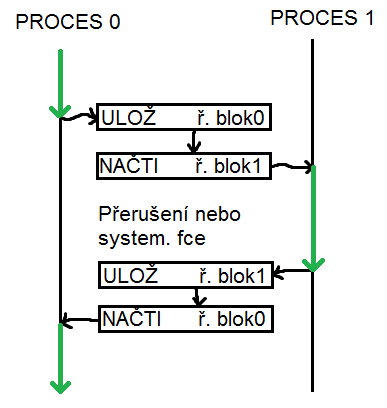
* Ukazatel - ukazatel na další PCB nebo konec seznamu
* Stav procesu- ready, wait
* Číslo procesu – PID - process ID
* PC- program counter – čítač instrukcí
* Registry – obsah procesoru
* Limit paměti – velikost přidělení OP
* Seznam souborů – soubory se kterými pracoval

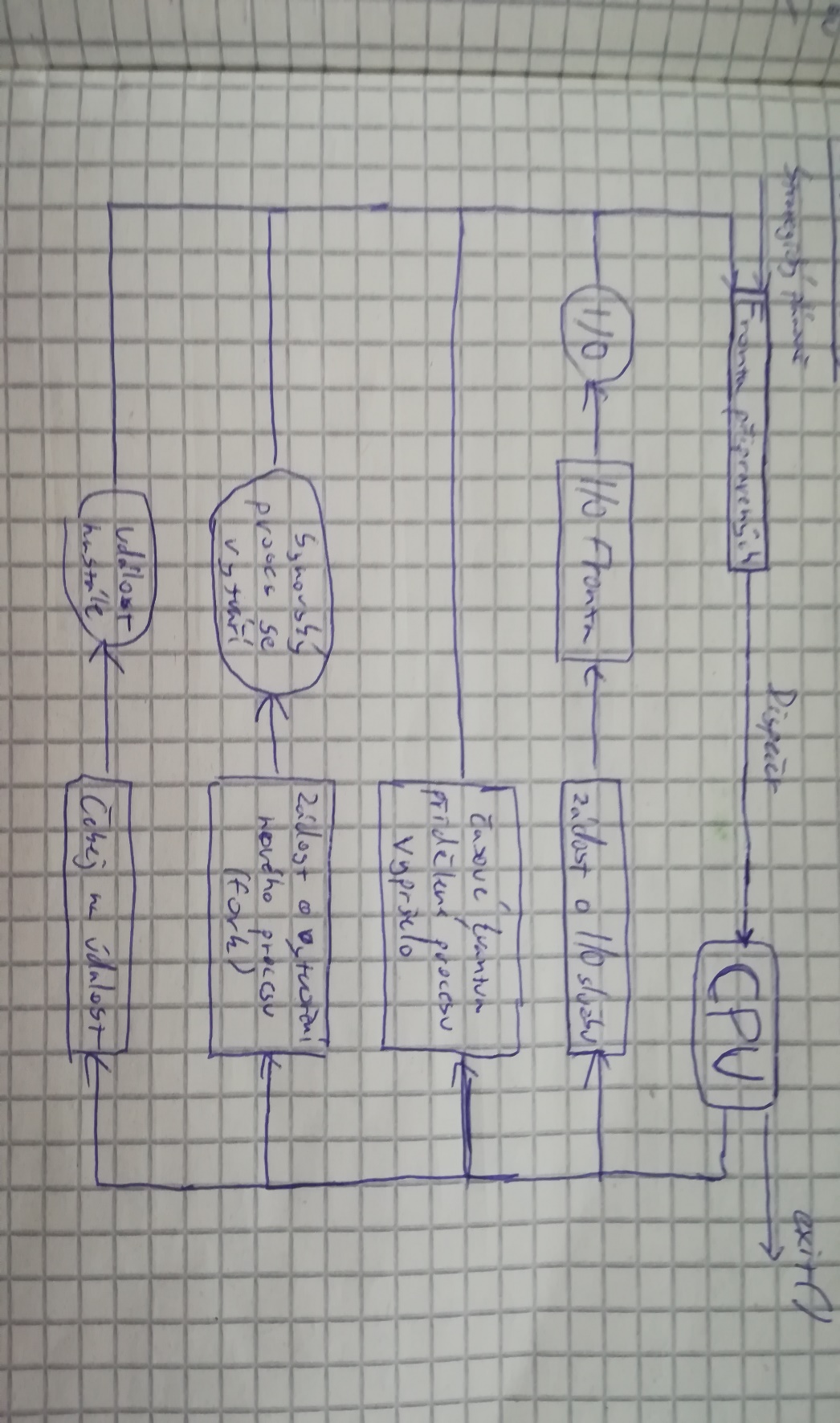
TCB

* Thread control block
* Datová struktura obsahující informace specifické pro vlákno potřebné pro jeho správu
* OBSAHUJE
  + Číslo vlákna – thread identifier TID
  + Ukazatel zásobníku – ukazuje na zásobník podprocesů v procesu
  + PC – program counter – čítač instrukcí
  + STAV vlákna – running, ready, waiting, start, done
  + Registr – obsah vlákna
  + Ukazatel na PCB procesu ve kterém je vlákno

**Přepínání kontextu**

* Vymění se procesy na procesoru
* Operace přepnutí z jednoho procesu na druhý
  + Opakuje se mnohokrát za sekundu
* Přepnutí trvá řádově desetiny milisekund, samotný proces jednotky a desítky milisekund
  + Dochází k němu i při obsluze přerušení nebo změně režimu
* Musí být velmi rychlé aby nebylo poznat „zamrznutí“ systému
  + Uživateli se jeví, že všechny procesy běží najednou
* Přepnutí může být řešeno jak HW tak SW
  + U HW je problém se správným načtením registru
  + Nejčastěji SW, ale je pomalejší



**Plánovače OS**

* Krátkodobý (short term)
* Dlouhodobý (long term)
* Středně dlouhý (mid term)

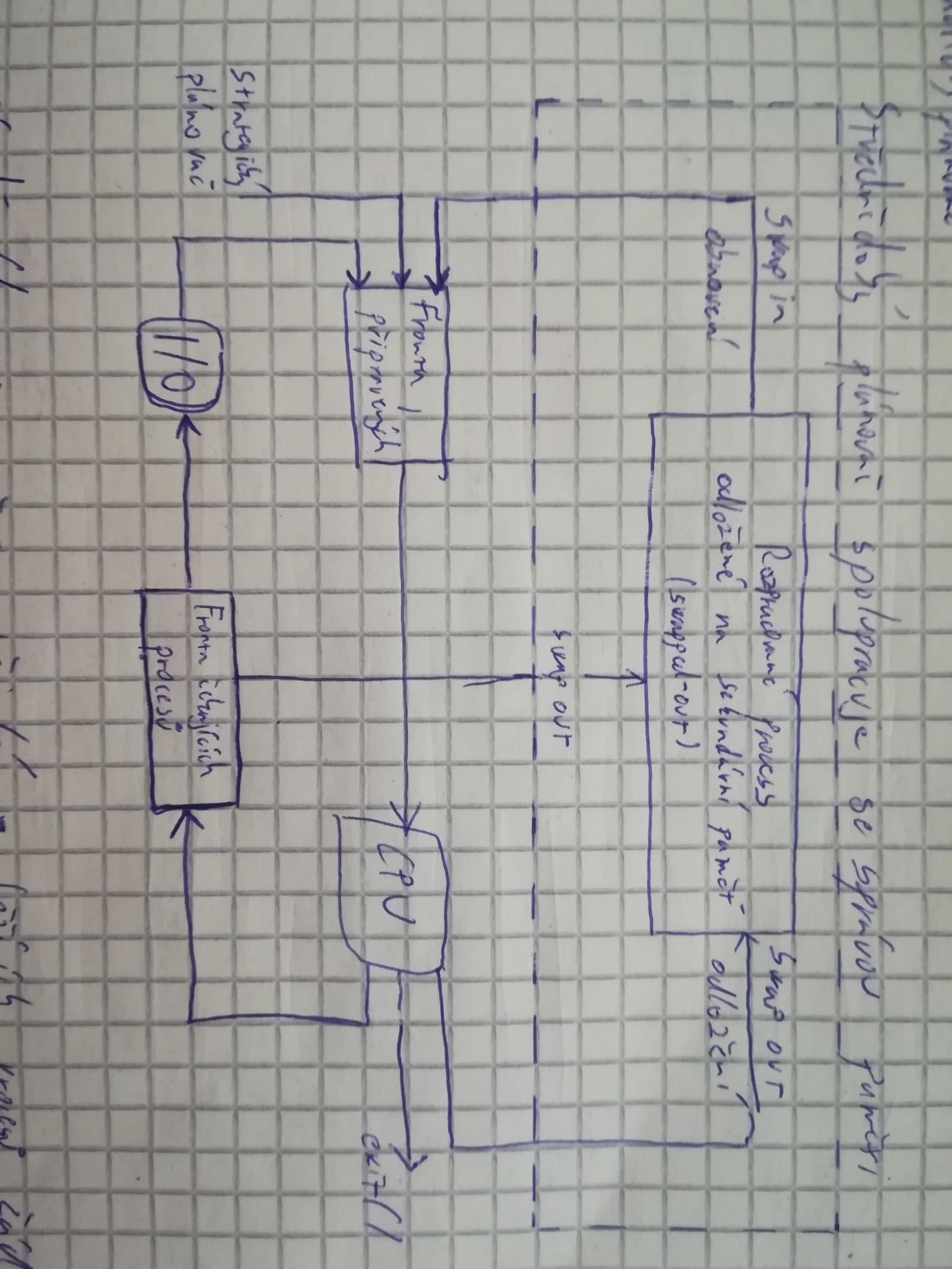
Krátkodobý

* 2 fronty
* Fronta I/O operací – postupně obsluhuje procesy až jsou obslouženy jdou do fronty připravených procesů
* Fronta připravených procesů – ready
* Procesor pracuje paralelně s I/O operacemi
* U běžícího procesu můžou nastat následující situace:

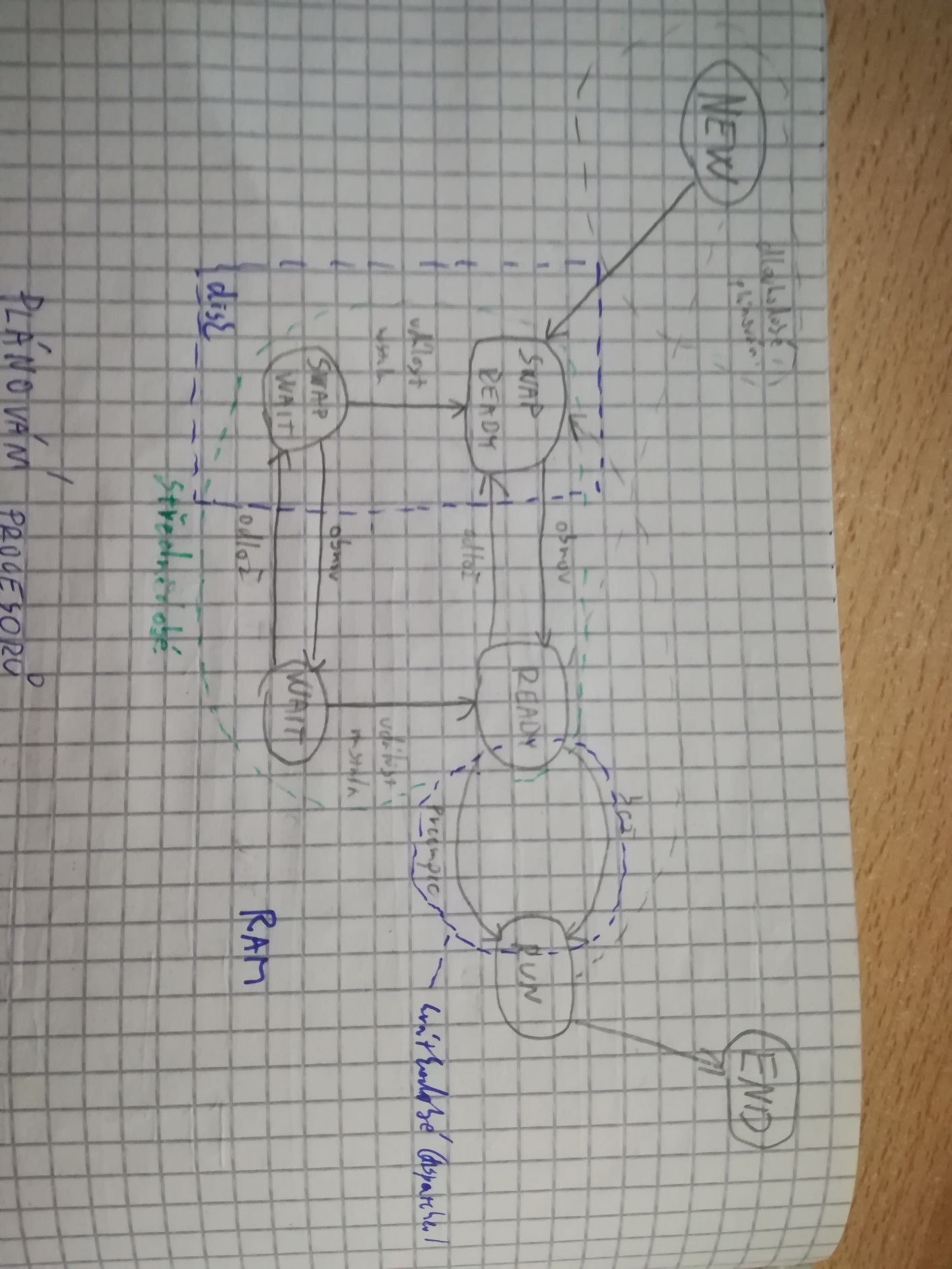
1. Proces požaduje nějakou I/O operaci
2. Proces vytvořil podproces a čeká na dokončení
3. Procesu je následně odebráno CPU z důvodu vypršení času a je
4. Umístěn do fronty připravených procesů -> proces může čekat na výstup z jiného procesu

* 1. 2. a 4. – stav wait
* Pokud je prázdná fronta, tak procesor zahálí, pracuje na prázdno a kdykoliv jej může vyrušit plánovač
* Vytváření potomka systémovým voláním fork -> rodičovský proces čeká až potomek zanikne
* Přerušení může být voláno např. při přijetí soketu prostřednictvím síťové karty – po obsloužení toho přerušení, které dostalo přednost pokračuje další proces fronty

Dlouhodobý

* Označován taky jako plánovač úloh
* Aktivace v řádově jednotkách až desítkách sekund -> krátkodobý vyvoláván neustále (jednotky ms)
* Jde o vhodnou kombinaci několika úloh náročnou na I/O operace a CPU (dávkové zpracování)
* V interaktivních systémech (widnows, atd..) se prakticky nepoužívá -> zastupuje ho krátkodobý plánovač
* Využití: systémy reálného času (dispatching)
* V případě spuštění více procesů než zvládne by nemohl plnit garantované limity

Středně dobý

* Využívá možnost ukládání na disk
* Taktický plánovač
* V případě kdy si začne nějaký z běžících procesů žádat větší paměťový prostor a začne provádět složitější výpočty, tak reaguje midterm
* Spolupracuje se správou hlavní paměti
* Vybírá, který proces je možné dočasně odložit přímo na disk (swapovací oddíl) a tím uvolní v hlavní paměti místo, pro náročnější proces
* Opětovně i vybírá, který proces z těch odložených procesů vrátí do RAM
* „každý“ OS má své způsoby jak předejít přetížení či zahlcení, bohužel real time OS ne
* Vznikají 2 nové stavy procesu

1. Odložený, čekající – čeká na událost a obnovuje se do WAIT
2. Odložený, připravený – obnovuje se do Ready
   * Připravený na běh, využití procesoru

Preemptivní vs nepreemptivní plánování

* Scheduler se rozhoduje, kterému procesu přidělí procesor na základě

1. Ukončení procesu
2. Změně stavu procesu z RAM do READY
3. Změně stavu procesu z RAM do WAIT (blokovaný)
4. Změně stavu procesu z WAIT do READY

* a)b)d) pro preemptivní
* a)c)d) pro nepreemptivní

Preemptivní

* OS má plnou kontrolu a může kdykoliv odebrat procesu procesor
* Spadá to do kategorie plánování s předbíháním
* Spadají zde všechny moderní OS
* Dochází k němu po uplynutí přidělené doby využití procesoru
* Výhoda
  + Neodchází k zamrznutí systému, protože je možno systém ukončit
* Nevýhoda
  + Složitější implementace samotného OS
  + Nutná HW podpora procesoru

Nepreemptivní

* OS nemá kontrolu nad převzetím od CPU, musí počkat až mu ji proces vrátí
* Bez předbíhání
* Použití u tzv. uzavřených systémech – všechny procesy jsou předem známy i jejich vlastnosti a jsou naprogramovány tak aby sami uvolňovali procesor pro další procesy
* Nevýhoda
  + Pokud je chybně napsaný program, který nevrátí proces procesoru tak dojde k zamrznutí
* Výhoda
  + Jednodušší implementace systému
* Př. OS: MS windows 95/98, starší MacOS

Cíle plánování

1. Využití CPU – maximalizace kontinuální činnosti procesoru
2. Propustnost – maximalizace ukončených procesů za jednotku času
3. Doba čekání – minimalizace doby čekání procesoru ve frontě ready
4. Doba obrátky – minimalizace doby potřebně pro provedené konkrétního procesu
5. Doba odpovědi – minimalizace doby, která uběhne od okamžiku zadání požadavku na spuštění procesu do jeho první realizace

* Podle váhy kritérií se rozlišují plánovací algoritmy (strategie plánování procesu nebo procesoru)

**Plánovací algoritmy**

FCFS

* First count first serve
* Proces, který požádá o procesor ho dostane
* Nepreemptivní
* Nevhodný pro OS se sdílením času (je zde více uživatelů)
* jeden proces může zablokovat ostatní na delší dobu
* delší proces zablokuje kratší – efekt kolony
* doba čekání na CPU je velká
* samostatně se skoro nepoužívá – v kombinaci s MFQS
* [výpočty](ODKAZY/fcfs.jpg)

SJF

* Nejkratší proces první
* nepreemptivní
* Je-li CPU volný, je přidělen procesu s nejkratším následujícím CPU cyklem
* Jestliže má více procesů stejnou dobu následujícího CPU cyklu , je mezi nimi rozhodnuto algoritmem FCFS
* Je-li kritériem kvality plánování průměrná doba čekání, je SJF optimální algoritmus
* [výpočet](ODKAZY/sjf.jpg)

SRTF

* Shortest remaining time first
* SJF, ale s preemptivním řízením
* [výpočet](ODKAZY/srtf.jpg)

Prioritní plánování

* Preemptivní, přiděluje na procesor procesy s největší prioritou
* Priorita běžícího procesu se porovnává s novým procesem, který žádá o procesor
* Nevýhodou může být stártnutí procesů, které čekají na přidělení procesoru (kvůli nízké prioritě)
* Pokud mají 2 procesy stejnou prioritu, je mezi nimi rozhodnuto FCFS
* [výpočet](ODKAZY/pp.jpg)

RR

* Round robin
* Preemptivní
* Je vhodný zejména pro OS sdílení času
* Je podobný algortimu FCFS obsahuje však možnost preemptivního plánování
* Je určené časové kvantum, proces příjde z fronty FIFO a pro uběhnutí čas. kvanta je buď dokončen nebo se zařadí na konec FIFO a čeká
* [výpočet](ODKAZY/rr.jpg)

MFQS

* Multi-level feedback queue scheduling
* [výpočet1](ODKAZY/mfqs1.jpg), [výpočet2](ODKAZY/mfqs2.jpg)