Operačné systémy

3. Procesy, thread-y a ich plánovanie

Ing. Martin Vojtko, PhD.

2024/2025

Procesy

Program a Proces

Program

je súbor obsahujúci množinu usporiadaných krokov, inštrukcií vedúcich k dosiahnutiu želaného cieľa, správania.

Proces

je vykonávaný program s pridelenými prostriedkami výpočtového systému.

Program a Proces

Program

je súbor obsahujúci množinu usporiadaných krokov, inštrukcií vedúcich k dosiahnutiu želaného cieľa, správania.

Proces

je vykonávaný program s pridelenými prostriedkami výpočtového systému.

Program a Proces

Program

je súbor obsahujúci množinu usporiadaných krokov, inštrukcií vedúcich k dosiahnutiu želaného cieľa, správania.

Proces

je vykonávaný program s:

- Vyhradeným adresným priestorom.
- Stavom (registre, PC, zásobník).
- Pridelenými I/O zariadeniami.
- Otvorenými súbormi.

Život bez procesov

- Vykonávanie jednotlivých úloh systému by prebiehal ako nekonečná slučka. Rad za radom by sa vykonávali podprogramy.
- Programátor by musel sám vždy brať do úvahy ostatné aplikácie. Musel by sa starať o dobrovolné prerušovanie činnosti.
- Prakticky nereálne riešiť pristupove opravnenia a bezpečnosť výpočtového systému.
- Nemožné bezpečne riešiť riadenie privilegovaného a neprivilegovaného režimu.
- Pre človeka nie je jednoduché riešiť problém paralelne. Návrh aplikácie ako poznáme by nebol možný.
- Procesy musia byť skompilovanou sučasťou OS.

Život s procesmi

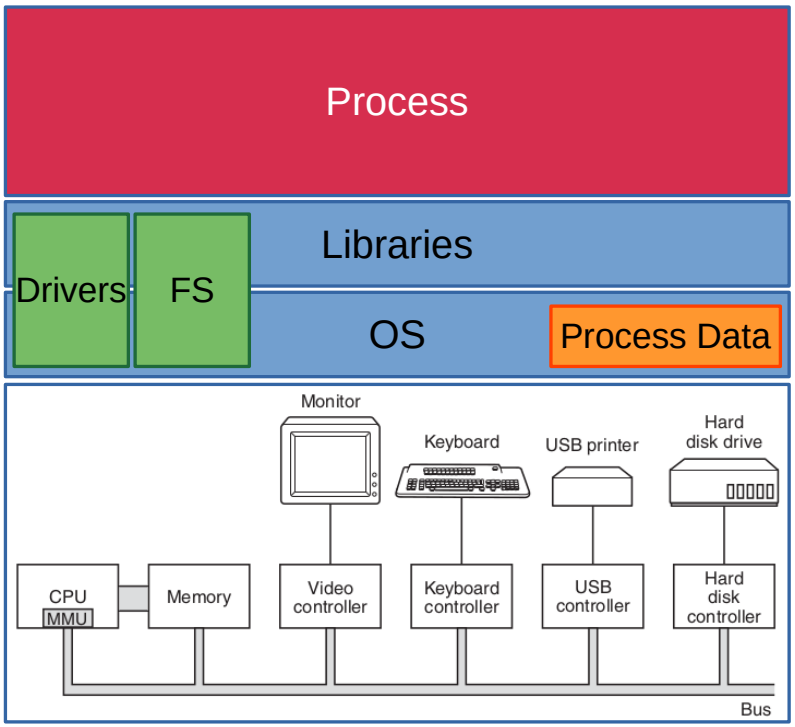
- Procesný model definuje proces ako sekvenčnú prácu a teda umožňuje človeku z komplexného problému spraviť množinu jednoduchších sekvenčných problémov.
- Jasné rozdelenie Privilegovaného a Neprivilegovaného režimu práce.
- V tomto pohľade proces vníma CPU akoby bol celý jeho. Preto sa programátor môže venovať návrhu svojej aplikácie.
- OS zaručuje, že procesy sa postupne striedajú na CPU v chránenom prostredí.
- Proces je model resp. abstrakciou vo výpočtovom systéme.

Process model

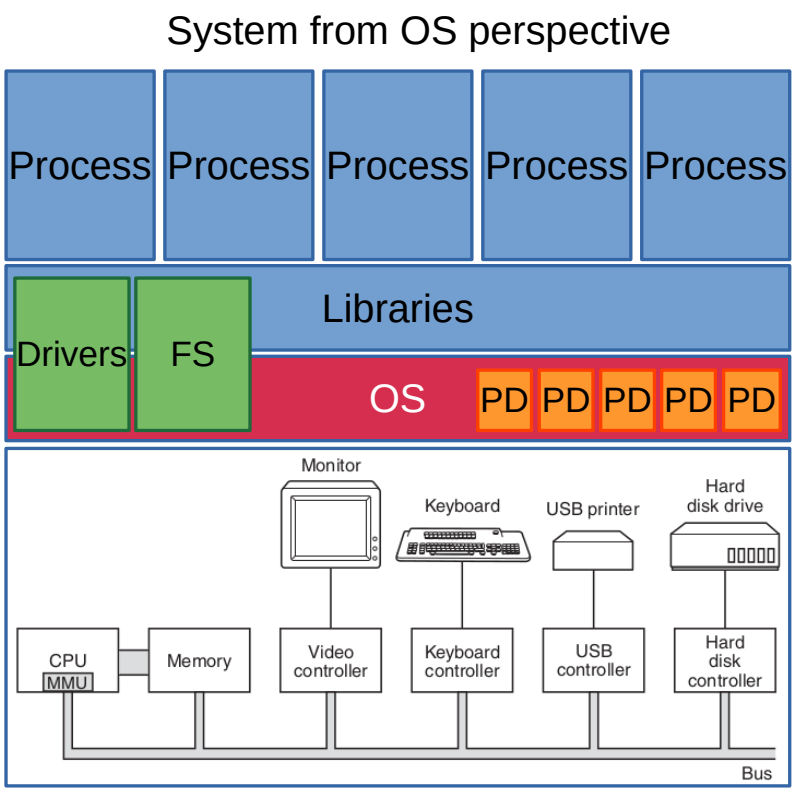
- Abstrakcia od riadenia prístupu ku zdrojom výpočtového systému.
- Proces je izolovaný na výpočtovom systéme (Neprivilegovaný režim).
- Proces a jeho prostredie je chránený OS.
- Proces sa nemusí deliť o prostriedky.
- Striedanie procesov na CPU riadi OS.
- Programátor sa nemusí zaoberať prerušeniami.
- Prideľovanie pamäte a prostriedkov riadi OS.
- Umožňuje **Multiprogramming**.

Process model

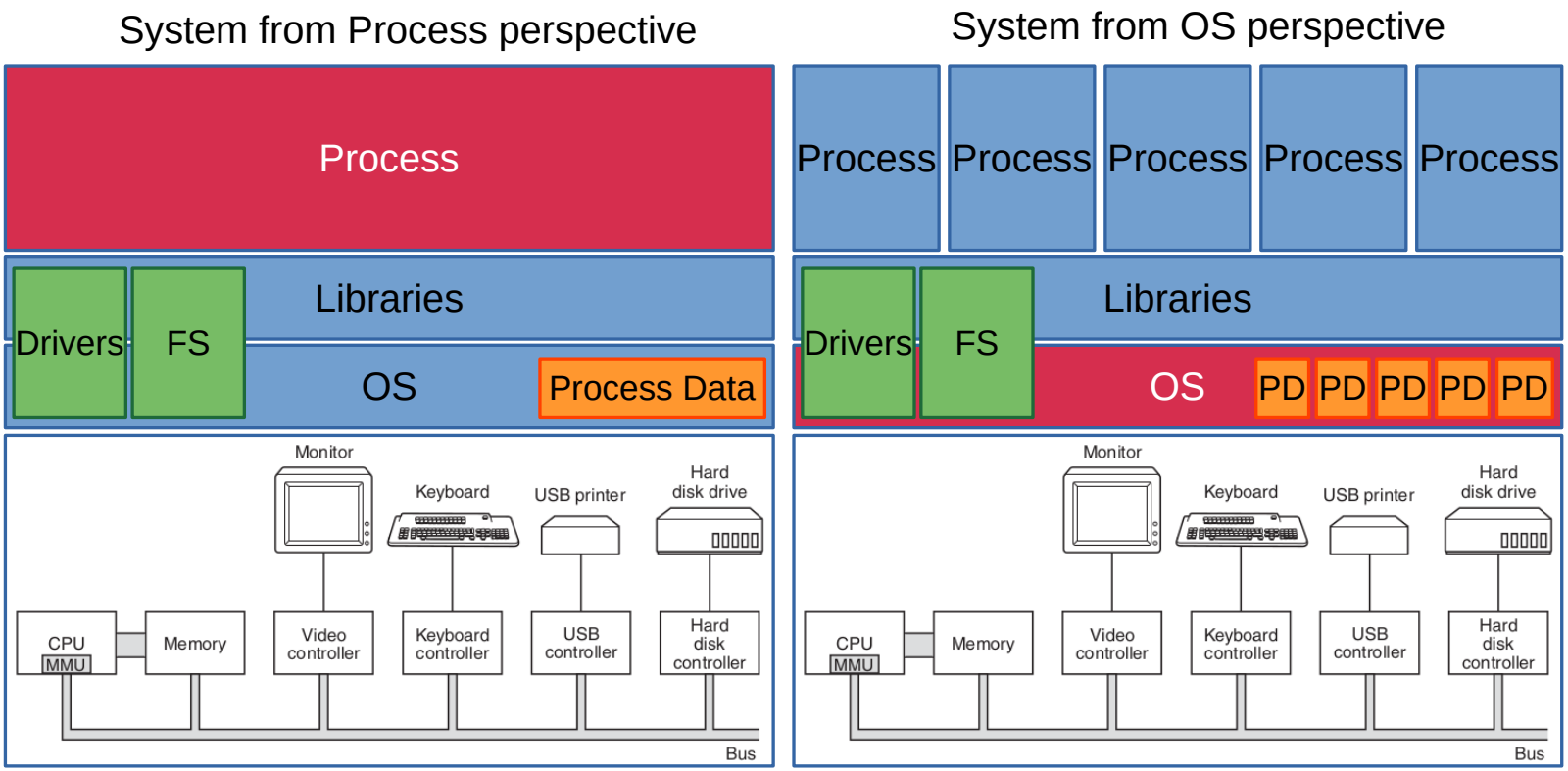
System from Process perspective



Process model



Process model



Process model - Paralelizmus

- V jednom okamihu je vykonávaný na jednom CPU práve jeden proces.
- Čas vykonania jedného procesu je niekoľko desiatok milisekúnd.
- V čase sa procesy na CPU striedajú.

Pseudo-paralelizmus

Časté striedanie procesov na jednom CPU budí dojem, že každý proces sa vykonáva súčasne. V takom prípade hovoríme o pseudo-paralelnom spracovaní.

Paralelizmus

Viac jadrové procesory umožňujú vykonávanie viacerých procesov súčasne. V takom prípade hovoríme o paralelnom spracovaní.

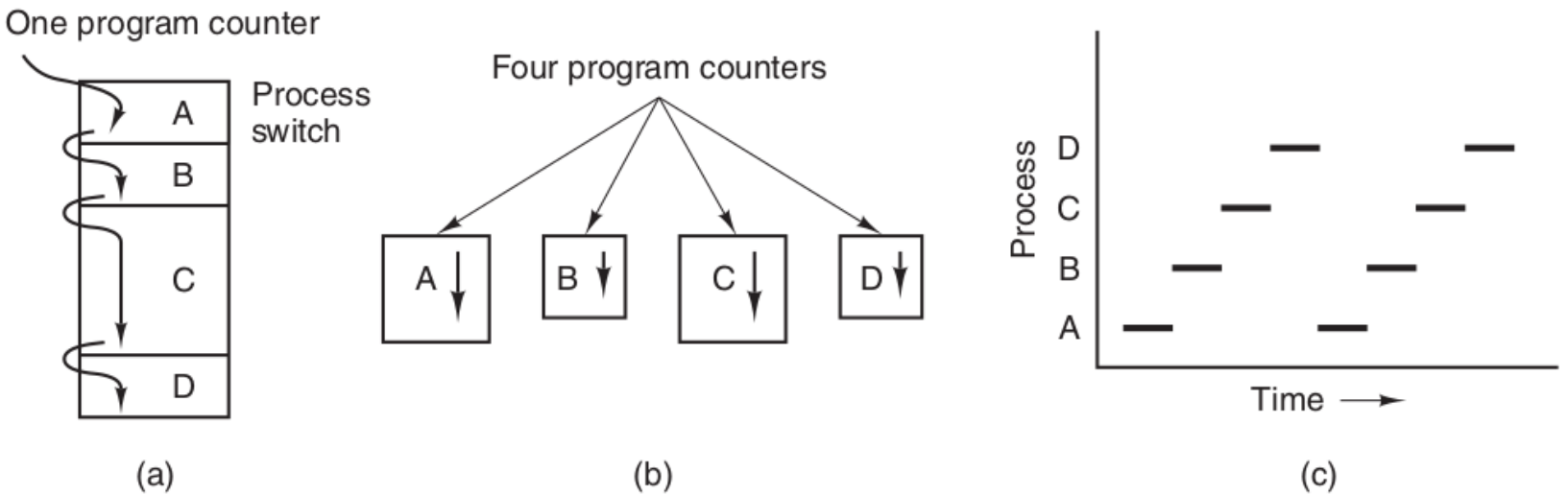
Process model - Multiprogramming

Multiprogramming

je udržiavanie viacerých procesov v hlavnej pamäti výpočtového systému.

- Procesy vykonávané na CPU často môžu čakať na odpoveď od I/O.
- Procesor v takom prípade nevykonáva žiadnu užitočnú prácu.
- Čakajúci proces môže OS preplánovať iným procesom.
- Ak by neboly procesy uložené v hlavnej pamäti OS by musel proces načítať z disku.
- Multiprogramming napomáha zvýšiť vyťaženie (utilizáciu) procesora.

Proces model - Multiprogramming

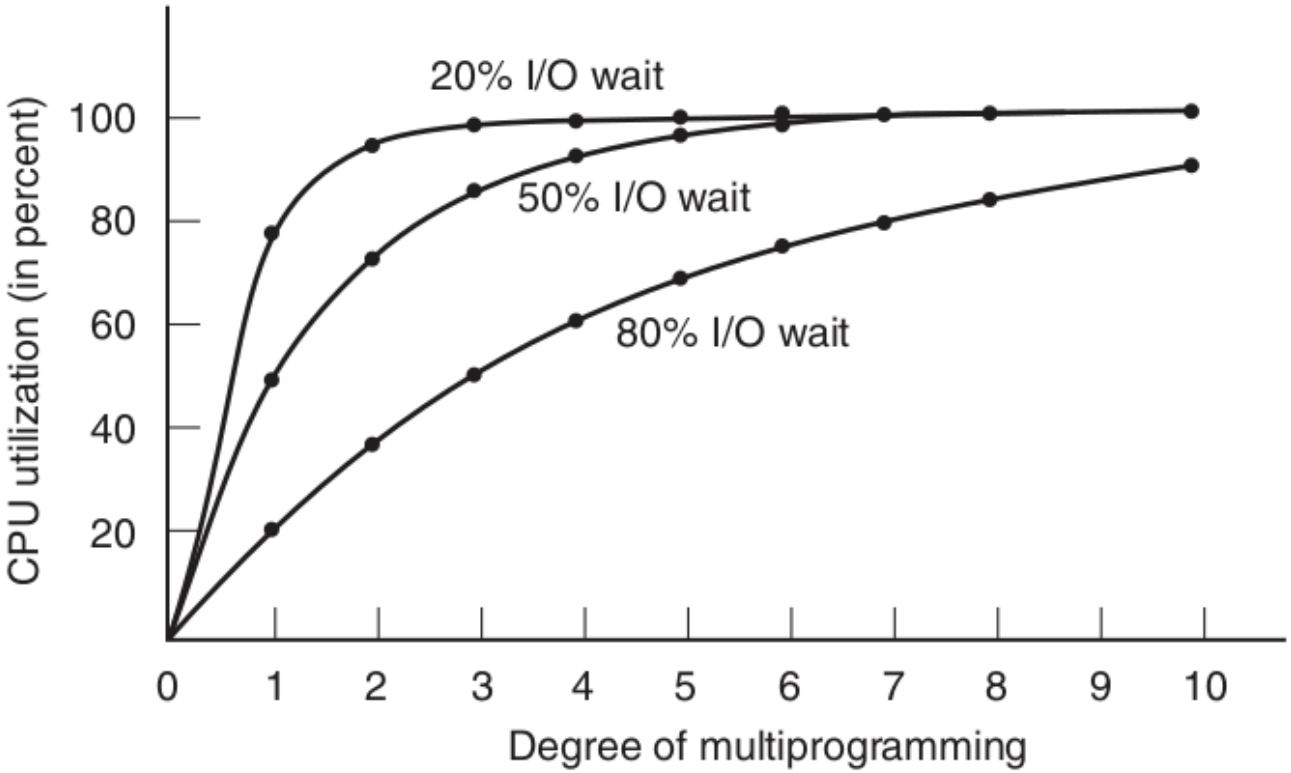


Proces model - Multiprogramming

utilizácia CPU

$$U = 1 - p^n$$

Kde p je pravdepodobnosť čakania na I/O a n je počet procesov.



Reprezentácia procesu v OS

Process Table

je tabuľka OS, kde sú uložené všetky vykonávané procesy.

Process Control Block (PCB)

je jeden záznam v Process Table, v ktorom OS udržuje všetky potrebné informácie o procese.

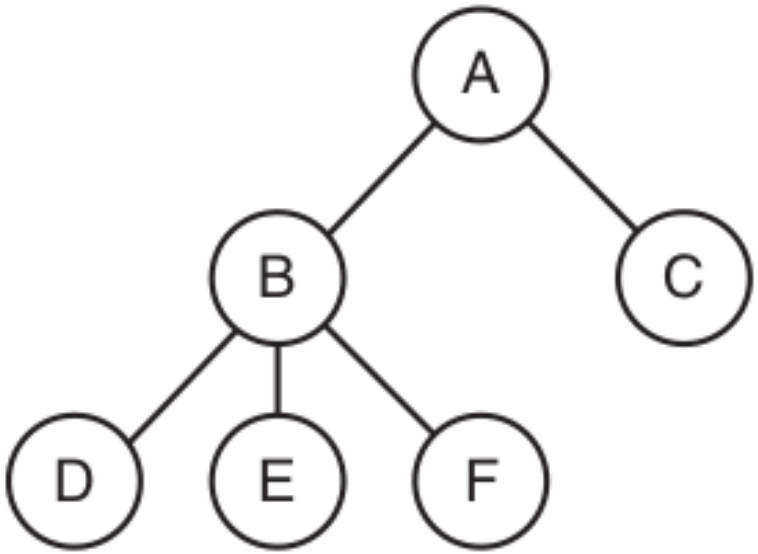
Process Context

je aktuálny stav procesu, ktorý sa odkladá do PCB v prípade preplánovania procesu.

Reprezentácia procesu v OS

Process Tree

V Unix OS procesy vznikajú inými procesmi. Preto vzniká vzťah rodič dieťa, kde rodič si pamätá identifikátory svojich detí. Takýmto spôsobom sú procesy udržované aj v strome procesov. Windows nemá takýto koncept.



Reprezentácia procesu v OS - PCB

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Reprezentácia procesu v OS - PCB

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Reprezentácia procesu v OS - PCB

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Reprezentácia procesu v OS - PCB

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

Životný cyklus procesu

Vznik procesu

- Pri spustení OS:
  - Bežne používaná metóda vo vnorených systémoch.
  - Pri štarte OS vždy vznikne aspoň jeden proces.
    Unix tento proces volá init process
- Systémovým volaním:
  - Prostredníctvom iného procesu.
  - Požiadavkou používateľa. (napríklad cez shell)
- Batch Jobom (napríklad plánovanou uzávierkou prevádzky sa vytvorí Batch úloha na spracovanie tržby.)

Vznik procesu - systémové volanie

fork() a exec() - Unix

V Unixe proces vzniká systémovým volaním fork(). fork() vytvorí presnú kópiu procesu so všetkými pridelenými prostriedkami. Pamäť programu nie je nutné kopírovať a rodič s dieťaťom ju zdieľajú. Ak chceme spustiť iný program Unix používa volanie exec(). exec() nahradí aktuálny program procesu novým programom.

spawn() - Windows

Windows používa volanie CreateProcess(). Toto volanie na základe parametrov vytvorí úplne samostatný proces.

Vznik procesu - systémové volanie

Copy-on-write

Napriek tomu, že počas volania fork() nedochádza ku kopírovaniu programu stále je nutné kopírovať dáta procesu. Táto operácia je tiež nákladná a preto v Unix-e sa používa metóda copy-on-write. V princípe sa pamäť dát nekopíruje kým do nej detský proces nechce zapísať.

Zánik procesu

- Dobrovoľný zánik:
  - Štandardné ukončenie programu.
  - Štandardné ukončenie s chybou.
- Nedobrovoľný zánik:
  - Ukončenie s fatálnou chybou. (Segmentation fault, division by zero)
  - Ukončenie iným procesom (Kill)

Zánik procesu - systémové volanie

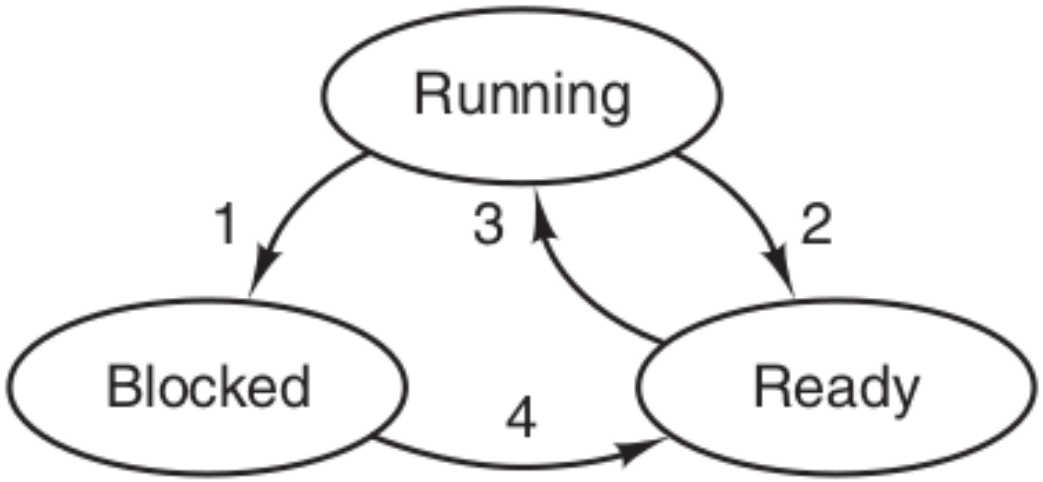
exit()

Pre dobrovoľné ukončenie procesu sa používa systémové volanie exit() (ExitProces() vo windows). Volanie sa automaticky vykoná po výstupe z funkcie main. Programátor však toto volanie môže vložiť aj do iných funkcií programu. Najčastejšie sa vkladá na miesta kde došlo k chybe.

Vykonávanie procesu

- Proces po vytvorení prechádza do fázy vykonávania programu. Počas vykonávania môže dosiahnuť nasledovné stavy:
  - **Running** - proces je vykonávaný na CPU
  - **Ready** - proces je pripravený na vykonávanie. Proces je plánovacím algoritmom vložený do poradovníka na vykonávanie.
  - **Blocked** - proces je blokovaný čakaním na udalosť. Proces nie je v poradí na vykonávanie.

Vykonávanie procesu



1. Proces je blokovaný lebo čaká na udalosť.
2. Proces je preplánovaný OS.
3. Proces je naplánovaný OS.
4. Proces je zobudený po príchode udalosti.

Threads (Vlákna)

Thread (Vlákno)

Proces

je vykonávaný program s
pridelenými prostriedkami výpočtového systému.

Thread

je vykonávaný program s vlastným stavom (registre, PC, zásobník).

Thread (Vlákno)

Proces

je vykonávaný program s
pridelenými prostriedkami výpočtového systému.

Thread

je vykonávaný program s vlastným stavom (registre, PC, zásobník).

Thread (Vlákno)

Proces

je jedno alebo viacej vykonávaných thread-ov programu s pridelenými prostriedkami výpočtového systému.

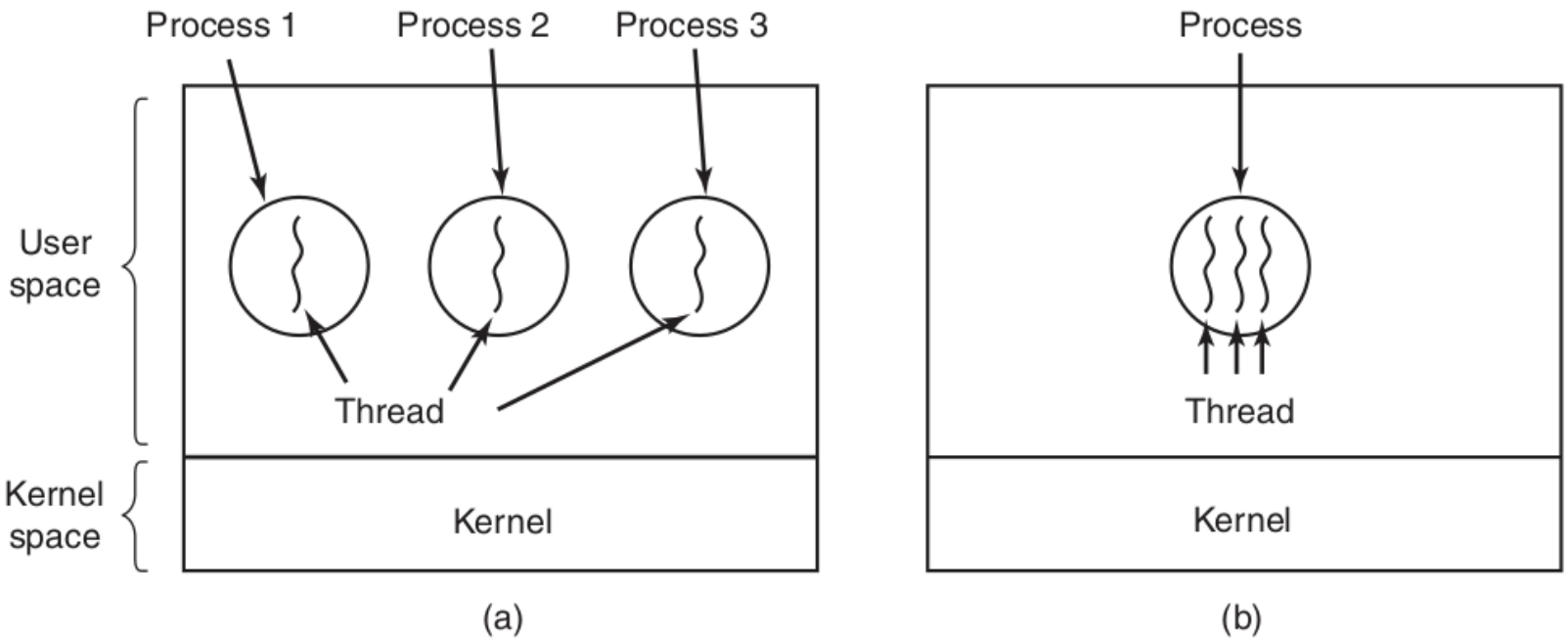
Thread

je vykonávaný program s vlastným stavom (registre, PC, zásobník).

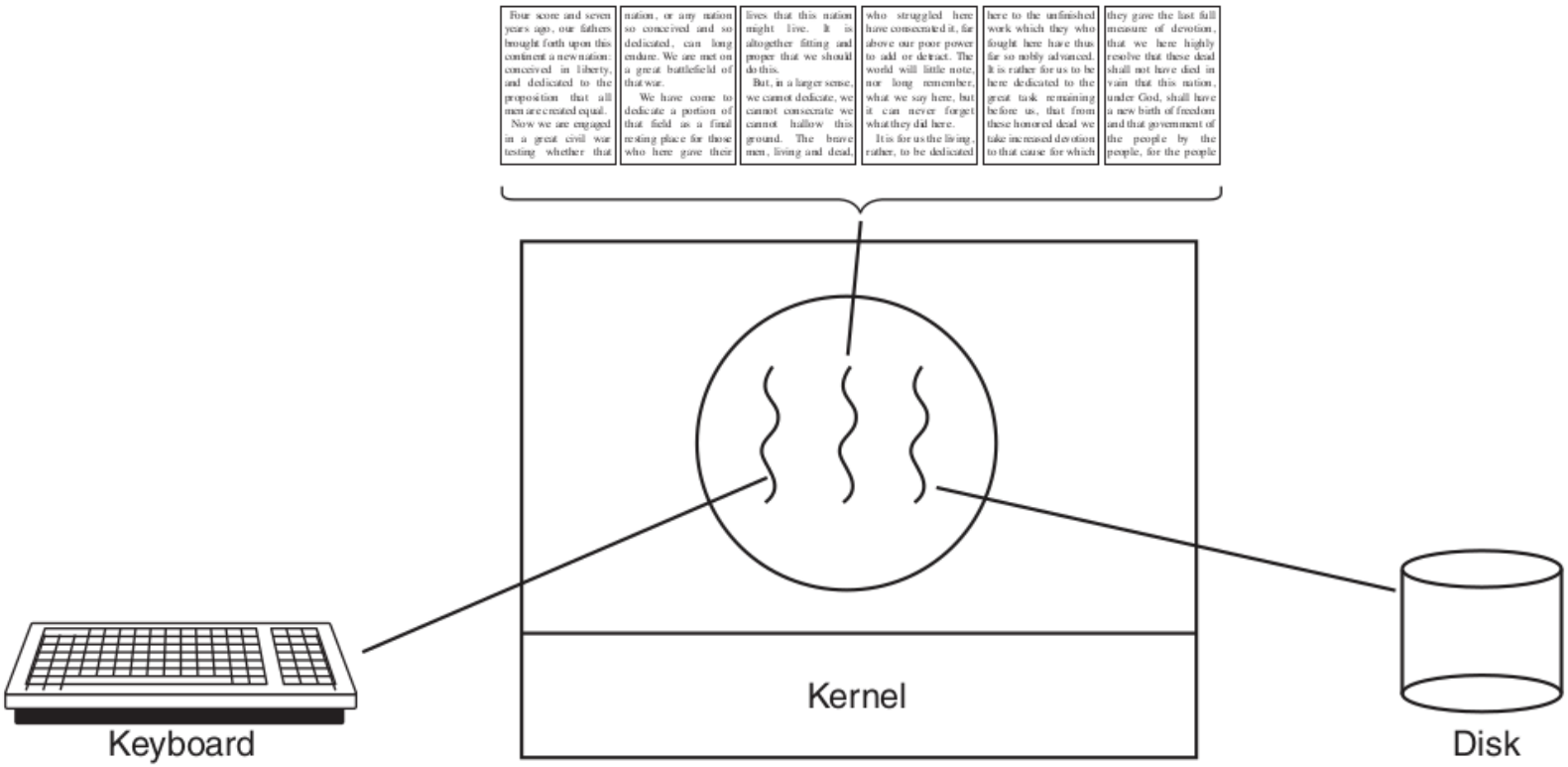
Thread (Vlákno)

- Vlákno je možné vidieť ako proces vykonávaný v kontexte procesu.
- Proces môže v tomto koncepte udržiavať viacero vláken.
  - V jednom momente len jedno vlákno je vykonávané na CPU.
  - Ostatné vlákna môžu byť pripravené alebo blokované.
- Vlákna sú v podstate veľmi ľahké procesy, ktoré zdieľajú v rámci jedného procesu program, dáta, súbory, signály...
- Výhody použitia vláken:
  - Rozdelenie programu do ešte menších sekvenčných úloh.
  - Vlákna navzájom môžu komunikovať a synchronizovať sa prostredníctvom spoločnej pamäti dát. U procesov to je tiež možné ale omnoho nákladnejšie.
  - Preplánovanie vlákna iným vláknom v rámci procesu je lacnejšie nakoľko nie je nutné pri zmene kontextu vymeniť pamäť programu resp. dát.
  - CPU s multithreading umožňuje jednoduchší prístup ku vláknu.

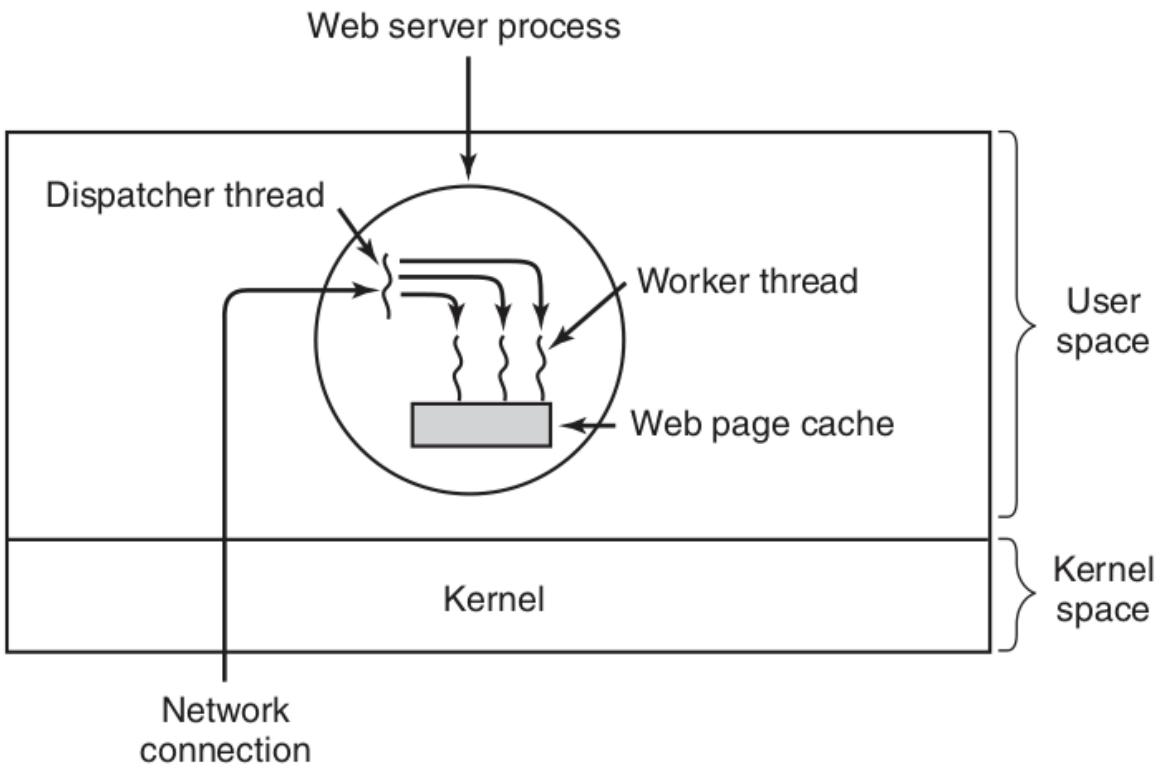
Thread vs Proces



Thread - príklad word



Thread - príklad web server



Thread - komplikácie

- Správanie fork() - keď vytváram nový proces vytvorím ho aj so všetkými existujúcimi vláknami?
- Zdieľanie prostriedkov - ak viacej vlákien pracuje so súborom. Čo sa deje ak jedno vlákno súbor zavrie?
- Alokácia pamäte - čo ak viacero vlákien súčasne požiada systém o pridelenie pamäte?
- Práca so stack-om - ak vlákno potrebuje väčší zásobník ako vie OS určiť kto ho potrebuje?
- Globálne premenné - nemajú chránený prístup.
- Signalizácia - kto reaguje na príchod signálu od OS? Všetci alebo aktuálne vlákno?
- Knižnice - knižnice musia byť implementované tak aby podporovali threading.

Klasický thread model

- Proces je kontajner udržiavajúci prostriedky.
- Thread má program, registre, PC, stack.
- Thread je entita využívajúca prostriedky procesu vykonávaná na CPU.
- V rámci procesu môže existovať niekoľko thread-ov.
- Thread nadobúda stavy running, ready, blocked.

Prečo má thread vlastný zásobník?

Thread je vykonávaný program a teda si musí udržovať vlastnú históriu vykonávania funkcií.

Klasický thread model - TCB

Thread Controll Block (TCB)

je štruktúra, v ktorej OS udržuje všetky potrebné informácie o vlákne.

Per-process items	**Per-thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

POSIX thread model

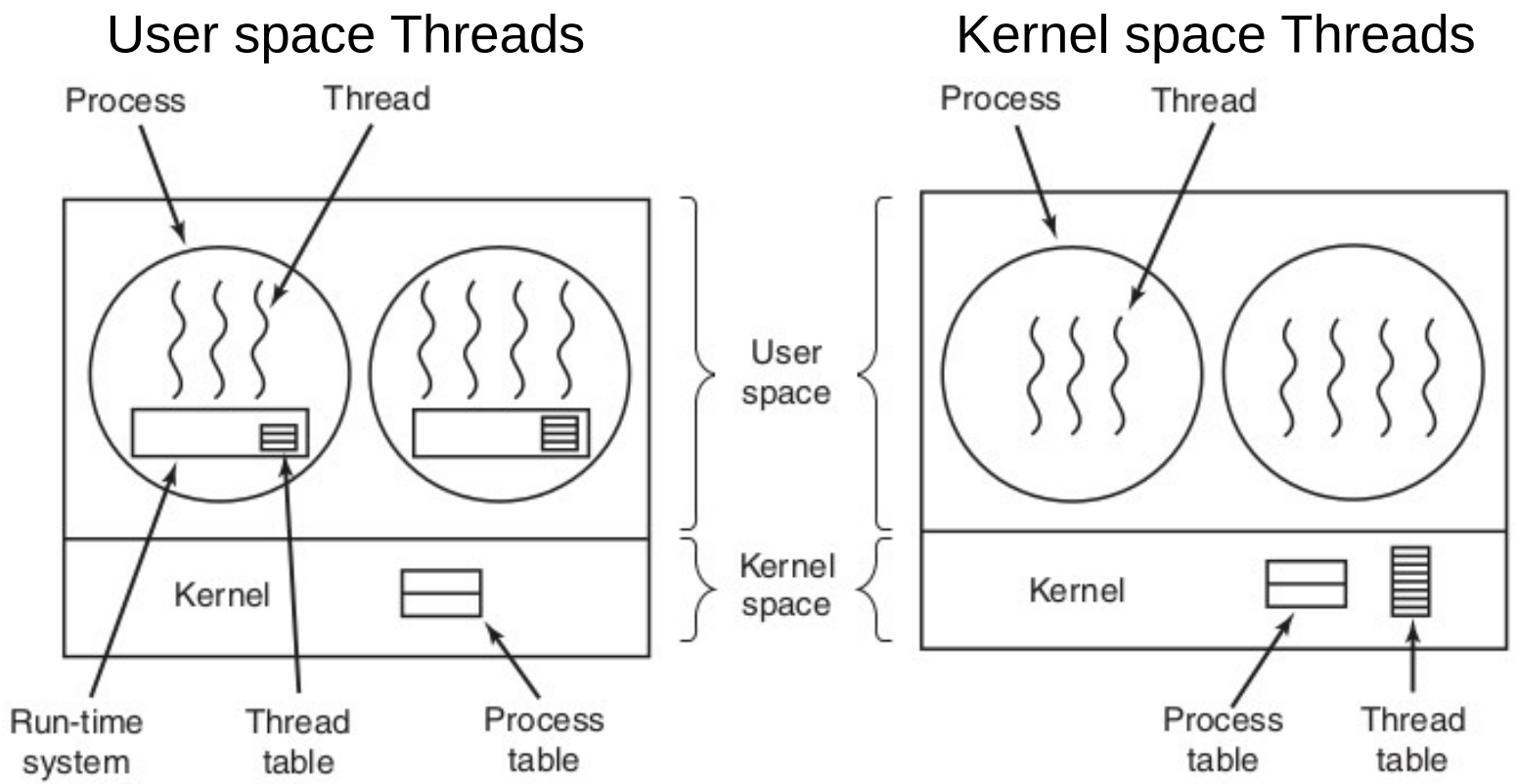
- Implementácia klasického modelu.
- POSIX vlákna štandardizujú a rozširujú klasický model vláken.
- POSIX definuje cez 60 volaní súvisiacich s vláknami.

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

POSIX Thread - hello world

```c
#include <pthread.h> //link with lib pthread -> gcc main.c -lpthread
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void * print_hello_world(void * tid)
{
    sleep(2);
    printf("Hello World! Greetings from thread %ld\n", (long int) tid);
}

int main(int argc, char * argv[])
{
    pthread_t threads[10];

    for (long int i=0; i < 10; i++) {
        printf("main: creating thread %ld\n", i);
        pthread_create(&threads[i], NULL, print_hello_world, (void *) i);
    }
    sleep(3);
}

```

Reprezentácia vlákien v OS



User space threads

- Knižnica - manažment vláken bez systémových volaní.
- Tabuľka vlákien je udržiavaná v kontexte procesu.
- Každý proces môže implementovať vlastný plánovací algoritmus.
- Vlákno ktoré čaká na prostriedky musí programovo používať yield().
- Vlákno blokované na systémovom volaní blokuje celý proces.
- Kernel OS:
  - nie je volaný pri práci s vláknami - rýchle operácie nad vláknami.
  - nemusí podporovať vlákna.
  - nedokáže prepínať vlákna.

Kernel space threads

- Kernel implementuje podporu vláken.
- Operácie nad vláknami obsahujú systémové volanie.
- Kernel sprostredkuje plánovanie vláken.
- Kernel umožňuje prepnutie vlákna bez potreby volania yield().
- Vlákno blokované na systémovom volaní neblokuje celý proces.
- Vytvorenie a ukončenie vlákna je rádovo pomalšie. (recyklácia?)

Plánovanie procesov

Plánovanie procesov

Scheduler (Plánovač)

je časť OS, ktorého úlohou je usporiadanie pripravených procesov, tak aby splnil definované kritéria plánovania na vybranom výpočtovom systéme.

Plánovací algoritmus

je algoritmus podľa ktorého sú pripravené procesy usporiadané.

Plánovanie procesov

Kedy OS potrebuje plánovať?

- Vykonávaný proces skončil.
- Vykonávaný proces vytvoril nový proces.
- Vykonávaný proces je blokovaný systémovým volaním.
- HW prerušil CPU.
  - I/O zdrojom prerušenia
  - Periodický časovač zdrojom prerušenia.

- To, že došlo k udalosti kedy je možné plánovať nemusí vyústiť k prepnutiu aktuálne vykonávaného procesu.

Klasifikácia plánovacích algoritmov podľa reakcie na HW prerušenie

Preemptive

plánovanie umožňuje prerušenie vykonávanej úlohy plánovačom.

Non-preemptive

plánovanie dovoľuje nahradiť vykonávaný proces len ak sa on dobrovoľne rozhodne alebo ak je blokovaný systémovým volaním.

Klasifikácia plánovacích algoritmov podľa typu OS

Batch

Batch OS bežne nemá aktívnych používateľov, a preto nie je potrebná preempcia. Ak je použité preemptívne plánovanie je nastavené na dlhé časové úseky. Dôležitými kritériami plánovania sú throughput (priepustnosť) a turnaround time (doba obratu) úlohy.

Interactive

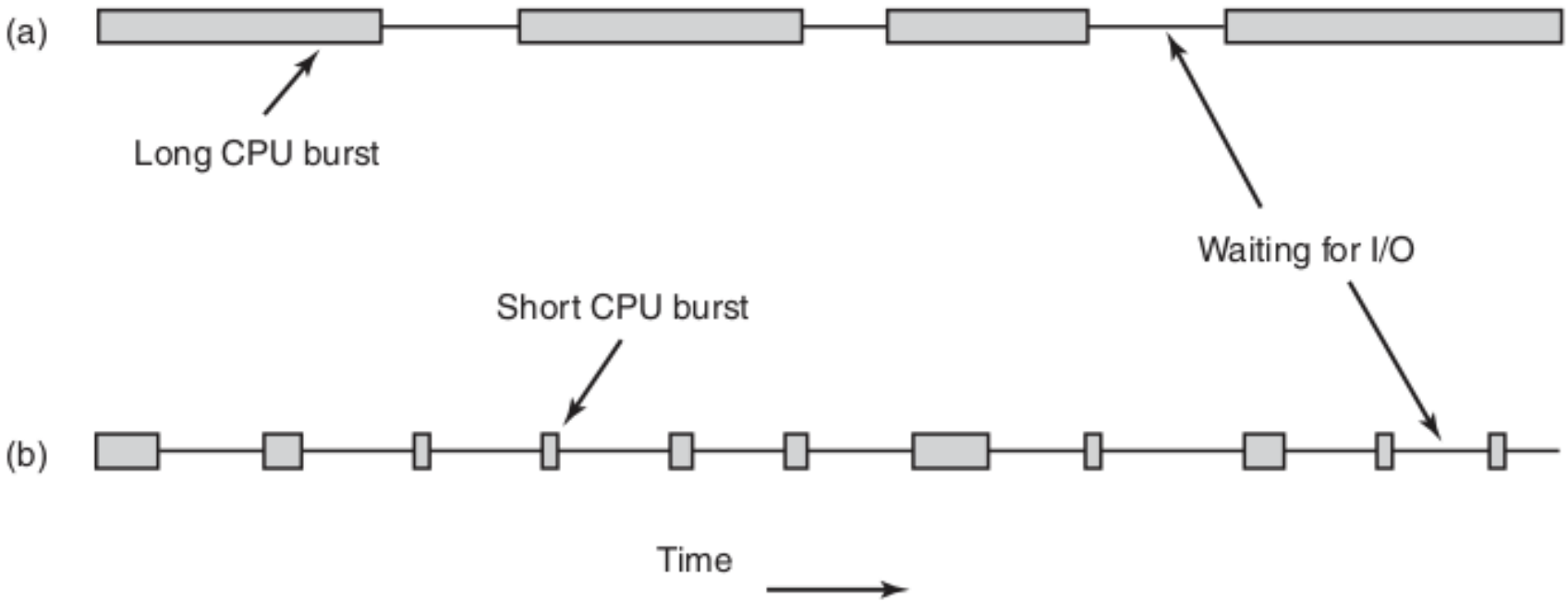
PC OS prípadne server OS kladie dôraz na rýchlu odozvu aplikácií preto preempcia je nutnosťou.

Real-time

RT OS kladú dôraz na dodržanie doby odozvy. Procesy sú často špecifické a čiastočne riadia plánovanie sami.

Klasifikácia procesov podľa väzby

- CPU-bound (viazané na CPU) - výpočtovo intenzívne.
- I/O-bound (viazané na I/O) - I/O intenzívne.



Kritériá plánovania

- Spoločné
  - Férovosť - každý proces by sa mal dostať k CPU.
  - Vynucovanie politiky - politika plánovania je dodržiavaná vždy.
  - Vyváženosť - Celý systém je rovnako zaneprázdnený.
- Batch systémy
  - Throughput - maximálny počet úloh spracovaných za čas.
  - Turnaround time - doba vykonania úlohy je čo najkratšia.
  - CPU utilizácia - CPU je vyťažené neustále.
- Interaktívne systémy
  - Doba odozvy - Požiadavky spracované čo najrýchlejšie.
  - Proporčnosť - Spĺňa očakávania používateľa.
- RT systémy
  - Dodržanie doby odozvy - Zamedzenie strate dát.
  - Predvídateľnosť - Plánovač sa správa rovnako za rovnakej situácie.

Nepreemptívne algoritmy

First Come First Served (FCFS, FIFO)

jednoduchý, férový algoritmu. Prepne na proces, ktorý prišiel ako prvý. Proces je vykonávaný kým nie je blokovaný. Potom je zaradený na koniec radu. Nevýhodou je, že CPU bound úlohy sú zvýhodnené oproti I/O bound.

Shortest Job First (SJF)

potrebuje presný odhad vykonávania úlohy nakoľko tá s najkratším časom je usporiadaná na začiatok radu úloh. Skracuje Turnaround time. Optimálny iba ak všetky úlohy sú spustené simultánne. Pri dostatočnom prúde krátkych procesov dlhé prakticky nedostávajú CPU.

Preemptívne algoritmy

Shortest Remaining Time

preemptívna obdoba SJF. Úloha, ktorá má najkratší zostávajúci čas do ukončenia má prednosť. Skracuje Turnaround time. Najoptimálnejší je pre systémy, kde všetky procesy sa spustia pri štarte systému. Nové krátke úlohy majú lepšiu šancu dostať sa k CPU, čo zvyšuje ich odozvu.

Round-robin

jeden z najstarších, najjednoduchších, najférovejších a najviac používaných algoritmov. Preemptívna podoba FCFS. Dôležitým prvkom je pridelenie časového quanta každému z procesov. Proces je vykonávaný kým nie je blokovaný alebo kým si neminie pridelené quantum. Potom je zaradený na koniec radu. Zásadným problémom algoritmu je správne stanovenie quanta. Malé quantum lepšia odozva ale vyššia réžia plánovania. Veľké quantum zlá odozva ale nízka réžia plánovania.

Preemptívne algoritmy

Prioritné plánovanie

jednotlivé procesy majú rôznu dôležitosť, ktorá je zohľadnená prioritou. Čím vyššia priorita tým je úloha vyššie v rade. Nekonečne dlho vykonávaná úloha s vysokou prioritou by vyhladovala ostatné úlohy a preto je priorita úlohy s časom postupne znižovaná alebo má úloha pridelené quantum. Priorita môže byť pridelená staticky alebo dynamicky. Prioritu si môžu vzájomne kooperujúce úlohy odovzdávať. Je bežné úlohy s rovnakou prioritou plánovať pomocou Round-robin.

Lotériové plánovanie

procesy dostávajú výherné lístky. Plánovač náhodne losuje nový proces vykonávaný na procesore. Prioritné procesy môžu dostať viac výherných losov. Procesy si môžu výherné losy navzájom darovať. V dlhodobom horizonte budú úlohy vykonávané proporčne k počtu výherných lístkov.

Preemptívne algoritmy

Garantované plánovanie

algoritmus ktorého úlohou je zabezpečiť aby všetky procesy v systéme dostali rovnaký prídel času vykonávania na CPU. Každý proces dostane $1/n$ výkonu CPU.

Plánovanie s primeraným podielom

obdoba Garantovaného plánovania s dôrazom na rovnomerné rozloženie času CPU pre používateľov alebo skupiny procesov. Napríklad ak jeden používateľ má 10 procesov a druhý má 3 tak s garantovaným plánovaním by aktívnejší používateľ dostal viac CPU. Aplikovaním primeraného podielu by obaja používatelia mali dostať rovnaký podiel CPU.

Plánovanie vláken

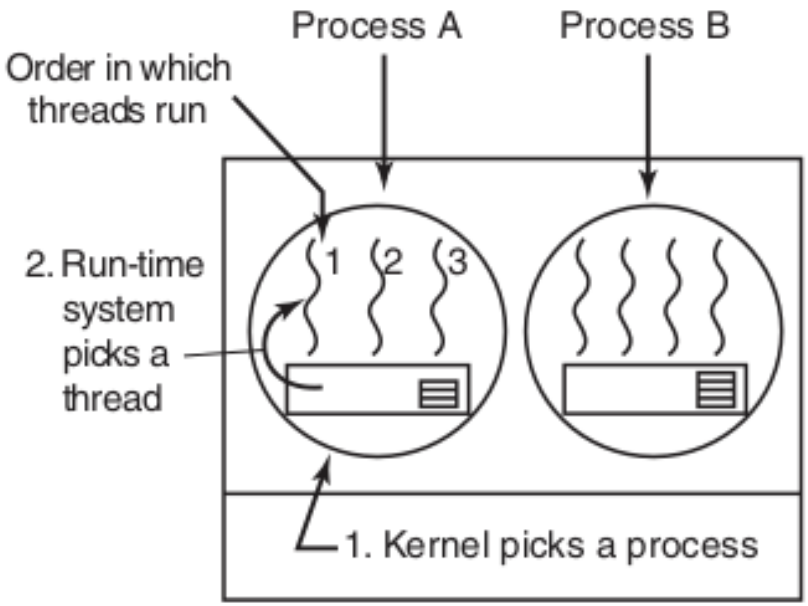
User Threads

Nakoľko OS nemá vedomosť o vláknach plánovač ich nedokáže zohľadniť. Ak sa vlákna často striedajú pri činnosti tak systém značne trpí.

Kernel Threads

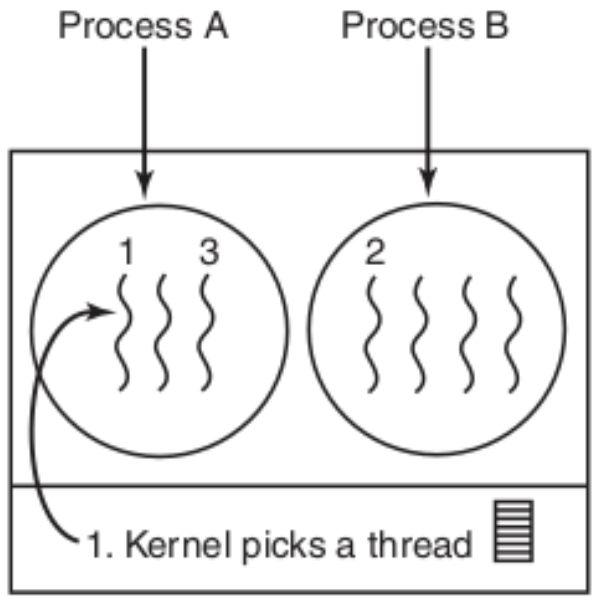
Plánovač môže zohľadniť prítomnosť vláken v procese. Ak vlákna sú často blokované (Častejšie ako preeemptívne quantum), tak môže dochádzať k častému a nákladnému prepínaniu úloh. Plánovač preto môže zohľadniť prepnutie vlákna v rámci toho istého procesu čím sa ušetrí odloženie programu procesu.

Plánovanie vláken



Possible: A1, A2, A3, A1, A2, A3
Not possible: A1, B1, A2, B2, A3, B3

(a)



Possible: A1, A2, A3, A1, A2, A3
Also possible: A1, B1, A2, B2, A3, B3

(b)

Zhrnutie

Zhrnutie

- Proces
  - Predstavuje abstrakciu, ktorá výrazne odbremeňuje programátora.
  - Komplexné úlohy rozdrobuje na menšie sekvenčné časti.
- Thread
  - Predstavuje abstrakciu, ktorá prináša mnohé výhody z hľadiska výkonu programu.
  - Komplexné programy rozdrobuje na menšie sekvenčné časti.
- Plánovanie
  - Súčasť OS, ktorá odbremeňuje programátora od mnohých aspektov riadenia výberu vykonávaných úloh.
  - v súčasnosti neexistuje univerzálny plánovací algoritmus.
  - výber plánovača v OS preto musí zohľadňovať systémovo špecifické kritéria plánovania.

Čo robiť do ďalšej prednášky

- Bash Bash Bash!!!!
- Prečítať kapitoly 2.3, 2.5 a 2.7 z Tanenbauma