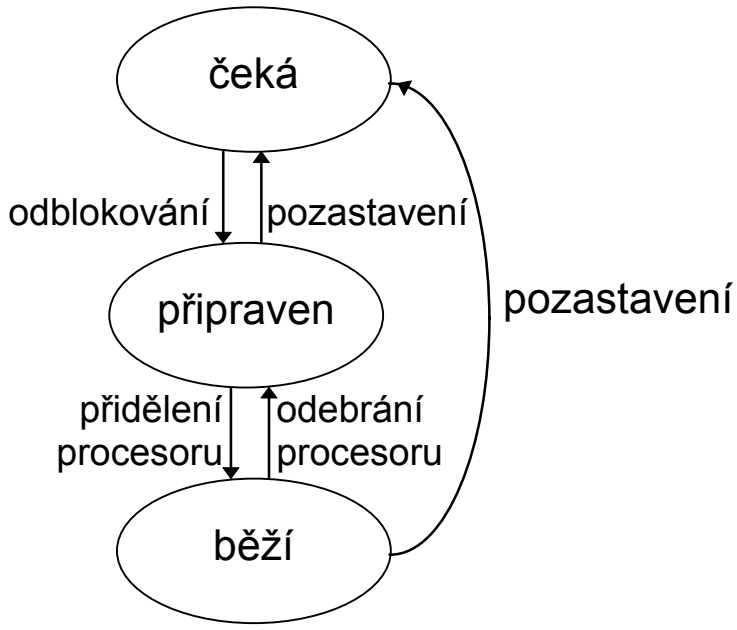
9. Plánování a přidělování procesoru

- plánování (scheduler) - volba strategie a řazení procesů
  - sdílený plánovač
  - samostatný plánovač
- přidělování (dispatcher) - přepínaní kontextu na základě naplánování

Cíle plánování:

- Minimalizace doby odezvy
- Efektivní využití prostředků
- Spravedlivé dělení času procesoru
- Doba zpracování
- Průchodnost (počet úloh/čas)

Stavový diagram procesů



fronta připravených procesů - seznam procesů, které mohou běžet
plánování - organizace fronty připravených procesů
přidělování procesoru - přidělení procesoru prvnímu procesu z fronty připravených

Přidělování procesoru

= přepínání kontextu podle plánování
Procesorově a systémově závislé, příklad jednoprocesorové jádro FreeBSD 4.x

Struktury:

```
struct  proc {          /* popis procesu */
    TAILQ_ENTRY(proc) p_procq; /* run/sleep q */
    LIST_ENTRY(proc) p_list;   /* all procs */
/* substructures: */
    struct  pcred *p_cred; /* owner's identity */
    struct  filedesc *p_fd;    * open files */
    struct  pstats *p_stats;   /* statistics */
    struct  plimit *p_limit;   /* limits */
    struct  procsig *p_procsig;
    int     p_flag;      /* P_* flags */
    char    p_stat;      /* S* process status */
    pid_t   p_pid;       /* Process identifier */
    ...
    struct  pargs *p_args;
    struct  user *p_addr;    /* u-area */
    ...
};
/* data, která lze odložit s procesem */
struct  user {
    struct  pcb u_pcb;
    struct  sigacts u_sigacts;
    struct  pstats u_stats;
    ...     /* kernel stack, TSS, etc. */
};
/* registry, stav */
struct pcb {
    int     pcb_cr3;
    int     pcb_edi,pcb_esi,pcb_ebp,pcb_esp;
    int     pcb_ebx,pcb_eip;
    ...
};
```

```
/* uložení stavu procesu, spuštění jiného */
cpu_switch:
    movl    _curproc,%ecx     /* proc pointer */
    testl   %ecx,%ecx
    je      sw1               /* no process */
    movl    P_ADDR(%ecx),%edx /* addr of user */
    movl    (%esp),%eax  /* hardware registers */
    movl    %eax,PCB_EIP(%edx) /* pcb in user */
    movl    %ebx,PCB_EBX(%edx)
    movl    %esp,PCB_ESP(%edx)
    movl    %ebp,PCB_EBP(%edx)
    movl    %esi,PCB_ESI(%edx)
    movl    %edi,PCB_EDI(%edx)
    movl    %gs,PCB_GS(%edx)
/* have we used fp, and need a save fp regs? */
    call    _npxsave /* do it in a big C func */
    movl    $0,_curproc    /* out of process */
/* save is done, now choose a new proc or idle */
sw1: cli
    call    _chooseproc    /* RT,NORMAL,IDLE */
    testl   %eax,%eax
    je      _idle          /* if no proc, idle */
    movl    %eax,%ecx      /* PCB pointer */
    movl    P_ADDR(%ecx),%edx /* addr of user */
/* switch address space? */
    movl    %cr3,%ebx      /* base of PT */
    cmpl    PCB_CR3(%edx),%ebx
    je      4f
    movl    PCB_CR3(%edx),%ebx
    movl    %ebx,%cr3      /* load new PT,flush*/
4:  /* restore context */
    movl    PCB_EBX(%edx),%ebx
    movl    PCB_ESP(%edx),%esp
    movl    PCB_EBP(%edx),%ebp
    movl    PCB_ESI(%edx),%esi
    movl    PCB_EDI(%edx),%edi
    movl    PCB_EIP(%edx),%eax
    movl    %eax,(%esp)
```

```
    movl    %edx, _curpcb
    movl    %ecx, _curproc /* into next proc */
    sti
    ret
```

Použití:
Pozastavení procesu a čekání ve frontě na událost (zjednodušeno, bez signálů)
ident = adresa na co se čeká (obvykle adresa datové struktury)
wmesg = označení čekání (pro ps)

```
int tsleep(void *ident, int priority,
            char *wmesg, int timo)
{
    struct proc *p = curproc;

    s = splhigh();      /* block interrupts */
    p->p_wchan = ident;    /* addr of wait */
    p->p_wmesg = wmesg;    /* comment */
    p->p_slptime = 0;
    p->p_priority = priority & PRIMASK;
    TAILQ_INSERT_TAIL(&slpque[LOOKUP(ident)],
        p, p_procq);
    if (timo)
        thandle = timeout(endtsleep, p, timo);
    p->p_stat = SSLEEP;
    cpu_switch();
    curpriority = p->p_usrpri;
    splx(s);
    if (timo)
        untimeout(endtsleep, (void *)p, thandle);
    return(0);
}
```

Fronty čekajících procesů jsou virtuální, nejsou součástí objektů, na které procesy čekají (byla by to velká režie). Všechny čekající

procesy jsou zařazeny do jedné hashovací tabulky, klíčem (funkce LOOKUP) je adresa objektu, na který se čeká.

Pokračování (odblokování) všech procesů čekajících v zadané frontě *ident*:

```
void wakeup(void *ident)
{
    struct slpquehead *qp;
    struct proc *p;
    int s;

    s = splhigh();
    qp = &slpque[LOOKUP(ident)];
restart:
    TAILQ_FOREACH(p, qp, p_procq) {
        if (p->p_wchan == ident) { /* hash ok */
            TAILQ_REMOVE(qp, p, p_procq);
            p->p_wchan = 0;
            if (p->p_stat == SSLEEP) {
                if (p->p_slptime > 1)
                    updatepri(p);
                p->p_slptime = 0;
                p->p_stat = SRUN;  /* running */
                if (p->p_flag & P_INMEM) {
                    setrunqueue(p);
                    maybe_resched(p);
                } else {
                    p->p_flag |= P_SWAPINREQ;
                    wakeup((caddr_t)&proc0);
                }
                goto restart;
            }
        }
    }
    splx(s);
}
```

viz ps axl

Univerzální plánovač

Ruschitzka, Fabry (1977) - popis běžných plánovacích algoritmů
Založen na prioritě procesů. Procesor je v okamžiku rozhodování přidělen procesu s nejlepší prioritou.

Definován třemi charakteristikami:
1. interval rozhodování
2. prioritní funkce
3. výběrové pravidlo

1. Interval rozhodování

Definuje časové okamžiky, ve kterých je aktivován plánovač.
Uspořádání procesů se nemůže během intervalu měnit.

Nepreemptivní - proces běží do ukončení nebo čekání
+ malá režie přepínání kontextu, jednoduchá implementace
– delší odezva, nevhodné pro systémy reálného času
Preemptivní - procesu může být odebrán procesor:
- v pravidelných intervalech - *časové kvantum*
- odblokování procesu s vyšší prioritou
- příchod nového procesu (přerušení, spuštění proces)

Selektivní preempce - pro každý proces (u_p, v_p):
$u_p = 1$, pokud p může přerušit jiný proces, jinak 0
$v_p = 1$, pokud p může být přerušen jiným procesem, jinak 0
Příklad:
- pro časově kritické procesy (1, 0)
- pro ostatní (0, 1)

Selektivní preempce podle priority:
- procesy s vysokou prioritou jsou nepreemptibilní
- ostatní preemptibilní

2. Prioritní funkce

Funkce určující prioritu procesu na základě parametrů:

- paměťové požadavky
- spotřebovaný čas procesoru a
- doba čekání na přidělení procesoru w
- doba strávená v systému $r = a+w$
- celkový čas procesoru t = a po dokončení
- externí priorita (důležitost)
- perioda d
- lhůta zpracování
- zátěž systému

3. Výběrové pravidlo

výběr z více procesů se stejnou prioritou:

- náhodně
- cyklicky
- FIFO

Univerzální plánovač vyhodnocuje v určených okamžicích dle (1) prioritní funkci (2) pro všechny připravené procesy a dle výsledné priority a výběrového pravidla (3) přiděluje procesor procesu s nejlepší prioritou (číselně nejvyšší).

Plánovací algoritmy závislé na časových parametrech:

P(a, r, t, d)

a – spotřebovaný čas procesoru
r – čas strávený v systému
t – celkový čas procesoru
d – perioda opakování procesu

1. FIFO (FCFS)

Zpracování procesů v pořadí jejich příchodů
(1) nepreemptivní, v čase příchodu procesu (odblokování)
(2) P(r) = r
(3) náhodný výběr

+ jednoduché, malá režie přepínání kontextu
+ deterministická odezva
– krátké procesy musí čekat na dříve zahájené dlouhé, delší celková doba zpracování, odezva

2. LIFO

Zpracování vždy posledního přišlého procesu
(1) nepreemptivní, v čase příchodu procesu (odblokování)
(2) P(r) = -r
(3) náhodný výběr
– stárnutí

3. SJF (Shortest Job First)

Zpracovat vždy nejkratší proces (nutná znalost t)
(1) nepreemptivní, v čase příchodu procesu
(2) P(t) = -t
(3) náhodný výběr nebo podle r

+ kratší celková doba zpracování
+ malý počet čekajících procesů
– odhad t (dávkové systémy)
– stárnutí dlouhých procesů při neustálém příchodu krátkých

4. SRT (Shortest Remaining Time)

Zpracovat vždy nejkratší proces k dokončení (nutná znalost t)
(1) preemptivní, v čase příchodu procesu
(2) P(a, t) = a - t
(3) náhodný výběr nebo podle r

+ minimální celková doba zpracování
– odhad t (dávkové systémy)

5. Statická priorita (prioritní plánování)

(1) preemptivní (v pravidelných intervalech) i nepreemptivní
(2) P(i) = $konstanta_i$ dle procesu
(3) cyklicky
Typický algoritmus pro RT systémy, dvě úrovně priorit = vyšší, bez časového kvanta, nižší s časovým kvantem

6. RR (Round Robin - cyklická obsluha)

Rovnoměrné přidělování procesoru po časových kvantech q. Procesu, který vyčerpá časové kvantum, je procesor odebrán a proces je zařazen na konec fronty připravených procesů.
(1) preemptivní, v pravidelných intervalech
(2) P() = konstanta
(3) cyklicky
+ dobrá odezva, spravedlivé dělení času procesoru
– celková doba zpracování
Volba velikosti q (10 - 100 ms):
- příliš malé - velká režie
- příliš velké - velká doba odezvy (průměrně q.n/2)

Nastavení *q* při pozastavení procesu čekáním a následném odblokování:
- ponecháno původní částečně vyčerpané kvantum
- nastaveno nové plné kvantum

7. MLF (MultiLevel Feedback)

(1) preemptivní, v pravidelných intervalech *q*

(2) P(a) = n - i, $2^i \le \frac{a}{T_0} + 1 \le 2^{i+1}, 0 \le i < n$, *n* - počet úrovní

(3) cyklicky

- úrovně priorit i = 0..n, počátečně i = 0 (nejvyšší priorita)
- procesy ve stejné úrovni *i* obsluhovány cyklicky
- po vyčerpání časového limitu T_i, je proces přeřazen do úrovně *i*+1
- délka časového limitu $T_i = 2^i.T_0$
- při dosažení i=n je proces ukončen nebo vrácen do i=n-1

Priorita dlouhodobě běžících procesů je postupně snižována

úroveň	**priorita**	**limit**	**zpracování**
0	n	T_0	→CPU→CPU→CPU→CPU
1	n-1	$2.T_0$	←CPU→CPU→CPU→CPU
...	...		←CPU→CPU→CPU→CPU
n	chyba	∞	←

Problémy:

- překročení maximální úrovně
- proces po delším výpočtu má navždy špatnou prioritu

Základ pro běžně používané algoritmy, s modifikací pro zlepšování priority procesu při čekání na přidělení procesoru nebo po odblokování.

8. Rate Monotonic (RM)

Plánovací algoritmus pro reálný čas
Proces se opakuje s dobou periody *d*
Každý běh procesu musí být dokončen dříve než bude spuštěn znovu. Spouštíme vždy nejdříve ten, který má nejkratší periodu.
(1) preemptivní, v čase příchodu procesu
(2) P(d) = -d
(3) náhodný výběr nebo podle r

9. Earliest Deadline First (EDF)

Plánovací algoritmus pro reálný čas
Proces se opakuje s dobou periody *d*
Každý běh procesu musí být dokončen dříve než bude spuštěn znovu. Spouštíme vždy nejdříve ten, který má nejkratší lhůtu k dokončení (který má nejkratší čas do dalšího začátku)
(1) preemptivní, v čase příchodu procesu
(2) P(r, d) = -(d – r%d)
(3) náhodný výběr nebo podle r

Plánovač FreeBSD 4.4

Priorita běžícího procesu je přepočítávána po 40 ms:

$$prio = PUSER + \frac{cpu\_used}{4} + 2.nice, prio = 0..127 \text{ (0 nejvyšší)}$$

cpu_used je čas souvislého přidělení procesoru, inkrementuje se při každém přerušení časovače (po 10 ms)

Každou sekundu je *cpu_used* přepočteno:

$$cpu\_used = \frac{2.load}{(2.load + 1)}.cpu\_used$$

load je zatížení systému (~ počet připravených procesů)

Příklad:

[10 ms]	P1		P2	
load=2	**cpu_used**	**prio**	**cpu_used**	**prio**
t = 0	8	52	0	50 běží
t = 4	8	52	4	51 běží
t = 8	8	52	8	52 běží
t = 12	8	52 běží	12	53
t = 16	12	53 běží	12	53
t = 96	52	63 běží	52	63
t = 100	37	59	34	58 běží
			nový P3	
t = 200	58	64	0	50
t = 300	38	59	66	66

Plánovač Unix SVR4

Konfigurovatelný plánovač, standardně 3 třídy:

RT (100-159) - statická priorita, různé časové kvantum podle priority

SYS (60-99) - procesy v systémové fázi, vlákna jádra - statická priorita, bez časového kvanta

TS, IA (0-59) - tabulkově parametrizovaný MLF

```
dispadmin -c TS -g
```

Time Sharing Dispatcher Configuration

ts_quantum	ts_tqexp	ts_slpret	ts_maxwait	ts_lwait	priority
200	0	50	0	50	# 0
...					
160	0	51	0	51	# 10
160	1	51	0	51	# 11
160	2	51	0	51	# 12
160	3	51	0	51	# 13
160	4	51	0	51	# 14
160	5	51	0	51	# 15
160	6	51	0	51	# 16
160	7	51	0	51	# 17
160	8	51	0	51	# 18
160	9	51	0	51	# 19
120	10	52	0	52	# 20
...					
40	41	58	0	59	# 51
40	48	58	0	59	# 58
20	49	59	32000	59	# 59

ts_tqexp - nová priorita po vyčerpání časového kvanta

ts_slpret - nová priorita po ukončení čekání (*sleep()*/*wakeup()*)

ts_maxwait - maximální doba čekání před použitím *ts_lwait*

ts_lwait - nová priorita po čekání *ts_maxwait* na vyčerpání časového kvanta

Plánovač Windows

Rozsah priorit – 0 až 31 (0 je nejnižší)

„normální“ priority – 1..15 – priorita je dynamická, podle kvanta, kvantum je 2 (short) nebo 12 (long) intervalů časovače
real-time – 16..31 – priorita je statická

Plánovaní procesoru je pro vlákna, vlákna mají počátečně prioritu podle třídy procesu, lze nastavit (*SetThreadPriority*) na:

THREAD_PRIORITY_IDLE	1 nebo 16
THREAD_PRIORITY_LOWEST	-2
THREAD_PRIORITY_BELOW_NORMAL	-1
THREAD_PRIORITY_NORMAL	0
THREAD_PRIORITY_ABOVE_NORMAL	+1
THREAD_PRIORITY_HIGHEST	+2
THREAD_PRIORITY_TIME_CRITICAL	15 nebo 31

Třídy procesů (*SetPriorityClass*) jsou
IDLE_PRIORITY_CLASS (4),
BELOW_NORMAL_PRIORITY_CLASS (6),
NORMAL_PRIORITY_CLASS (8),
ABOVE_NORMAL_PRIORITY_CLASS (10),
HIGH_PRIORITY_CLASS (13),
REALTIME_PRIORITY_CLASS(24)

Při probuzení po odblokování nebo čekání na procesor > 4s je priorita vlákna nastavena na bázovou+X (závisí na ovladači, lze zakázat pomocí *SetProcessPriorityBoost*), do max. 15. Po vyčerpání časového kvanta je dynamická priorita vlákna snížena o 1, do bázové priority vlákna.
Vlákna systému běží v intervalu RT priorit, kromě idle procesu nulování stránek, který má prioritu 0.

Plánování pro víceprocesorové systémy

Různé požadavky na univerzální systém:

- víceuživatelský,
- web server,
- databázový server,
- paralelní aplikace.

Je třeba zohlednit architekturu (SMP, NUMA), organizaci a propustnost paměťových sběrnic, vícejádrové/SMT jádra procesorů (nejprve rozdělit mezi fyzické procesory, teprve pak zatěžovat sdílené logické procesory), sdílení cache L2/L3, atd.

A) Plánovač společný pro všechny procesory:

- neodpovídá požadavkům, přiděluje procesory náhodně,
- omezující při větším počtu procesorů.

B) Jedna fronta připravených procesů, každý procesor vybírá samostatně procesy z této fronty (varianta předchozího):

- globální zámek fronty omezuje paralelní plánování

C) Fronta připravených procesů pro každý procesor:

- problém vyrovnávání zátěže procesorů a migrace procesů („kradení“ procesů z nejdelší fronty)
- řazení nových/probuzených procesů do správné fronty

Efektivní využití cache:

- vícevláknové programy omezit na jeden procesor/jádra v jednom procesoru – efektivní využití cache, TLB, ale ne výpočetního výkonu

Procesor-affinity – plánovač zohledňuje, na kterém procesoru vlákno běželo, a snaží se nemigrovat vlákno na jiné procesory (pouze v případě, že jiný je volný a nemá co dělat)

Plánovač ULE ve FreeBSD

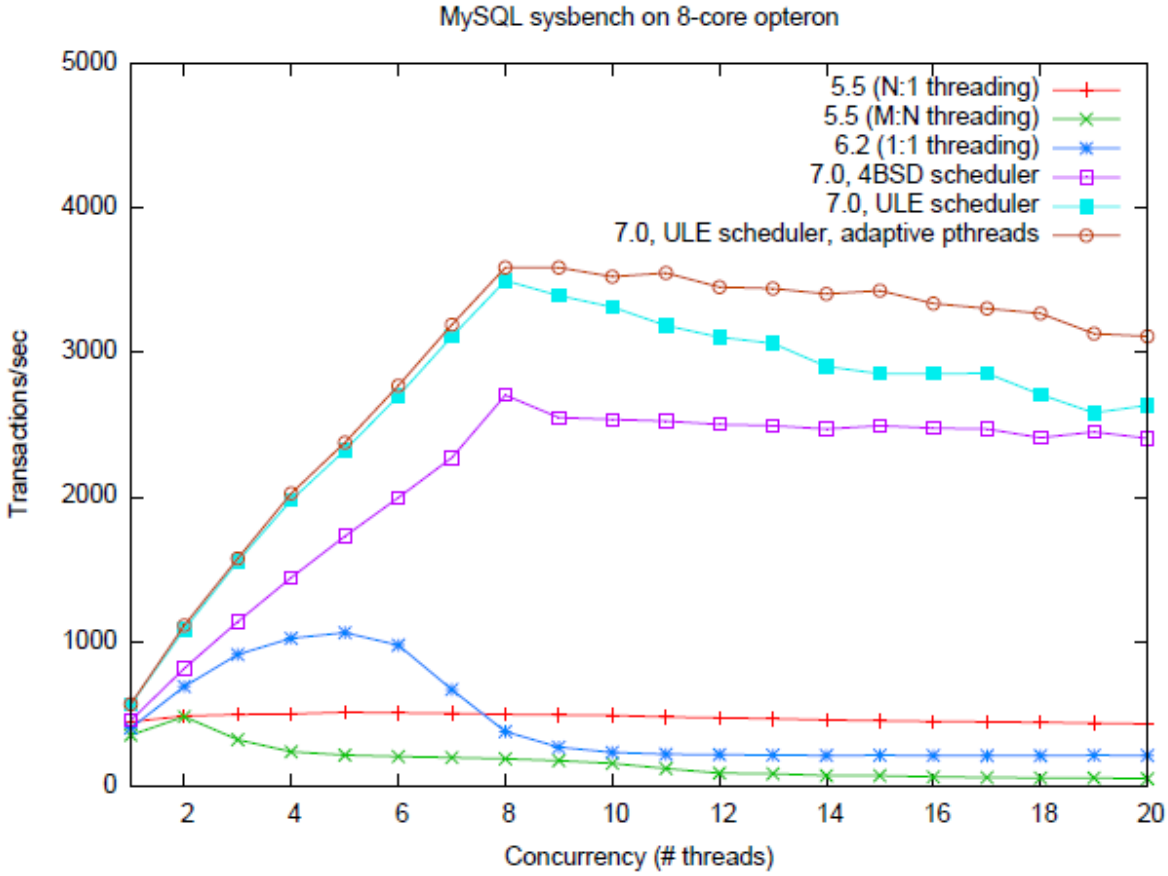
Problém klasického plánovače – musí procházet všechny procesy a přepočítávat priority pro implementaci MLF algoritmu – O(n) (Linux 2.4, BSD plánovač). Princip O(1) plánovače (Linux<2.6.23) – použít pro každý procesor dvě fronty: první - ze které se bere od začátku, druhou - do které se vkládá podle priority. Po vyprázdnění první se fronty prohodí. Prohozením dojde k tomu, že proces s nízkou prioritou nemůže být trvale odstaven od procesoru, dostane jej přidělen alespoň jednou během 2 zpracování front.

Problémy:

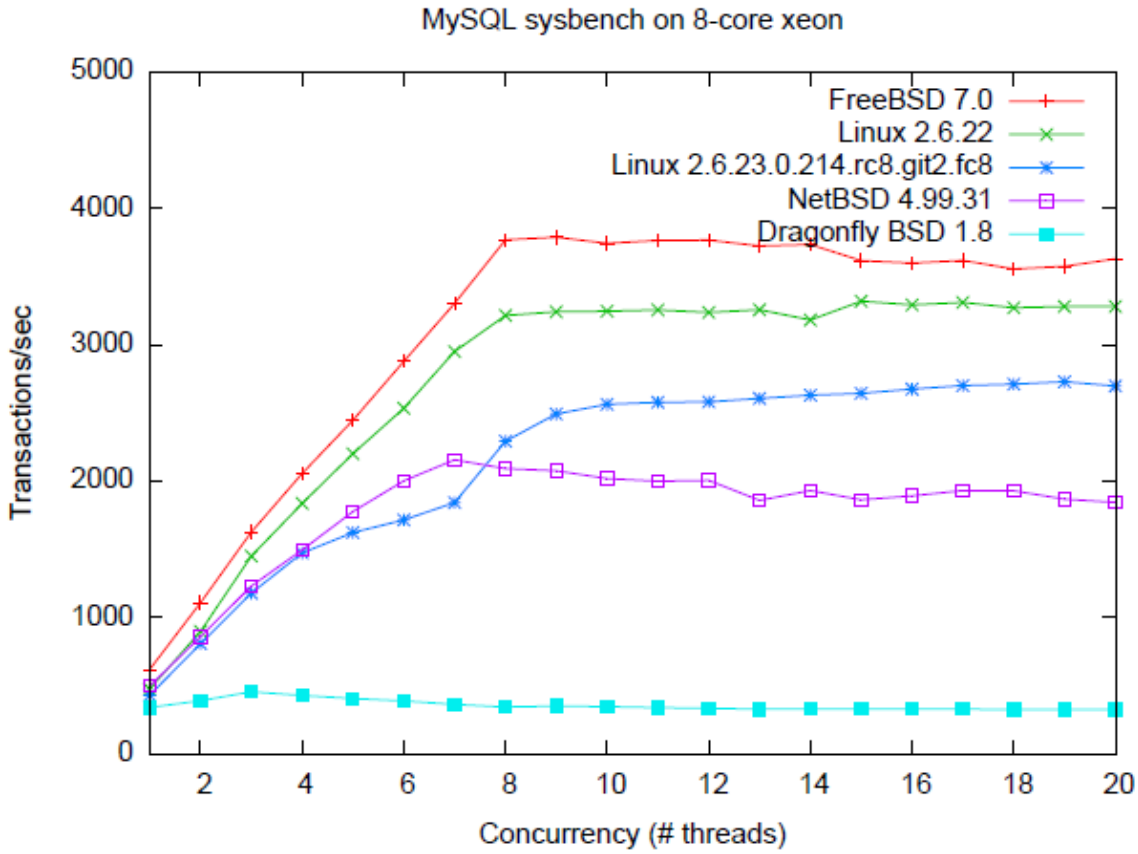
- nelze přímo implementovat „nice“
- vkládání do fronty podle priority je náročná operace

Plánovač ULE ve FreeBSD od verze 7:

- pro každý procesor 3 fronty: *Idle*, *Current*, *Next*
- prováděná vlákna vybírána postupně z *Current* podle priority
- když je *Current* prázdná, prohodí se *Current* a *Next*
- vlákna z *Idle* se provádí pouze, když je *Current* i *Next* prázdná
- pokud nemá procesor co dělat, „krade“ vlákna z front ostatních procesorů (vybírá první z nejzatíženějšího procesoru)
- může se stát, že *Idle* vlákno na jednom procesoru nedovolí „kradení“ a tím vybalancování zátěže – periodicky (co 0,5s) se zvolí nejméně a nejvíce zatížený procesor a jeho nadbytečná vlákna se přesunou na nejméně zatížený tak, aby měli stejně
- vlákna obsluhy přerušení a realtime se vkládají vždy do *Current* (tím předbíhají ostatní v *Next*)
- „interaktivní“ vlákna se také vkládají do *Current*, ostatní do *Next*
- prioritní vkládání je řešeno indexem podle priority a frontou pro každou úroveň priority (přímé vkládání, bit ve slově indikuje, zda je fronta prázdná, lze jednou instrukcí otestovat, zda jsou všechny prázdné a která je první neprázdná) – obdobně O(1) plánovač v Linuxu.



Pozn: FreeBSD 7.0 i Linux používají model vláken N:N (1:1)



Linux 2.6.22 – O(1), Linux 2.6.23 – CFS plánovač

Completely Fair Scheduler (CFS)

Linux >= 2.6.23 (Ingo Molnar)
Princip: výpočetní kapacita systému ve formě volného strojového času procesorů je rozdělena spravedlivě mezi procesy podle jejich „priority“ podobně jako kapacita komunikační linky. Procesory s normální prioritou dostávají stejný díl, procesy s nižší prioritou se podělí o patřičně menší díl (poloviční). Na rozdíl od předchozích algoritmů není časové kvantum, ale granularita (jak moc může překročit „spravedlivý díl“).

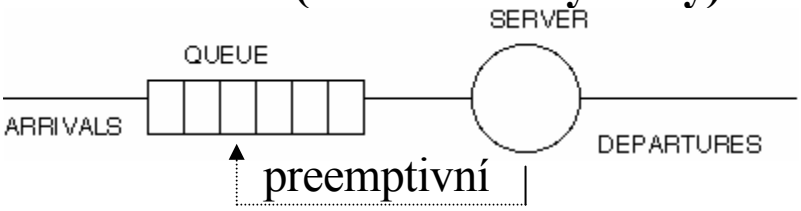
Procesy, které čekají na přidělení procesoru, jsou uloženy v balancovaném binárním stromu (red/black) podle virtuálního času procesoru (*vruntime*). Proces, který dostal nejméně virtuálního času, je nejlevější ve stromu, je vyjmut ze stromu a dostane přidělen procesor jako první. Běží tak dlouho, dokud jeho virtuální čas nepřekročí čas nyní nejlevějšího ve stromu. Jeho virtuální čas se zvětší o (dobu běhu/váha) a je zařazen zpět do stromu. Nově spuštěný proces dostane počáteční virtuální čas rovný minimálnímu v systému (jinak by mohl dlouhodobě předbíhat všechny právě běžící).
Náročnost plánovače: O(log N) – náročnost operace vložení do vybalancovaného binárního stromu

Hlavní problém všech plánovačů – jak identifikovat interaktivní procesy a dát jim větší prioritu pro lepší odezvu. Co když se stane z interaktivního procesu proces výpočetně náročný a naopak? Jak rychle na to plánovač zareaguje?
O(1) plánovač – složitá heuristika aktualizace dynamické priority
CFS – *vruntime* po probuzení z čekání nastaveno na minimum
ULE plánovač – poměr doby synchronního čekání a běhu, po čase se redukuje a připočítá znovu aktuální (historie + nové chování), vlákna překračující nastavený limit se považují za interaktivní a vkládají se do *Current*

Hodnocení plánovacích algoritmů

1. matematický model - systémy hromadné obsluhy
2. simulace
3. měření na reálném systému - monitorování

Model M/M/1 (Markovské systémy)



- počet příchodů je nekonečný
- distribuce příchodů a doby obsluhy má exponenciální rozložení
- režim fronty je FIFO
- zkoumá se ustálený stav

$$\rho = \frac{\lambda}{\mu}$$

ρ = intenzita provozu (zátěž), musí být < 1
λ = 1/střední doba mezi příchody (parametr exp. rozložení)
μ = 1/střední doba obsluhy (parametr exp. rozložení)

průměrný počet zpracovávaných požadavků (musí být < 1):

$$l_p = \frac{\rho}{1-\rho}$$

průměrný počet požadavků ve frontě:

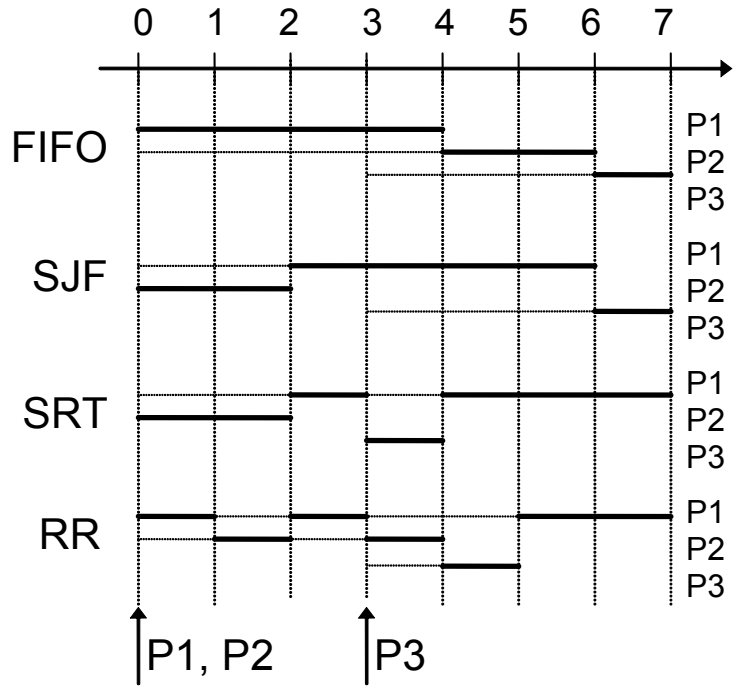
$$l_q = \frac{\rho^2}{1-\rho}$$

střední doba čekání, doba strávená v systému:

$$w = \frac{\rho}{\mu - \lambda} \qquad s = \frac{1}{\mu - \lambda}$$

Simulace

Ganttův diagram



r_1=4
r_2=6 R=14 s=4,7
r_3=4
r_1=6
r_2=2 R=12 s=4
r_3=4
r_1=7
r_2=2 R=10 s=3,3
r_3=1
r_1=7
r_2=4 R=13 s=4,3
r_3=2

R - celková doba strávená v systému (všech procesů)
s - střední doba strávená v systému