SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 627

Raspoređivanje procesa u Win32 okruženju

Berislav Hunjadi

ZAGREB, lipanj 2009.

Zahvaljujem mentoru doc. dr. sc. Leonardu Jelenkoviću na predloženoj temi i pruženoj stručnoj pomoći i savjetima kod izrade rada.

Sadržaj

1. Uvod

Operacijski sustav je danas baza svakog računala. Prije četrdeset i više godina računala su bila programirana tako da su mogla obavljati samo specifične zadatke. Pojavom operacijskih sustava 60-tih godina 20. stoljeća računala su postala neovisna o samom sklopovlju, npr. programi mogu raditi na više različitih procesora i slično.

Prvi operacijski sustavi bili su jednozadaćni i jednokorisnički. Računala su postajala sve brža i javila se ideja o višezadaćnim operacijskim sustavima (prvi UNIX je bio višezadaćni). Odvijanje više procesa u isto vrijeme otvorilo je i neke nove mogućnosti, ali i probleme u razvoju operacijskih sustava. Da bi višezadaćni rad uopće bio moguć, operacijski sustav treba upravljati memorijom i raspoređivati procesorsko vrijeme na procese koji su pokrenuti na računalu. Procesorsko vrijeme treba biti tako raspoređeno da svaki proces dobije svoj dio vremena. Na današnjim računalima nakon samog podizanja operacijskog sustava može biti pokrenuto i više od 50 procesa te više od 500 dretvi. Za normalan rad potrebno je implementirati dobar algoritam raspoređivanja zadataka i rad s memorijom mora biti efikasan.

U radu se objašnjava rad raspoređivača zadataka u operacijskim sustavima te različiti algoritmi raspoređivanja dretvi i procesa. Objašnjeno je kako radi raspoređivač u Windows XP operacijskom sustavu te su prikazani rezultati dobiveni ispitivanjem raspoređivača zadataka programima koji računaju trajanje promjene konteksta, način rada raspoređivača i slično.

2. Raspoređivač zadataka

Cilj raspoređivača zadataka u operacijskom sustavu je omogućiti istodobno izvođenje više procesa na računalu. Na računalu se najčešće odvija mnogo više procesa u neko vrijeme nego što postoji procesora. Pitanje je kako je moguće da se ti procesi rasporede po procesorima (procesoru) i izvode istodobno. U praksi, procesi se ne izvode istodobno (osim na višeprocesorskim sustavima), nego operacijski sustav svakom procesu dodijeli određeni dio vremena za izvođenje. Korisniku taj rad izgleda kao istodoban jer su ti vremenski odsječci veoma kratki (reda veličine deset milisekundi). Na višeprocesorskim sustavima istodobno se može vrtjeti onoliko procesa koliko postoji procesora. Ako procesor podržava *hyper-threading* onda se na jednom procesoru mogu izvršavati i dvije dretve istovremeno.

Raspoređivač zadataka je, dakle, program koji je zadužen za optimalno iskorištenje procesora, protok (tj. broj procesa koji završe izvođenje u određenom vremenskom intervalu), vrijeme za izvršavanje određenog procesa, vrijeme čekanja, vrijeme odgovora i pravednost (dati isto vrijeme svakoj dretvi).

2.1. Procesi i dretve

Aplikacije na našim računalima sastoje se od jednog ili više procesa. Najjednostavnije rečeno, proces je izvršni program. Svaki proces se sastoji od jedne ili više dretvi (*engl.* thread) pa su dretve zapravo osnovne jedinice kojima operacijski sustav dodjeljuje procesorsko vrijeme.

Svaki proces sadrži resurse potrebne da se program može izvesti. Operacijski sustav procesu dodjeljuje virtualni adresni prostor. Svaki proces sastoji se od izvršnog koda, upravljača na objekte sustava (*engl.* handle), sigurnosnog konteksta, unikatnog identifikatora, varijabli okoline (*engl.* environment variables), klase prioriteta, minimalne i maksimalne veličine prostora rada (*engl.* working set size) i najmanje jedne dretve. Svaki proces na početku sadrži samo jednu, primarnu dretvu, ali može kasnije stvoriti nove dretve.

Dretva je dio procesa kojem može biti dodijeljeno vrijeme za izvršavanje. Dretve dijele virtualnu memoriju i resurse sustava s procesom koji ih je stvorio. Dodatno, dretve sadrže upravljače iznimaka (*engl.* exception handle), prioritet raspoređivanja, lokalne podatke, unikatni identifikator i strukture koje će koristiti operacijski sustav pri promjeni konteksta.

Kontekst dretve uključuje registre procesora koje koristi dretva, stog jezgre, blok okruženja dretve i korisnički stog, a mogu imati još i sigurnosni kontekst.

2.2. Višezadaćni rad

Operacijski sustavi koji podržavaju višezadaćni rad (*engl.* multitasking) raspoređuju slobodno procesorsko vrijeme između dretvi koje ga trebaju. Raspoređivač dodjeljuje procesorsko vrijeme dretvama. Kada procesor aktivno izvršava neki zadatak, tj. dio koda neke dretve, kažemo da je taj zadatak trenutno aktivan (*engl.* running). Raspoređivač na temelju nekog algoritma raspoređuje vrijeme na dretve koje su aktivne u tom sustavu. Koriste se mali vremenski odsječci koji se daju pojedinoj dretvi tako da korisnik dobiva iluziju paralelnog rada više procesa. Promjena konteksta je posljedica višezadaćnog rada i uključuje pohranjivanje sadržaja svih registara procesora u opisnik dretve i obnavljanje stanja procesora kojeg koriste mnogi procesi i dretve.

Raspoređivač zadataka može raditi na dva osnovna načina – raspoređivanjem po prioritetu i raspoređivanjem podjelom vremena.

Operacijski sustavi mogu imati različite strategije raspoređivanja: multiprogramski sustavi, sustavi s raspodjelom vremena (*engl.* time-sharing, multitasking) i sustavi u realnom vremenu (*engl.* real time).

2.2.1. Promjena konteksta

Promjena konteksta (*engl.* context switch) je jedan od ključnih elemenata u operacijskom sustavu. Što spada pod kontekst ovisi o procesoru te operacijskom

sustavu. Vrijeme promjene konteksta nastoji se minimizirati te se dosta pažnje posvećuje optimiranju u korištenju promjena konteksta.

Pseudokod promjene konteksta izgleda otprilike ovako:

```
promjena_konteksta() {
      Stavi registre na stog;
      Spremi pokazivače na kod i podatke;
      Spremi pokazivač stoga;

      Uzmi sljedeći proces za izvršavanje;

      Uzmi pokazivač stoga tog procesa;
      Uzmi pokazivače na kod i podatke;
      Uzmi registre sa stoga;
}
```

Algoritam raspoređivanja definira sljedeći proces koji će se izvršiti (u pseudokodu je to funkcija `Uzmi sljedeći proces za izvršavanje`).

Informacije o kontekstu pohranjuju se u strukturu procesni informacijski blok (*engl.* Process Control Block, PCB). Struktura se nalazi u jezgri operacijskog sustava i sadrži sve potrebne informacije za upravljanje određenim procesom. Ta struktura ovisna je o implementaciji, ali najčešće se u njoj nalaze:

- identifikator procesa
- vrijednosti registara i programskog brojača (*engl.* program counter, PC)
- adresni prostor procesa
- prioritet
- informacije o ulazu i izlazu (lista otvorenih datoteka i slično)
- pokazivač na sljedeću PCB strukturu (tj. pokazivač na sljedeći proces koji se treba izvršiti)

PCB sadrži vrlo važne informacije o procesu te se stoga nalazi u memoriji jezgre koja je zaštićena od pristupa korisnika.

2.2.2. Multiprogramski sustavi

Na početku razvoja računala, procesorsko je vrijeme bilo skupo, a periferni uređaji su bili veoma spori. Kada je program trebao pristupiti vanjskom uređaju, procesor je trebao čekati na taj uređaj. To je bilo veoma neefikasno. 1960.-ih su izvršeni prvi pokušaji stvaranja multiprogramskih sustava. Ideja je bila da se pokrene nekoliko procesa. Kad prvi dođe do naredbe koja treba čekati na vanjski uređaj, spremi se kontekst tog procesa i pokreće se drugi proces i tako sve dok se ne završe svi procesi.
Glavni nedostatak ovog principa je da neki proces ne mora komunicirati s okolinom, pa ništa ne garantira da će se ostali procesi izvršiti.

2.2.3. Sustavi s raspodjelom vremena

Nakon što je korištenje računala uznapredovalo na interaktivnu razinu, multiprogramski pristup više nije bio dobar izbor. Svaki korisnik je želio da izgleda kao da je njegov program jedini koji se izvodi. Princip u kojem se vremenski odsječci daju procesima je omogućio takav rad. Dva su osnovna načina na koja se to može ostvariti: suradnički (*engl.* cooperative) i istiskujući (*engl.* preemptive) rad. Kod suradničkog rada, svaki proces odlučuje u kojem će trenutku predati procesorsko vrijeme nekom drugom procesu. Prednost ovog načina rada naspram istiskujućeg je da nema zamjene konteksta, ali veliki nedostatak je da jedan loš program može usporiti cijeli sustav. Istiskujući način rada omogućava sustavu da s većom pouzdanošću garantira da će neki proces dobiti dio procesorskog vremena. Također je moguće dodjeljivanje prioriteta procesima u slučaju bitnih odsječaka koda koji se moraju brzo izvoditi ili kod ulaznih podataka. Istiskujući višezadaćni rad je podržavala i prva verzija UNIX-a 1969. Procesi su podijeljeni u dvije grupe: oni koji čekaju na ulazne ili izlazne podatke i na one koji koriste procesor. Kad je proces čekao na neki ulaz, sustav nije ništa radio. Pojavom prekida (*engl.* interrupt) i istiskujućeg rada, bilo je moguće drugim procesima dodijeliti procesor dok jedni čekaju na ulaze. Promjena konteksta je karakteristična za ovaj način rada.

2.2.4. Sustavi u realnom vremenu

Višezadaćni rad je veoma bitan u sustavima u realnom vremenu (*engl.* real-time system). U takvim sustavima postoji mnogo nepovezanih vanjskih aktivnosti koje kontrolira jedan procesor. Da bi se osiguralo da glavne aktivnosti dobivaju veće odsječke vremena, spajaju se u sustav hijerarhijskih prekida (*engl.* hierarchical interrupt system) i prioriteta.

2.3. Vrste raspoređivača zadataka

U operacijskom sustavu mogu postojati tri vrste raspoređivača: dugoročni, srednjoročni i kratkoročni. Imena im određuju učestalost korištenja.

2.3.1. Dugoročni raspoređivač

Dugoročni raspoređivač odlučuje koji će procesi biti propušteni u red čekanja spremnih procesa. Npr. kada se želi izvršiti neki program, raspoređivač može dopustiti da proces postane dio trenutačno izvršavajućih procesa ili je odgođen. Dugoročni raspoređivač odlučuje koji procesi će se izvoditi na računalu i koji će stupanj istodobnosti biti podržan u određenom trenutku. Primjerice, dugoročni raspoređivač odlučuje hoće li mnogo ili malo procesa biti istovremeno izvršavano te kako upravljati procesima koji intenzivno rade neke ulazno-izlazne operacije i onima koji intenzivno koriste procesor.

2.3.2. Srednjoročni raspoređivač

Srednjoročni je raspoređivač prisutan u svim sustavima koji imaju virtualnu memoriju. Zadatak mu je da privremeno premjesti procese iz glavne memorije u sekundarnu (npr. čvrsti disk) ili obrnuto. Srednjoročni raspoređivač može zamijeniti

procese koji dulje vrijeme nisu bili aktivni, one s niskim prioritetom, one koji koriste veliku količinu memorije, itd.

2.3.3. Kratkoročni raspoređivač

Kratkoročni raspoređivač odlučuje koji će procesi u redu čekanja biti izvršeni, tj. kojima će biti dodijeljeno procesorsko vrijeme, nakon prekida sata, prekida ulazno-izlaznog uređaja, sustavskog poziva ili nekog drugog oblika signala. Kratkoročni raspoređivač odlučuje mnogo češće od ostalih, najmanje jedanput u vremenskom odsječku koji je jako kratak. Taj raspoređivač može biti suradnički ili istiskujući. Istiskujući može sam izbaciti procese iz procesora i procesorsko vrijeme dodijeliti drugom procesu, dok suradnički ne može.

2.4. Algoritmi raspoređivanja

Algoritmi raspoređivanja koriste se i u usmjerivačima pri upravljanju prometom paketa, u čvrstim diskovima pri ulazno-izlaznim operacijama, pisačima i sl. Glavna zadaća algoritama je minimiziranje izgladnjivanja resursa (*engl.* resource starvation) da bi se osigurala pravednost između strana koje koriste neke resurse.

U operacijskim sustavima algoritam raspoređivanja je metoda kojom dretve, procesi i tokovi podataka dobivaju pristup sredstvima sustava. Potreba za algoritmima proizlazi iz zahtjeva za višezadaćnim radom kojeg većina modernih sustava podržava i multipleksiranjem.

Najjednostavniji algoritam raspoređivanja je kružno posluživanje (*engl.* round-robin). Taj algoritam svakom procesu dodijeli isti vremenski odječak (obično između 1 ms do 100 ms). Procesi se stavljaju u ciklički red što znači da taj algoritam ne razlikuje prioritete. Napredniji algoritmi uzimaju u obzir prioritet ili važnost procesa što omogućuje da bitniji procesi dobivaju više vremena od onih manje bitnih.

Najpoznatiji i najviše korišteni algoritmi su: višerazinski povratni red čekanja, potpuno pravedan raspoređivač i O(1).

2.4.1. Implementacija raspoređivača u operacijskim sustavima

Rane verzije MS-DOS i Microsoft Windows operacijskih sustava nisu bile višezadaćne. Windows 3.1x nisu koristili istiskujući raspoređivač pa nisu prekidali procese u izvođenju. Windows 95 bili su prvi koji su koristili istiskujući raspoređivač, ali su se 16-bitne aplikacije pokretale bez prevencije. Microsoftovi operacijski sustavi bazirani na Windows NT (to su svi poslije Windows 2000) koristili su višerazinski povratni red čekanja (istiskujući algoritam). Postoje 32 razine prioriteta, od kojih su prvih 16 normalni prioriteti, a sljedećih 16 prioriteti realnog vremena. Korisnik može koristiti šest prioriteta. Jezgra može promijeniti razinu prioriteta ovisno o ulazno-izlaznim naredbama i korištenju procesora. Windows Vista ima malo modificiran raspoređivač koji koristi ciklički registar modernih procesora i prati koliko je procesorskih taktova određena dretva dobila. Prijašnji raspoređivači su funkcionirali tako da su dodjeljivali vremenske odsječke.

Mac OS 9 koristi suradnički raspoređivač u kojem jedan proces kontrolira više suradničkih dretvi. Za dodjeljivanje procesorskog vremena procesima jezgra koristi algoritam kružnog posluživanja. Svaki proces ima svoju kopiju upravljača dretvi koji dodjeljuje procesorsko vrijeme dretvama. Jezgra zatim koristi istiskujući algoritam omogućava da svi zadaci dobe procesorsko vrijeme.

Linux do verzije 2.5 koristi višerazinski povratni red čekanja sa prioritetima od 0 do 140. Prioriteti od 0 do 99 su rezervirani za zadatke u realnom vremenu, a ostalo je za normalne zadatke. Za zadatke u realnom vremenu sustav dodjeljuje odsječke od oko 200 ms, a za normalne zadatke 10 ms. Raspoređivač gleda prvo procese s najvišim prioritetom, pa kad završi njihov odsječak stavljaju se u red isteklih procesa. Kad u redu aktivnih više nema procesa, red isteklih se proglasi aktivnim. Od verzije 2.5 do verzije 2.6.23 koristi se O(1) raspoređivač, a poslije 2.6.23 koristi se potpuno pravedan raspoređivač koji koristi crveno-crna stabla umjesto redova čekanja.

FreeBSD koristi višerazinski povratni red čekanja sa prioritetima od 0-255. Do 63. su rezervirani za prekide, 64-127 za gornju polovicu jezgre, 128-159 za korisničke dretve u realnom vremenu, 160-223 za korisničke dretve koje dijele resurse s ostalima i 224-255 za korisničke dretve niskog prioriteta.

Solaris također koristi višerazinski povratni red čekanja, samo s prioritetima od 0 do 169. Za razliku od Linux-a procesu koji je završio dodijeljen je novi prioritet i tek onda se stavlja natrag u red čekanja.

2.4.2. Višerazinski povratni red čekanja

Algoritam koji želi zadovoljiti zahtjeve heterogenih sustava. Cilj je dati prednost procesima koji obavljaju kratke poslove, procesima koji su vezani za ulazno-izlazne naredbe te brzo ustanoviti prirodu procesa i dodijeliti mu vrijeme sukladno tome. Algoritam koristi više FIFO redova čekanja. Princip rada:

- novi proces se pozicionira na kraj najvišeg FIFO reda
- nakon nekog vremena proces dospije na početak reda te mu se dodijeli procesorsko vrijeme
- ako je proces gotov, izlazi iz sustava
- ako proces svojevoljno prepusti kontrolu, izlazi iz reda čekanja, a tek poslije kad opet postane spreman, ulazi u sustav u isti red čekanja ili čak i red više
- ako proces iskoristi cijeli vremenski odječak, stavlja ga se na kraj reda nižeg prioriteta
- postupak se ponavlja sve dok proces ne završi izvođenje ili dosegne red osnovne razine, tj. početno pridijeljenog prioriteta)

Na osnovnoj razini procesi cirkuliraju na temelju algoritma kružnog posluživanja sve dok nisu gotovi.

2.4.3. O(1) raspoređivač

To je raspoređivač koji dodjeljuje procesima vremenski odsječak unutar nekog konstantnog vremena neovisno o tome koliko procesa je pokrenuto na računalu. Osnovna ideja ovog algoritma je da se smanje resursi koje koristi sam

raspoređivač. O(1) ne znači da je to najbrži algoritam, ali mu za svaki izračun treba isto vrijeme, neovisno o broju procesa.

2.4.4. Potpuno pravedan raspoređivač

Koristi se u Linux-u od 2.6.23 verzije jezgre. Upravlja alokacijom procesorskih resursa za procese koji se izvršavaju i maksimizira iskorištenje procesora uz zadovoljavajuće raspoređivanje interaktivnih aplikacija. Algoritam se temelji na starijem algoritmu pravednog raspoređivanja. Algoritam nije baziran na redovima čekanja, nego na crveno-crnim stablima koja označavaju vremensku liniju budućeg izvršavanja zadataka. Raspoređivač računa u nanosekundama i svakom procesu dodjeljuje određeno vrijeme, a ne vremenske odsječke. Kompleksnost algoritma izražena u veliko O notaciji je O(log N). Odabir zadatka je u konstantnom vremenu, a za ponovno stavljanje treba log N operacija zbog toga jer je red čekanja implementiran u obliku stabla.

3. Windows XP i raspoređivanje procesa i dretvi

32-bitni operacijski sustav Windows XP za raspoređivanje procesa koristi algoritam višerazinskog povratnog čekanja. Podržana su maksimalno 32 fizička ili logička procesora. Isto tako postoje 32 razine prioriteta. Sve dretve istog prioriteta se jednako tretiraju. Sustav dodjeljuje jednake vremenske odsječke dretvama najvećeg prioriteta. Kad niti jedna dretva najvišeg prioriteta nije spremna za izvršavanje, prelazi se na dretve sljedećeg nižeg prioriteta. Ako je pokrenuta dretva višeg prioriteta od one koja se izvodi, sustav prekida izvođenje te dretve i pokreće onu s višim prioritetom. U Win32 okruženju razlikujemo osnovni i dinamički prioritet. Osnovni prioritet dretve (*engl.* base priority) je definiran klasom prioriteta procesa pod kojim se dretva izvršava te razinom prioriteta dretve.

3.1. Dinamičko podizanje prioriteta

Dinamički prioritet koristi raspoređivač kako bi mogao odlučiti koju dretvu treba izvršavati. Svaka dretva ima dinamički prioritet koji je na početku jednak osnovnom. Operacijski sustav može smanjiti ili povećati prioritet kako bi spriječio izgladnjivanje i poboljšao odziv. Samo dretvama s prioritetima od 0 do 15 sustav može podići prioritet (*engl.* priority boost).

Dinamičko podizanje prioriteta obavlja se u sljedećim slučajevima:

- kada se proces normalne klase prioriteta dovede u prednji plan (tj. prema korisniku)
- kada prozor prima neke ulazne podatke
- kada se dretva odblokira

Nakon što je podignut dinamički prioritet dretve, raspoređivač smanjuje taj prioritet nakon svakog odsječka vremena koji dretva dobije i u potpunosti iskoristi, sve dok ne dođe do svog inicijalnog prioriteta.

Provjera da li nekoj dretvi ili procesu operacijski sustav može podići prioritet iznad osnovnog obavlja se sljedećim funkcijama:

```
BOOL GetThreadPriorityBoost(HANDLE hThread,
                            PBOOL pDisablePriorityBoost);
BOOL GetProcessPriorityBoost(HANDLE hProcess,
                            PBOOL pDisablePriorityBoost);
```

Postavljanje mogućnosti dinamičnog postavljanja prioriteta omogućavaju:

```
BOOL SetThreadPriorityBoost(HANDLE hThread,
                            BOOL disablePriorityBoost);
BOOL SetProcessPriorityBoost(HANDLE hProcess,
                            BOOL disablePriorityBoost);
```

3.2. Inverzija prioriteta

Inverzija prioriteta je slučaj kada se dvije ili više dretvi različitih prioriteta nalaze u sukobu oko nekog resursa. Pretpostavimo da postoje tri dretve u sustavu. Jedna dretva je visokog prioriteta (dretva A), druga srednjeg prioriteta (dretva B), a treća je niskog prioriteta (dretva C). Dretva C je u kritičnom odsječku, a počinje se izvršavati dretva A. Dretva A čeka na resurse dretve C, tj. dretva A je blokirana. U sustavu još postoji i dretva B koja se izvršava neovisno o dretvama A i C. Budući da dretva A čeka, izvršava se dretva B jer je ona srednjeg prioriteta. Dretva C ne dobiva procesorsko vrijeme, a dretva A je zablokirana iako je visokog prioriteta. Raspoređivač rješava taj problem tako da naizmjenično povećava prioritete dretvi u redu pripravnih dretvi.

3.3. Prioriteti

Prioriteti u Windows XP se dijele na dvije osnovne grupe – klase prioriteta (*engl.* priority class) i razine prioriteta (*engl.* priority level). Klase prioriteta

pridjeljuju se procesima, a razina prioriteta dretvama. Prioritet procesa i prioritet dretve zajedno daju osnovni prioritet u sustavu. Na slici 1. prikazano je kako se postavlja osnovni prioritet dretvi na temelju prioriteta dretvi i procesa.

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Slika 1. Osnovni prioritet dretve

Na slici 1. nisu prikazane sve mogućnosti dobivanja osnovnog prioriteta. Postoje još razine prioriteta od -7 do -3 i od 3 do 6. Te se razine mogu koristiti samo u slučaju kada je klasa prioriteta postavljena na REALTIME_PRIORITY_CLASS.

Osnovni prioritet može poprimiti vrijednosti od 0 do 31 gdje je 0 najniži prioritet, a 31 najviši. Vrijednosti od 16 do 31 su prioriteti dretvi koje koriste raspoređivanje u realnom vremenu (*engl.* real time).

3.3.1. Klase prioriteta procesa

Prioritet nekog procesa pripada jednoj od sljedećih klasa:

- IDLE_PRIORITY_CLASS,
- BELOW_NORMAL_PRIORITY_CLASS,
- NORMAL_PRIORITY_CLASS,
- ABOVE_NORMAL_PRIORITY_CLASS,
- HIGH_PRIORITY_CLASS,
- REALTIME_PRIORITY_CLASS.

Podrazumijevana klasa pri stvaranju procesa je NORMAL_PRIORITY_CLASS i svakom procesu je dodijeljen dinamički prioritet. U funkciji CreateProcess() možemo zadati željenu klasu prioriteta novog procesa. U slučaju da ne specificiramo koju klasu prioriteta želimo pri kreiranju, dodijeljena klasa prioriteta bit će NORMAL_PRIORITY_CLASS, osim u slučaju ako početni proces spada pod IDLE_PRIORITY_CLASS ili BELOW_NORMAL_PRIORITY_CLASS. U tom slučaju dijete poprima klasu prioriteta roditelja.

Funkcije kojima možemo dobiti osnovnu klasu prioriteta procesa i postaviti prioritet:

```
DWORD GetPriorityClass(HANDLE hProcess);
DWORD SetPriorityClass(HANDLE hProcess, DWORD dwPriorityClass);
```

Klasu HIGH_PRIORITY_CLASS treba koristiti s oprezom. Ako dretva s tim prioritetom radi dulje vrijeme, ostale dretve neće dobiti procesorsko vrijeme. Tu klasu prioriteta treba dodijeliti samo onim dretvama koje obavljaju vremenski kritične događaje. U praksi je najbolje povećati prioritet dretve neposredno prije vremenski kritičnog dijela koda, a odmah poslije smanjiti prioritet.
Praktično se nikad ne koristi REALTIME_PRIORITY_CLASS jer ometa sustavske dretve koje rukuju ulazom od miša i tipkovnice. Ta klasa je pogodna samo za aplikacije koje komuniciraju direktno sa sklopovljem.

3.3.2. Razine prioriteta dretve

Prioritet neke dretve određen je razinama:

- THREAD_PRIORITY_IDLE,
- THREAD_PRIORITY_LOWEST,
- THREAD_PRIORITY_BELOW_NORMAL,
- THREAD_PRIORITY_NORMAL,
- THREAD_PRIORITY_ABOVE_NORMAL,
- THREAD_PRIORITY_HIGHEST,
- THREAD_PRIORITY_TIME_CRITICAL.

Razina svih kreiranih dretvi je postavljena na THREAD_PRIORITY_NORMAL. To znači da je prioritet dretve isti kao i prioritet procesa. Za dobivanje i postavljanje osnovnog prioriteta dretve koriste se sljedeće funkcije:

```
DWORD GetThreadPriority(HANDLE hThread);
DWORD SetThreadPriority(HANDLE hThread, int nPriority);
```

Najčešće se dretvi koja obavlja ulazno-izlazne operacije dodjeljuje prioritet THREAD_PRIORITY_ABOVE_NORMAL ili THREAD_PRIORITY_HIGHEST da bi se osigurao dobar odziv aplikacije. Pozadinske dretve koje više koriste procesor se stavljaju na THREAD_PRIORITY_BELOW_NORMAL ili THREAD_PRIORITY_LOWEST kako bi mogle biti prekinute kada je potrebno (npr. ulaz od korisnika).

3.4. Višeprocesorski sustavi

Računala s više procesora su dizajnirana za jednu od dvije arhitekture: neuniformni pristup memoriji (*engl.* non-uniform memory access, NUMA) ili simetrični multiprocesorski rad (*engl.* symmetric multiprocessing, SMP). U NUMA arhitekturi svaki je procesor bliže jednom dijelu memorije nego ostalima. Dretve se dodjeljuju procesima koji su blizu memoriji koja se koristi. Procesori u SMP arhitekturi dijele isti memorijski prostor. Bilo koja dretva može biti dodijeljena nekom procesoru. Raspoređivanje dretvi u SMP arhitekturi je praktično isto kao i kod jednoprocesorskih računala.

3.4.1. Afinitet dretvi i procesa

Afinitet dretvi određuje na kojem se podskupu procesora one mogu pokretati. U praksi se izbjegava postavljanje afiniteta dretvi jer sam raspoređivač raspoređuje dretve po procesorima. Sustav sadrži masku procesorskog afiniteta. Svaki bit u 32-bitnoj maski određuje jedan procesor.

Dobivanje i postavljanje maske postiže se funkcijama:

```
BOOL GetProcessAffinityMask(HANDLE hProcess,
                            PDWORD_PTR lpProcessAffinityMask,
                            PDWORD_PTR lpSystemAffinityMask);
BOOL SetProcessAffinityMask(HANDLE hProcess,
                            DWORD_PTR dwProcessAffinityMask);
BOOL SetThreadAffinityMask(HANDLE hThread,
                            DWORD_PTR dwThreadAffinityMask);
```

Također, afinititeti procesa i dretvi se mogu postaviti i u *Task Manager*-u. Ako se neki proces izvodi na jednom procesoru u višeprocesorskom sustavu i prekinut je od raspoređivača zadataka, neki dijelovi procesa mogu ostati u priručnoj memoriji (*engl.* cache). Sljedeći puta kad raspoređivač dodijeli procesorsko vrijeme tom procesu, najbolje je da ga dodijeli tom procesoru jer će izvođenje biti efikasnije nego da je dodijeljen nekom drugom procesoru.

Postavljanjem afiniteta postiže se bolja iskoristivost priručne memroije, ali može se dogoditi da se loše odrede afiniteti procesa te da se na jednom procesoru izvodi mnogo više procesa nego na ostalim procesorima.

4. Ispitivanje rada raspoređivača u Win32 okruženju

Već je spomenuto da Windows XP operacijski sustav koristi višerazinski povratni red čekanja. U sljedećim će se primjerima ispitivati karakteristike raspoređivača. Cilj je provjeriti način rada raspoređivača. Provjeravat će se kako raspoređivač dodjeljuje procesorsko vrijeme dretvama i procesima različitih prioriteta, kako funkcionira inverzija prioriteta te sama brzina promjene konteksta procesa i dretvi.

U Windowsima XP također je moguće odrediti način rada raspoređivača. To je moguće uraditi u opcijama performansi (*engl.* Performance Options). Postoje dvije opcije koje možemo odabrati – programi i pozadinski servisi. Windowsi su tako podešeni da su podrazumijevana opcija programi zbog toga jer to omogućuje korisniku najjednostavniji i najbrži rad. Druga opcija dodjeljuje više procesorskog vremena pozadinskim dretvama i preporuča se koristiti u slučaju da se na računalu izvršava neki server.

Sva su mjerenja provedena na procesoru Intel Core 2 Duo 3.00 GHz.

4.1. Raspoređivanje procesa

Program koji računa omjere dobivenog procesorskog vremena za određene procese zamišljen je tako da korisnik upiše broj procesa. Zatim treba odrediti vrijeme u sekundama za koje će se ispitivati. Potom korisnik unosi željene prioritete za svaki proces te program kreće s radom. Korisnik ovdje ne može unijeti REAL_TIME_PRIORITY_CLASS zbog toga jer previše opterećuje sustav te je moguće da sve zablokira. Glavni program kreira procese zadanih prioriteta. Procesi čekaju na semafor koji je kreiran u glavnom programu s ciljem da svi procesi počnu istodobno s izvršavanjem. Nakon kreiranja svih procesa, glavni program oslobodi semafor te procesi kreću s radom. Svaki proces ima beskonačnu petlju u kojoj se izvodi funkcija `Racunaj()` i povećava brojač. Funkcija `Racunaj()` „troši“ procesorsko vrijeme tako da pridružuje nasumične elemente dvodimenzionalnom polju te zatim još i zbraja, oduzima i množi neke elemente.

Svaki proces ima brojač inicijalno postavljen na nulu. Glavni program zaustavlja procese nakon zadanog vremena i šalje prekidni signal (SIGBREAK) svim procesima vezanim na njegovu konzolu. U procesima je taj signal maskiran tako da se prilikom tog signala ispiše brojač i završi proces. U brojaču je zapravo pohranjen broj prolaska procesa kroz petlju. Budući da su procesi zadani tako da imaju različit prioritet, vrijednosti brojača svakog procesa su omjeri procesorskog vremena dodijeljenog pojedinom procesu. Poželjno je da se program izvršava kada nema pokrenutih nekih pozadinskih aplikacija niti procesa visokog prioriteta. Svi su procesi ograničeni na korištenje samo jednog procesora (jezgre) u višeprocesorskom sustavu.

Svaki proces sadrži jednu dretvu kojoj je automatski pridijeljen prioritet THREAD_PRIORITY_NORMAL na temelju kojeg su vršena mjerenja.

4.1.1. Rezultati ispitivanja programa

Mjerenja su vršena na 2, 3, 4 i 5 procesa istovremeno. Trajanje ispitivanja je bilo 60 sekundi i korišteno je pet prioriteta.

U tablici 1. prikazani su rezultati mjerenja na dva procesa. Uz klasu prioriteta i brojač za svaki proces naveden je još i osnovni prioritet.

Tablica 1. Raspoređivanje dva procesa s razinom prioriteta dretve THREAD_PRIORITY_NORMAL (svaki redak predstavlja rezultate jednog mjerenja)

Proces 1			**Proces 2**		
Prioritet	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**
HIGH	13	6932	ABOVE_NORMAL	10	266
NORMAL	8	6310	BELOW_NORMAL	6	222
ABOVE_NORMAL	10	6910	BELOW_NORMAL	6	246
HIGH	13	7133	IDLE	4	68
BELOW_NORMAL	6	6460	IDLE	4	58
ABOVE_NORMAL	10	6857	NORMAL	8	228
NORMAL	8	3237	NORMAL	8	3303

Iz tablice 1. vidljivo je da raspoređivač procesima nižeg prioriteta daje uvijek jednako procesorsko vrijeme bez obzira na razliku prioriteta. U slučaju kad postoje dva procesa od kojih je veći normalnog ili nižeg prioreta, tom procesu dodijeljeno je manje procesorskog vremena zbog procesa i servisa koji su nužni za rad operacijskog sustava.

Jedina se razlika javlja kad je procesu postavljen prioritet IDLE_PRIORITY_CLASS i u tom slučaju dobiva četiri puta manje procesorskog vremena nego ostali procesi.

U tablici 2. prikazani su rezultati mjerenja raspoređivanja na temelju prioriteta na tri procesa.

Tablica 2. Raspoređivanje tri procesa s razinom prioriteta dretve THREAD_PRIORITY_NORMAL

Proces 1			**Proces 2**			**Proces 3**		
Prioritet	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**
HIGH	13	6682	ABOVE_ NORMAL	10	225	NORMAL	8	201
ABOVE_ NORMAL	10	6686	NORMAL	8	219	BELOW_ NORMAL	6	243
HIGH	13	6859	NORMAL	8	246	IDLE	4	50
ABOVE_ NORMAL	10	6842	BELOW_ NORMAL	6	208	IDLE	4	59
ABOVE_ NORMAL	10	3470	ABOVE_ NORMAL	10	3470	BELOW_ NORMAL	6	241
ABOVE_ NORMAL	10	6700	BELOW_ NORMAL	6	205	BELOW_ NORMAL	6	232

Proces najvišeg prioriteta dobiva najviše vremena, dok drugi procesi dobivaju jednako procesorsko vrijeme neovisno o prioritetu. Opet su iznimka procesi prioriteta IDLE_PRIORITY_CLASS koji dobivaju četiri puta manje vremena.

U slučaju sa četiri i pet pokrenutih procesa proces najvišeg prioriteta također dobiva gotovo svo procesorsko vrijeme, a procesi nižeg prioriteta isto vrijeme neovisno o klasi prioriteta (osim procesi s najnižim prioritetom).

Raspoređivač pokreće proces najvišeg prioriteta i nakon nekog vremena pušta sve procese koji su u redu čekanja neovisno o prioritetu. Svakom procesu daje nekoliko vremenskih odsječaka i nastavlja dalje. Dok su svi procesi izvršeni, vraća se početni proces te se neko dulje vrijeme dodjeljuje samo tom procesu. Isto tako, nije bitno koliko je procesa u redu čekanja, niti koji je njihov prioritet jer će svi dobiti istu količinu vremena, tj. nekoliko odsječaka. Zbog toga je u mjerenju na dva procesa svaki proces nižeg prioriteta dobio isto vrijeme neovisno o svom prioritetu, a u mjerenju s 3 procesa su svi procesi nižeg prioriteta dobili isto vrijeme. Jedina je iznimka u tome najniži prioritet. Raspoređivač provjerava svaki četvrti put da li ima procesa u redu čekanja tog prioriteta i onda im daje određeno procesorsko vrijeme.

Ova mjerenja vršena su s opcijom raspoređivanja postavljenom na programe, a u tablici 3. su prikazani rezultati mjerenja na tri procesa i opcijom postavljenom na pozadinske servise.

Tablica 3. raspoređivanje tri procesa s razinom prioriteta dretve THREAD_PRIORITY_NORMAL (opcija pozadinski servisi)

Proces 1			Proces 2			Proces 3		
Prioritet	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**
HIGH	13	6280	ABOVE_ NORMAL	10	402	NORMAL	8	404
ABOVE_ NORMAL	10	6329	NORMAL	8	400	BELOW_ NORMAL	6	433
HIGH	13	6716	NORMAL	8	389	IDLE	4	56
ABOVE_ NORMAL	10	6670	BELOW_ NORMAL	6	444	IDLE	4	63
ABOVE_ NORMAL	10	3365	ABOVE_ NORMAL	10	3356	BELOW_ NORMAL	6	441
ABOVE_ NORMAL	10	6343	BELOW_ NORMAL	6	441	BELOW_ NORMAL	6	451

Raspoređivač u ovom slučaju više procesorskog vremena daje procesima nižeg prioriteta (kako i samo ime opcije govori). Uspoređujući tablicu 2. i tablicu 3. može se zaključiti da raspoređivač daje duplo više procesorskog vremena procesima nižeg prioriteta. Opet je najniži prioritet ovdje iznimka jer dobiva isto vrijeme bez obzira na način rada raspoređivača.

4.2. Raspoređivanje dretvi

Ovaj program radi na istom principu kao i program kojim se provjerava raspoređivanje procesa. Unosi se broj dretvi koje želimo stvoriti te njihovi prioriteti. Postoji sedam prioriteta dretvi, no nije moguć unos prioriteta THREAD_PRIORITY_TIME_CRITICAL zbog mogućnosti da računalo stane. Glavni program kreira dretve i čeka zadano vrijeme. Dretve su u beskonačnoj petlji i izvršavaju funkciju `Racunaj()` i povećavaju svoj brojač (u globalnoj memoriji).

Kada istekne vrijeme, glavni program terminira dretve te ispisuje sadržaje brojača. Izvršavanje dretvi ograničeno je na samo jedan procesor.

Sve dretve imaju svoj pripadajući proces. Njegov prioritet postavljen je automatski na NORMAL_PRIORITY_CLASS.

4.2.1. Rezultati ispitivanja programa

Mjerenja su vršena na dvije do šest dretvi. Upotrebljeno je šest prioriteta na vremenu od 60 sekundi.

U tablici 4. prikazani su rezultati mjerenja sa dvije dretve.

Tablica 4. Raspoređivanje dvije dretve čiji je proces prioriteta NORMAL_PRIORITY_CLASS

Dretva 1			**Dretva 2**		
Prioritet	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**
HIGHEST	10	6382	ABOVE_NORMAL	9	236
NORMAL	8	6344	BELOW_NORMAL	7	236
HIGHEST	10	6369	NORMAL	8	246
ABOVE_NORMAL	9	6348	BELOW_NORMAL	7	226
HIGHEST	10	6397	LOWEST	6	226
ABOVE_NORMAL	9	6339	IDLE	1	246
BELOW_NORMAL	7	6344	IDLE	1	228

Vidljivo je da je raspoređivanje slično kao i kod procesa. Dretva višeg prioriteta uvijek dobiva više procesorskog vremena u istom omjeru naspram dretve nižeg prioriteta neovisno o tom nižem prioritetu. Za razliku od raspoređivanja procesa raspoređivač ne daje manje vremena dretvama s THREAD_PRIORITY_IDLE prioritetom obzirom na ostale dretve nižeg prioriteta iako je osnovni prioritet tih dretvi manji nego u slučaju kada neki proces ima IDLE_PRIORITY_CLASS. Iz toga se zaključuje da raspoređivač raspoređuje dretve prvo na temelju klase prioriteta, a tek onda gleda razinu prioriteta.

Mjerenje na tri dretve prikazano je u tablici 5.

Tablica 5. raspoređivanje tri dretve čiji je proces prioriteta NORMAL_PRIORITY_CLASS

Dretva 1			Dretva 2			Dretva 3		
Prioritet	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**
HIGHEST	10	6142	ABOVE_ NORMAL	9	235	NORMAL	8	247
ABOVE_ NORMAL	9	6143	NORMAL	8	226	BELOW_ NORMAL	7	207
HIGHEST	10	6192	NORMAL	8	226	LOWEST	6	207
ABOVE_ NORMAL	9	6132	BELOW_ NORMAL	7	226	IDLE	1	227
ABOVE_ NORMAL	9	3195	ABOVE_ NORMAL	9	3159	BELOW_ NORMAL	7	226
HIGHEST	10	6193	BELOW_ NORMAL	7	225	BELOW_ NORMAL	7	207

Iz rezultata mjerenja vidljivo je da raspoređivač s dretvama radi na isti način kao i s procesima. Izvodi se dretva najvišeg prioriteta koja se nakon nekog vremena prekine i daje nekoliko vremenskih odsječaka svakoj dretvi nižeg prioreta. Također, kod dretvi se isto vrijeme daje svim nižim prioritetima pa i najnižem prioritetu.

Raspoređivanje na četiri ili više dretvi daje iste rezultate. Dretva najvišeg prioriteta dobiva najviše vremena, dok ostale dobivaju jednako procesorsko vrijeme.

U tablici 6. prikazana su mjerenja vršena na temelju tri dretve i s opcijom raspoređivanja postavljenom na pozadinske servise.

Tablica 6. raspoređivanje tri dretve čiji je proces prioriteta NORMAL_PRIORITY_CLASS (opcija pozadinski servisi)

Dretva 1			Dretva 2			Dretva 3		
Prioritet	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**	**Prioritet**	**OP**	**Brojač**
HIGHEST	10	6299	ABOVE_ NORMAL	9	433	NORMAL	8	412
ABOVE_ NORMAL	9	6249	NORMAL	8	433	BELOW_ NORMAL	7	412
HIGHEST	10	6320	NORMAL	8	414	LOWEST	6	412
ABOVE_ NORMAL	9	6364	BELOW_ NORMAL	7	412	IDLE	1	372
ABOVE_ NORMAL	9	3120	ABOVE_ NORMAL	9	3123	BELOW_ NORMAL	7	412
ABOVE_ NORMAL	9	5814	BELOW_ NORMAL	7	414	BELOW_ NORMAL	7	412

S uključenom opcijom raspoređivača na pozadinske servise raspoređivanje dretvi se izvršava slično kao i kod procesa. Usporedbom tablice 5. i tablice 6. uočava se da svaka dretva nižeg prioriteta dobiva duplo više procesorskog vremena (pa i dretva s THREAD_PRIORITY_IDLE prioritetom).

4.3. Inverzija prioriteta

Inverzija prioriteta je scenarij u kojem zadatak niskog prioriteta zadržava resurs koji je potreban zadatku visokog prioriteta. Proces ili dretva visokog prioriteta je blokiran. Općenito, scenarij u kojem neki proces ili dretva ne može doći do resursa zove se izgladnjivanje (*engl.* starvation). U sustavu se još nalaze i zadaci srednjeg prioriteta koji onemogućavaju da zadatak niskog prioriteta oslobodi resurs za zadatak visokog prioriteta. U tom slučaju raspoređivač zamijeni prioritete procesa ili dretvi tako da se oslobode resursi koji su potrebni zadatku s visokom prioritetom.

Program je napravljen tako da se može odrediti broj dretvi srednjeg prioriteta (to su dretve koje će „smetati“ dretvi nižeg prioriteta u izlasku iz kritičnog odsječka). Postoji još i dretva niskog prioriteta i dretva visokog prioriteta. Dretve srednjeg i visokog prioriteta su na početku zablokirane na semaforu kojeg kontrolira dretva niskog prioriteta. Dretva niskog prioriteta je na početku zablokirana sa semaforu koji joj omogućuje ulazak u kritični odsječak i na kojem čeka dretva visokog prioriteta.

Pseudokod dretve niskog prioriteta:

```
DretvaNP {
	while (1) {
		Racunaj();
		Cekaj(sem1);
		Racunaj();
		Oslobodi_sve(ulaz);
		Oslobodi(sem);
	}
}
```

Pseudokod dretve visokog prioriteta:

```
DretvaVP {
	while (1) {
		Cekaj(ulaz);
		Racunaj();
		RacunajVrijeme();
		Cekaj(sem);
		ZaustaviVrijeme();
		Racunaj();
		Oslobodi(sem);
	}
}
```

Semafor `ulaz` odblokira dretve srednjeg i visokog prioriteta. Čim se oslobodi semafor, sve dretve izlaze iz reda čekanja i dretva niskog prioriteta se više ne izvodi. Dretva visokog prioriteta se nakon nekog vremena zablokira jer želi ući u kritični odsječak kontroliran semaforom `sem`, ali taj semafor još nije oslobođen od strane

dretve niskog prioriteta. Dretva postane blokirana i više se ne izvodi, ali zato su tu dretve srednjeg prioriteta koje se trenutačno izvode i ne daju dretvi niskog prioriteta da otpusti semafor `sem`.

Pseudokod dretve srednjeg prioriteta:

```
DretvaSP {
      while (1) {
            Cekaj(ulaz);
            Racunaj();
      }
}
```

Kod dretve srednjeg prioriteta bitno je da funkcija `Racunaj()` traje više vremena nego što je potrebno raspoređivaču da preda procesorsko vrijeme drugim dretvama. Primjerice, ako raspoređivač ide mijenjati prioritete nakon 10 sekundi, a funkcija `Racunaj()` traje 5 sekundi, dretva srednjeg prioriteta bit će blokirana na semaforu `ulaz`. Jedina dretva koja nije blokirana je dretva niskog prioriteta i ona se izvršava. Oslobodit će se semafor `sem` i dretva visokog prioriteta će se moći izvršavati, a rezultat će biti krivi (5 umjesto 10 sekundi).

4.3.1. Rezultati ispitivanja programa

U programu je moguće upisati broj dretvi srednjeg prioriteta. U tablici 7. prikazani su rezultati mjerenja u ovisnosti o broju dretvi srednjeg prioriteta. Vrijeme u tablici označava koliko je vremena prošlo od trenutka kada je dretva visokog prioriteta postala blokirana na semaforu pa do trenutka kad je semafor oslobođen i dretva je ušla u kritični odsječak. Prikazana su tri izmjerena vremena.

Tablica 7. Vrijeme potrebno za inverziju prioriteta

Broj dretvi	Vrijeme 1 [s]	Vrijeme 2 [s]	Vrijeme 3 [s]
1	4.671	4.998	5.353
2	4.866	4.798	4.785
3	4.776	4.922	5.153
4	4.794	4.672	5.020
5	4.917	4.797	5.009
10	3.831	3.964	4.796
15	3.841	4.801	3.541

Iz rezultata prikazanih u tablici može se zaključiti da dodjeljivanje procesorskog vremena niskoj dretvi koja drži resurs ne ovisi o broju dretvi srednjeg prioriteta. U dokumentaciji o raspoređivaču u Win32 okruženju se govori da je problem inverzije prioriteta riješen tako da se prioritet podiže nasumičnim dretvama prioriteta nižeg od blokirane dretve. To je u načelu točno jer se dretve pokreću nasumično, ali se uvijek pokrenu sve dretve nižeg prioriteta neovisno o tome koliko ih ima tako da je vrijeme potrebno za inverziju prioriteta dretvi nižeg prioriteta uvijek približno isto i to oko 5 sekundi.

4.4. Vrijeme promjene konteksta procesa i dretvi

Promjena konteksta nezaobilazna je u višezadaćnim sustavima. Kako raspoređivač daje procesorsko vrijeme procesima i dretvama, mijenja se stanje u samom procesoru (registri, kazaljka stoga, zastavice itd.). Budući da raspoređivač može nekom procesu ili dretvi uzeti procesorsko vrijeme u bilo kojem trenutku, kontekst se mora spremiti da bi se taj proces ili dretva kasnije mogli izvršavati bez pogrešaka. Brzina promjene konteksta vrlo je bitna jer brže promjene omogućavaju efikasnije raspoređivanje i bolje iskorištenje procesora.

4.4.1. Promjena konteksta dretvi

Promjena konteksta dretvi kraća je od promjene konteksta procesa. Dretve dijele neke resurse (npr. globalne varijable, memorijski prostor).

Izračunavanje vremena promjene konteksta dretvi u programu je implementirano tako da postoje dvije dretve u beskonačnoj petlji koje se naizmjenično izvode.

Pseudokod prve dretve:

```
Dretva1 {
      Cekaj(sem1);
      brojPromjeneKonteksta++;
      Cekaj(sem2);
      While (1) {
            Cekaj(sem1);
            brojPromjeneKonteksta++;
            Oslobodi(sem2);
      }
}
```

Pseudokod druge dretve:

```
Dretva2 {
      While (1) {
            Cekaj(sem2);
            brojPromjeneKonteksta++;
            Oslobodi(sem2);
      }
}
```

Dretve sadrže dva semafora. `sem1` je u glavnoj dretvi postavljen na prohodnog tako da prva dretva može ući u kritični odsječak, povećati brojač i osloboditi semafor `sem2` koji omogućuje izvođenje druge dretve koja onda oslobađa prvu dretvu itd. Vidljivo je da se u jednom trenutku može izvoditi samo jedna dretva i da se pri oslobađanju semafora događa promjena konteksta dretvi. Zato postoji

brojač kojeg pri svakom prolazu kroz petlju povećavamo i na kraju nam je u njemu pohranjen broj promjena konteksta dretve.

Glavna dretva potom kreira prvu i drugu dretvu te oslobađa semafor `sem1`. Nakon toga počinje štopati vrijeme i čeka 20 sekundi. Nakon tog vremena terminira sve dretve i izračuna ukupno vrijeme. Uzima se broj pohranjen u brojaču kojeg su povećavale obje dretve. Vrijeme provedeno na jednu iteraciju u petlji računa se prema:

$$t_{cs} = \frac{T_{uk}}{n}, \tag{1}$$

gdje je n broj pohranjen u brojaču, a T_{uk} vrijeme izvršavanja

Ta iteracija je zapravo vrijeme promjene konteksta. Povećavanje brojača za jedan možemo zanemariti budući da je vrijeme potrebno za izvođenje te naredbe mnogo kraće nego vrijeme potrebno za promjenu konteksta.

4.4.2. Promjena konteksta procesa

Promjena konteksta procesa drugačija je od promjene konteksta dretvi. Kontekst procesa je veći od konteksta dretve pa je i vrijeme promjene duže.

Izračunavanje konteksta napravljeno je slično kao i kod dretvi. Glavni proces ovdje stvara semafore i dva procesa. Procesi na isti način unakrsno oslobađaju i zauzimaju semafore i svaki proces ima svoj brojač. Zbog toga što svaki proces ima svoj brojač taj brojač ovdje nije jednak broju promjene konteksta tih dvaju procesa. Za svako povećanje brojača dogode se dvije promjene konteksta. Prva je kad jedan proces oslobodi semafor, a druga kad zauzme drugi semafor.

Na početku izvođenja svaki proces otvara semafore (budući da procesi ne dijele memoriju s glavnim procesom). Jedan semafor je inicijalno postavljen tako da je prolazan, dok je drugi neprolazan. Procesi sadrže jednu beskonačnu petlju koja je identična onoj kod dretvi. Prije te petlje počinje računanje vremena. Glavni proces generira signal SIGBREAK nakon zadanog vremena. Procesi imaju zamaskiran taj signal, a funkcija koja se poziva nakon prekida izazvanog tim signalom zaustavlja štopericu, uzima brojač procesa i ispisuje prosječno vrijeme potrebno za jednu

iteraciju u petlji. Dovoljno je da samo jedan proces ispiše te vrijednosti, dok drugi samo izlazi prilikom prekida.

Vrijeme jedne iteracije računa se slično kao i kod dretvi:

$$t_{it} = \frac{T_{uk}}{2n}, \tag{2}$$

gdje n vrijednost brojača, a T_{uk} ukupno vrijeme kroz koje se petlja vrtjela. Uzima se dvostruka vrijednost brojača zbog toga jer svaki proces ima svoj brojač pa je broj iteracija za neki vremenski odsječak zapravo dvostruko veći.

4.4.3. Rezultati ispitivanja programa

U tablici 8. prikazani su rezultati dobiveni testiranjem dretvi. U tablici se nalazi broj dretvi na kojem je testirano, vrijeme testiranja i prosječno vrijeme promjene konteksta koje je dobiveno na temelju 10 ispitivanja.

Iz tablice je vidljivo da broj dretvi u sustavu utječe na vrijeme promjene konteksta. Više dretvi znači i veće opterećenje priručne memorije procesora pa se kontekst nekih dretvi pohranjuje u memoriju ili priručnu memoriju više razine. Na korištenom procesoru Intel Core 2 Duo E8400 3.00 GHz postoji priručna memorija prve razine od 128 kB koja se dijeli na dvije jezgre (svakoj po 64 kB). Od tih 64 kB pola otpada na kod, a pola na podatke. Dakle, mjesta za spremanje konteksta ima 32 kB. Postoji i druga razina priručne memorije koja ima 6 MB i brža je od radne memorije, ali sporija od prve razine. Ako svi konteksti ne stanu u priručnu memoriju prve razine, spremaju se ovdje. Zbog toga je uočljivo da je za promjenu konteksta potrebno više vremena kad je prisutno više dretvi u sustavu.

Ovi rezultati dobiveni su tako da je svaka dretva ograničena na korištenje iste jezgre procesora. U slučaju kada jednom dijelu dretvi dodijelimo jednu jezgru, a drugoj polovici drugu jezgru dretve će se izmjenjivati tako da raspoređivač prvo oslobađa dretve jedne jezgre koje su blokirane na semaforu, a tek onda dretve druge jezgre. No, u slučaju s dvije dretve kojima je dodijeljena različita jezgra koriste se naizmjenično prva i druga jezgra. To znači da se koristi priručna memorija druge

razine. Za taj slučaj vrijeme promjene konteksta je 1.71 μs. U slučaju kada raspoređivaču prepustimo izbor procesora vrijeme promjene konteksta je 1.56 μs.

Tablica 8. Vremena promjene konteksta dretve

Broj dretvi	Vrijeme izvršavanja programa [s]	Vrijeme promjene konteksta [μs]
2	1	0.754
2	2	0.750
2	5	0.733
4	1	0.758
4	2	0.736
4	5	0.735
16	1	0.817
16	2	0.781
16	5	0.769
64	1	0.803
64	5	0.793
64	10	0.790
128	1	0.823
128	10	0.799
256	1	0.821
256	10	0.803

Kako je korišten procesor brzine 3 GHz, jedan takt na tom procesoru traje približno $3.33 * 10^{-10}$ s. Pretpostavit ćemo da promjena konteksta dretve traje približno $0.8 * 10^{-6}$ s. Da bi izračunali koliko je procesorskih taktova potrebno za promjenu konteksta dretve treba koristiti sljedeću formulu:

$$N = \frac{t_{cs}}{T_{proc}} \ , \tag{3}$$

gdje je T_{proc} vrijeme potrebno za jedan takt, a t_{cs} vrijeme promjene konteksta.

Na temelju jednadžbe (3) dobiva se da je za promjenu konteksta dretve potrebno malo više od 2100 ciklusa procesora.

U tablici 9. prikazani su rezultati dobiveni ispitivanjem vremena promjene konteksta kod dva procesa.

Tablica 9. Vremena promjene konteksta procesa

Broj procesa	Vrijeme izvršavanja programa [s]	Vrijeme promjene konteksta [µs]
2	1	1.098
2	2	1.090
2	5	1.063

Ispitivanje je vršeno na dva procesa zbog toga jer je potrebno kreirati onoliko semafora koliko ima procesa da bi oni jedan drugom mogli dodjeljivati kritične odsječke. Za slučaj kada svi procesi imaju isti semafor kojeg međusobno puštaju može se dogoditi da jedan proces uđe dva i više puta u kritični odsječak nakon što sam sebi oslobodi semafor.

Za promjenu konteksta procesa potrebno je više vremena nego za dretve što je i logično jer je kontekst procesa opsežniji. Potrebno je otprilike 40% više vremena nego u slučaju promjene konteksta dretvi.

4.5. Ispitivanje duljine vremenskih odsječaka

Raspoređivač zadataka brine se i o vremenskim odsječcima koji će biti dodijeljeni određenoj dretvi. Ti odsječci su većinom reda veličine 10 milisekundi, a kod prioriteta u realnom vremenu i više od 100 milisekundi. Ispitivanje je osmišljeno tako da u jednom procesu možemo zadati proizvoljan broj dretvi prioriteta THREAD_PRIORITY_LOWEST, THREAD_PRIORITY_BELOW_NORMAL, THREAD_PRIORITY_NORMAL i THREAD_PRIORITY_ABOVE_NORMAL. Osim tih dretvi, stvara se još jedna dretva koja je prioriteta THREAD_PRIORITY_HIGHEST koja je u beskonačnoj petlji i troši procesorsko vrijeme. Ostale dretve su također u beskonačnoj petlji, ali svaka računa vrijeme potrebno za jednu iteraciju i piše to u tekstualni datoteku.

Pseudokod dretvi:

```
Dretva {
      PostaviPrioritet();
      While (1) {
            MjeriVrijeme();
            Racunaj(broj_iteracija);
            ZaustaviVrijeme();
            PisiUDatoteku(vrijeme);
      }
}
```

Prije nego program počinje kreirati dretve računa se vrijeme potrebno za funkciju `Racunaj()` za određeni broj iteracija. Broj iteracija je implicitno postavljen na 10, a računalu treba oko 3.52 ms za izračun te funkcije. Program funkcionira tako da se ispisuje identifikator dretvi te vrijeme koje prođe od `MjeriVrijeme()` pa do `ZaustaviVrijeme()`. Funkcija `Racunaj()` troši najviše vremena u iteraciji pa je procesorsko vrijeme potrošeno na računanje vremena i pisanje u datoteku zanemarivo. Zbog toga se pretpostavlja da raspoređivač predaje vremenski odsječak drugoj dretvi u trenutku kad se izvodi funkcija `Racunaj()`. Ako neka dretva u datoteku ispiše vrijednost približnu onoj koja treba da se izvrši funkcija `Racunaj()`, znači da dretva nije bila prekinuta od raspoređivača. U slučaju mnogo većeg vremena to znači da je dretva bila prekinuta dok se izvršavala funkcija `Racunaj()`. Da bi se odredio vremenski odsječak koji raspoređivač daje dretvama za svaku se dretvu gledaju vremena koja su ispisana. Trajanje vremenskog odsječka je zbroj svih vremena nakon što je dretva dobila vremenski odsječak pa sve dok je neka druga dretva nije prekinula.

Za funkciju `Racunaj()` može se zadati i veći broj iteracija, ali je onda računanje trajanja odsječka nepreciznije. Preciznost je ovisna o trajanju funkcije `Racunaj()`, ali treba pripaziti da funkcija traje puno duže od računanja vremena i pisanja u datoteku jer to također uzrokuje nepreciznost u računanju trajanja odsječka (veća je vjerojatnost da raspoređivač u tom slučaju preda vremenski odsječak drugoj dretvi kada se ne izvršava funkcija `Racunaj()`).

4.5.1. Rezultati ispitivanja programa

Mjerenja su provedena pod opterećenim sustavom, slabo opterećenim sustavom i jako opterećenim sustavom. Rezultati su prikazani u tablici 10. Testiranje se vršilo sa jednom dretvom iz svakog prioriteta koji je moguće izabrati. Rezultati se odnose na vremenske odsječke koje raspoređivač daje dretvama koje nisu najvišeg prioriteta. Dretvi najvišeg prioriteta daje mnogo više vremena – približno 5 sekundi. U tablici su još i podaci o tome da li je dinamički prioritet dretvi podignut zbog toga jer je označen prozor u kojem se izvode dretve.

Tablica 10. Trajanje vremenskih odsječaka

Opterećenje sustava	Podizanje prioriteta	Vrijeme odsječka [ms]
Neopterećen (0%)	Ne	60
Neopterećen (0%)	Da	170
Slabo opterećen (5-10%)	Ne	25
Slabo opterećen (5-10%)	Da	90
Jako opterećen (>80%)	Ne	25
Jako opterećen (>80%)	Da	90

Iz podataka prikazanih u tablici vidi se da postoje različita vremena koja sustav daje dretvama. Trajanje vremenskog odsječka ovisno je o opterećenosti sustava. Ako je sustav opterećen trajanje je otprilike duplo kraće nego kod neopterećenog sustava. Kod dinamičkog podizanja prioriteta isti je slučaj – u neopterećenom sustavu dretva dobije dva puta više vremena nego u opterećenom. Raspoređivač dinamički podiže prioritet prozora koji je označen. No zanimljivo je da je pri prokretanju programa prozor označen automatski, ali u tom trenutku raspoređivač ne podiže osnovni prioritet dretve. Da bi se to postiglo, potrebno je označiti bilo koji drugi prozor i onda ponovno željeni prozor.

U slabo opterećenom sustavu izvršava se prikazivanje video odsječka kodiranog u MPEG-2 formatu. U slučaju bez dinamičkog podizanja prioriteta zadanih dretvi, označen je prozor programa koji prikazuje video sadržaj. U jako opterećenom sustavu pretvara se taj isti odsječak iz MPEG-2 formata u DiVX. Opterećenje procesora je redovito iznad 80%. U takvom je sustavu trajanje

vremenskih odsječaka jednako kao i u slabo opterećenom sustavu. Vremenski odsječci su jednaki za sve dretve osim one najvišeg prioriteta.

5. Zaključak

Raspoređivač zadataka vrlo je bitan element višezadaćnog operacijskog sustava. Ključan je za dodjeljivanje procesorskog vremena dretvama i procesima. U operacijskom sustavu Windows XP se kao algoritam raspoređivanja koristi višerazinski povratni red čekanja. Postoje dva načina na koji taj raspoređivač može raditi – tako da je usredotočen na programe ili na pozadinske programe (servise). Ta dva načina rada su u principu skoro ista, jedino što ovaj drugi daje nešto više procesorskog vremena procesima i dretvama nižeg prioriteta.

Raspoređivač može sam povećavati prioritet dretvi koje rade ulazno-izlazne operacije, koje su postavljene u prednji plan ili koje su dulje vrijeme u pripremnom stanju, ali su niskog prioriteta. Na taj je način riješena i inverzija prioriteta. Dretve visokog prioreta su blokirane, a dretva niskog prioriteta koja drži resurs je spriječena od dretvi srednjeg prioriteta. Dretvi niskog prioriteta se nakon nekog vremena poveća dinamički prioritet te ona oslobodi resurs.

Pokazano je kako raspoređivač dodjeljuje procesorsko vrijeme dretvama. Vrijeme se dodjeljuje na temelju klase prioriteta i razine prioriteta koje zajedno čine osnovni prioritet. Ispitivanja su pokazala kako se prvo u obzir uzima klasa prioriteta, a tek onda razina prioriteta neovisno o osnovnom prioritetu dretve.

Promjene konteksta odvijaju se veoma brzo kada se koristi priručna memorija. Raspoređivač sam odabire koje će procesore koristiti za procese u sustavu. Cilj je rasporediti procese tako da se minimizira vrijeme utrošeno na promjene konteksta, ali i omogući normalan rad procesa.

U Windows XP (i svim ostalim operacijskim sustavima baziranim na Windows NT) može se podesiti mnogo parametara – od prioriteta dretvi i procesa, uključivanja i isključivanja podizanja dinamičkog prioriteta pa sve do postavljanja procesora na kojima se smije izvoditi neka dretva ili proces.

Na temelju rezultata dobivenih u ovom radu može se zaključiti da će i loše organizirani programi više-manje raditi zbog karakteristika raspoređivača (dinamičko podizanje prioriteta, rješenje za inverziju prioriteta), no na programeru ostaje da pažljivo oblikuje sustav jer rješenja raspoređivača često nisu najefikasnija (kao npr.

kod inverzije prioriteta ili zagušenja sustava dretvama visokog prioriteta koje rade procesorski zahtjevan posao).

Literatura

1. Scheduling, 6.4.2009., *Scheduling,* http://msdn.microsoft.com/en-us/library/ms685096.aspx, 15.4.2009.
2. Silberschatz, A., Galvin, P.B., Gagne, G. Operating System Concepts. 7. izdanje. : John Wiley & Sons, 2004.

Sažetak

Naslov rada: Raspoređivanje procesa u Win32 okruženju

U radu su prikazane teoretske osnove i eksperimentalni primjeri na raspoređivaču zadataka u Win32 okruženju.

U teoretskom dijelu opisani su procesi i dretve koji su osnovne jedinice za raspoređivanje procesorskog vremena. Potreba za raspoređivačem zadataka ukazuje se u višezadaćnim sustavima koji nastoje više zadataka izvoditi paralelno pa su u radu ukratko prikazani različiti višezadaćni sustavi kao što su sustavi u realnom vremenu, sustavi s raspodjelom vremena i multiprogramski sustavi. Raspoređivači zadatka dijele se na tri vrste: dugoročni, srednjoročni i kratkoročni. Svaki raspoređivač mora implementirati neki algoritam raspoređivanja. Postoje različiti algoritmi različitih složenosti, a najčešći su O(1), višerazinski povratni red čekanja i potpuno pravedan raspoređivač. U radu su ukratko opisani najčešći algoritmi raspoređivanja.

Drugi dio rada bavi se radom raspoređivača u Win32 okruženju. Ukratko su objašnjene teoretske osnove raspoređivanja u tom okruženju na temelju dokumentacije, a kasnije su prikazani rezultati eksperimenata.

U teoretskom dijelu govori se o načinu rada raspoređivača u Win32 okruženju. Objašnjeni su pojmovi dinamičkog podizanja prioriteta i inverzije prioriteta te način na koji Windows-i dodjeljuju prioritete procesima i dretvama – klase prioriteta i razine prioriteta.

Eksperimentalni dio temelji se na programima koji ispituju način rada raspoređivača te je li u skladu s teoretskim dijelom. Ispitivanje je usmjereno na raspoređivanje procesa i dretve, promjene konteksta procesa i dretvi te inverzije prioriteta. Za svako ispitivanje prikazani su i objašnjeni dobiveni rezultati.

Ključne riječi: raspoređivač zadataka, Win32, procesi, dretve, promjena konteksta, vremenski odsječak

Summary

Title: Process scheduling in Win32 environment

This thesis presents theoretical basis and experimental examples regarding the task scheduler in Win32 environment.

The theoretical part covers processes and threads which are basic units in task scheduling. Task scheduler is needed in multitasking systems which have to execute parallel multiple tasks. This thesis contains a short introduction into real-time systems, multitasking systems and multiprogramming system. There are three types of task scheduler: long-term, mid-term and short-term scheduler. Every scheduler is based on a scheduling algorithm. There are different scheduling algorithms of different complexity. The most often used algorithms are O(1), multilevel feedback queue and completely fair scheduler. These algorithms are briefly explained.

The second part deals with the task scheduler in Win32 environment. Theoretical foundation of allocation in this environment on the basis of documentation is explained in short, and the results of the experiments are presented subsequently.

The theoretical part is focused on the way of work of task scheduler in Win32 environment. The concepts of dynamic priority boost and priority inversion are explained as well as the way in which Windows schedules priorities to processes and threads – priority classes and priority levels.

The experimental part is based on the programmes that examine the way of work of task scheduler and if it is in compliance with the theoretical part. The examination is directed towards the scheduling of processes and threads, changes of process and thread contexts and the priority inversion. Each examination contains and explains the results obtained.

Key words: task scheduler, Win32, process, thread, context switch, time slice