Diana Šantavec

Implementacija kružnog algoritma sa zadržavanjem u planeru procesa na MP/M II operativnom sistemu

U radu se opisuje implementacija modifikovanog kružnog algoritma koji procesima dodeljuje uzastopne kvante procesorskog vremena. Ideja projekta je da se implementira algoritam koji će omogućiti da se svi procesi izvršavaju pseudo-paralelno, ali i da prioritet procesa ima uticaj na njihovo sortiranje. Prvo je implementiran kružni algoritam (Round Robin), koji je omogućio da se svi procesi izvršavaju naizmenično, pri čemu su svi oni, bez obzira na prioritet, imali isto vreme i istu učestalost izvršavanja. Potom je implementiran predloženi algoritam koji je, u zavisnosti od prioriteta, procesu dodeljivao različit broj uzastopnih vremenskih kvanti. Predloženi algoritam je upoređen sa originalnim algoritmom i sa kružnim algoritmom. Prednost predloženog algoritma je što je ukupno izvršavanje nekog skupa procesâ kraće i, što se proces nižeg prioriteta izvršava pseudo-paralelno sa procesima višeg prioriteta. Međutim, za razliku od kružnog, kod predloženog algoritma prioritet procesa ima uticaj na vreme izvršavanja procesa. Loša osobina predloženog algoritma je slab odziv procesa visokog prioriteta. Razlog ovome je što se proces uvek stavlja na poslednje mesto u listi i što se zadržava uzastopno više puta na prvom mestu (što se nije dešavalo kod kružnog algoritma).

Uvod

U poslednje vreme nema puno istraživanja vezanih za planer procesa. Glavna tema istraživanja operativnih sistema većinom je zaštita (kako od spoljnih napada, tako i od samog korisnika), virtualizacija, rad u cloud-u i ekonomičnije korišćenje hardverskih resursa (Tanenbaum i Bos 2015: 172-173). Međutim, programi koji se danas pišu zahtevaju puno računarksih resursa, što može predstavljati problem, iako su računari brži i imaju više memorije.

U toku rada računara pokrene se više procesa, međutim, procesorsko jezgro može da izvršava samo jedan proces u datom trenutku. Iz tog razloga procesi moraju, ili u potpunosti završiti svoj rad, ili prepustiti procesor drugom procesu, kako bi korisnik imao utisak paralelnog izvršavanja procesa (Tanenbaum i Bos 2015: 86). Smenom procesa postiže se i bolja iskorišćenost procesora, pošto procesor ne mora da čeka proces dok isti izvršava neku ulazno/izlaznu operaciju (Marić 2017: 50). Kada se ostvari utisak paralelnog izvršavanja, onda se kaže da se procesi izvršavaju pseudo-paralelno, odnosno samo prividno paralelno.

Da bi se procesi mogli smenjivati, neophodno je zapamtiti trenutno stanje procesa u trenutku kada mu se oduzmu procesorski resursi, što se postiže uvođenjem kontrolnog bloka procesa.

Diana Šantavec (2001), Vrbas, učenica 3. razreda elektrotehničke škole „Mihajlo Pupin" u Novom Sadu

MENTOR: *Dragan Toroman*
Istraživačka stanica Petnica

Kontrolni blok procesa je struktura podataka koja pored trenutnog stanja procesa, sadrži i druge podatke o procesu, koji se razlikuju u različitim implementacijama operativnih sistema (prioritet, status, jedinstveni naziv (PID), naziv korisnika koji ga je pokrenuo, zauzeti ulazno-izlazni uređaji itd.) (Tanenbaum i Bos 2015: 94).

Kako kontrola prepuštanja procesorskih resursa iz samog programa otežava programiranje i dovodi do slabijih performansi operativnog sistema (gledajući sisteme koji se danas koriste), uveden je planer procesa (u daljem tekstu: planer). Planer je algoritam koji vodi računa o rasporedu i dužini izvršavanja procesa u operativnom sistemu. Kako postoje različite upotrebe operativnih sistema i različiti tipovi procesa (I/O bound i CPU bound procesi), postoje i različiti algoritmi za planere, koji obezbeđuju različite performanse operativnog sistema.

Istraživanje se zasniva na modifikaciji planera procesa MP/M II (2.0) operativnog sistema (akronim od MultiProgramming Monitor Control Program for Microprocessors). Objavljen je 1981. godine kao jedna od verzija naslednika CP/M operativnog sistema. Podržava do 8 terminala, i na svakom omogućava izvršavanje više procesa (jedan od prvih koji je to omogućio). Zahteva 48 kB RAM memorije (random access memory), od kojih sam sistem koristi 26 kB. Takođe podržava do 400 kB banked RAM memorije. Može da se pokreće na Intel 8080 i 8085 mikroprocesorima, kao i na Zilog-ovom Z80 mikroprocesoru (DRI 1982*b*: Foreword).

Originalni planer omogućava pseudo-paralelno izvršavanje samo procesa istog prioriteta. Usled toga dolazi do situacije da procesi manjeg prioriteta previše dugo čekaju na procesorske resurse, što se sa strane korisnika, može manifestovati kao „kočenje" sistema, pošto ne reaguje na korisničke komande. Iz navedenog razloga, ideja je bila da se omogući da se pseudo-paralelno izvršavaju procesi različitih prioriteta, a da se pri tom zadrži i uticaj prioriteta procesa na redosled i dužinu izvršavanja procesa i da kompleksnost algoritma ostane u okvirima hardvera iz 1980-ih godina. Dalji tekst opisuje princip rada originalnog planera procesa na MP/M II operativnom sistemu, predloženu i implementiranu izmenu istog, kao i rezultate koji su dobijeni.

Procesi i planer procesa u MP/M II operativnom sistemu

Procesi. Svaki proces je određen kontrolnim blokom procesa (u daljem tekstu: kontrolni blok) veličine 50 bajtova. Najbitniji podaci u kontrolnom bloku za ovo istraživanje su sledeći (DRI 1982*a*: 111-6):

1. PL (prva dva bajta) – adresa sledećeg kontrolnog bloka u jednostruko spregnutoj listi ili 0 ukoliko je u pitanju poslednji kontrolni blok.

2. STATUS (treći bajt) – opisuje stanje procesa (da li proces čeka na neki događaj, na procesorsko vreme, neki drugi sistemski resusrs, ili treba da se završi njegovo izvršavanje). Ukupno ima 14 vrednosti, a najbitniji statusi za ovaj projekat su:
   - 0 – proces je spreman da se pokrene (READY)
   - 9 – poziv funkcije INSERT PROCESS (funkcija koja sortira procese u okviru planera) (INTERRUPT)

3. PRIORITET (četvrti bajt) – kreće se od 0 do 255. Što je manja numerička vrednost prioriteta, to je sam prioritet procesa veći.

Proces se može nalaziti u jednom od tri osnovna stanja (DRI 1982*a*: 4):
- spreman – proces je spreman za izvršavanje i čeka procesorsko vreme
- aktivan – proces koji se trenutno izvršava na procesoru, ili proces čije je izvšavanje prekinuto, ali se još nije pokrenuo drugi proces
- na čekanju – proces sa ovim statusom je blokiran (ne može da se izvršava)

Organizacija procesa. Procesi koji imaju aktivan ili spreman status nalaze se sortirani u jednostruko spregnutoj listi (u daljem tekstu READY LIST). Ostali procesi (sa statusom „na čekanju") se nalaze u drugim listama u zavisnosti od razloga čekanja (DRI 1982*a*: 5; Comp.os.cpm 2019). Na adresu prvog bajta prvog kontrolnog bloka u READY LIST pokazuje RLR pokazivač. Lista je povezana PL pokazivačima, i na kraju liste se nalazi IDLE proces koji pokazuje na vrednost 0 (proces koji uvek postoji i njegovo izvršavanje nije nikada sprečeno, i iz tog razloga treba da se izvršava samo kada ne postoji ni jedan drugi proces u listi, kako se ne bi izvršavala ne-

potrebna HALT instrukcija) (DRI 1982*a*: 5; DRI 1982*b*: 43). Kada se proces ubacuje u listu, tj. njegov kontrolni blok, uvek se ubacuje odmah posle svih drugih procesa istog prioriteta. Na ovaj način procesi nižeg prioriteta nikada neće dobiti procesorsko vreme. Još jedan problem je ukoliko je dodavani proces najnižeg prioriteta, jer onda može doći do nepotrebnog pokretanja IDLE procesa (DRI 1981: linija 456).

Planer procesa. U planer se može ući na dva načina: na vremenski interapt procesora kroz PDISP ulaz (50 procesorskih tikova u seknudi (P112 2019)), ili iz nekih drugih delova samog jezgra (kernel-a) kroz DISPATCH ulaz (DRI 1982*b*: 35; Comp.os.cpm. 2019). Ukoliko se u planer uđe na PDISP ulaz, trenutno aktivnom procesu će se prvo dodeliti status 9 da bi se pozvala funkcija INSERT PROCESS (DRI 1981: linija 555).

Funkcija INSERT PROCESS kao parametre uzima koren liste i adresu kontrolnog bloka procesa koji se ubacuje u listu. Kada se uđe u ovu funkciju, u slobodan registarski par se upisuje adresa kontrolnog bloka koji je prvi u listi. Sledeći korak je provera da li je vrednost prioriteta ubačenog procesa veća od vrednosti prioriteta sledećeg procesa u listi (da li je vrednost prioriteta manja od vrednosti prioriteta sledećeg procesa). Ukoliko uslov nije ispunjen, u registrar u kome je bio koren liste, upisuje se adresa prvog kontrolnog bloka, a u registarski par u kome je bila adresa prvog kontrolnog bloka – adresa drugog kontrolnog bloka procesa u listi. Ovim postupkom se pomera kroz listu dok se uslov ne ispuni ili ne dođe do kraja liste. Ukoliko se uslov ispuni (vrednost prioriteta ubacivanog procesa je manja ili jednaka vrednosti prioriteta sledećeg procesa u listi), proces se ubacuje u listu menjanjem vrednosti PL pokazivača odgovarajućih kontrolnih blokova procesa. Ako se proces ubacuje na prvo mesto, pokazivač RLR će promeniti vrednost, tako da pokazuje na kontrolni blok koji se ubacuje. Ukoliko se proces ubacuje na poslednje mesto, vrednost PL pokazivača ubacivanog kontrolnog bloka će dobiti vrednost 0 (DRI 1981: linija 456).

Pre ulaska u planer, stanje procesa biva zapamćeno u kontrolnom bloku (DRI 1982*b*: 15, 17, 35; Comp.os.cpm 2019). Obrađivač prekida (interrupt handler) određuje razlog prekida. Ukoliko je uzrok procesorski tik, u planer će se ući kroz PDISP ulaz (i biće promenjen status procesa), a u svakom drugom slučaju kroz DISPATCH ulaz. Potom planer proverava status prekinutog procesa, i u slučaju da taj status ima vrednost 0, biće provereno da li se neki proces treba dodati u listu, i pokreće prvi proces na listi. U slučaju da status ima bilo koju drugu vrednost, prekinuti proces će biti izbačen iz liste i biće izvršena potrebna operacija, u zavisnosti od statusa. U slučaju statusa 9, pozvaće se funkcija INSERT PROCESS, i proveriti da li ima da se doda još neki proces u listu. Ovim postupkom je omogućeno da proces najvišeg prioriteta ne ostane na prvom mestu, ako u listi već postoji proces istog prioriteta, ali ukoliko se proces istog prioriteta tek dodaje, prekinuti proces će ostati na prvom mestu. Planer potom uzima iz liste READY LIST proces koji je na prvom mestu (koji ima najviši prioritet) i daje mu procesorsko vreme (DRI 1981: linije 545-1156). Proces se izvršava dok ne dođe do sledećeg procesorskog prekida, ili dok on sam ne izvrši prekid (sistemskim pozivom), nakon čega se planer ponovo poziva iz obrađivača prekida.

Implementacija algoritama

Kružni algoritam

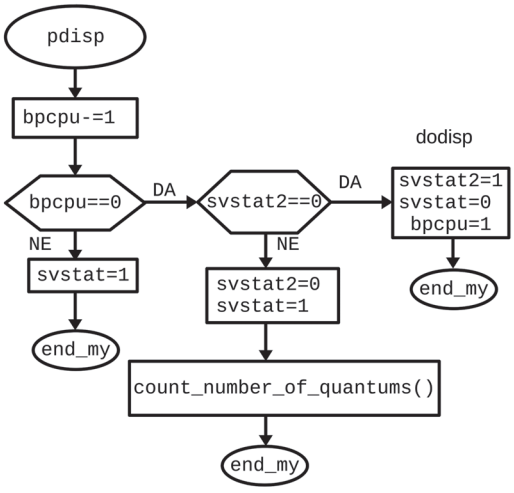
Prvo je urađena implementacija kružnog algoritma (Round Robin), da bi se sprečila mogućnost da proces nižeg prioriteta ostane dugo bez procesorskog vremena (resource starvation). Implementacija je ostvarena izmenom INSERT PROCESS funkcije, tako što je izbačena provera prioriteta procesa i promenjen uslov za ubacivanje procesa u listu. U originalnom algoritmu, konačni uslov da se proces ubaci u listu bio je da se ubaci na poslednje mesto, što se proveravalo očitavanjem vrednosti koja se nalazila na adresnom prostoru koji bi odgovarao PL pokazivaču sledećeg. Uslov je izmenjen tako da se proces ubaci pre poslednjeg kontrolnog bloka procesa u listi (ukoliko `*PL_next_process_descriptor == 0`) da se ne bi nikada ubacio posle IDLE procesa.

Predloženi algoritam

Implementacijom kružnog algoritma je postignuto da procesi različitih prioriteta dobijaju jednako procesorsko vreme i da ga dobijaju istom učestalošću. Iz ovog razloga je sledeći korak bio unapređenje navedene implementacije kružnog algoritma dodavanjem brojača kvanti procesorskog vremena (vreme koje proces dobije na procesoru između dva prekida) procesa i zadržavanjem procesa na prvom mestu u listi u zavisnosti od prioriteta. Izmene se nalaze na početku koda, gde se pre provere statusa procesa definišu i inicijalizuju promenljive u PDISP ulazu i, kasnije, u DISPATCH ulazu.

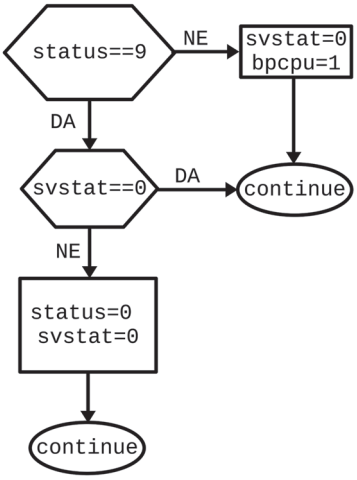
U delu koda u kojem se inicijalizuju promenljive dodate su još tri promenljive veličine jednog bajta: `bpcpu`, `svstat` i `svstat2`. Promenljiva `bpcpu` služi da se zapamti broj uzastopnih kvanti procesorskog vremena koje su dodeljene prvom procesu u listi, i inicijalizovana je na 1 (objašnjenje u daljem tekstu). Bajt `svstat` je inicijalizovan na 0, i ukoliko ima vrednost 1 treba da se preskoči premeštanje procesa sa prvog mesta u listi. Bajt `svstat2` je inicijalizovan na 0, i koristi se u kombinaciji sa `svstat` bajtom da se odredi da li proces treba biti pomeren sa prvog mesta, ili treba da se izvrši računanje broja uzastopnih kvanti koje treba da se dodele procesu.

U PDISP ulazu se prvo umanjuje broj uzastopnih kvanti procesorskog vremena koje su preostale (`bpcpu = bpcpu - 1`), i iz ovog razloga `bpcpu` mora biti inicijalizovan na 1. Sledeći korak je provera da li je proces potrošio sve dodeljene kvante procesorskog vremena, nakon čega se kôd deli na dva slučaja (slika 1). Ukoliko proces nije potrošio sve dodeljene kvante procesorskog vremena, `svstat` bajt uzima vrednost 1, kako bi se preskočilo premeštanje procesa sa prvog mesta u listi. Ukoliko je proces potrošio sve dodeljene uzastopne kvante procesorskog vremena, potrebno je proveriti vrednost `svstat2` bajta, kako bi proverili da li je prilikom prethodnog ulaska u planer novi proces postavljen na prvo mesto u listi. Ukoliko je vrednost ovog bajta 1, potrebno je izračunati broj uzastopnih kvanti za taj proces i postaviti vrednost `svstat` bajta na 1, kako bi se izbeglo pomeranje procesa sa prvog mesta u listi. Uko-



Slika 1. Provera brojača uzastopnih kvanti procesorskog vremena i postavljanje zastavica

Figure 1. Checking the counter for the successive quanta of processor time and setting the flags



Slika 2. Izmena statusa procesa u zavisnosti od vrednosti zastavica

Figure 2. Changing the status depending on the status of the flags

liko je vrednost `svstat2` bajta 0, potrebno je proces premestiti sa prvog mesta u listi, i postaviti vrednost ovog bajta na 1, da bi se prilikom sledećeg ulaska u planer izračunao broj uzastopnih kvanti za taj proces. Takođe je potrebno postaviti vrednost `bpcpu` bajta na 1, kako prilikom sledećeg ulaska u planer ovaj bajt ne bi dobio negativnu vrednost, tj. vrednost 255.

Ideja za računanje brojača uzastopnih kvanti koje proces treba da dobije jeste da se prioritet procesa podeli sa 16 (da bi se dobio opseg 0–15), a dobijena vrednost potom oduzme od 16, da bi procesi sa većim prioritetom (kojima odgovara manja vrednost brojača) dobili više puta uzastopno procesorsko vreme. Time je omogućeno da se procesi sa najvećim prioritetom izvršavaju šesnaest puta uzastopno, dok se procesi sa najnižim prioritetom mogu izvršiti samo jednom (qn = 16 – prioritet/16, gde je qn brojač uzastopnih kvanti).

Sledeći korak za implementaciju predloženog algoritma je u delu koda koji se izvršava pre skoka koji omogućava sprečavanje pomeranja procesa sa prvog mesta u listi i koji se izvršava samo ako je status procesa READY. Pre tog skoka bitno je proveriti da li status procesa glasi INTERRUPT (slika 2), zato što proces prilikom ulaska u planer, u slučaju redovnog procesorskog prekida, uvek dobija onu vrednost koja označava da isti treba biti pomeren sa prvog mesta. Ukoliko se u planer ušlo na bilo koji drugi način, status može biti INTERRUPT (proces sam traži da prepusti procesor drugim procesima), READY (proces treba da nastavi izvršavanje); u svim ostalim slučajevima proces treba biti prebačen u neku drugu listu.

Ukoliko je status procesa READY, potrebno je postaviti vrednost `svstat` bajta na 0 (jer proces treba da ostane na prvom mestu u listi), a vrednost `bpcpu` bajta na 1 (kako se prilikom sledećeg ulaska u planer ne bi dobilo 255 uzastopnih kvanti procesorskog vremena za proces koji je prvi u listi).

Ukoliko je status procesa INTERRUPT, potrebno je proveriti vrednost `svstat` bajta, i ukoliko je ona 1, potrebno je postaviti status procesa na READY (kako bi sada proces dobio status „ready to run”, i kako ne bi bio premešten sa prvog mesta u listi), a vrednost `svstat` bajta na 0, kako se sledeći put ne bi preskočilo pomeranje procesa sa prvog mesta u listi ukoliko to nije potrebno.

Dalje planer vrši premeštanje procesa sa prvog mesta u listi (ukoliko nije preskočeno), ubacuje nove kontrolne blokove u listu i vraća stanja procesa koji je prvi u listi, i nastavalja njegovo izvršavanje.

Rezultati

Maksimalan broj procesa u MP/M II operativnom sistemu je 16 (P112 2019), pa je iz tog razloga testiranje vršeno pomoću tri programa pisana u asembleru (Intel 8080 asembler): PROCES 1, PROCES 2 i PROCES 3. PROCES 1 i PROCES 2 su isti programi prioriteta 199 koji svoje izvršavanje prvo postavljaju u pozadinu, a potom 100 puta prolaze kroz 255^3 petlji i ispisuju broj 1 ili 2 (zavisno koji je program pokrenut), a na samom kraju ispisuje se karakter koji označava kraj izvršavanja. PROCES 3 je program prioriteta 255 koji svoje izvršavanje postavlja u pozadinu, i prolazi 25 puta kroz 255^3 petlji i ispisuje broj 3, dok na samom kraju ispisuje karakter koji označava kraj izvršavanja.

Test programi PROCES 1, PROCES 2 i PROCES 3 su pokrenuti redom na operativnom sistemu MP/M II upotrebom z80pack emulatora (Z80-SIM 2019) sa 2 sekunde razlike. Brzina procesora je bila ograničena na 60 MHz, a vreme izvršavanja procesa je mereno štopericom, usled problema sa ispisom vremena iz samog procesa. Merenje je vršeno više puta, ali kako su greške nastale usled merenja u okvirima od ±1s, u daljem tekstu se govori samo o srednjoj vrednosti rezultata.

Prilikom testiranja u originalnom i implementiranim sistemima konstatovan je bag koji se javlja ukoliko se pokrenu prvo procesi nižeg, pa tek onda procesi višeg prioriteta. U ovom slučaju, čim neki proces završi izvršavanje, prvi proces koji je posle njega na listi će takođe biti izvezan iz liste. Nije utvrđen uzrok baga, kao ni to da li se bag nalazi u operativnom sistemu ili u samom emulatoru.

Program PROCES 1 je pokrenut kao jedini proces (ne računajući IDLE proces) na originalnom i implementiranim planerima, i izvršavao se isto vreme. Ovim se došlo do zaključka da

samo izvršavanje kôda planera, bez premeštanja procesa, nema primetno drugačije vreme izvršavanja.

Razliku u ukupnom vremenu izvršavanja, kada je pokrenuto više procesa, možemo primetiti u zasenčenim ćelijama u tabeli (tabela 1). Pošto kružni algoritam nema računanje prioriteta i njihovo poređenje, ima neznatno kraće vreme izvršavanja od originalnog algoritma. Međutim, predloženi algoritam ima 4 s kraće vreme izvršavanja od originalnog, a 3 s kraće vreme od kružnog, jer ima smanjeno premeštanje kontrolnih blokova u listi.

Predloženi algoritam, za razliku od preostala dva, ne daje istu količinu procesorskog vremena procesima različitih prioriteta. Ovo je dokazano pokretanjem tri ista procesa kada se dobilo da se kraći proces (PROCES 3) izvršavao 12 min i 34 s, dok su se duži izvršavali 18 min i 52 s.

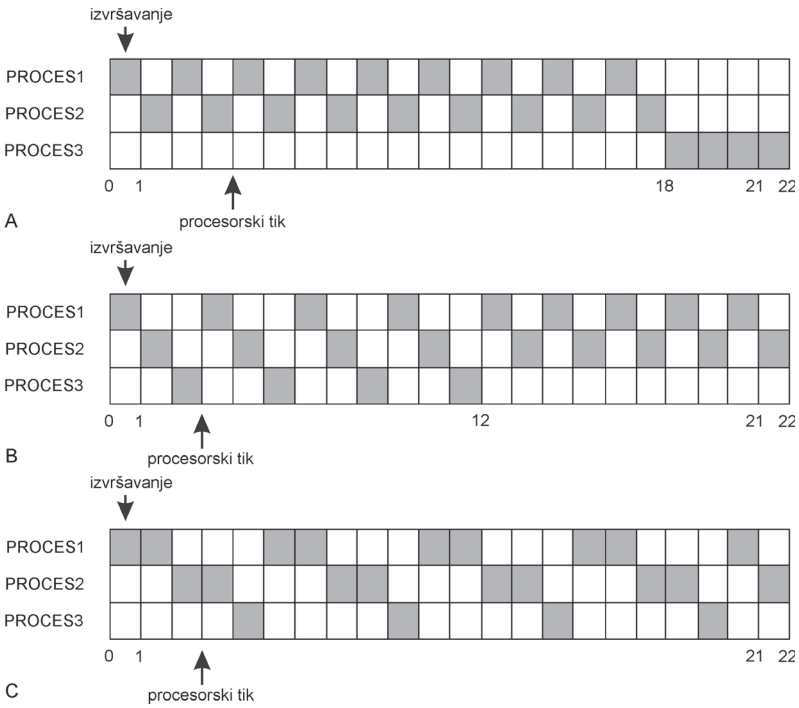
Navedeno je da originalni algoritam omogućava da se prvo izvršavaju naizmenično samo procesi najvišeg prioriteta (slika 3A), što je i dokazano merenjem vremena (tabela 1).

Kružni algoritam omogućava da svi procesi dobiju isto vreme na procesoru sa istom učestalošću (slika 3B), što je i dokazano puštanjem istog skupa procesâ kao u originalnom algoritmu. U kružnom algoritmu se proces nižeg prioriteta izvršavao 12 min i 39 s kraće, ali su se zato procesi višeg prioriteta izvršavali 2 min i 7 s duže u odnosu na originalni algoritam. Kako je bitno da se procesi višeg prioriteta ipak kraće izvršavaju, zato što su to uglavnom sistemski procesi, a za procese nižeg prioriteta nije bitno da se toliko brzo izvrše, bolje rezultate daje kružni algoritam.

Tabela 1. Ukupno vreme izvršavanja* seta opisanih procesa pomoću originalnog i implementiranih algoritama

Planer procesa	Program		
	PROCES 1	PROCES 2	PROCES 3
Originalni	1008	1010	1138
Kružni	1135	1137	379
Predloženi	1132	1134	758

* Vreme je navedeno u sekundama. Osenčena su polja koja predstavljaju proces koji se poslednji završio.



Slika 3. Izvršavanje procesa kroz vreme:
A – originalni algoritam;
B – kružni algoritam;
C – predloženi algoritam.

Figure 3. Process execution over time:
A – Original algorithm;
B – Round Robin algorithm;
C – Round Robin algorithm with retention.

Predloženi algoritam omogućava da se neki proces izvrši više puta uzastopno, ali da se broj dodeljenih uzastopnih kvanti procesorskog vremena razlikuje za procese različitog prioriteta (slika 3C), pa se može primetiti promena u rezultatima (tabela 1). Proces koji je kraći i manjeg prioriteta se sada izvršava duplo duže u odnosu na kružni algoritam, ali se i dalje izvršava kraće za 6 min i 20 s u odnosu na originalni algoritam. Međutim, predloženi algoritam omogućava da se procesi višeg prioriteta izvšavaju 3 s kraće nego kod kružnog algoritma, ali 2 min i 4 s duže nego u originalnom algoritmu.

Primenom kružnog ili originalnog algoritma takođe je isključena eventualna mogućnost nepotrebnog izvršavanja IDLE procesa, što je dokazano dodavanjem ispisa znaka „+" prilikom izvršavanja IDLE procesa. Test je činio proces prioriteta 255 pokrenut primenom originalnog i implementiranih planera. Kako se „+" nije ispisivao primenom implementiranih algoritma, zaključeno je da se proces ne izvršava.

Diskusija

Zbog brzine rada procesora i količine dostupne memorije, nije bilo moguće implementirati neki složeniji algoritam kao što je CFS (Completely Fair Scheduler) – planer procesa koji se koristi u operativnom sistemu Linuks (CFS 2019), ili EAS (Energy Aware Scheduling ili Energy Priority Scheduling) – planer procesa koji se koristi u operativnom sistemu Android, ali je uveden i u kernel operativnog sistema Linuks (Perret 2019), ili neki hibridni algoritam. Dugačka obrada ovakvih algoritama mogla bi da dovede do predugačkog izvršavanja planera u odnosu na same procese, i da dovede do gubljenja procesorskog prekida (INTERRUPT-a). Primenom ovih algoritama rezultati ne bi imali veliko poboljšanje, jer je u to vreme najčešće pokretan mali broj procesa i njihov odziv je predstavljao veći značaj u odnosu na malo smanjenje vremena koje bi ovaj algoritam mogao da donese. Tokom podizanja sistema procesi se izvršavaju jedan po jedan, pa nikakva izmena planera ne bi mogla uticati na brzinu, već bi bila potrebna izmena samog učitavanja sistema.

Predloženi algoritam je omogućio rad sa procesima nižeg prioriteta, pa čak i da proces nižeg prioriteta završi izvršavanje, dok se izvršava neki proces višeg prioriteta. Primer gde se vidi ova promena je pokretanje nekog sistemskog podprocesa (koji je CPU bound) iz korisničkog procesa. Primenom ovog algoritma, korisnik će biti u mogućnosti da nastavi rad sa korisničkim programom (ukoliko ne čeka povratnu informaciju), dok se sistemski proces izvršava. Realni primer je snimanje velikog tekstualnog fajla. Kako se tom prilikom poziva proces koji radi sa ulazno-izlaznim operacijama, rad samog tekstualnog editora bi bio moguć, što nije slučaj sa originalnim planerom. Sa originalnim planerom, sve i da proces većeg prioriteta bude postavljen da radi u pozadini, proces nižeg prioriteta ne bi mogao da se izvršava, jer proces nižeg prioriteta, nikada ne bi dobio procesorsko vreme i time bi sistemski proces blokirao rad korisnika.

Problem predloženog algoritma predstavlja odziv u slučaju novonastalog procesa. Ovo predstavlja problem zato što se može desiti da proces koji je pokrenut ima kratko izvršavanje na procesoru nakon čega bi mogao raditi ulazno-izlazne operacije. Kako predloženi algoritam nov proces uvek stavlja na kraj liste, poboljšanje algoritma bi moglo biti da se novom procesu da određen broj uzastopnih kvanti procesorskog vremena (ne nužno broj koji se dobija kao brojač u implementaciji predloženog algoritma) nakon čega bi se trebao prebaciti na kraj liste i dalje tretirati kao i ostali procesi.

Zaključak

U ovom radu je odrađena izmena originalnog planera procesa MP/M II operativnog sistema. Cilj je bio da se postigne brži i efikasniji rad sa procesima poštujući resurse koji su bili dostupni kada je taj planer implementiran. Implementirani su kružni algoritam i kružni algoritam sa zadržavanjem.

Kružni algoritam je omogućio da svi procesi dobiju isto vreme na procesoru sa istom učestalošću, ali je zanemario uticaj prioriteta procesa. Predloženi algoritam je omogućio da se neki proces izvrši više puta uzastopno, ali da se broj dodeljenih uzastopnih kvanti procesorskog

vremena razlikuje za procese različitog prioriteta. Obe modifikacije originalnog planera procesa nisu dovele do bitnije promene u vremenu izvršavanja samog planera i postigle su da procesi nižeg prioriteta ne ostanu dugo bez procesorskih resursa.

Dalji rad bi mogao da se usmeri na preciznije dobijanje rezultata, grupisanje procesa po terminalu sa kog su pokrenuti i bolji odziv novonastalih procesa. Grupisanjem bi se omogućila ravnomernija raspodela procesorkog resursa između više korisnika, a na bolji odziv bi se moglo uticati privremenim menjanjem prioriteta novonastalog procesa.

Literatura

CFS (CFS Scheduler) 2019. https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt

Comp.os.cpm. 2019. https://groups.google.com/forum/#!forum/comp.os.cpm

DRI (Digital Research) 1981. MP/M II V2.0 Dispatcher, verzija: 2.0, izvorni kod, http://www.cpm.z80.de/source.html (dsptch.asm)

DRI (Digital Research) 1982*a*. MP/M II Operating System Programmer's Guide. Pacific Grove (CA): Digital Research

DRI (Digital Research) 1982*b*. MP/M II Operating System User Guide. Pacific Grove (CA): Digital Research

Perret Q. 2019. Energy Aware Scheduling (EAS) in Linux 5.0. https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/energy-aware-scheduling-in-linux

Marić M. 2017. *Operativni sistemi*. Beograd: Univerzitet u Beogradu – Matematički fakultet

P112 (P112 Support Page) 2019. http://p112.sourceforge.net/index.php?mpm2

Tanenbaum S. A., Bos H. 2015. *Modern Operating Systems*. London: Pearson

Z80-SIM (Z80-SIM 8080-SIM) 2019. https://www.autometer.de/unix4fun/z80pack/

Diana Šantavec

Implementation of the Round Robin Algorithm with Retention in the MP/M II Operating System

Process scheduling in multiprogramming operating systems is an issue on which the performance of the entire system depends. By analyzing the scheduling algorithm of the MP/M II operating system, it was observed that it allows pseudo-parallel execution only of processes with the same priority. Because the algorithm in this system always begins the execution from the highest priority process, we can come to the conclusion that resource starvation of processes with a lower priority can be reached. Because of the above reason, two algorithms have been implemented: the Round Robin algorithm and the Round Robin algorithm with retention. These algorithms have been chosen based on their performance and because they are less hardware demanding.

Implementation of the Round Robin algorithm removed the above problem, but process priority no longer had an impact neither on sorting processes nor on the duration of the process execution on the processor. The advantage of the above trait is that processes with shorter execution time will finish their execution more quickly. The disadvantage of the above trait is when a lot of long execution processes with low priority have been started. That could lead to rare execution of processes with higher priority. This problem was manifested to the user by the blockage of the system.

Because of that new problem, the Round Robin algorithm with retention has been implemented. This algorithm introduces the counter of successive quantum of processor time which allows the higher priority process to receive processor resources in multiple successive quanta. Implementation of this counter has led to slower execution of the processes with a lower priority, but the execution of processes with different priorities was more balanced than it was when the Round Robin algorithm was implemented. How

the recommended scheduling algorithm works is represented in Figure 3C.

The final conclusion was that the execution time of the original and the recommended scheduling algorithm depends on a number of processes which are currently in the list of processes ready for execution. The execution time of the scheduling algorithm with the Round Robin algorithm implemented depends only on a number of processes in the list of processes that are ready for execution.

The recommended algorithm has achieved the desired goal, but all the listed algorithms could have a usage in some operating systems, depending on the needs and characteristics of that specific system.