Ograniczanie przepustowości operacji zapisu i odczytu

Piotr Truszkowski

18 lutego 2007

Spis treści

1 Wstęp

W następnych rozdziałach przedstawię jak działa algorytm limitowania, a na koniec symulację jego działania krok po kroku. Ów opis jest potrzebny ze względu na zrozumienie wyników testów oraz poprawną ich analizę. W rozdziałach 4 i 5 są przedstawione 4 typy testów. Każdy test składa się z kilku innych pod-testów, opierających się na tym samych schemacie jednak z innymi wartościami limitów, kopiowanych danych itp. Każdy z tychże testów był przeprowadzany 10 krotnie i na podstawie wyników wyliczana była wartość średnia oraz odchylenie standardowe. Ponadto w wyniku tych właśnie testów wyklarował się sposób przyznawania czasu do zaśnięcia.

2 Środowisko

Testy przeprowadziłem na osobistym komputerze z zainstalowaną dystrybucją Slackware 10.2 oraz wersją jądra 2.6.17.13. Komputer z jednym procesorem Intel Celeron 2.666GHz, cache 256KB, Pamięcią RAM 256MB. Testy przeprowadzałem w trybie konsolowym bez aplikacji X-owych. Urządzeniami blokowymi były partycje z systemami plików ext3 oraz fat32.

3 Algorytm

3.1 Opis

Po pierwsze zastosowałem pojęcie kwantu czasu. Na ten czas proces ma do dyspozycji tyle bajtów ile wynosi nałożony limit(przeskalowany względem długości kwantu). Przed każdym zapisem/odczytem algorytm przyznaje ile można w tej chwili skopiować danych z puli przeznaczonej na obecny kwant czasu. Tuz po przekopiowaniu danych algorytm pobiera wartość czasu jaki musi przeczekać by limit nie został przekroczony. W kodzie służą do tego procedury get_count() oraz get_sleep(). W uproszczeniu algorytm wygląda tak:

```
while (true) {
  operacja(file,buf,get_count();,pos);
  sleep(get_sleep());
  ...;
}
```

Procedura get_count() potrafi zmniejszyć porcję jeśli zauważy, ze jest zbyt duża. Natomiast get_sleep() usypia proces na czas proporcjonalny do tego na ilu danych operowaliśmy w stosunku do tych jakie jeszcze zostały do przydzielenia w aktualnym kwancie czasu. Przyznajemy czas z puli czasu jaki pozostał do końca „epoki”. Jest to wyliczane dokładnie wg wzoru:

$$czas\_uspienia = (ile\_minelo) * (ile)/(ile\_pozostalo + ile)$$

3.2 Symulacja

Poniżej fragment kodu(C++) symulującego działanie algorytmu.

```
C = (C <= (L >> 3) ? C : (L >> 3));
while (true) {
  K -= C;
  S[i] = (K > 0 ? (T - t)*C/(K+C) : T - t );
  t += S[i] + W;
  ++i;
}
```

Jego działanie jest pomocne w zrozumieniu i interpretacji wyników przeprowadzonych testów.

4 Testy read(), write()

4.1 Test 1

Plik test_1.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla jednorazowych dużych zapisów.

4.1.1 Wyniki

Nr	Limit	Dane	Średnia	Odch. std.
0	10kB/s	200000B	9.940kB/s	0.000B/s
1	20kB/s	200000B	19.882kB/s	0.005B/s
2	30kB/s	200000B	29.809kB/s	0.011B/s
3	40kB/s	200000B	39.734kB/s	0.077B/s
4	50kB/s	200000B	49.758kB/s	0.022B/s
5	100kB/s	500000B	98.881kB/s	0.887B/s
6	150kB/s	500000B	148.699kB/s	0.419B/s
7	200kB/s	500000B	198.490kB/s	0.234B/s
8	500kB/s	2000000B	497.532kB/s	0.059B/s
9	1000kB/s	5000000B	943.809kB/s	65.416B/s
10	2000kB/s	5000000B	1767.566kB/s	233.394B/s
11	0kB/s	5000000B	262375.976kB/s	187738.807B/s

4.1.2 Analiza

Widzimy, ze limit nie jest przekraczany, a także odchylenie standardowe wyników wykazuje dużą stabilizację. Dla wyższych limitów możemy jednak obserwować większe wahania, wiąże się to z małą przewidywalnością transferu dla wielkości jakich tu użyłem, 5MB danych z 2MB/s jest trudniej monitorować niźli np: 30kB/s.

4.2 Test 2

Plik test_2.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla wielu częstych zapisów małej ilości danych.

4.2.1 Wyniki

Nr	Limit	Dane	Średnia	Odch. std.	Porcja	Szacowanie	Czas zapisu
0.1	10kB/s	200000B	11.236kB/s	0.001kB/s	2000	–	0.188s
0.2	10kB/s	200000B	9.615kB/s	0.001kB/s	2000	100kB/s	–
1.1	20kB/s	200000B	25.539kB/s	0.117kB/s	400	–	0.025s
1.2	20kB/s	200000B	8.758kB/s	0.013kB/s	400	20kB/s	–
2.1	30kB/s	200000B	33.246kB/s	0.146kB/s	400	–	0.022s
2.2	30kB/s	200000B	9.513kB/s	0.014kB/s	400	20kB/s	–
3.1	40kB/s	200000B	51.161kB/s	0.530kB/s	1000	–	0.029s
3.2	40kB/s	200000B	20.181kB/s	0.088kB/s	1000	50kB/s	–
4.1	50kB/s	200000B	63.643kB/s	0.710kB/s	1000	–	0.025s
4.2	50kB/s	200000B	21.869kB/s	0.090kB/s	1000	50kB/s	–
5.1	100kB/s	500000B	127.165kB/s	1.536kB/s	2500	–	0.029s
5.2	100kB/s	500000B	50.193kB/s	0.303kB/s	2500	125kB/s	–
6.1	150kB/s	500000B	180.116kB/s	3.376kB/s	2500	–	0.024s
6.2	150kB/s	500000B	56.672kB/s	0.458kB/s	2500	125kB/s	–
7.1	200kB/s	500000B	253.479kB/s	5.934kB/s	5000	–	0.031s
7.2	200kB/s	500000B	99.799kB/s	0.990kB/s	5000	250kB/s	–
8.1	500kB/s	2000000B	595.877kB/s	29.423kB/s	10000	–	0.027s
8.2	500kB/s	2000000B	210.231kB/s	5.757kB/s	10000	500kB/s	–
9.1	1000kB/s	2000000B	1001.372kB/s	42.328kB/s	10000	–	0.021s
9.2	1000kB/s	2000000B	242.087kB/s	5.891kB/s	10000	500kB/s	–
10.1	2000kB/s	3000000B	904.168kB/s	49.620kB/s	6000	–	0.017s
10.2	2000kB/s	3000000B	161.356kB/s	2.202kB/s	6000	300kB/s	–
11.1	0kB/s	6000000B	74769.620kB/s	145945.743kB/s	12000	–	0.012s
11.2	0kB/s	6000000B	369.800kB/s	7.263kB/s	12000	600kB/s	–

4.2.2 Analiza

Testy oznaczone „numer.1” oznaczają pomiar prędkości SAMEGO zapisu danych, testy „numer.2” do pomiaru prędkości zapisu wliczając przerwy miedzy każdym zapisem. Obserwujemy zatem, iż w każdym wypadku prędkość samego zapisu jest ponad limit! Wiąże się to z tym iż algorytm limitujący „przyswaja” sobie przerwy między kolejnymi zapisami, wlicza w koszty „sleep-ów”. To zatem tłumaczy podwyższoną prędkość testów z indeksami *.1. Zauważmy iż prędkości testów z indeksami *.2 bywają rażąco mniejsze od tych pierwszych. Znów powodem zaistniałej anomalii jest czas przerwy miedzy kolejnymi zapisami. Otóż nawet jakbyśmy się starali zapisywać dane natychmiast bez usypiania to moglibyśmy się nie wyrobić. Powód jest prosty, zapisujemy N bajtów, przerwa T czasu, zapisujemy N bajtów i znów przerwa T

czasu, itd. Zatem po sekundzie zapiszemy(zakładam, że T mniejsze od 1 sekundy) N/T bajtów w optymistycznym planie. Planie którego nie możemy przekroczyć i tak w końcu zakładamy, że zapisujemy natychmiast. W przyrządzonym teście $T = 0.02s$, wartości w przedostatniej kolumnie przedstawiają optymistyczną wartość prędkości całkowitej. Nie jest ona osiągana nawet dla zapisu bez limitu(ostatni test) ciężko więc spodziewać się by była osiągana dla zapisów z nałożonymi limitami, którym to z założenia zarzucone jest, ze po zapisie mają chwilą odczekać. Ostatnia kolumna przedstawia właśnie średni czas operacji write(). Biorąc pod uwagę, iż proces bez limitu zużywa $0.012sek$ a więc narzut związany z usypianiem procesu na koniec zapisu jest rzędu $0.01sek$, a to skolei jest minimalną jednostką czasu(zaraz po zerze) na jaki proces może być usypiany po zapisie.

4.3 Test 3

Plik test_3.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla operacji kopiowania z urządzenia objętego limitem odczuty do urządzenia objętego limitem zapisu. U mnie pierwszym urządzeniem był dysk z systemem plików ext3, drugim dysk z systemem plików fat32.

4.3.1 Wyniki

Nr	Odczyt	Zapis	Dane	Średnia	Odch. std.	Szacowanie
0	10kB	15kB	200000B	7.365kB/s	0.036kB/s	6kB/s
1	20kB	40kB	200000B	19.098kB/s	0.124kB/s	13kB/s
2	50kB	30kB	200000B	26.487kB/s	0.010kB/s	18kB/s
3	40kB	10kB	200000B	9.599kB/s	0.029kB/s	8kB/s
4	20kB	100kB	200000B	19.276kB/s	0.228kB/s	16kB/s
5	100kB	20kB	200000B	18.952kB/s	0.053kB/s	16kB/s
6	150kB	150kB	2000000B	105.475kB/s	1.921kB/s	75kB/s
7	200kB	350kB	2000000B	186.818kB/s	3.908kB/s	127kB/s
8	500kB	1000kB	2000000B	485.637kB/s	1.008kB/s	333kB/s
9	300kB	0kB	2000000B	297.392kB/s	3.549kB/s	300kB/s
10	0kB	300kB	2000000B	291.996kB/s	11.283kB/s	300kB/s
11	0kB	0kB	2000000B	338776.755kB/s	3909.867kB/s	?kB/s

4.3.2 Analiza

Rozpatrzmy najpierw na czym powinniśmy się skupić analizując powyższe wyniki. Ostatnią kolumną w tabeli są wartości prędkości szacowanej jaką uzyskać powinniśmy. Zakładając, że N danych odczytamy w czasie N/V_{read} i te N danych zapiszemy w czasie N/V_{write} to całkowita prędkość tejże operacji wyrazi się następująco: $V_{szacowana} = \frac{N}{N/V_{read}+N/V_{write}} = \frac{1}{\frac{1}{V_{read}}+\frac{1}{V_{write}}}$. Każdorazowo obserwujemy jednak prędkość wyższą od szacowanej. Wynika to, podobnie jak w teście poprzednim z „przywłaszczania” przerw miedzy kolejnymi operacjami zapisu jak i odczytu. Zatem write() korzysta z czasu w jakim działa read() oraz read() korzysta z czasu w jakim

działa write(), mamy do czynienia z niejawnym „współdzieleniem” operacji sleep(). Warto zauważyć, że w każdym przypadku wyniki testowe zbiegają do wartości minimum z założonych ograniczeń.

4.4 Test 4

Plik test_4.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla operacji kopiowania z urządzenia objętego limitem odczuty do dwóch plików z urządzenia objętego limitem zapisu. Znów u mnie pierwszym urządzeniem był dysk z systemem plików ext3, drugim dysk z systemem plików fat32.

4.4.1 Wyniki

Nr	Odczyt	Zapis	Dane	Średnia	Odch. std.	Szacowanie
0	10kB	15kB	200000B	12.248kB/s	0.104kB/s	8kB/s
1	20kB	40kB	200000B	28.922kB/s	0.165kB/s	19kB/s
2	50kB	30kB	200000B	29.063kB/s	0.145kB/s	23kB/s
3	40kB	10kB	200000B	9.857kB/s	0.084kB/s	8kB/s
4	20kB	100kB	200000B	38.206kB/s	0.219kB/s	28kB/s
5	100kB	20kB	200000B	19.767kB/s	0.150kB/s	18kB/s
6	150kB	150kB	2000000B	127.434kB/s	3.685kB/s	99kB/s
7	200kB	350kB	2000000B	220.393kB/s	3.104kB/s	186kB/s
8	500kB	1000kB	2000000B	908.646kB/s	69.120kB/s	499kB/s
9	300kB	0kB	2000000B	589.352kB/s	17.172kB/s	599kB/s
10	0kB	300kB	2000000B	297.221kB/s	0.810kB/s	299kB/s
11	0kB	0kB	2000000B	425194.655kB/s	919.448kB/s	0kB/s

4.4.2 Analiza

Sytuacja dość podobna do poprzedniego testu. Tutaj dodatkowo kopiuje się dane nie do jednego pliku lecz do dwóch. Szacowana prędkość ulegnie zmianie, gdyż zapisuje w sumie dwa razy więcej niż poprzednio. $V_{szacowana} = \frac{2N}{N/V_{read}+2N/V_{write}} = \frac{2}{\frac{1}{V_{read}}+\frac{1}{2V_{write}}}$. Podobnie i jak poprzednio uzyskane realnie prędkości są lepsze od szacowanych. Znów to samo – write() i read() „odpoczywają” gdy drugi pracuje. Podobnie jak poprzednio i w tym teście występuje równanie do minimum z dwóch liczb. Ponieważ teraz zapisywaliśmy do dwóch plików to $predkosc = minimum(2Limit_{read}, Limit_{write})$.

5 Testy readv(), writev()

5.1 Test 1

Plik testv_1.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla jednorazowych dużych zapisów.

5.1.1 Wyniki

Nr	Limit	Dane	Średnia	Odch. std.
0	10kB/s	2000B	9.939kB/s	0.004kB/s
1	20kB/s	2000B	19.864kB/s	0.035kB/s
2	30kB/s	2000B	29.709kB/s	0.018kB/s
3	40kB/s	2000B	39.744kB/s	0.054kB/s
4	50kB/s	2000B	49.606kB/s	0.212kB/s
5	100kB/s	5000B	99.238kB/s	0.296kB/s
6	150kB/s	5000B	146.710kB/s	0.771kB/s
7	200kB/s	5000B	191.416kB/s	0.667kB/s
8	500kB/s	20000B	471.859kB/s	29.726kB/s
9	1000kB/s	50000B	851.485kB/s	65.402kB/s
10	2000kB/s	50000B	1607.796kB/s	559.197kB/s
11	0kB/s	50000B	270274.572kB/s	177421.074kB/s

5.1.2 Analiza

Podobnie jak w przypadku testu 1 dla write() widzimy wyraźne trzymanie się założonego ograniczenia. Wyniki wykazują też dużą stabilizację(Odch. std.).

5.2 Test 2

Plik testv_2.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla wielu częstych zapisów małej ilości danych. Dokładniej to każdy writev() zapisuje dane jako wektor buforów, w każdym wypadku wektor ma dlługość 100.

5.2.1 Wyniki

Nr	Limit	Dane	Średnia	Odch. std.	Szacowanie	Czas
0.1	10kB/s	100x100x20B	11.117kB/s	0.003kB/s	–	0.189s
0.2	10kB/s	100x100x20B	9.528kB/s	0.002kB/s	100kB/s	–
1.1	20kB/s	500x100x20B	3043.121kB/s	247.275kB/s	–	0.130s
1.2	20kB/s	500x100x20B	13.322kB/s	0.006kB/s	100kB/s	–
2.1	30kB/s	500x100x20B	2465.539kB/s	573.206kB/s	–	0.130s
2.2	30kB/s	500x100x20B	13.304kB/s	0.022kB/s	100kB/s	–
3.1	40kB/s	200x100x20B	5511.584kB/s	1376.747kB/s	–	0.040s
3.2	40kB/s	200x100x20B	33.195kB/s	0.112kB/s	100kB/s	–
4.1	50kB/s	200x100x20B	4547.078kB/s	1989.570kB/s	–	0.040s
4.2	50kB/s	200x100x20B	32.935kB/s	0.558kB/s	100kB/s	–
5.1	100kB/s	200x100x50B	5381.139kB/s	1964.249kB/s	–	0.040s
5.2	100kB/s	200x100x50B	81.923kB/s	0.496kB/s	250kB/s	–
6.1	150kB/s	200x100x50B	5697.405kB/s	2316.002kB/s	–	0.040s
6.2	150kB/s	200x100x50B	82.069kB/s	0.382kB/s	250kB/s	–
7.1	200kB/s	100x100x50B	16921.349kB/s	17834.221kB/s	–	0.010s
7.2	200kB/s	100x100x50B	162.480kB/s	3.331kB/s	250kB/s	–
8.1	500kB/s	200x100x200B	24945.086kB/s	24113.021kB/s	–	0.044s
8.2	500kB/s	200x100x200B	311.103kB/s	10.719kB/s	1000kB/s	–
9.1	1000kB/s	200x100x200B	46970.100kB/s	41180.903kB/s	–	0.044s
9.2	1000kB/s	200x100x200B	309.893kB/s	8.747kB/s	1000kB/s	–
10.1	2000kB/s	500x100x300B	8100.895kB/s	2506.154kB/s	–	0.137s
10.2	2000kB/s	500x100x300B	190.648kB/s	1.607kB/s	1500kB/s	–
11.1	0kB/s	500x100x600B	23127.533kB/s	17555.626kB/s	–	0.144s
11.2	0kB/s	500x100x600B	363.728kB/s	7.983kB/s	3000kB/s	–

5.2.2 Analiza

Ten test pokazuję, dobitniej niż analogiczny dla write(), wykorzystywanie przerw między kolejnymi zapisami. Widać, że czas trwania operacji writev(ostatnia kolumna) daje większy czas niż dla testu 2 dla write(). Wynika to z tego iż tam liczony był czas pojedynczego write() tutaj writev() w istocie wykonuje 100 writev() wewnętrznych. Prędkości średnie wliczając przerwy między zapisami są ustabilizowane(Odch. std.). Widać, że zapis nie przyjmuje maksymalnej przepustoości ze względu na czas trwania writev().

5.3 Test 3

Plik testv_3.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla operacji kopiowania z urządzenia objętego limitem odczuty do urządzenia objętego limitem zapisu. U mnie pierwszym urządzeniem był dysk z systemem plików ext3, drugim dysk z systemem plików fat32. Długość wektora z buforami danych ustaliłem na 10.

5.3.1 Wyniki

Nr	Odczyt	Zapis	Dane	Średnia	Odch. std.	Szacowanie
0	10kB	15kB	20000B	5.830kB/s	0.021kB/s	6kB/s
1	20kB	40kB	20000B	15.385kB/s	0.388kB/s	13kB/s
2	50kB	30kB	20000B	24.515kB/s	2.072kB/s	18kB/s
3	40kB	10kB	20000B	7.633kB/s	0.000kB/s	8kB/s
4	20kB	100kB	20000B	18.004kB/s	0.763kB/s	16kB/s
5	100kB	20kB	20000B	17.989kB/s	0.524kB/s	16kB/s
6	150kB	150kB	200000B	57.865kB/s	0.832kB/s	75kB/s
7	200kB	350kB	200000B	164.857kB/s	7.885kB/s	127kB/s
8	500kB	1000kB	200000B	484.807kB/s	1.435kB/s	333kB/s
9	300kB	0kB	200000B	303.053kB/s	54.952kB/s	300kB/s
10	0kB	300kB	200000B	302.493kB/s	52.967kB/s	300kB/s
11	0kB	0kB	200000B	381471.026kB/s	3317.206kB/s	?kB/s

5.3.2 Analiza

Podobnie jak dla read() i write() i tutaj zachodzi do „współdzienia” przerw. Operacja readv() wyrabia limit przy pomocy operacji writev() i na odwrót. Mimo trzymania się limitów wyniki są lepsze od szacowanych(ostatnia kolumna). W czasie przerwy readv() operacja writev() zapisuje dane. Dzięki czemu jak wcześniej następuje próba równania do minimum z wartości obu założonych limitów.

5.4 Test 4

Plik testv_4.c przedstawia przykładowy test mający na celu sprawdzenie algorytmu dla operacji kopiowania z urządzenia objętego limitem odczuty do dwóch plików z urządzenia objętego limitem zapisu. Znów u mnie pierwszym urządzeniem był dysk z systemem plików ext3, drugim dysk z systemem plików fat32. Długość wektora z buforami danych ustaliłem na 10.

5.4.1 Wyniki

Nr	Odczyt	Zapis	Dane	Średnia	Odch. std.	Szacowanie
0	10kB	15kB	20000B	8.439kB/s	0.001kB/s	8kB/s
1	20kB	40kB	20000B	18.015kB/s	0.009kB/s	19kB/s
2	50kB	30kB	20000B	27.364kB/s	0.731kB/s	23kB/s
3	40kB	10kB	20000B	8.658kB/s	0.000kB/s	8kB/s
4	20kB	100kB	20000B	32.831kB/s	1.183kB/s	28kB/s
5	100kB	20kB	20000B	17.857kB/s	0.000kB/s	18kB/s
6	150kB	150kB	200000B	75.281kB/s	0.890kB/s	100kB/s
7	200kB	350kB	200000B	247.771kB/s	18.332kB/s	186kB/s
8	500kB	1000kB	200000B	885.867kB/s	60.037kB/s	500kB/s
9	300kB	0kB	200000B	607.979kB/s	116.318kB/s	600kB/s
10	0kB	300kB	200000B	294.901kB/s	18.983kB/s	300kB/s
11	0kB	0kB	200000B	491074.935kB/s	3058.360kB/s	?kB/s

5.4.2 Analiza

Zupełnie analogicznie jak w poprzednim teście. Tutaj jedynie występuje równanie do wartości $minimum(2Limit_{read}, Limit_{write})$.

6 Testy ekstremalne

Na koniec postanowiłem przeprowadzić kilka ekstremalnych testów. Spisane są one w plikach extreme.c i extremev.c. A poniżej wyniki testów dla read() i write().

Nr	Odczyt	Zapis	Ilość	Porcja	Średnia	Odch. std.	Szacowanie
0	0kB	0kB	1000	1B	198.079kB/s	0.501kB/s	?kB/s
1	0kB	1kB	1000	5B	0.855kB/s	0.282kB/s	1kB/s
2	0kB	2000kB	5000000	10B	1811.063kB/s	197.509kB/s	2000kB/s
3	2000kB	5000kB	10000000	1000B	1924.406kB/s	251.188kB/s	1428kB/s
4	3000kB	1000kB	10000000	100000B	890.519kB/s	87.058kB/s	750kB/s

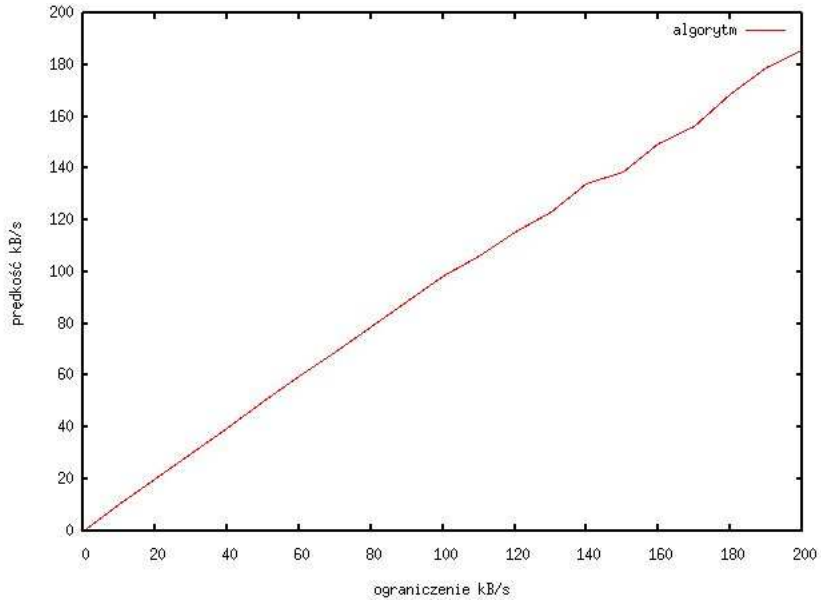
Oraz dla readv(), writev().

Nr	Odczyt	Zapis	Ilość	Porcja	Średnia	Odch. std.	Szacowanie
0	0kB	0kB	100	10B	13219.504kB/s	152.412kB/s	?kB/s
1	0kB	80kB	100	5B	3196.234kB/s	2081.798kB/s	80kB/s
2	0kB	2000kB	700	100B	1928.073kB/s	507.258kB/s	2000kB/s
3	2000kB	5000kB	500	200B	2099.060kB/s	385.313kB/s	1428kB/s
4	3000kB	1000kB	200	500B	962.448kB/s	72.694kB/s	750kB/s

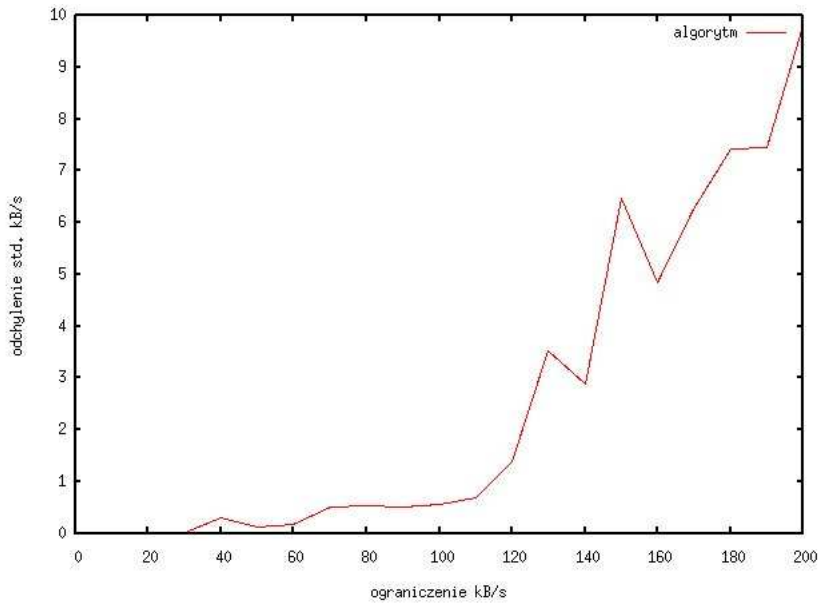
W obu przypadkach występuje równanie do $minimum(Limit_{read}, Limit_{write})$. Wyniki dla read()/write() cechują się stabilnością, dla readv()/writev() stabilność jest mniejsza jednak pozostająca w ramach 25%. Anomalię stanowi wynik dla writev()/readv() z limitem 80kB/s.

7 Podsumowanie

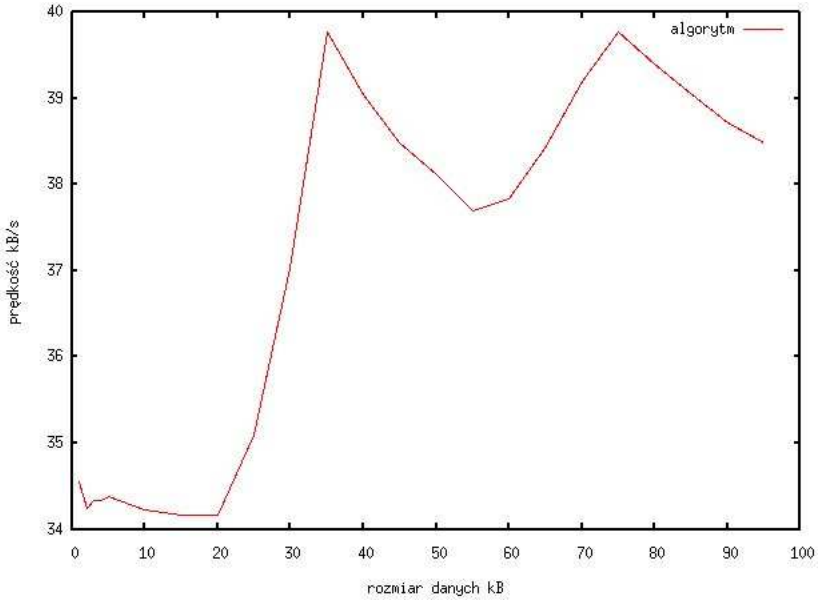
Niebagatelny wpływ na przebieg odczytów/zapisów ma czas przerwy między kolejnymi operacjami. Z jednej strony przerwa może pełnić rolę zastępczego sleep'a z innej, gdy jest zbyt długa, może sama nakładać własny limit. Pierwsze oblicze widać było wyraźnie w testach 3 i 4, gdzie uzyskiwaliśmy prędkości lepsze od szacowanych. Drugie oblicze w 2 teście, gdzie czasem się okazywało, że przerwa między operacjami nakłada własny limit.
Testy 3 i 4 pokazują także, iż zastosowanie limitów nie musi odbijać się na przepustowości sumarycznej transakcji. Wyniki szacunkowe jakich się można było spodziewać po przeliczeniu czasu jaki będzą potrzebować operacje read(), write(), readv(), writev() okazywały się zaniżone. Lepszym oszacowaniem okazywało się minimum z obu nałożonych limitów.
Poniżej kilka wykresów.



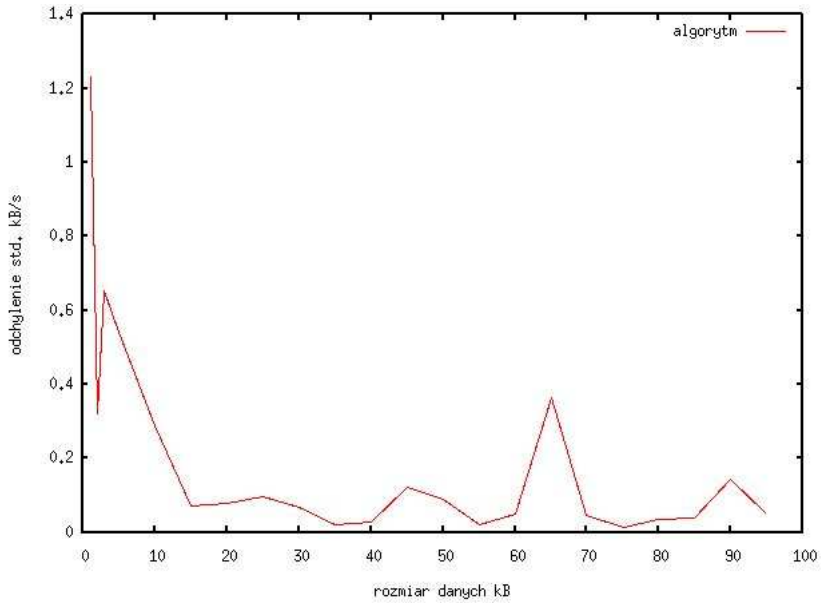
Rysunek 1: zależność między nałożonym limitem a rzeczywistym.



Rysunek 2: Wartość odchylenia std. wobec nałożonego limitu.



Rysunek 3: Zależność rzeczywistego limitu wobec liczby zapisywanych danych(limit = 40kB/s).



Rysunek 4: Wartość odchylenia std. wobec liczby zapisywanych danych(limit = 40kB/s).