Tema 4

Secvential		23.42866ms
Pr=1	Pw=4	334.69896ms
	Pw=6	525.16463ms
	Pw=8	753.03452ms
	Pw=16	1651.61944ms
Pr=2	Pw=4	118.495601ms
	Pw=6	205.99784ms
	Pw=8	262.91336ms
	Pw=16	529.79212ms

Analiza timpilor de executie

- Varianta secventiala se comporta cel mai bine, de 10..20 de ori mai rapida decat P-C
- Cresterea numarului de workers aduce delay semnificativ, din cauza busy-waitingului combinat cu faptul ca din cauza numarului mic de producer majoritatea workerilor doar asteapta fara sa consume ceva.
- Cresterea numarului de readers creste numarul de taskuri si ca atare scade numarul workerilor inactivi/idle, conducand la rezultate cu pana la 60% mai eficiente.
- Incercarea de a atenua impactul busy-waitingului printr-un sleep(1ms) aduca mici imbunatatiri pentru pr=1, dar dubleaza timpul de executie in cazul pr=2

Descrierea algoritmului

Se foloseste implementarea clasica de linked list/queue, iar in cazul paralel intervine un mutex care asigura siguranta datelor intre threaduri. Singura modificare notabila se intampla in functia queue-pop cand se trateaza cazul cozii vide. Pentru a nu bloca structura de date, functia de pop returneaza un flag boolean care specifica daca s-a putut extrage un element din coada sau nu, in loc sa blocheze coada pana la introducerea unui nou element. Astfel, ramane in responsabilitatea consumatorului sa verifice daca a primit un element sau va continua sa interogheze coada.

Pentru a asigura terminarea programului, se foloseste o variabila atomica (initial, are valoarea fals) care retine daca workerele ar trebui sa ramana active sau sa se opreasca. Aceasta devine true abia dupa ce toti producatorii si-au incarcat toate datele in coada. Conditia de oprire a unui consumator este: “a primit semnal de oprire?” si “daca da, este coada vida?”.