1. Згенеруйте випадкові адреси:

Аргументи: -s 0 -n 10, -s 1 -n 10, -s 2 -n 10.

Змініть політику з FIFO на LRU та OPT.

Оцініть, чи кожен доступ у трасі адреси — це збіг чи промах.

bash

Copy code

./paging-policy.py -s 0 -n 10 -c

./paging-policy.py -s 0 -n 10 -c --policy=LRU

./paging-policy.py -s 0 -n 10 -c --policy=OPT

2. Для кешу розміром 5 згенеруйте найгірші потоки посилань для FIFO, LRU та MRU, щоб спричинити максимальну кількість промахів. Визначте, наскільки потрібно збільшити кеш для значного покращення продуктивності до рівня OPT.

bash

Copy code

./paging-policy.py --addresses=0,1,2,3,4,5,0,1,2,3,4,5 --policy=FIFO --cachesize=5 -c

./paging-policy.py --addresses=0,1,2,3,4,5,0,1,2,3,4,5 --policy=LRU --cachesize=5 -c

./paging-policy.py --addresses=0,1,2,3,4,5,4,5,4,5,4,5 --policy=MRU --cachesize=5 -c

3. Згенеруйте випадкову трасу (через Python або Perl). Порівняйте ефективність різних політик для такої трасування.

bash

Copy code

./paging-policy.py -s 0 -n 10 -c

./paging-policy.py -s 0 -n 10 -c --policy=LRU

./paging-policy.py -s 0 -n 10 -c --policy=OPT

./paging-policy.py -s 0 -n 10 -c --policy=UNOPT

./paging-policy.py -s 0 -n 10 -c --policy=RAND

./paging-policy.py -s 0 -n 10 -c --policy=CLOCK

4. Згенеруйте трасу з локальністю та оцініть ефективність LRU, RAND та CLOCK із різною кількістю бітів.

bash

Copy code

./generate-trace.py

./paging-policy.py --addresses=3,0,6,6,6,6,7,0,6,6 --policy=LRU -c

./paging-policy.py --addresses=3,0,6,6,6,6,7,0,6,6 --policy=RAND -c

./paging-policy.py --addresses=3,0,6,6,6,6,7,0,6,6 --policy=CLOCK -c -b [біти]

5. Використовуйте Valgrind для аналізу реальної програми.

Генеруйте трасу доступу до віртуальної пам’яті:bash  
Copy code  
  
  
valgrind --tool=lackey --trace-mem=yes ls &> ls-trace.txt

Перетворіть її на віртуальні номери сторінок:bash  
Copy code  
  
  
./transform.py

./run.sh

./plot.py

Визначте розмір кешу, що забезпечує високу продуктивність, і побудуйте графік залежності робочого набору від розміру кешу.