CONCEPTES AVANÇATS DE SISTEMES OPERATIUS (CASO)

Facultat d'Informàtica de Barcelona, Dept. d'Arquitectura de Computadors, curs 2018/2019 – 2Q

Pràctiques de laboratori — Mesures de rendiment

Material

La vostra instal·lació de **Linux (Ubuntu, Debian...)**.
Un disc dels antics (els haureu de compartir, planificats en RR).

Mesures de rendiment

Per mesurar el rendiment d'aplicacions i sistemes operatius habitualment usem diverses mètriques:

- Temps d'execució
- Acceleració (speedup): la relació entre el temps d'execució en seqüencial i el temps d'execució en paral·lel
- Ample de banda (bandwidth): la relació entre les dades transmeses i el temps que s'ha invertit en transmetre-les
- Latència: Cost d'iniciar operacions o comunicacions

Les mesures preses poden tenir una certa variabilitat, per això és bo repetir els experiments un cert número de cops. Per exemple, fem l'experiment 10 cops i...

- Temps d'execució: fem la mitjana dels temps obtinguts en els 10 experiments
- Speedup: fem les mitjanes de les 10 executions seqüencials i de les 10 execucions en paral·lel i l'speedup es calcula com a la seva relació
- Bandwidth: fem la mitjana dels temps d'execució dels 10 experiments i calculem el bandwidth o bé fem la mitjana dels diferents bandwidths obtinguts
- Latència: mitjana de les latències obtingudes en N experiments

També és bo calcular la desviació estàndard de les mesures, que ens donarà una idea de la seva variabilitat respecte la mitjana.

Eines de suport

L'eina més bàsica per a prendre mesures de temps en Linux/UNIX és **gettimeofday**(...). És una crida a sistema que retorna el temps transcorregut des de l'1 de gener del 1970[1]. Veure el seu manual ($man gettimeofday).

Amb aquesta crida podem obtenir el temps abans i després del codi que volem mesurar i fer la diferència.

La pàgina de manual del **gettimeofday**(…) fa referència a la més moderna **clock_gettime(…)**. Determineu les diferències entre les dues crides – veure Exercici 1 i els codis que us proporcionem: **gettimeofday.c**, **getclock.c** i **getclock-syscall.c**.

Linux proporciona altres eines, en forma de comandes, que ens permeten analitzar l'execució del sistema o de les aplicacions: *top*, *ps*, *vmstat*, *iostat*...

En el codi de suport d'aquest laboratori us proporcionem un exemple que mostra com aconseguir la data/hora actuals i un altre per comprovar quina és la fórmula correcte per calcular diferències dels temps presos amb *gettimeofday*.

1 http://en.wikipedia.org/wiki/Unix_time

Exercicis

1. Determineu les diferències entre les crides gettimeofday(…) i clock_gettime(…).

2. Comproveu que **getclock.c** i **getclock-syscall.c** tenen un comportament diferent respecte a l'ús de crides a sistema. Quina és la causa? Pista: veieu la pàgina de manual de vdso (7). Quina eina podeu usar per determinar les crides a sistema que fan els programes?

3. Escriviu un programa que calculi el temps que triga una crida a sistema senzilla. Per exemple, les següents són interessants. Podeu basar el vostre programa en el codi **getclock.c** que us proporcionem, simplement substituint la crida a **usleep(…)** per la nova crida que volgueu provar:

- **sbrk(0)**, que només ha de retornar el *break point*, apuntant a la primera adreça invàlida després del *heap*. Veieu la diferència provant també amb **sbrk(inc)**, amb inc > 0. Quin "inc" podeu posar sense comprometre el sistema?
- **sched_yield()**, que només suggereix al sistema operatiu que si troba un altre flux per executar, canviï de context.
- **getpid()**, que retorna el pid del procés.
- **fork()/waitpid()**, per crear i esperar un procés fill.

  Per a fer-ho, feu programes que executin aquestes crides milions de vegades (desenes de milers en el cas del fork()) i preneu el temps.

4. Comproveu que cada programa executa realment la crida a sistema (I que no aprofita el resultat retornat per la crida anterior o usa alguna altra tècnica per estalviar-se-la).

Feu una taula amb els temps d'execució que obteniu a la pregunta 3. Per què els temps d'execució són tan diferents? Indiqueu també si es fan totes les crides a sistema o s'estalvien (pregunta 4):

Syscall	sbrk(0)	sbrk(inc)	sched_yield()	getpid()	fork/waitpid
Execution time (in microsecs.)					
OS is actually called (yes/no)?					

5. Des d'un usuari no privilegiat (no root), executeu els dos programes que teniu cadascun de vosaltres de l'exercici de teoria (diguem-los write-to-disk1 i write-to-disk2), sobre el disc actual i el disc antic. Per exemple, suposant que hem muntat el disc antic (dev/sdc) a /mnt/point:

   $ write-to-disk1 /tmp/file-sdb $ write-to-disk2 /tmp/file-sdb

   $ write-to-disk1 /mnt/point/file-sdc $ write-to-disk2 /mnt/point/file-sdc

   Modifiqueu els programes per poder canviar la mida de les dades enviades al disc. Un cop executats, feu una gràfica amb els resultats que obteniu de bandwidth, comparant els dos discos i els dos programes. A l'eix de les X situeu el tamany de les dades transmeses i a l'eix de les Y, el bandwidth. Podeu usar LibreOffice scalc, el GNU plot, el jgraph…

6. Si executeu el programa com a administradors (root), obteniu alguna diferència en els resultats?

7. Escriviu i executeu una aplicació similar a la del punt 2, però que llegeixi del disc. Feu una gràfica similar a la del punt 2, que mostri ara el bandwidth obtingut per les lectures. Compareu els resultats.

8. Useu les eines del sistema **vmstat** i **iostat** per veure el bandwidth obtingut en lectura i escriptura de fitxers quan executeu els programes write-to-disk i read-from-disk, per separat i alhora. Per una mida concreta (p. ex. 500 Mb), coincideixen amb els vostres números?

L'entrega, a través del Racó, abans del 25/04/19

Entregareu la taula dels apartats 3 i 4, i els programes i les gràfiques obtingudes als apartats 5, 6, 7 i 8.