Universidad de Barcelona

Arthur Font Gouveia 20222613

Marc Colominas Casabella 20081692

Sistemes Operatius II

Pràctica 3

Barcelona

2021

Índex

1. Introducció

En aquesta pràctica farem una breu aproximació al concepte de ciberseguretat i ens centrarem en algunes tècniques que es poden utilitzar per aprofitar els buffer overflows així com altres vulnerabilitats de corrupció de memòria.

2. Anàlisi de l'eficiència d'execució de mainSave

2.1. Operacions sobre la CPU

• Quines són les funcions que triguen més aexecutar-se? Quin és el percentatge de temps que "ocupa" l'execució de cadascuna de les funcions?

R: La funció que triguen més en executar-se són createEmptyRecommendation (46,88%), fillRecommendationMatrix (32,58%) i setRecommendationMatrixValue (31,60%).

• Analitzem entre altres la funció createEmptyRecommendationMatrix. Quina creieu que és la raó per la qual triga tant aexecutar-se?

R: La funció createEmptyRecommendationMatrix tarda molt a executar-se perquè crida la funció setRecommendationMatrixValue 795 976 398. Si mirem bé, createEmptyRecommendationMatrix ocupa (46,88%) sent que setRecommendationMatrixValue ocupa (31,25%) d'aquest temps. Això passa perquè la seva funció filla inicialitza a 0 tots els valors de la matriu.

• Observeu la funció fillRecommendationMatrix. Quines son les línies/crides que més temps "ocupen" en l'execució del codi?

R: Les crides que més temps ocupen són lookForUser i lookColForMovie, sparse matrix (molts més 0 que valoracions).

• La funció anterior crida a les funcions lookRowForUser i lookColForMovie. Quina és la funció que fa "alentir" aquestes dues funcions.

R: La funció que fa alentir aquestes dues funcions és el sprintf(), que guarda la string en el buffer.

• Què opineu de la funció getMoviesFromFile? Quines són les línies/crides que més temps "ocupen" en l'execució del codi?

R: Les crides que més temps ocupen són getRating (8,33%), fgets (2,92%) i lineIsMovie (2,43%).

• Quins valors (mitjans) d'execució real, d'usuari i de sistema us surten? Per què la suma del temps d'usuari i de sistema dóna, aproximadament, el temps real d'execució?

R: Ens ha sortit els següents valors:

```
real    0m14.934s
user    0m12.916s
sys     0m2.004s
```

El temps real és el temps de rellotge que passa des que executem la comanda fins que acaba. El temps d'usuari és la quantitat de temps que la CPU ha gastat en mode usuari (fora del nucli) dins del procés, durant l'execució. I el temps de sistema és la quantitat de temps que la CPU es passa al nucli dins del procés.

Amb el que hem definit podem veure que la suma del temps d'usuari i de sistema tenim el temps total que tarda el procés, és a dir el temps real.

Primera modificació: Modificar el codi de forma que s'inicialitzin els valors dels elements de la matriu fent servir punters i un únic bucle.

```
int size = numberOfUsers*numberOfMovies;
  for(int i=0; i < size; i++, grid++) {
    *grid = 0;
  }
```

• Mesureu l'eficiència de la implementació del nou codi fent servir el callgrind/kcachegrind i compareu els resultats obtingus amb el codi original a la secció 3.1.2. Hi ha hagut millora? Segons callgrind, és útil fer servir punters per recórrer els elements de la matriu de recomanació?

R: Si, ara la funció createEmptyRecommendationMatrix tarda molt menys (17,62%) en executar-se a causa de l'alteració feta. Hi ha hagut millora, com ens indica el callgrind, és més eficient fer servir punters amb un únic bucle que accedir un array amb dos bucles per inicialitzar els valors de la matriu a 0.

• Mesureu el temps d'execució del codi amb la instrucció time. Com han canviat els valors del temps d'execució respecte a els valors obtinguts amb el codi original? Raoneu quins han canviat i per què d'altres no ho fan.

R: Ens ha sortit els següents valors:

```
real    0m12.695s
user    0m10.517s
sys     0m2.133s
```

Podem observar que el temps d'execució ha baixat, en concret el temps d'usuari.

Segona modificació: En comptes de fer servir punters es farà servir la funció **memset** de la llibreria d'usuari per inicialitzar tots els valors de la matriu a zero (amb una única crida a memset). De nou, es demana fer l'anàlisi feta fa un moment:

```
int size = numberOfUsers*numberOfMovies;

memset(grid,0,size);
```

• Mesureu l'eficiència de la implementació del nou codi fent servir el callgrind/kcachegrind i compareu els resultats amb els obtinguts amb el codi original a la secció 3.1.2 així com el codi que fa servir punters. Hi ha hagut millora?

R: Si, el que primer ens sobta és el percentatge de temps que tarda el createEmptyRecommendationMatrix comparat amb abans on tardava 46,88% sense cap modificació i 17,62% amb punters i ara ha passat a 2,71%. Podem observar com hi ha hagut millora respecte a els procediments anteriors.

• Mesureu el temps d'execució del codi amb la instrucció time. Com han canviat els valors del temps d'execució respecte a els valors obtinguts abans? Raoneu quins han canviat i per què d'altres no ho fan.

R: Ens ha sortit els següents valors:

```
real    0m10.646s
user    0m8.607s
sys     0m2.031s
```

Podem observar que el temps d'execució ha baixat, en concret el temps d'usuari, de la mateixa manera que ho havia fet amb la primera modificació.

2.2. Operacions d'entrada-sortida

• Mesureu la mida de la matriu (nombre de columnes, nombre de files) i calculeu el que ocupa la matriu a memòria. Coincideix amb la mida del fitxer matrix.bin que es genera?

R:

Files: 442701

Columnes: 1798

Mida que ocupa: Files*columnes*4 = 3.183.905.592 bytes ± 3GiB

Matrix.bin : 3.183.905.592 bytes ± 3GiB

Com podem veure sí que coincideix la mida amb el que ocupa el fitxer matrix.bin.

• Quin és el temps d'execució del codi original (sense les optimitzacions de CPU)? Què observeu al codi? Ha canviat el temps d'usuari? I el de sistema? I el temps real? Raoneu

per què el temps d'execució real és més gran que la suma del temps d'usuari i de sistema? Recordar que cal executar la instrucció time en un Linux natiu.

R: Ens ha sortit els següents valors:

```
real    2m29.117s
user    0m17.482s
sys     0m15.264s
```

Ens ha semblat estrany aquest resultat, però com ens ha comentat amb en Lluis a classe de pràctiques deu ser a causa les limitacions de l'ordinador (memòria i disc dur), i ho veiem amb la diferència de la suma entre user i sys amb el que dóna real, que a priori hauria de ser igual com passa amb la resta d'exercicis on calculem el temps.

Modificació: Es proposa modificar el codi perquè els elements de la matriu siguin chars en comptes de floats

• Quan ocupa (teòricament) la matriu si es fan servir chars en comptes de floats per emmagatzemar cadascun dels elements de la matriu?

R: Anteriorment per calcular la mida havíem multiplicat per 4, al ser el que ocupa un float. Ara en fer servir chars que ocupen 1 byte només hem de multiplicar files per columnes i ens dóna 795.976.398 bytes.

• Mesureu el temps d'execució del codi. Com han canviat els valors del temps d'execució respecte a els valors obtinguts amb la matriu de floats? Raoneu la resposta

R: Ens ha sortit els següents valors:

```
real    0m18.259s
user    0m8.720s
sys     0m1.991s
```

• Quines conclusions traieu dels experiments realitzats?

La conclusió que trèiem és que per molt que un programa funcioni, encara se'l pot millorar i fer més eficient. I en el nostre cas, ens és més ràpid i eficient construir la matriu amb la funció memset i treballar amb chars en lloc de floats.

3. Conclusions

En aquesta pràctica hem vist com analitzar un programa que s'executa de forma seqüencial i hem après eines que ens han permès veure quines eren les parts del programa més lentes en executar-se. A més hem vist la manera de fer aquestes parts més ràpides en el programa que se'ns ha donat.