

시스템 프로그래밍 과제 2 report

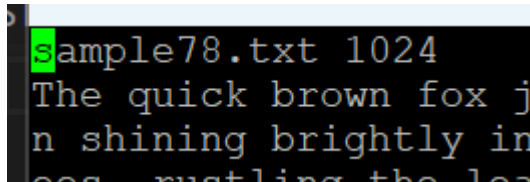
Email: jhss4475@dgist.ac.kr

202111133 이석민

A. Brief explanation about your implementation strategies for Part 2.

1. pack function

- 먼저 arcx.c file의 첫 부분에는 pack 함수가 정의되어 있습니다. pack 함수는 src directory에 있는 모든 파일들을 읽어서 하나의 archive file에 저장합니다.
- 만약 src directory안에 하위 directory가 있다면, 코드 23~25줄에 의해 무시됩니다.
- src directory에 있는 파일이 archive file로 옮겨질때, file은 file header 부분과 file context부분으로 나누어져서 옮겨지게 됩니다.
- file header는 밑의 그림과 같이, {파일 이름 + 공백 + 파일크기}로 이루어져 있습니다.



<그림 1 : file header>

- file context부분은 src directory에 있는 file의 내용을 그대로 archive file에 저장합니다.
- 즉 archive file은 header와 context가 번갈아 나오는 file이 됩니다. 이때 header와 context는 모두 "\n"으로 구분되어 있습니다.

2. unpack function

- arcx.c file의 두 번째 부분에는 unpack함수가 정의되어 있습니다. 이 함수는 archive file에 zip되어있는 파일들을 source directory에 저장합니다.
- 이 함수에서는 fgets함수를 이용하여 file header를 line변수에 저장하고, sscanf함수를 이용하여 file header에서 filename과 filesize를 읽어옵니다.
- 이후 해당 file의 context를 archive file에서 읽어서 source directory에 해당 파일을 만들어 저장합니다.
- context를 read하는 과정에서 해당 file의 크기만큼archive file의 open file table의 file offset이 증가되어 있으므로, 다음 fgets함수에서는 다음 file header가 읽히게 됩니다.
- 이를 반복하면 unpack을 완벽하게 수행할 수 있습니다.

3. add function

- add function은 target file을 archive file에 추가합니다.
- archive file의 마지막 부분에 추가할 target file의 header와 context를 추가하는 방식으로 작동됩니다.
- 코드의 110 ~ 120 부분에서는 미리 archive file에 존재하는 file과 같은 이름의 target file이 들어오면, 프로그램을 중단하고 에러메세지를 출력해줍니다.

4. delete function

- del function은 target file을 archive file에서 삭제합니다.
- 먼저 코드의 161~174줄을 보면, archive file에 없는 file을 삭제하라고 할 때에는 프로그램을 중단하고 에러메세지를 출력해줍니다.
- 이외의 정상적인 상황에서는 새로운 new_archive_file을 만들고 new_archive_file에 archive file로부터 삭제할 파일의 내용을 제외하고 복사해줍니다.
- 이후 원래의 archive file을 삭제하고 new_archive_file의 이름을 원래의 archive file의 이름으로 바꿔줍니다.

5. list function

- list function은 archive file에서 file header의 정보만 읽어서 출력해주는 함수입니다.
- 이는 unpack function과 비슷하게 작동합니다.
- 단, unpack function에서 file context를 읽었던 것을 list함수에서는 context를 읽는 것이 목표가 아니라 file offset만 증가시키면 되므로, 간단히 fseek 함수를 이용해서 구현했습니다.
- 즉 file header를 읽고, file context부분은 pass하고, file header부분을 읽는 것을 반복적으로 수행합니다.
- file header의 내용을 출력해줍니다.

B. PMU measurement results including

Case1: When packing 100 files, where each file is 1MB.

- 밑의 그림을 보면, 100개의 1MB sample이 ~/test/realsample directory에 있는 것을 확인할 수 있습니다. 위 샘플들은 노트북에서 제작하여 github를 통해 clone하여 라즈베리파이에서 사용할 수 있도록 했습니다.

sample1	2023-05-30 오후 8:03	텍스트 문서	1,024KB
sample2	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample3	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample4	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample5	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample6	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample7	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample8	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample9	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample10	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample11	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample12	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample13	2023-05-30 오후 8:05	텍스트 문서	1,024KB
sample14	2023-05-30 오후 8:05	텍스트 문서	1,024KB

<그림 2 : 1MB 파일들, 노트북>

```
seokmin@raspberrypi:~/test/realsample $ ls
sample100.txt  sample28.txt  sample46.txt  sample64.txt  sample82.txt
sample10.txt   sample29.txt  sample47.txt  sample65.txt  sample83.txt
sample11.txt   sample2.txt   sample48.txt  sample66.txt  sample84.txt
sample12.txt   sample30.txt  sample49.txt  sample67.txt  sample85.txt
sample13.txt   sample31.txt  sample4.txt   sample68.txt  sample86.txt
sample14.txt   sample32.txt  sample50.txt  sample69.txt  sample87.txt
sample15.txt   sample33.txt  sample51.txt  sample70.txt  sample88.txt
sample16.txt   sample34.txt  sample52.txt  sample71.txt  sample89.txt
sample17.txt   sample35.txt  sample53.txt  sample72.txt  sample90.txt
sample18.txt   sample36.txt  sample54.txt  sample73.txt  sample91.txt
sample19.txt   sample37.txt  sample55.txt  sample74.txt  sample92.txt
sample1.txt    sample38.txt  sample56.txt  sample75.txt  sample93.txt
sample20.txt   sample39.txt  sample57.txt  sample76.txt  sample94.txt
sample21.txt   sample3.txt   sample58.txt  sample77.txt  sample95.txt
sample22.txt   sample40.txt  sample59.txt  sample78.txt  sample96.txt
sample23.txt   sample41.txt  sample60.txt  sample79.txt  sample97.txt
sample24.txt   sample42.txt  sample61.txt  sample80.txt  sample98.txt
sample25.txt   sample43.txt  sample62.txt  sample81.txt  sample99.txt
sample26.txt   sample44.txt  sample63.txt  sample82.txt  sample9.txt
sample27.txt   sample45.txt  sample64.txt  sample83.txt  sample0.txt
```

<그림 3 : 1MB파일들, 라즈베리파이>

- 밑의 그림을 보면, make 명령어로 arcx.c가 arcx로 잘 compile된 것을 확인할 수 있습니다.

```
seokmin@raspberrypi:~/test $ ls
arcx.c  bye  Makefile  prac2  realbye  spassignsample  unpack_  yeah
bench.sh  hh  pmuon.ko  prac4  realendunpack  test  unpackfile
seokmin@raspberrypi:~/test $ make
gcc -o arcx arcx.c
seokmin@raspberrypi:~/test $ ls
arcx  bye  pmuon.ko  realbye  test  yeah
arcx.c  hh  prac2  realendunpack  unpack_
bench.sh  Makefile  prac4  spassignsample  unpackfile
seokmin@raspberrypi:~/test $
```

<그림 4 : 성공적 compile>

- 밑의 그림은 100개의 file을 pack 할때의 PMU measurement를 측정한 것입니다.

```
seokmin@raspberrypi:~/test $ ./bench.sh ./arcx pack pack.arcx realsample
vm.drop_caches = 3
successfully pack files
[ 4892.403929] Turn PMU on Core 2
[ 4892.403929] Turn PMU on Core 1
[ 4892.403929] Turn PMU on Core 3
[ 4892.403925] Turn PMU on Core 0
[ 4892.403948] We have 6 configurable event counters on Core 1
[ 4892.403948] We have 6 configurable event counters on Core 0
[ 4892.403949] We have 6 configurable event counters on Core 3
[ 4892.403954] We have 6 configurable event counters on Core 2
[ 4892.403976] Ready to use PMU
[ 4892.762496] sysctl (1707): drop_caches: 3
[ 4898.780773] PMU Kernel Module Off
[ 4898.780801] [Core 1] Insts: 342000156, Misses: 600531, Refs: 118385123, Cycles: 258860795
[ 4898.780803] [Core 0] Insts: 5486444607, Misses: 6746629, Refs: 2167022741, Cycles: 4541104278
[ 4898.780803] [Core 3] Insts: 201771810, Misses: 219363, Refs: 74275072, Cycles: 157099583
[ 4898.780804] [Core 2] Insts: 44057850, Misses: 302027, Refs: 17038356, Cycles: 55585155
```

<그림 5 : pack PMU result>

CASE 2: When unpacking the archive file having 100 files.

- 밑의 그림은 unpack할 때의 PMU Measurement를 측정한 것입니다.

```
seokmin@raspberrypi:~/test $ ./bench.sh ./arcx unpack pack.arcx real_unpack
vm.drop_caches = 3
successfully unpacked
[ 5734.970420] Turn PMU on Core 1
[ 5734.970416] Turn PMU on Core 0
[ 5734.970420] Turn PMU on Core 2
[ 5734.970420] Turn PMU on Core 3
[ 5734.970438] We have 6 configurable event counters on Core 0
[ 5734.970438] We have 6 configurable event counters on Core 2
[ 5734.970438] We have 6 configurable event counters on Core 1
[ 5734.970443] We have 6 configurable event counters on Core 3
[ 5734.970467] Ready to use PMU
[ 5735.296943] sysctl (1821): drop_caches: 3
[ 5737.816746] PMU Kernel Module Off
[ 5737.816767] [Core 1] Insts: 31147699, Misses: 166472, Refs: 11672801, Cycles: 46088928
[ 5737.816768] [Core 2] Insts: 56635548, Misses: 360091, Refs: 21572733, Cycles: 71603191
[ 5737.816768] [Core 3] Insts: 27281315, Misses: 230834, Refs: 10728592, Cycles: 38670132
[ 5737.816768] [Core 0] Insts: 1204910578, Misses: 4741812, Refs: 450581693, Cycles: 1289061976
```

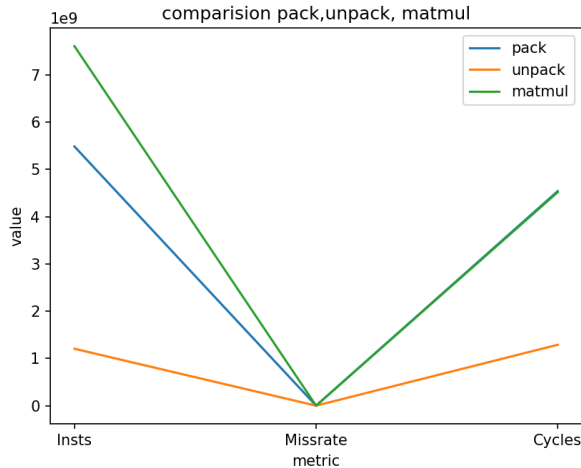
<그림 6 : unpack PMU measurement>

- 밑의 그림은 provided된 matmul의 PMU Measurement를 측정한 것입니다.

```
seokmin@raspberrypi:~/team/hw2_resource $ ./bench.sh ./matmul
vm.drop_caches = 3
[ 5875.826708] Turn PMU on Core 0
[ 5875.826708] Turn PMU on Core 2
[ 5875.826708] Turn PMU on Core 1
[ 5875.826705] Turn PMU on Core 3
[ 5875.826727] We have 6 configurable event counters on Core 0
[ 5875.826728] We have 6 configurable event counters on Core 1
[ 5875.826727] We have 6 configurable event counters on Core 2
[ 5875.826732] We have 6 configurable event counters on Core 3
[ 5875.826754] Ready to use PMU
[ 5876.043594] sysctl (1970): drop_caches: 3
[ 5878.705072] PMU Kernel Module Off
[ 5878.705094] [Core 2] Insts: 44882676, Misses: 325959, Refs: 17225558, Cycles: 55827600
[ 5878.705095] [Core 3] Insts: 63413520, Misses: 509754, Refs: 24620654, Cycles: 78020485
[ 5878.705095] [Core 0] Insts: 7610172087, Misses: 3062866, Refs: 3396995722, Cycles: 4519941464
[ 5878.705097] [Core 1] Insts: 82875661, Misses: 230246, Refs: 28206072, Cycles: 77615922
```

<그림 7 : PMU Measurement of matmul>

i. Provide a simple discussion that compares each case with the matmul program provided.



<그림 8 : comparision pack,unpack,matmul>

- miss rate가 단위문제로 그래프상에서 3가지 경우가 모두 비슷하게 출력되는데, 각각의 miss rate는 (pack, unpack, matmul) = (0.3%, 1%, 0.9%)입니다.

- pack function은 1024*1024 Byte 파일 100개를 읽습니다. 또한 약 1024*1024*100 Byte 파일을 작성합니다. load와 write를 전부 합해서 약 $2 \times 1024 \times 1024 \times 100$ 번, 즉 100×2^{21} 번을 실행합니다.

- unpack function은 약 1024*1024*100 Byte 파일을 한개를 읽습니다. 또한 1024*1024 Byte 파일 100개를 작성합니다. load와 write를 전부 합해서 약 $2 \times 1024 \times 1024 \times 100$ 번, 즉 100×2^{21} 번을 실행합니다.

- matmul function은 load와 write를 전부 합해서 약 $100 \times 128 \times 128 \times 128 \times 3$ 번, 즉 $100 \times 3 \times 2^{21}$ 번을 실행합니다. 단, 피연산자가 float이므로, 한번에 4Byte를 읽거나 쓰게 됩니다.

- pack과 unpack의 instruction개수와 cycle, miss rate가 차이나는 이유는, 제가 arcx.c를 구현할때, pack function은 source file들의 내용을 1byte씩 읽고, archive file에 1 byte씩 작성한 반면, unpack function은 source file들의 내용을 archive file에서 1024*1024 byte씩 읽고, source file에 1024*1024 byte씩 작성해서 차이가 발생하는 것 같습니다.

- 물론 내부적으로 한번에 1024*1024byte씩 한번에 읽지는 못하겠지만, 의도적으로 계속 1byte씩 읽는것과 자동으로 k byte씩 읽는 것에서 instruction개수와 cycle, miss rate의 차이가 발생한 것 같습니다.

- 왜 다르게 구현했는지 궁금해 하실 수 있을 것 같아 설명드리면, 처음에 전부 1byte 씩 읽고 쓰는것으로 구현했는데, 왜인지 모르게 unpack함수에서 그렇게 했을 때 버그가 발생해서 1024*1024 byte를 한번에 읽는 것으로 고쳤습니다. 아직 왜 그런지 이유를 발견하지는 못했습니다.

- matmul의 경우에는 한 번에 처리하는 data가 4byte 이기 때문에 pack function에 비해서 cache miss rate가 높은 것을 볼 수 있습니다.

- 크기가 16byte인 block이 있다고 가정하면, 연속적인 값을 읽어올 때, data의 크기가 1byte이면 miss = 1/16이지만, data의 크기가 4Byte이면 1/4 이기 때문입니다.

- 한 word(메모리에서 한번에 읽어오는 data의 크기)의 크기가 4Byte로 알고있는데, 아마 unpack function을 수행할때, 자동으로 word크기인 4Byte만큼 한 instruction에서 읽어온 것 같습니다. 따라서 miss rate가 unpack과 matmul에서 비슷하게 출력되었다고 생각합니다.

ii. You can also discuss whatever you want by testing more cases.

- 위에서 만들었던 pack.arcx file에 1MB file을 add 했을 때의 PMU 출력 결과입니다.

```
seokmin@raspberrypi1:~/test $ ./bench.sh ./arcx add pack.arcx smp.txt
vm.drop_caches = 3
successfully add file
[ 8828.396176] Turn PMU on Core 0
[ 8828.396179] Turn PMU on Core 1
[ 8828.396179] Turn PMU on Core 3
[ 8828.396178] Turn PMU on Core 2
[ 8828.396196] We have 6 configurable event counters on Core 0
[ 8828.396196] We have 6 configurable event counters on Core 3
[ 8828.396196] We have 6 configurable event counters on Core 2
[ 8828.396202] We have 6 configurable event counters on Core 1
[ 8828.396224] Ready to use PMU
[ 8828.621817] sysctl (1966): drop_caches: 3
[ 8828.855450] PMU Kernel Module Off
[ 8828.855471] [Core 3] Insts: 27802083, Misses: 167153, Refs: 10475961, Cycles: 35043344
[ 8828.855473] [Core 2] Insts: 66233284, Misses: 500837, Refs: 25630641, Cycles: 86570985
[ 8828.855473] [Core 1] Insts: 107419197, Misses: 455072, Refs: 37661513, Cycles: 103126875
[ 8828.855473] [Core 0] Insts: 48590460, Misses: 254580, Refs: 18285422, Cycles: 59804021
```

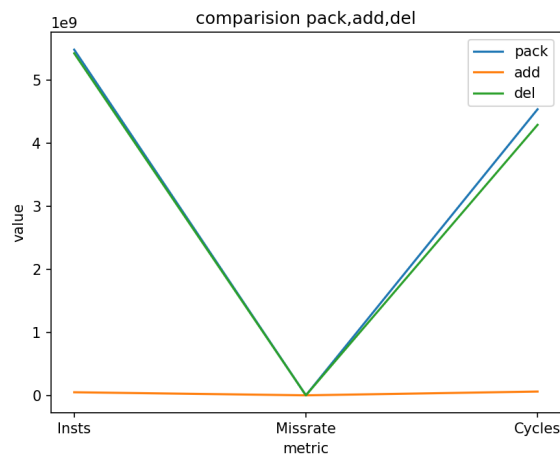
<그림 9 : add>

-위에서 만들었던 pack.arcx file에서 1MB file을 delete했을 때의 출력 결과입니다.

```
seokmin@raspberrypi:~/test $ ./bench.sh ./arcx del pack.arcx smp.txt
vm.drop_caches = 3
successfully delete the file from archive file
[ 9045.470146] Turn PMU on Core 0
[ 9045.470147] Turn PMU on Core 2
[ 9045.470145] Turn PMU on Core 3
[ 9045.470148] Turn PMU on Core 1
[ 9045.470168] We have 6 configurable event counters on Core 3
[ 9045.470168] We have 6 configurable event counters on Core 2
[ 9045.470169] We have 6 configurable event counters on Core 1
[ 9045.470172] We have 6 configurable event counters on Core 0
[ 9045.470198] Ready to use PMU
[ 9045.641777] sysctl (1984): drop_caches: 3
[ 9048.217737] PMU Kernel Module Off
[ 9048.217757] [Core 1] Insts: 32381074, Misses: 194289, Refs: 12149520, Cycles: 47270612
[ 9048.217758] [Core 0] Insts: 5428646474, Misses: 6913420, Refs: 2161479286, Cycles: 4293916795
[ 9048.217758] [Core 3] Insts: 134037824, Misses: 640520, Refs: 48118980, Cycles: 150271252
[ 9048.217760] [Core 2] Insts: 21546431, Misses: 140519, Refs: 8085269, Cycles: 34891818
```

<그림 10 : del>

- 밑의 그래프는 결과를 분석한 그래프입니다.



- add를 보면, archive file 끝에 한 파일을 작성 할 뿐이므로 instruction개수와 cycle개수가 압도적으로 적은 것을 볼 수 있습니다.

- 반면, del을 보면, archive file을 한번 전부 읽은 다음 새로 archive file을 작성하므로, instruction과 cycle개수가 pack과 거의 동일함을 알 수 있습니다.

감사합니다.