시스템프로그래밍 5번째 과제: "최적화 실습: 성능 측정 및 논의"

소프트웨어학과 32203660 이현정

1. 소개

이용한 프로그램

: 수업시간에 했던 combine1~6의 내용을 참고해서 소프트웨어적으로 성능이 다른 프로그램을 만들어 보았습니다. 해당 프로그램은 인자로 "add"와 "mul" 중 1개와 해당하는 연산을 수행할 값들을 char * argv로 전달하면, 주어진 인자에 따라서 받은 값들에 대한 덧셈 또는 곱셈을 시행하는 것으로 설정하였습니다.

총 6개의 프로그램으로 ex1 ~ ex6까지로 프로그램의 내용 중 일부를 수정하여 성능에 영향을 줄 수 있도록 구성해보았습니다. 성능은 main함수의 시작부터 결과를 출력하기 위해 저장하는 부분에 대해서 시간을 재기 위해서 "gettimeofday"를 이용하였습니다.

2. 본문

각 프로그램에 대한 세부 설명으로 앞 프로그램에 비해 변화시킨 점을 위주로 기술하였습니다.

2-1) ex1

```c
/*Performance Comparison Program 1 , HyeonJoeng Lee(32203660@dankook.ac.kr) */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

int add(int n, int x) { n = n+x; return n; }

int mul(int n, int x) { n = n*x; return n; }

int main(int argc, char* argv[]) {
  struct timeval stime, etime, gap;
  gettimeofday(&stime, NULL);
  if (argc < 2 || argc > 20) { printf("Wrong number of arguments!\n"); exit(1); }

  int result;
  if (!strcmp(argv[1], "add"))
    result = 0;
  else if (!strcmp(argv[1], "mul"))
    result = 1;
```

```
  for (int i=2; i < argc; i++) {
    if (!strcmp(argv[1], "add"))
      result = add(result, atoi(argv[i]));
    else if (!strcmp(argv[1], "mul"))
      result = mul(result, atoi(argv[i]));
  }

  gettimeofday(&etime, NULL);
  gap.tv_sec = etime.tv_sec - stime.tv_sec;
  gap.tv_usec = etime.tv_usec - stime.tv_usec;
  if (gap.tv_usec < 0) {
    gap.tv_sec = gap.tv_sec-1;
    gap.tv_usec = gap.tv_usec + 1000000;
  }
  printf("time %ldsec : %ldusec\n", gap.tv_sec, gap.tv_usec);
  printf("result >> %d\n", result);
  return 0;

}
```

: 받은 인자에 대해서 초기값을 덧셈이면 0으로 설정하고, 곱셈인 경우에는 1로 설정합니다. 그리고 반복문을 돌면서 주어진 값들에 대한 연산을 한 뒤, 결과를 계속 지역변수에 저장합니다. 그리고 다시 반복문을 돌아가면, 해당 지역변수를 이용해서 값들이 누적되어서 계산되게 하였고, 해당하는 내용을 통해 모든 인자에 대한 결과를 만드는 작업의 시간을 측정할 수 있게 구성하였습니다.

2-2) ex2

```
  int result, state;
  if (!strcmp(argv[1], "add"))
    result = 0, state = 0;
  else if (!strcmp(argv[1], "mul"))
    result = 1, state = 1;

  for (int i=2; i < argc; i++) {
    if (state == 0)
      result = add(result, atoi(argv[i]));
    else if (state == 1)
      result = mul(result, atoi(argv[i]));
  }
```

: ex1에서는 반복문의 조건문에 들어갈 필요 없이 이미 정해진 값이 있는 부분이 있었습니다. 해당하는 부분을 제거하면, 반복문 내부에서 연산이 수행될 때마다 정해진 부분에 대해 계산 후 확인을 해야 하는데, 계산하는 부분을 반복문 밖으로 내보내서 1번 계산 한 결과를 state 변수에 저장하고 반복문 내부에서는 state 값이 무엇인지에 따라서 원하는 계산을 실행할 수 있도록 구성하였습니다.

2-3) ex3

```
if (state == 0) {
  for (int i=2; i < argc; i++) {
    result = add(result, atoi(argv[i]));
  }
}
else if (state == 1) {
  for (int i=2; i < argc; i++) {
    result = mul(result, atoi(argv[i]));
  }
}
```

: ex2에 대해서 더 생각을 해보면, 반복문 내부에 조건문이 있기 때문에 반복할 때마다 확인이 필요합니다. ex2의 방식에서 반복문과 조건문의 위치를 반대로 조건문에 따라 반복문이 돌아가게 하면 1번의 확인만으로 조건문이 해당 내용에 따라 계속 돌아갈 수 있게 되었습니다.

2-4) ex4

```
if (state == 0) {
  for (int i=2; i < argc; i++) {
    result += atoi(argv[i]);
  }
}
else if (state == 1) {
  for (int i=2; i < argc; i++) {
    result *= atoi(argv[i]);
  }
}
```

: main내에서 직접 연산을 하도록 구성하면, ex3에서는 함수를 통해 덧셈이나 곱셈을 하도록 만들어서 함수를 호출하고, 함수의 결과를 저장하는 등의 작업을 줄일 수 있습니다.

2-5) ex5

```
if (state == 0) {
  for (int i=2; i < argc-1; i+=2)
    result = result + atoi(argv[i]) + atoi(argv[i+1]);
}
else if (state == 1) {
  for (int i=2; i < argc-1; i+=2)
    result = result * atoi(argv[i]) * atoi(argv[i+1]);
}
```

: 반복문이 1번 돌아갈 때 2개 이상의 정수 연산이 작동하도록 구성해서 분기 예측으로 다음 반복문이 실행되는 과정에서 정수형 연산의 개수로 인해서 미뤄야 했던 한계를 극복할 수 있게 되기에 시간을 줄을 것이라고 예상을 했습니다.

2-6) ex6

```
if (strcmp(argv[1], "add") != 0 && strcmp(argv[1], "mul") != 0)
  exit(1);

int result = 1;
if (!strcmp(argv[1], "add")) {
  result = 0;
  for (int i=2; i < argc; i++)
    result += atoi(argv[i]);
}
else {
  result = 1;
  for (int i=2; i < argc; i++)
    result *= atoi(argv[i]);
}
```

: if-else는 실행할지 말지 확인하는 단계를 거쳐야 하므로 시간이 소비되어 최대한 조건 확인을 줄여서 작동할 수 있게 구성하는 것이 성능 향상에 도움을 줄 것이라고 생각하였습니다. 그래서 우선 덧셈이나 곱셈 중 하나에 해당하는 연산이 맞는지 확인하고, state라는 지역변수로 덧셈인지 곱셈인지 연산을 확인하는 대신에 각각을 확인해서 초기화하는 동시에 해당하는 반복문이 동작할 수 있게 구성하였습니다.

3. 결론

앞 선 프로그램의 성능과 비교해서 변화된 점을 기술하고, 예상과 다른 경우 제가 생각하는 원인을 작성하였습니다.

3-1) ex1 vs ex2

```
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex1 add  1 2 3 4
time 0sec : 23usec
result >> 10
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex2 add  1 2 3 4
time 0sec : 20usec
result >> 10
```

반복문 내부 조건문에서의 연산이 이뤄져야하는 부분이 strcmp() 없이 비교만으로 계산이 될 수 있게 구성을 하였기 때문에 해당 함수가 돌아가는 시간을 절약할 수 있었습니다.

3-2) ex2 vs ex3

```
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex2 add  1 2 3 4
time 0sec : 25usec
result >> 10
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex3 add  1 2 3 4
time 0sec : 19usec
result >> 10
```

반복문 내에서는 조건을 확인하지 않도록 구성해서 성능을 높일 수 있었습니다.

3-3) ex3 vs ex4

```
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex3 add  1 2 3 4
time 0sec : 27usec
result >> 10
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex4 add  1 2 3 4
time 0sec : 19usec
result >> 10
```

함수 이용에 대한 부분을 줄여서 성능을 높였습니다.

3-4) ex4 vs ex5

```
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex4 add  1 2 3 4 5 6
time 0sec : 20usec
result >> 21
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex5 add  1 2 3 4 5 6
time 0sec : 19usec
result >> 21
```

```
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex4 add  1 2 3 4 5
time 0sec : 19usec
result >> 15
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex5 add  1 2 3 4 5
time 0sec : 20usec
result >> 15
```

반복문이 6번 동작되도록 인자를 주고 1번 돌아갈 때 2번의 덧셈이나 곱셈이 작동되게 만들어보니, 인자가 짝수인 경우에는 ex4보다 ex5에서 성능이 좋았습니다.

하지만, 인자가 5개인 홀수인 경우에는 2개씩 계산하다 보니 마지막 인자에 대한 연산을 하기 위해서 따로 인자의 수를 확인 후 해당하는 연산을 시행해야 해서 시간이 ex5에서 ex4에 비해 더 오래 걸려서 성능이 도리어 나빠질 수 있다는 사실을 확인하였습니다.

```
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex4 add  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
time 0sec : 28usec
result >> 120
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex5 add  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
time 0sec : 22usec
result >> 120
```

하지만 인자 수가 많은 경우에는 반복문에서 훨씬 효율적으로 계산이 가능해져서 ex5의 성능이 좋다는 것을 파악하였으며, 어떤 경우가 더 많은지에 미리 판단을 하고서 프로그램이 다르게 작성이 필요하다는 것을 확인하였습니다.

3-5) ex4 vs ex6

```
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex4 add  1 2 3 4 5
time 0sec : 19usec
result >> 15
hyeonjeong@LAPTOP-I49JJJGR:~/project$ ./ex6 add 1 2 3 4 5
time 0sec : 20usec
result >> 15
```

ex6 결과에서는 ex4에 비해서 성능이 낮게 나왔습니다. 코드를 보면 ex4는 strcmp()를 2번 시행하고 if-else문을 2번 시행한 반면, ex6은 strcmp()를 3번 시행하고 if-else를 1번 시행했습니다. 이것을 보면, 함수를 통한 실행이 조건문에서 해당 값이 맞는지 확인하는 것보다 더 많은 시간이 걸려서 함수의 동작이 조건문보다 더 오랜 시간이 걸려서 각 동작에 대한 성능 비교를 통해 결과적으로 높은 성능을 가지게 해야 한다는 사실을 파악할 수 있었습니다.