неудачный выбор приоритетов на PDP-11 и его наследие на Си/Си++

крис касперски, ака мыщъх, no-emal

введение

Отлаживал как-то мыщъх одну свою программу, написанную на Си? и периодически делающую из чисел винегрет или выдающую критическую ошибку access violation при трудно воспроизводимых обстоятельствах. Тщательная проверка исходного текста "глазами" ровным счетом ничего не дала. Программа продолжала выпендриваться, сроки сдачи проекта поджимали, дедлайн нависал над головой Дамокловым мечом, мыщъх нервничал, много курил, нервничал, закидывался ноотропами, не спал ночами, высаживался на жуткую измену, а глубоко укоренившийся баг игнорировал всякие попытки вытащить его из норы.

Под конец, подозрения пали на компилятор и мыщъх, переключивший отладчик в ассемблерный режим, начал шаг за шагом исследовать каждую строчку программы, пока не вышел на конструкцию, компилирующуюся не так, как предполагалась (читай: подсказывал здравый смысл и мои познания языка Си).

источник проблемы

Виновницей оказалась мерзопакостная конструкция типа "`*p[a]++`", которая вопреки логике увеличивает отнюдь _не_ содержимое ячейки, на которую указывает "`*(p+a)`", а значение самого указателя `p`, то есть транслируется в следующий ассемблерный код:

```
mov     eax, dword ptr [p]     ; прочитать адрес переменной p
mov     ecx, dword ptr [eax]   ; прочитать значение указателя, на который указывает p
add     ecx, 1                 ; увеличить содержимое указателя на кот. указывает p
mov     edx, dword ptr [p]     ; прочитать адрес переменной p
mov     dword ptr [edx], ecx   ; занести в переменную p ее обновленное значение
```

Листинг 1 ассемблерный код в который транслируется *p[a]++

Специально написанный для этого дела демонстрационный пример (см. листинг 3) в отладчике выглядел так:

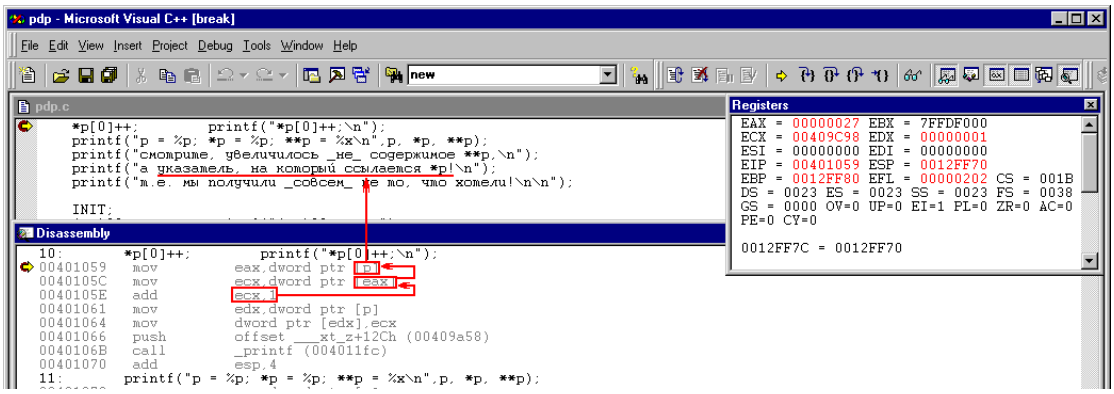

Рисунок 1 откомпилированная конструкция *p[a]++ под лупой отладчика

В то время, как _ожидаемый_ вариант трансляции должен был выглядеть так:

```
mov     eax, dword ptr [p]     ; прочитать адрес переменной p
mov     ecx, dword ptr [eax]   ; прочитать значение указателя на который указывает p
mov     dl, byte ptr [ecx]     ; прочитать значение переменной на которую указывает *p
add     dl, 1                  ; увеличить значение переменной на 1
mov     eax, dword ptr [p]     ; прочитать адрес переменной p
mov     ecx, dword ptr [eax]   ; прочитать значение указателя на который указывает p
mov     byte ptr [ecx], dl     ; занести обновленное значение переменной *p[a]
```

Листинг 2 ожидаемый вариант трансляции конструкции *p[a]++

решение проблемы

Мыщь решил проблему очень просто — явно навязав свое намерение компилятору путем расстановки скобок: "`(*p)[a]++`". Аналогичного результата было можно достичь заменой оператора "++" на оператор "+=" и тогда коварная конструкция принимала вид "`*p[a]+=1`"

причины, следствия или почему так устроен мир?

Выгнав бага из его норы, мыщъх решил провести широкомасштабные археологические раскопки, чтобы добраться до _смысла_ происходящего. Причастность компилятора была отведена сразу, как только остальные компиляторы выдали идентичный результат. Значит, собака зарыта вовсе не в компиляторе, а в самом языке Си.

Странно. Очень странно. Ведь основное кредо Си — краткость. И тут... вдруг такое расточительство! Ведь, чтобы использовать оператор "*" необходимо расставлять скобки, а это — целых два нажатия на Клаву. Зачем? Может быть, есть такие ситуации, где именно такой расклад приоритетов дает выигрыш? Вообще: о чем думали в этот момент разработчики языка? В доступных мне книжках никаких вразумительных объяснений мыщъх так и не нашел.

...прозрение наступило внезапно и причина, как выяснилась, оказалась даже не в самом языке, а... в особенностях косвенной автоинкрементной/автодекрементной адресации процессора PDP-11, из которого, собственно, и вырос Си. Команда "`MOV @(p)+,xxx`" пересылала содержимое `**p` в `xxx`, а затем увеличивала значение `p`. Да! Именно `p`, а отнюдь не ячейки, на которую `**p` ссылается!!!

Так стоит ли удивляться тому, что люди, взращенные на идеологии PDP-11, перенесли ее поведение и на разрабатываемый ими язык?!

демонстрационный пример

Ниже приводится демонстрационный пример, который можно погонять на различных Си/Си++ компиляторов, с неизменностью получая один и тот же результат.

```
main()
{
        char buf; char* p_buf[2]; char **p;

        #define INIT buf=0x66; *p_buf=&buf; *(p_buf+1)=&buf; p=&p_buf;

        INIT;
        printf("char **p;\n");
        printf("p = %p; *p = %p; **p = %x\n\n",p, *p, **p);

        *p[0]++; printf("*p[0]++;\n");
        printf("p = %p; *p = %p; **p = %x\n",p, *p, **p);
        printf("смотрите, увеличилось _не_ содержимое **p,\n");
        printf("а указатель, на который ссылается *p!\n");
        printf("т.е. мы получили _совсем_ не то, что хотели!\n\n");

        INIT;
        (*p)[0]++; printf("(*p)[0]++;\n");
        printf("p = %p; *p = %p; **p = %x;\n",p, *p, **p);
        printf("хорошо, заключаем *p в скобки, тем самым явно\n");
        printf("навязывая компилятору последовательность действий\n\n");

        INIT;
        *p[0]+=1; printf("*p[0]+=1;\n");
        printf("p = %p; *p = %p; **p = %x;\n",p, *p, **p);
        printf("забавно, но замена оператора ++ на оператор +=\n");
        printf("эту проблему как рукой снимает!\n");
}
```

Листинг 3 демонстрационный пример pdp.c

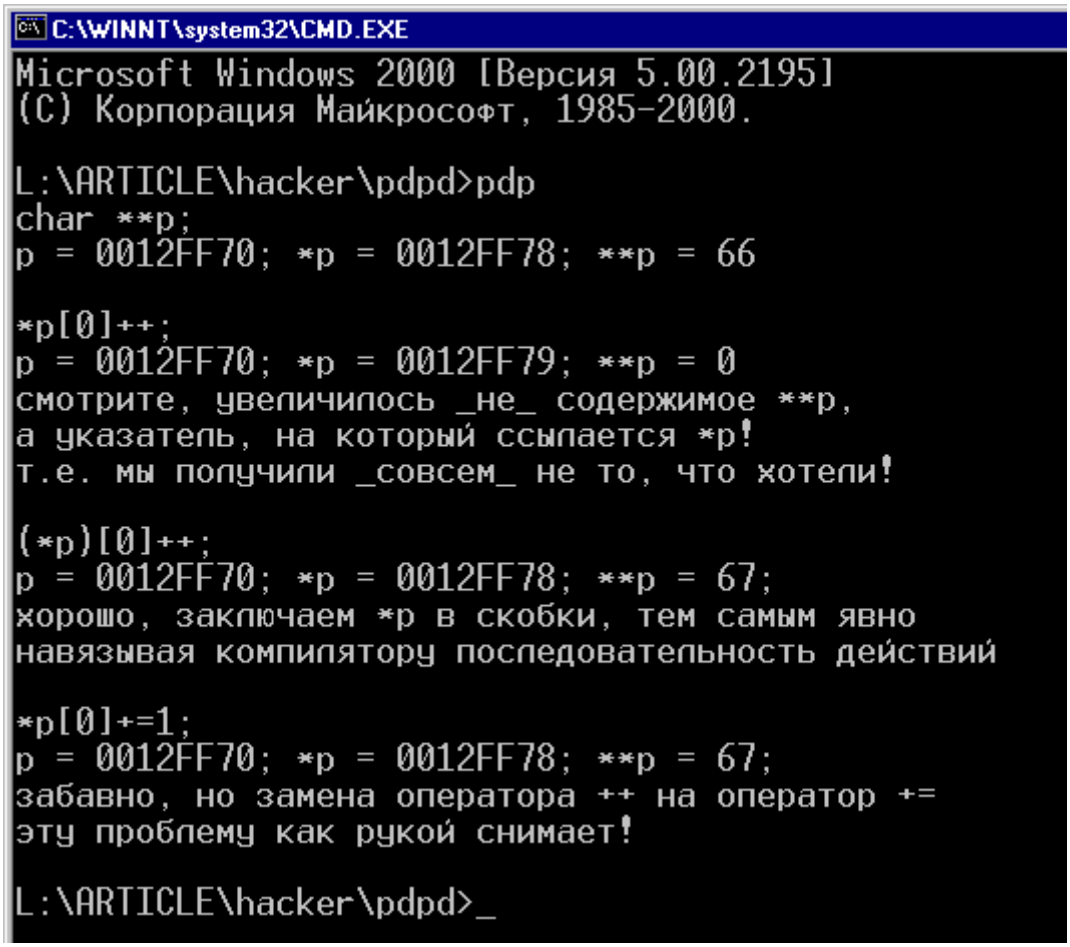

Рисунок 2результат прогона pdp.exe