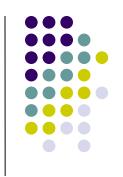




Поравнање података



Поравнање



- За меморијску адресу се каже да је поравната на n бајтова уколико је она умножак броја n.
- Многе физичке архитектуре намећу захтеве за поравнањем адреса одређених објеката зарад потпуне искоришћености могућности које нуде.
- Дакле, приступ и непоравнатим подацима је увек могућ, али није ефикасан.
- Пример: ширина меморије је 8 бита, а меморији је омогућен 32битни приступ али само са адреса које су умножак четворке.



Уобичајена поравнања



- Уобичајена поравнања за 32битну архитектуру х86:
 - char (један бајт) поравнат на 1 бајт.
 - short (два бајта) поравнат на 2 бајта.
 - int (четири бајта) поравнат на 4 бајта.
 - float (четири бајта) поравнат на 4 бајта.
 - double (осам бајтова) поравнат на 8 бајтова под оперативним системом Windows, поравнат на 4 бајта на о.с. Linux (посебном компајлерском опцијом се поравнање подесити на 8 бајтова).
 - показивач (четири бајта) поравнат на 4 бајта

```
char a;
int b;
char c;
short d;
```



Шта ако подаци нису правилно поравнати?

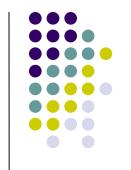


- Компајлер претпоставља правилно поравнање и сам ће се за њега постарати, али у одређеним случајевима програмерском непажњом може доћи до непоравнатог приступа меморији.
- У случају непоравнатог приступа може се десити следеће:
 - Програм неће дати добар резултат
 - Ако приступ меморији иде кроз оперативни систем, онда ће се можда он постарати да се све, иако неефикасније, ипак добро заврши

```
int8_t buffer[4];
int32_t x = *(int32_t*)buffer;
```



Поравнање чланова структуре



- Код структуре чланови морају бити у меморији поређани редом којим су наведени.
- Величина променљиве типа struct S је већа од суме појединачних величина њених чланова.
- То је зато што долази до уметања бајтова између чланова да би се обезбедило њихово одговарајуће поравнање.



Још један пример уметања

- Иако би само са једним уметањем (Pad1) сви елементи понаособ били поравнати, ипак се бајтови умећу и на крају (Pad2) да би се обезбедило правилно поравнање и у случају низа структура.
- Дакле, величина структуре ће увек бити умножак поравнања највећег основног типа њених чланова.



Једнакост структура

```
struct MixedData s1 = {a, b, c};
   struct MixedData s2;
   s2.x = a; s2.y = b; s2.z = c;
if (memcmp(&s1, &s2, sizeof(struct MixedData)) == 0)
   s1 = s2;
if (memcmp(&s1, &s2, sizeof(struct MixedData)) == 0)
  memcpy(&s1, &s2, sizeof(struct MixedData));
  if (memcmp(&s1, &s2, sizeof(struct MixedData)) == 0)
```



Паковање структура



- Баратање са поравнатим структурама и њиховим елементима је брже, али уметнути бајтови повећавају утрошак меморије.
- Смањење утрошка меморије без успорења рада може се остварити променом редоследа елемената у структури.
- Стандард гарантује редослед елемената, тако да компајлер не може самоиницијативно преуредити структуру, али програмер може.
- Међутим, такво преуређење може само донекле смањити величину. Ако желимо још мање меморије да утрошимо, то можемо урадити, али по цену брзине.
- Већини Це компајлера је могуће наредити да упакују елементе структуре на жељено поравнање, нпр. pack (2) значи да компајлер треба да поравна податке на максимално 2 бајта, што значи да уметнута поља неће бити већа од једног бајта.
- Паковање структура се најчешће користи за смањење утрошка меморије, али може се користи и за припрему података за мрежно слање и слично.



Преуређење елемената структуре

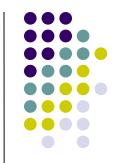


Пример структуре MixedData. Лево са преуређеним елементима, а десно је дата првобитна структура:

Са оваквим преуређењем више није потребно уметнути ни један бајт. Подсећање: char - 1 бајт, short - 2 бајта, int - 4 бајта на x86 Колико бајтова заузимају лева и десна структура?



Паковање структуре

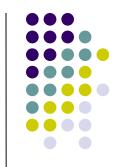


- Замислимо да структура MixedData нема поље Data 2. У том случају ни приликом најбољег редоследа чланова структуре не би могли имати мање од два уметнута бајта.
- Тада се можемо ослонити на могућност коју многи компајлери нуде, али није део стандарда, а то је паковање структура. Већина компајлера (Microsoft, Borland, GNU...) користе #pragma директиве као механизам задавања жељеног паковања.

```
#pragma pack(push)  /* push current alignment to stack */
#pragma pack(1)  /* set alignment to 1 byte boundary */
struct MixedData
{
    char Data1;
    int Data3;
    char Data4;
};
#pragma pack(pop)  /* restore original alignment from stack */
```



GNU __attribute__ проширење



- У GCC-у је уведена нова кључна реч __attribute__ која омогућава да се одређене особине придруже променљивама (укључујући и поља структуре) и функцијама.
- Дат је код који превођен GCC-ом даје исти резултат као и код на претходном слајду:

```
struct MixedData
{
    char Data1;
    int Data3;
    char Data4;
}__attribute__ ((packed));
```

• У случају потребе за посебним поравнањем, које не произилази из потреба за поравнањем основних типова, у GCC-у је могуће искористити aligned атрибут. Њиме се наређује компајлеру да одређену променљиву поравна на задат начин.

```
\_attribute\_ ((aligned (16))) int data = 0;
```

Компајлер ће променљиву data поставити на адресу која је умножак броја 16.

NIT

С11 новине у вези са поравнањем



- Уведено ново стандардно заглавље <stdalign.h>
 - Нуди лепша имена за нове кључне речи _Alignof и _Alignas
- alignof(tip)

```
size_t x = alignof(int);
size t y = alignof(struct {char c; int n;});
```

alignas(tip) или alignas(izraz)

```
alignas(16) int data = 0;
alignas(int) char array[57];
struct alignas(2 * alignof(int)) _s {
  char Data1;
  int Data3;
  char Data4;
}
```

- aligned_alloc();
 - Своди се на основни тип са највећим поравнањем