Jak Windows zarządza pamięcią?

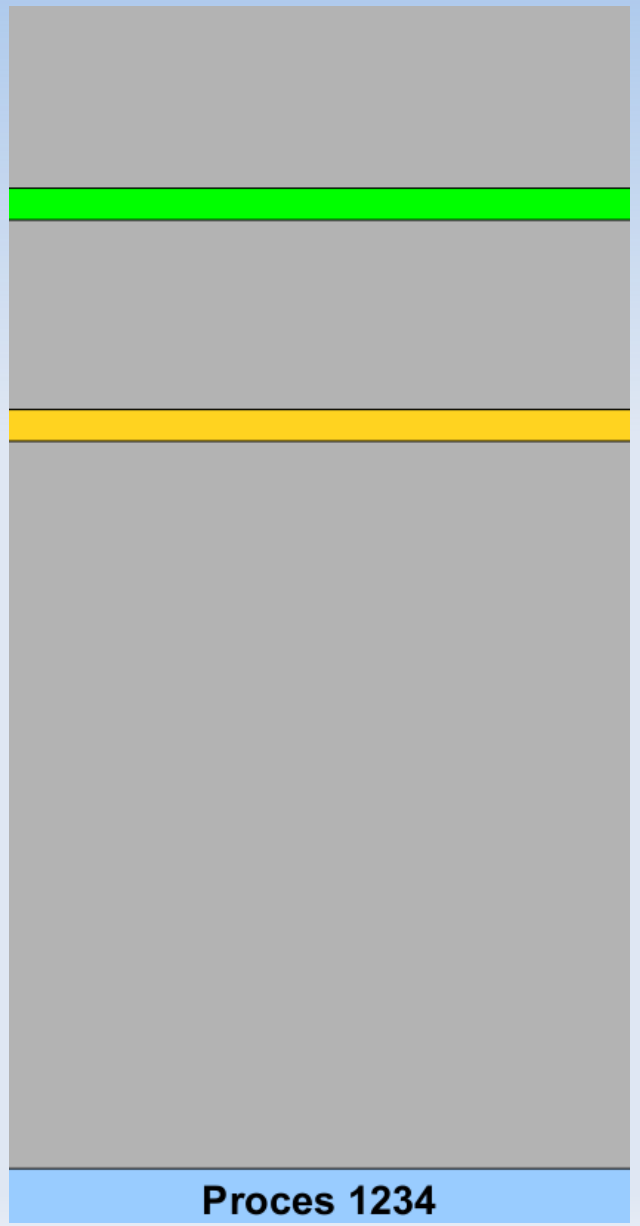
- System Windows definiuje dwa typy pamięci, często mylone przez użytkowników.
- Pamięć fizyczna (pamięc RAM zainstalowana w komputerze)
- Pamięć widziana przez daną aplikację (pamięć wirtualna procesu)
- Do pamięci fizycznej dostęp ma tylko system operacyjny.

Jak Windows zarządza pamięcią?

- Na systemie 32-bitowym możemu zaadresować maksymalnie 4GB pamięci wirtualnej.
- Pamięć jest podzielona na strony (strona ma 4096 bajtów)
-

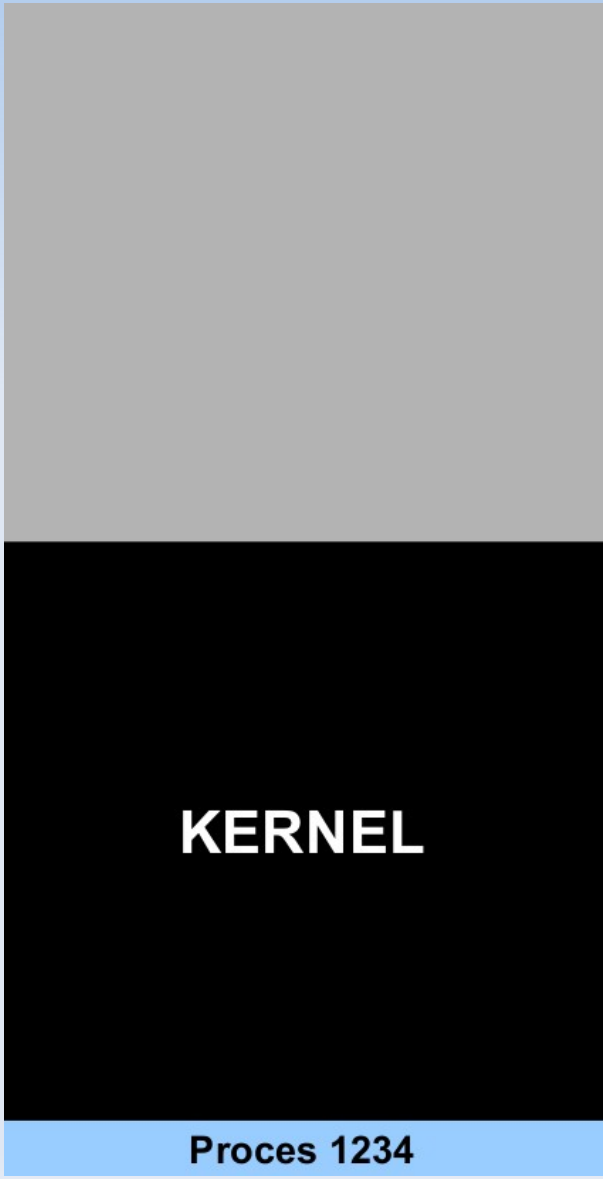
Jak Windows zarządza pamięcią?

- Kolor zielony-strona zamapowana, ma odzwierciedlenie w pamięci fizycznej
- Kolor pomarańczowy-strona zarezerwowana, nie ma jeszcze swojego odzwierciedlenia w pamięci fizycznej
- Kolor szary-strona nieistniejąca



Jak Windows zarządza pamięcią?

- Mimo że przestrzeń adresowa pozwala zaadresować 4GB, proces może wykorzystać jedynie 2GB.
- Do drugiej części pamięci, w ktorej znajduje się kernel, nie mamy dostępu.



Jak Windows zarządza pamięcią?

- Istnieje możliwość zwiększenia pamięci dla procesu do 3GB.
- Aby tego dokonać, należy w pliku boot. Ini podać przełącznik /3G oraz ustawić plikowi wykonalnemu flagę IMAGE_FILE_LARGE_ADDRESS_AWARE
- Inną możliwością jest na zwiększenie pamięci dla procesu jest funkcja CreateFileMapping.

Dynamiczne przydzielanie pamięci

- Do zarządzania pamięcią w języku C++ służa operatory **new** i **delete**.

- Można używać operatorów **malloc** i **free**, ze względu na to, że C++ jest zgodny wstecz z językiem C.
- Nie należy mieszać operatorów, np zwalniać obiektów utworzonych przy pomocy **new** uzywając komendy **free**!

Dynamiczne przydzielanie pamięci

- Porównanie operatorów **new** i **malloc**

Dynamiczne przydzielanie pamięci

- Porównanie operatorów **new** i **malloc**
- Tworzenie wektora przy pomocy malloca:

```
int *Wektor = (int*)malloc(sizeof(int)*10);
free(Wektor);
```

- Oraz przy pomocy **new**

```
int *Wektor = new int[10];
delete [ ] Wektor;
```

Dynamiczne przydzielanie pamięci

- Operator **new** oprócz przydzielania miejsca w pamięci obiektom, może je również inicjalizować

  X *p = **new** X inicjalizator;

- inicjalizator może być dowolnym wyrażeniem inicjalizującym

Dynamiczne przydzielanie pamięci

- Istnieje kilka wersji funkcji operator **new()**

- Forma placement-nie przydziela pamięci, umieszcza obiekt w określonym miejscu w pamięci

  void* operator new(std::size_t size, void* ptr) throw() {return ptr;};

Dynamiczne przydzielanie pamięci

- Druga forma przydziela pamięć obiektowi. Jeśli pamięc zostanie przydzielona, to funkcji zwraca wskaźnik na obiekt, a jeśli pamięć nie zostanie przydzielona, to funkcja zwraca wyjątek std::bad_alloc.

- **void* operator new(std::size_t size) throw(std::bad_alloc) ;**

Dynamiczne przydzielanie pamięci

- Trzecia forma operatora new dostarczonego w bibliotece standardowej to wersja no_throw. Jeśli pamięć zostanie przydzielona, to funkcja zwraca wskaźnik na obiekt, a jeśli operacja się nie powiedzie, to zwraca wskaźnik zerowy.

- **void* operator new(std::size_t size,**

  const std::nothrow_t&) throw();

-

Dynamiczne przydzielanie pamięci

- Aby zwolnić pamięć, używamy operatora **delete**

- Operator **delete** najpierw wywołuje destruktor klasy, a następnie zwalnia pamięć. Jeśli wywołanie destruktora nie powiedzie się, to pamięc nie zostanie zwolniona!

Dynamiczne przydzielanie pamięci

- Funkcja new pozwala też przydzielać pamięć tablicom.

X *px = new X[10];

- Jeśli chcemy zwolnić pamięć przydzieloną dla tablicy, nie musimy wywoływać oepratora delete dla każdego elementu. Następująca konstrukcja zwolni całą tablicę:

- **Delete [ ] X;**

Dynamiczne przydzielanie pamięci

- Przykładowy program, który pyta użytkownika o rozmiar tablicy, następnie wczytuje jej elementy, wypisuje je, i zwalnia pamięć.

```
#include <iostream>
#include <new>
using namespace std;
int main ()
{
  int i,n;
  int * p;
  cout << "Ile licz chcesz podać? " << endl;
  cin >> i;
  p= new (nothrow) int[i];
```

Dynamiczne przydzielanie pamięci

```
if (p == 0)
    cout << "Błąd. Pamięć nie została przydzielona";
  else   {
    for (n=0; n<i; n++)
    {
      cout << "Podaj liczbę: " << endl;
      cin >> p[n];
    }
    cout << "Wprowadziłeś następujące liczby:  ";
    for (n=0; n<i; n++)
      cout << p[n] << ", ";
    delete[] p;
  }  return 0;
}
```

Dynamiczne przydzielanie pamięci

- Istnieje jednak znacznie łatwiejszy sposób tworzenia dynamicznych tablic i zarządzania nimi-jest to tzw. Wektor, wchodzący w skład biblioteki STL. Klasa vector reprezentuje obudowaną, zwykłą tablicę znaną z C, wyposażoną w kilka dodatkowych mechanizmów. Elementy wektora mogą być dowolnego typu.

Dynamiczne przydzielanie pamięci

- Obiekt vector możemy tworzyć przy pomocy kilku różnych konstruktorów.
- Konstruktor tworzący wektor pusty:

  vector < typ_elemetow > nazwa_tablicy;
- Możemy też podać wielkość, co wcale nas nie ogranicza do tej wielkości. Może to być zabieg optymalizacyjny.

  vector < int > tab(20);
- Wektor tworzący tablicę o określonym rozmiarzę i inicjalizujący ją podaną wartoscią:

  vector < int > tablica(20, 0);

Dynamiczne przydzielanie pamięci

- Podstawowe metody klasy wektor:
- **void push_back(const T obj)** **-**dodaje na końcu wektora kopię przekazanego argumentu
- **void swap(vector<T>& vec)** -zamienia zawartości dwóch wektorów miejscami
- **void pop_back()** **-**usuwa ostatni element z wektora
- **void clear()** usuwa wszystkie elementy z wektora
- **iterator insert(iterator pos, T obj)** -wstawia element obj przed wskazywaną przez iterator pos pozycją i zwraca iterator do dostawionego elementu

Dynamiczne przydzielanie pamięci

- Metoda push_back() dodając nowy element do tablicy, dba o to, aby tablica była odpowiedniego rozmiaru. Za każdym razem, gdy brakuje miejsca, tablica jest powiększana - rezerwowana jest nowa, większa przestrzeń, stare elementy są kopiowane, aby do większej tablicy móc dodać dany element. Z reguły rezerwowana jest pamięć dwa razy większa od poprzedniej. W skrajnej sytuacji, może się zdarzyć, że zarezerwowana pamięć jest prawie dwa razy większa niż ilość elementów znajdujących się w niej!
- Ponieważ kopiowanie tablicy jest powolnym procesem, w przypadku tablicy dużych struktur, na przykład klas, warto zastanowić się nad utworzeniem tablicy wskaźników na te struktury.

Dynamiczne przydzielanie pamięci

- Kolejnym kontenerem udostępnianym przez STL jest klasa list ,będąca listą dwukierunkową. Oto główne różnice między wektorami a listami:

aspekt	vector	list
dostęp	przez podanie indeksu lub przez iteratory	tylko przez iteratory
dodawanie/usuwanie	jeśli nie chodzi o końcowy element - powolne	dodawanie i usuwanie elementów jest bardzo szybkie i odbywa się w stałym czasie
adresy elementów	ulegają zmianom, wskaźniki często tracą ważność	niezmienne

Dynamiczne przydzielanie pamięci

- Metody klasy list:
- **void push_back(const T obj)** -dodaje na końcu listy kopię przekazanego argumentu
- **void pop_back()** -usuwa ostatni element z listy
- **void push_front(const T obj)** -dodaje na początku listy kopię przekazanego argumentu
- **void pop_front()** -usuwa pierwszy element listy
- **void clear()** -usuwa wszystkie elementy z listy

Dynamiczne przydzielanie pamięci

- **void remove(const T& wartosc)** -usuwa wszystkie elementy równe argumentowi wartosc
- **void merge(list<T> ls**) -dostawia zawartość ls do obecnej listy i całość sortuje rosnąco
-

Memory Mapped Files

- Rozmiar pamięci, ktorą dysponuje proces, możemy zwiększyć przy pomocy funkcji CreateFileMapping
- HANDLE WINAPI CreateFileMapping(
- __in HANDLE hFile,
- __in_opt LPSECURITY_ATTRIBUTES lpAttributes,
- __in DWORD flProtect,
- __in DWORD dwMaximumSizeHigh,
- __in DWORD dwMaximumSizeLow,
- __in_opt LPCTSTR lpName
-);

Memory Mapped Files

- Funkcja zwracająca wskaźnik do zamapowanej pamięci
- LPVOID WINAPI MapViewOfFile(
- __in HANDLE hFileMappingObject,
- __in DWORD dwDesiredAccess,
- __in DWORD dwFileOffsetHigh,
- __in DWORD dwFileOffsetLow,
- __in SIZE_T dwNumberOfBytesToMap
-);
-