Podstawy programowania komputerów

Wykład 10: „Sterowanie pamięcią w C”

swalover@ie.tu.koszalin.pl

Pamięć „ na stosie”

- Każdy program napisany w języku C ma dostęp do dwóch obszarów pamięci - stosu i sterty, w których może być przechowywana zmienna ilość danych.
- Na stosie automatycznie przechowywane są zmienne lokalne.
- Wywołania funkcji również zostawiają ślad na stosie, aby było wiadomo dokąd program ma powrócić po zakończeniu funkcji.
- Stos obsługiwany jest automatycznie i programista nie ma możliwości ingerencji w tą obsługę.

Pamięć „na stercie”

- Sterta jest obszarem pamięci udostępnianym przez system operacyjny wszystkim wykonującym się procesom (**sprawdź w WinTop lub PrcView**).
- W systemach wielodostępnych są zabezpieczenia, które uniemożliwiają dostęp procesu do pamięci, z której korzysta inny proces.
- Ponieważ wielkość sterty jest ograniczona, więc procesy działające w systemie współzawodniczą o dostęp do tego obszaru.

System operacyjny a pamięć

- Przydziałem pamięci zajmuje się system operacyjny, może on jednak przydzielić tylko tyle pamięci ile w danej chwili jest wolne.
- Procesy, którym pamięć została przydzielona, powinny informować system, że z niej nie korzystają - zwalniać pamięć natychmiast po tym jak ona przestaje być im potrzebna.

Przydział pamięci w ruchu programu (Declaration or Runtime Memoty Allocation)

- pamięć operacyjna w każdym komputerze ma charakter zasobów zdecydowanie ograniczonych;
- często pisząc program nie wiemy, jakie wymiary będzie mieć obszar danych;
- zajmowanie zbędnej pamięci zwiększa częstotliwość operacji dyskowych i spowalnia pracę całego środowiska;

Przydzielanie pamięci: malloc ( )

- Dla przechowywania danych automatycznie rezerwuje się obszar pamięci, potrzebny zgodnie z deklaracją w kodzie źródłowym C.
- Możliwe jest rezerwowanie pamięci w trakcie działania programu za pomocą funkcji **malloc ( )**

```
double * wsd;
wsd = (double *) malloc(30 *
                        sizeof(double));
```

Wskaźnik do zarezerwowanego obszaru dla 30 wartości typu double

Obsługa funkcji malloc( )

- Uogólniony format:

```
#include<stdlib.h>
void *malloc(size_t size);
```

- Wskaźnik typu void (nieokreślony)

```
//można konwertować automatycznie
int *ptr ;
. . .
ptr= malloc(100*sizeof(int));
//lub stosować wymuszoną konwersje
int *ptr ;
. . .
ptr = (int*) malloc(100*sizeof(int));
```

Tablica statyczna i dynamiczna

Taki samy obszar pamięci można zarezerwować drogą zadeklarowania tablicy statycznej.

```
float dane[100];
```

Stosowanie malloc( ) pozwala utworzyć tablicę dynamiczną, czyli taką, której rozmiar określany jest w trakcie działania programu:

```
int n;
wsdane = (float *) malloc(n*sizeof(float));
     // nie można napisać tak:
                              float dane[n];
```

Zwalnianie pamięci: free( )

```
int *ptr_liczby, max = 0;
      . . .
scanf(„%d”, &max);
ptr_liczby = malloc(max*sizeof(int));
//można lokować dane w pamięci liczby
      . . .
      free (ptr_liczby);
          //liczby już nie są dostępne
```

Zalety i wady „ręcznej” alokacji

+ Trwałość pamięci jest kontrolowana przez programistę, a nie przez zbiór sztywnych reguł.

+ Możliwe jest utworzenie bloku pamięci w jednej funkcji i usuniecie go w innej.

- Obszar pamięci przeznaczony do alokacji może ulec fragmentacji - aktywne bloki mogą przeplatać się z nieużywanymi.

- Korzystanie z „ręcznie” alokowanej pamięci jest wolniejsze niż korzystanie ze stosu.

Niepowodzenie w alokacji pamięci

Programista ma przewidzieć, że **malloc( )** może zwrócić pusty wskaźnik. Na przykładzie dynamicznego wiersza:

```
d_wiersz(  )//funkcja obsługi d_wiersza
     {char wiersz[  ] ;
      . . .
      ptr_str = malloc(strlen(wiersz)+1);
      if ( ptr_str != NULL )
          { StrCopy(wiersz, ptr_str);
           termination = 0}
      else{termination = 1}
      return termination}
```

Stała NULL

- W pliku nagłówkowym **stdio.h** zdefiniowana jest stała o nazwie **NULL**. Jej wartością jest 0 rzutowane na typ wskaźnikowy.
- Żadna funkcja służąca do przydziału (alokacji) pamięci, nie zwróci tej wartości, jeśli pamięć już została przydzielona.
- Wskaźnik, którego wartość jest równa **NULL** nie wskazuje na żaden obszar pamięci, wartość ta służy do informowania, że wskaźnik jest pusty i pamięć nie jest przydzielona.

Alokacja zerowanej pamięci: calloc()

- Uogólniony format funkcji:

```
#include <stdlib.h>
void *calloc(size_til_elem,
                    size_trozm_elem);
```

- Funkcja **calloc()** alokuje dane na stercie w jednym ciągłym obszarze.
- Rezerwuje pamięć jak **malloc()** i dodatkowo powoduje wyzerowanie bajtów.
- Zawartość pamięci, zarezerwowanej **malloc()** należy traktować jako nieprzewidywalną!!!

Ponowna alokacja pamięci: realloc( )

- Funkcja pozwala na zmianę rozmiaru bloku w obszarze, już przyporządkowanym dynamicznie:

```
void *realloc(void*block, size_t size);
```

- Dane, które znajdowały się w poprzednim obszarze pozostają **nie zmienione.**

```
realloc(NULL, 10*sizeof(float));
     /* to jest równoważne stosowaniu
     funkcji malloc(10*sizeof(float)) */

realloc(ptr, 0); // równoważne free(ptr);
```