Zajęcia 12

C – program ls

Podstawy programowania w języku C w Linux'ie, impelemtacja ls

1. Informacje ogólne

Programy pisane w języku C mogą być bardziej wydajne, niż pisane w języku CPP. Ma to znaczenie m.in. w programach przeznaczonych na urządzenia mobilne, programach obliczeniowych

gcc – kompilator do C
g++ - kompilator do CPP

Za pomocą g++ można skompilować program w C, różnice są, np. komunikaty błędu mogą być inne (co ma znaczenie przy bardziej skomplikowanych błędach).

W Window do programowania używa się środowisk zintegrowanych (tj. Visual Studio). W Linux'ie środowiska zintegrowane również istnieją, jednak możemy programować rozdzielając na:
- edytor, w którym piszemy kod,
- kompilator, który kod przetwarza.

2. Hello world

Po wpisaniu gcc plik.c domyślnie powstaje po udanej kompilacji plik a.out

Jak chcemy nadać nazwę pliku wynikowago do po parametrze –o, np.
gcc –o hello hello.c

ZADANIE DLA STUDENTÓW

Zapisać, skompilować i uruchomić program który wypisuje w terminalu *hello world*
(korzystając z Intertnetu, wyszukiwarki i własnej wiedzy)

3. Operacje na pamięci

Nie należy pisać programu w taki sposób, że używa pamięć, która nie została zaalokowana!
Pojawiają się błędy: naruszenie ochrony pamięci LUB segmentation fault!

Przykład statycznego alokowania pamięci:

```
#include <stdio.h>

void main() {
int tab[5];

tab[4]=7;

printf("Wartosc rowna jest: %d\n",tab[4]);
}
```

Czy jeśli zamiast tab[4]=7; będzie tab[5]=7; to program będzie poprawnie napisany?
NIE
Czy zawsze dojdzie do naruszenia ochrony pamięci?
NIE, jest to zależne od sposobu alokowania pamięci przez system operacyjny i innych zmiennych alokowanych w programie.

Przykład statycznego alokowania pamięci używając wskaźników:

```
#include <stdio.h>

void main() {
int tab[5];
int *pTab;

tab[4]=7;
pTab = &tab[0];

printf("Wartosc rowna jest: %d\n",pTab[4]);
}
```

Przykład dynamicznego alokowania pamięci:

```
#include <stdio.h>
#include <stdlib.h>

void main() {

int *pTab;

pTab = (int*) malloc(5 * sizeof(int));
pTab[4]=7;

printf("Wartosc rowna jest: %d\n",pTab[4]);

free(pTab);
}
```

4. Implementacja uproszczonego ls'a

```
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>


int main( int argc, char *argv[] ) {
DIR *pDIR;
struct dirent *pDirEnt;

pDIR = opendir(".");

if ( pDIR == NULL ) {
    fprintf( stderr, "%s %d: opendir() failed (%s)\n",
             __FILE__, __LINE__, strerror( errno ));
    exit( -1 );
}

pDirEnt = readdir( pDIR );
while ( pDirEnt != NULL ) {
    printf( "%s\n", pDirEnt->d_name );
    pDirEnt = readdir( pDIR );
}

closedir( pDIR );

return 0;
}
```

Jeśli funkcja nie zadziała to zwraca NULL. errno – zmienna przechowująca numer błędu. Podobnie jest dla innych funkcji, np. malloc.

ZADANIE DLA STUDENTÓW
samodzielnie, korzystając z Internetu, rozbudować program tak, aby wypisywał również rozmiar pliku

ZADANIE DO DOMU na 9 punktów (do oddania na zajęciach 14.):
zaimlementować ls z:
opcją – l - ze szczegółami
opcją – R - w podkatalogach
trzema dowolnymi innymi opcjami z „prawdziwego ls”
możliwość wywołania ls w dowolnym katalogu, np. ls -l /etc