**Wojskowa Akademia Techniczna**

**Wydział Elektroniki, Instytut Telekomunikacji**

ul. Urbanowicza 2, 00-908 Warszawa

INSTRUKCJA DO ĆWICZEŃ LABORATORYJNYCH

z przedmiotu**:**

**Programowanie w Linux Unix**

Opracował:

mgr inż. Jakub Banaszek

mgr inż. Cezary Wilkowski

**Laboratorium NR 1**

**ZAD. 1.**

**1.1 Wykorzystując powłokę BASH, napisać program „Hello World!”** w wybranym edytorze: vi[dodatkowe punkty]//vim/nano**.** Zmodyfikować kod tak aby:

1. możliwe było podanie tekstu na wejściu i wyświetlenie go na wyjściu - terminal
2. podany na wejściu tekst był wyświetlany z odwróconą wielkością liter i kolejnoscią
3. opóźnić wyświetlenie podanego tekstu o 5 sekund
4. utworzyć instrukcję warunkująca wpisany tekst równy = Hello Word

**1.2 Wykorzystując powłokę BASH, napisać program, który:**

Wypisuje szczegółowe informacje o pliku txt – „szczegółowe informacje” – trzy informacje:

1. liczba linii tekstu,
2. ilość znaków,
3. właściciel/e pliku (skorzystać z poleceń wc, ls).

**1.3. Wykorzystując powłokę BASH, napisać program, który:**

1. Tworzy konto nowego użytkownika (**nie** wykorzystując w tym celu polecenia **adduser**) :
2. program prosi o pełną nazwę użytkownika,
3. program prosi o nazwę użytkownika do wyświetlania (w shellu),
4. przypisuje hasło użytkownikowi i prosi o ponowne jego wpisanie.
5. program ma wyświetlać błąd przy próbie uruchomienia go jako użytkownik nieuprzywilejowany (brak uprawnień ROOT).
6. Pomocnicze wyrażenia:

* /etc/passwd
* /etc/shadow
* /etc/group
* /home
* hostname
* uid – numer id użytkownika (jak najwyższy)
* chmod
* chown

chmod +x [nazwa skryptu]

./[nazwa skryptu] – wywołanie skryptu

**ZAD. 2.**

**2.1 Przeanalizować poniższy przykład:**

**Plik main.c**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include "reciprocal.hpp"  int main (int argc, char \*\*argv)  {  int i;  i = atoi (argv[1]);  printf ("The reciprocal of %d is %g\n", i, reciprocal (i));  return 0;  } |

**Plik reciprocal.cpp**

|  |
| --- |
| #include <cassert>  #include "reciprocal.hpp"  double reciprocal (int i) {  // I should be non-zero.  assert (i != 0);  return 1.0/i;  } |

**Plik reciprocal.hpp**

|  |
| --- |
| #ifdef \_\_cplusplus  extern "C" {  #endif  extern double reciprocal (int i);  #ifdef \_\_cplusplus  }  #endif |

**Polecenia kompilacji:**

|  |
| --- |
| % gcc -c main.c //kompilacja samego pliku main.c  % g++ -c reciprocal.cpp //kompilacja samego pliku reciprocal.cpp  % g++ -c -I ../include reciprocal.cpp  % g++ -c -I ../include reciprocal.cpp  % g++ -c -D NDEBUG=3 reciproca.cpp  % g++ -c -O2 reciprocal.cpp //-02 optymalizacja kompilowania  % info gcc |

**Polecenia linkowania:**

|  |
| --- |
| % g++ -o reciprocal main.o reciprocal.o  %  % ./reciprocal 7  The reciprocal of 7 is 0.142857 |

Przeprowadzić kompilacje na swoich urządzeniach, odpoiwedzieć na pytanie co jest wynikiem program.

**2.2 Użycie paramterów argc i argv**

1. Utworzyć nowy projekt dla języka C w wybranym edytorze: vi [dodatkowe punkty]/vim/nano utworzyć program z funkcją main:

|  |
| --- |
| int main (int argc, char\* argv[]) |

1. W kodzie programu wypisać nazwę programu
2. Napisać instrukcję sprawdzającą podanie argumentów z wiersza poleceń i wypisać liczbę wywoływanych argumentów.

**ZAD. 3.**

**3.1 Procesy**

1. Utworzyć nowy projekt dla języka C w wybranym edytorze: vi/vim/nano
2. Utworzyć proces, wprowadzić identyfikator procesu, wyświetlić jego wartość, utworzyć przykładową zmienną (np. int test = 5), sprawdzić poleceniem ps jak widziany jest proces przez system
3. Utworzyć proces potomny do wcześniej wywołanego procesu – duplikat za pomocą funkcji fork(), wyświetlić wartosc „test” w procesie macierzystym i potomnym (w funkcji warunkowej). Następnie zmodyfikować ich wartości i ponownie wyświetlić. Zdefiniować w obu procesach nową zmienna (np. int test2) i przypisać im różne wartości. Wyświetlić wartości obu procesów poza funkcją warunkową – jedno polecenie.
4. Utworzyć nowy proces potomny i wyświetlić wartość identyfikatora procesu.
5. Wywołać funkcje systemową ls -l i ps -aux | grep *nazwa programu* w procesie potomnym i macierzystym. Zmodyfikować proces potomny jako za pomocą funkcji exec() i fork() wykorzystanych razem.
6. Zakończyć proces potomny a następnie macierzysty – kolejno funkcje exit() i wait()
7. Na podstawie poprzedniego programu (np. 3) utworzyć procesy zombie (defunct)
8. Dodać funkcję czyszczącą procesy potomne

**Wywołanie funkcji systemowej**

|  |
| --- |
| #include <stdlib.h>  int main ()  {  int return\_value;  return\_value = system (“ls -l /”);  return return\_value;  } |

**Użycie funkcji fork**

|  |
| --- |
| #include <stdio.h>  #include <sys/types.h>  #include <unistd.h>  int main ()  {  pid\_t child\_pid;  printf (“the main program process ID is %d\n”, (int) getpid ());  child\_pid = fork ();  if (child\_pid != 0) {  printf (“this is the parent process, with id %d\n”, (int) getpid ());  printf (“the child’s process ID is %d\n”, (int) child\_pid);  }  else  printf (“this is the child process, with id %d\n”, (int) getpid ());  return 0;  } |

**Użycie funkcji fork i exec**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <sys/types.h>  #include <unistd.h>  int spawn (char\* program, char\*\* arg\_list)  {  pid\_t child\_pid;  /\* Duplicate this process. \*/  child\_pid = fork ();  if (child\_pid != 0)  /\* This is the parent process. \*/  return child\_pid;  else {  /\* Now execute PROGRAM, searching for it in the path. \*/  execvp (program, arg\_list);  /\* The execvp function returns only if an error occurs. \*/  fprintf (stderr, “an error occurred in execvp\n”);  abort ();  }  }  int main ()  {  /\* The argument list to pass to the “ls” command. \*/  char\* arg\_list[] = {  “ls”, /\* argv[0], the name of the program. \*/  “-l”,  “/”,  NULL /\* The argument list must end with a NULL. \*/  };  spawn (“ls”, arg\_list);  printf (“done with main program\n”);  return 0;  } |

**Użycie funkcji czyszczenia potomków.**

|  |
| --- |
| #include <signal.h>  #include <string.h>  #include <sys/types.h>  #include <sys/wait.h>  sig\_atomic\_t child\_exit\_status;  void clean\_up\_child\_process (int signal\_number)  {  int status;  wait (&status);  child\_exit\_status = status;  }  int main ()  {  struct sigaction sigchld\_action;  memset (&sigchld\_action, 0, sizeof (sigchld\_action));  sigchld\_action.sa\_handler = &clean\_up\_child\_process;  sigaction (SIGCHLD, &sigchld\_action, NULL);  return 0;  } |

Pomocne funkcje

* system()
* getpid()
* fork()
* exec()
* instrukcje warunkowe if/else lub switch/case

Kompilacje należy przeprowadzić poprzez wywołanie polecenia gcc

**3.2** Punkty 3-4 wykonać w plikach nagłówkowe, odwołać się do poniższego przykładu

**Laboratorium NR 2**

**ZAD. 1**

**1.1 Dla przykładowego programu (składającego się z kilku plików źródłowych) podanego poniżej utworzyć plik makefile**, w którym zdefiniowane będą następujące główne cele:

* + lib\_stat - tworzenie biblioteki statycznej
  + lib\_dyn - tworzenie biblioteki współdzielonej
  + calc\_stat - kompilacja programu z użyciem biblioteki statycznej
  + calc\_dyn - kompilacja programu z użyciem biblioteki współdzielonej
  + all - wykonuje automatycznie cztery powyższe cele
  + clean - usuwanie z katalogu bieżącego wszystkich plików za wyjątkiem plików źródłowych

Wszędzie tam, gdzie jest to możliwe używać zmiennych automatycznych programu make (takich

jak na przykład: $@, $\*, $<, $^, $?) oraz predefiniowanych zmiennych i nazw programów

(takich jak na przykład: AR, CC, CFLAGS, RM).

**Plik mymath.h**

|  |
| --- |
| **/\* mymath.h \*/** #ifndef \_MYMATH #define \_MYMATH double myadd (double, double); double mysub (double, double); double mymul (double, double); double mydiv (double, double); #endif /\* \_MYMATH \*/ |

**Plik mymath.c**

|  |
| --- |
| **/\* mymath.c \*/** #include "mymath.h" double myadd (double a, double b) { return a + b; }  double mysub (double a, double b) { return a - b; }  double mymul (double a, double b) { return a \* b; }  d/ouble mydiv (double a, double b) { return a / b; } |

**Plik calc.c**

|  |
| --- |
| **/\* calc.c \*/** #include <stdio.h> #include "mymath.h"  int main (void) { double a=5, b=20; printf("\na = %8.2f\nb = %8.2f\n\n",a,b); printf ("a + b = %8.2f\n", myadd(a,b)); printf ("a - b = %8.2f\n", mysub(a,b)); printf ("a \* b = %8.2f\n", mymul(a,b)); printf ("a / b = %8.2f\n", mydiv(a,b)); printf("\n"); return 0; } |

**1.2 Utworzyć drugi plik makefile** analogicznie jak w poprzednim podpunkcie dla poniższego programu (składającego się z kilku plików źródłowych):

|  |
| --- |
| /\* pl.c \*/  #include<stdio.h>  void hello\_pl(void) {  printf("Witaj w swiecie bibliotek !\n");  }  void linux\_pl(void) {  printf("Witaj w LINUX-ie !\n");  } |

|  |
| --- |
| /\* eng.c \*/  #include<stdio.h>  void hello\_eng(void) {  printf("Hello in the library world !\n");  }  void linux\_eng(void) {  printf("Welcome to LINUX !\n");  } |

|  |
| --- |
| /\* pleng.h \*/  void hello\_pl(void);  void hello\_eng(void);  void linux\_pl(void);  void linux\_eng(void); |

|  |
| --- |
| /\* pleng.c \*/  /\* wstawiamy kod nagłówkowy naszej biblioteki \*/  #include "pleng.h"  int main (void) {  hello\_pl();  hello\_eng();  linux\_pl();  linux\_eng();  return(0);  } |

**1.3 Program calc.c podany w zad 1.1 przerobić w taki sposób**, aby wykorzystywał interfejs dl.

**ZAD. 2**

* 1. **Dla programu podanego poniżej** zrealizować następujące polecenia:

1. przeanalizować strukturę programu i zastosowane w nim funkcje,
2. sprawdzić odpowiednim poleceniem jak widziany jest proces przez system (który użytkownik go uruchomił, jakie zasoby zajmuje itp.),
3. zmodyfikować program tak, aby możliwe było zatrzymanie jego działania za pomocą dowolnego przycisku na klawiaturze,
4. zmodyfikować program tak, aby możliwe było zatrzymanie jego działania za pomocą dowolnej kombinacji przycisków na klawiaturze (za wyjątkiem ALT+F4, ALT+TAB, CTRL+ALT+DEL itp.),

|  |
| --- |
| #include <pthread.h>  #include <stdio.h>  /\* Prints x’s to stderr. The parameter is unused. Does not return. \*/  void\* print\_xs (void\* unused)  {  while (1)  fputc (‘x’, stderr);  return NULL;  }  /\* The main program. \*/  int main ()  {  pthread\_t thread\_id;  /\* Create a new thread. The new thread will run the print\_xs  function. \*/  pthread\_create (&thread\_id, NULL, &print\_xs, NULL);  /\* Print o’s continuously to stderr. \*/  while (1)  fputc (‘o’, stderr);  return 0;  } |