Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

Факультет електроніки та комп’ютерних технологій
Кафедра радіоелектронних і комп’ютерних систем

# Звіт

про виконання лабораторної роботи №5

“ Керування процесами і потоками”

Виконав
студент групи ФеІ-23
Морозенко Богдан Андрійович
Перевірив
ас. Сінькевич О. О.

Львів – 2019

Дата виконання роботи 19.11.19
ОС – Ubuntu 12.04 LTS

## Хід роботи

1. Написати функцію, виклик якої приведе до знищення всіх процесів-зомбі. створених поточним процесом.

## Код тестової програми, яка створює процес зомбі

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

int main()
{
    int pid = fork();
    if(pid == 0) {
        /* child */
        while(1) pause();
    } else {
        /* parent */
        sleep(1);
        kill(pid, SIGKILL);
        printf("pid %d should be a zombie\n", pid);
        while(1) pause();
    }
}
```

## Демонстрація роботи

Функція для демонстрації наявності процесів-зомбі

```
bohdan@bohdan-VirtualBox:~$ function Z_Display
> {
> echo ""
> echo "PID - PPID - State - User - Proc"
> UNIX95= ps -eo pid,ppid,state,user,comm | awk 'BEGIN { count=0
 } $3 ~ /Z/ { count++; print $1,$2,$3,$4,$5 } END { print "n" co
unt " Zombie(s) to slay." }'
> echo ""
> }
bohdan@bohdan-VirtualBox:~$ Z_Display

PID - PPID - State - User - Proc
n0 Zombie(s) to slay.
```

Створюємо процес-зомбі

```
bohdan@bohdan-VirtualBox:~$ ./test
pid 3061 should be a zombie
^Z
[1]+  Зупинено        ./test
bohdan@bohdan-VirtualBox:~$ Z_Display

PID - PPID - State - User - Proc
3061 3060 Z bohdan test
n1 Zombie(s) to slay.
```

Функція для завершення процесу-зомбі

```
bohdan@bohdan-VirtualBox:~$ function Z_Kill
> {
>  read -p "Enter PPID to kill or 'exit' : " SLAY_PPID
> if [ "$SLAY_PPID" = "exit" ] || [ "$SLAY_PPID" = "" ]
> then
> exit
> fi
> ps -p $SLAY_PPID | grep -q $SLAY_PPID
> if [ $? -eq 0 ]
> then
> UNIX95= ps -o pid,user,state,comm -p $SLAY_PPID |
> awk '$1 ~ /^[0-9]*$/ { print "The program " $4 " with PID " $1
 " is being run by the user " $2 " and is currently in state " $
3 }'
> read -p "Are you sure you want to kill PID $SLAY_PPID ? Y|N :
" COMMIT_KILL
> if [ "$COMMIT_KILL" = "Y" ] || [ "$COMMIT_KILL" = "y" ]
> then
> kill -9 $SLAY_PPID
> echo ""
> read -p "Killed PID $SLAY_PPID. Run again? Y/N : " GO_AGAIN
> else
> Z_Kill
> fi
```

```
> fi
> else
> echo "Invalid PID. Try again."
> echo ""
> Z_Kill
> fi
> }
```

Спроба вбивства процесу-зомбі

```
bohdan@bohdan-VirtualBox:~$ Z_Kill
Enter PPID to kill or 'exit' : 3061
The program test with PID 3061 is being run by the user bohdan a
nd is currently in state Z
Are you sure you want to kill PID 3061 ? Y|N : Y

Killed PID 3061. Run again? Y/N : N
bohdan@bohdan-VirtualBox:~$ Z_Display

PID - PPID - State - User - Proc
3061 3060 Z bohdan test
n1 Zombie(s) to slay.
```

Вбивство-батьківського процесу

```
bohdan@bohdan-VirtualBox:~$ Z_Kill
Enter PPID to kill or 'exit' : 3060
The program test with PID 3060 is being run by the user bohdan a
nd is currently in state T
Are you sure you want to kill PID 3060 ? Y|N : Y

Killed PID 3060. Run again? Y/N : N
[1]+  Вбито                 ./test
bohdan@bohdan-VirtualBox:~$ Z_Display

PID - PPID - State - User - Proc
n0 Zombie(s) to slay.
```

2. Розробити простий командний інтерпретатор для Linux.

## Код програми

```
#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>
```

```
/*
  Function Declarations for builtin shell commands:
 */
int lsh_cd(char **args);
int lsh_help(char **args);
int lsh_exit(char **args);

/*
  List of builtin commands, followed by their corresponding functions.
 */
char *builtin_str[] = {
  "cd",
  "help",
  "exit"
};

int (*builtin_func[]) (char **) = {
  &lsh_cd,
  &lsh_help,
  &lsh_exit
};

int lsh_num_builtins() {
  return sizeof(builtin_str) / sizeof(char *);
}

/*
  Builtin function implementations.
*/

int lsh_cd(char **args)
{
  if (args[1] == NULL) {
    fprintf(stderr, "lsh: expected argument to \"cd\"\n");
```

```
  } else {
    if (chdir(args[1]) != 0) {
      perror("lsh");
    }
  }
  return 1;
}

int lsh_help(char **args)
{
  int i;
  printf("Type program names and arguments, and hit enter.\n");
  printf("The following are built in:\n");

  for (i = 0; i < lsh_num_builtins(); i++) {
    printf("  %s\n", builtin_str[i]);
  }

  printf("Use the man command for information on other programs.\n");
  return 1;
}

int lsh_exit(char **args)
{
  return 0;
}

int lsh_launch(char **args)
{
  pid_t pid, wpid;
  int status;

  pid = fork();
  if (pid == 0) {
```

```
    // Child process
    if (execvp(args[0], args) == -1) {
      perror("lsh");
    }
    exit(EXIT_FAILURE);
  } else if (pid < 0) {
    // Error forking
    perror("lsh");
  } else {
    // Parent process
    do {
      wpid = waitpid(pid, &status, WUNTRACED);
    } while (!WIFEXITED(status) && !WIFSIGNALED(status));
  }

  return 1;
}

int lsh_execute(char **args)
{
  int i;

  if (args[0] == NULL) {
    // An empty command was entered.
    return 1;
  }

  for (i = 0; i < lsh_num_builtins(); i++) {
    if (strcmp(args[0], builtin_str[i]) == 0) {
      return (*builtin_func[i])(args);
    }
  }

  return lsh_launch(args);
```

```
}

#define LSH_RL_BUFSIZE 1024
char *lsh_read_line(void)
{
  int bufsize = LSH_RL_BUFSIZE;
  int position = 0;
  char *buffer = malloc(sizeof(char) * bufsize);
  int c;

  if (!buffer) {
    fprintf(stderr, "lsh: allocation error\n");
    exit(EXIT_FAILURE);
  }

  while (1) {
    // Read a character
    c = getchar();

    // If we hit EOF, replace it with a null character and return.
    if (c == EOF || c == '\n') {
      buffer[position] = '\0';
      return buffer;
    } else {
      buffer[position] = c;
    }
    position++;

    // If we have exceeded the buffer, reallocate.
    if (position >= bufsize) {
      bufsize += LSH_RL_BUFSIZE;
      buffer = realloc(buffer, bufsize);
      if (!buffer) {
        fprintf(stderr, "lsh: allocation error\n");
```

```
      exit(EXIT_FAILURE);
    }
   }
  }
}

#define LSH_TOK_BUFSIZE 64
#define LSH_TOK_DELIM " \t\r\n\a"
char **lsh_split_line(char *line)
{
  int bufsize = LSH_TOK_BUFSIZE, position = 0;
  char **tokens = malloc(bufsize * sizeof(char*));
  char *token;

  if (!tokens) {
    fprintf(stderr, "lsh: allocation error\n");
    exit(EXIT_FAILURE);
  }

  token = strtok(line, LSH_TOK_DELIM);
  while (token != NULL) {
    tokens[position] = token;
    position++;

    if (position >= bufsize) {
      bufsize += LSH_TOK_BUFSIZE;
      tokens = realloc(tokens, bufsize * sizeof(char*));
      if (!tokens) {
        fprintf(stderr, "lsh: allocation error\n");
        exit(EXIT_FAILURE);
      }
    }

    token = strtok(NULL, LSH_TOK_DELIM);
```

```
  }
  tokens[position] = NULL;
  return tokens;
}

void lsh_loop(void)
{
  char *line;
  char **args;
  int status;

  do {
    printf("> ");
    line = lsh_read_line();
    args = lsh_split_line(line);
    status = lsh_execute(args);

    free(line);
    free(args);
  } while (status);
}

int main(int argc, char **argv)
{
  // Load config files, if any.

  // Run command loop.
  lsh_loop();

  // Perform any shutdown/cleanup.

  return EXIT_SUCCESS;
}
```

# Результат роботи програми

```
bohdan@bohdan-VirtualBox:~$ gcc comand.c -o main3
bohdan@bohdan-VirtualBox:~$ ./main3
> help
Type program names and arguments, and hit enter.
The following are built in:
  cd
  help
  exit
Use the man command for information on other programs.
> cd Lab6
lsh: No such file or directory
> cd Lab8
> ./main

Information about CPU

processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 23
model name      : Intel(R) Core(TM)2 Duo CPU     E7200  @ 2.53GHz
stepping        : 6
microcode       : 0x60b
cpu MHz         : 2514.514
cache size      : 3072 KB
```

```
Information about memory

MemTotal:         506936 kB
MemFree:           13408 kB
Buffers:           10760 kB
Cached:           136640 kB
SwapCached:         2136 kB
Active:           205296 kB
Inactive:         224168 kB
Active(anon):     122764 kB
Inactive(anon):   163476 kB
Active(file):      82532 kB
Inactive(file):    60692 kB
Unevictable:           0 kB
Mlocked:               0 kB
HighTotal:             0 kB
HighFree:              0 kB
LowTotal:         506936 kB
LowFree:           13408 kB
SwapTotal:        522236

> exit
bohdan@bohdan-VirtualBox:~$
```

3. Розробити застосування для Linux, що реалізує паралельне виконання коду двома потоками.

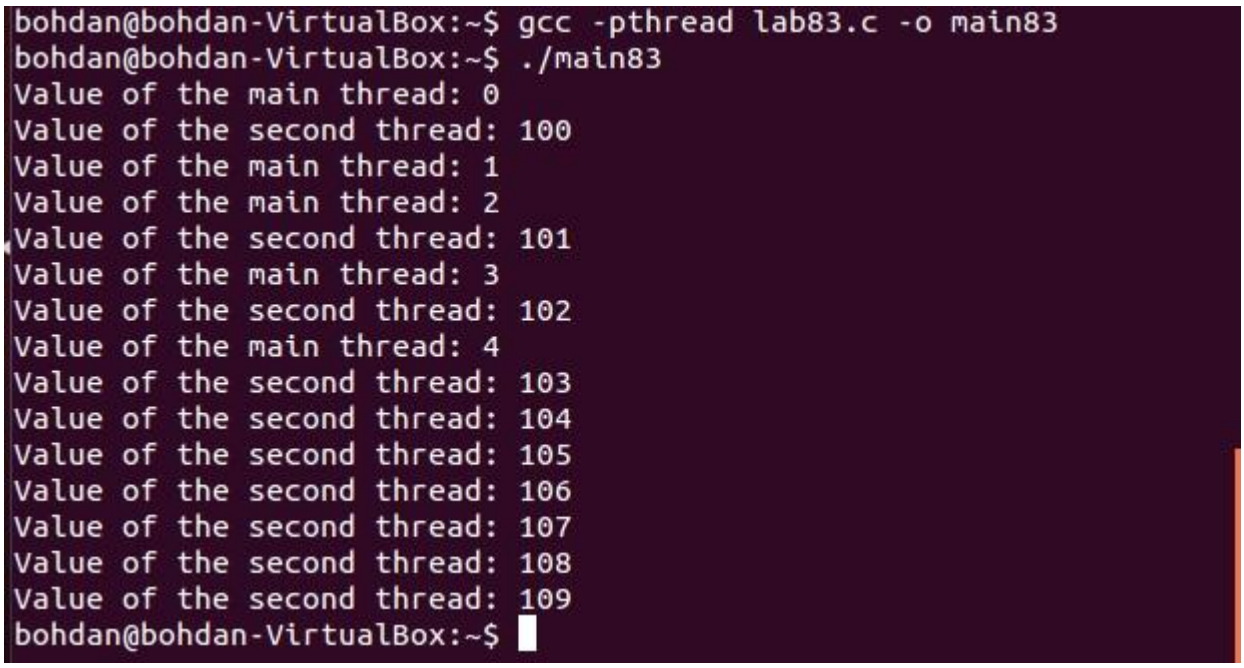
## Код програми

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

void *secondTh(void *value)
{
int i=0;
for(i=100;i<110;i++)
{
printf("Value of the second thread: %d\n",i);
sleep(2);
}
}

int main(int argc, char const *argv[])
{
pthread_t mainTh;
pthread_create(&mainTh,NULL,secondTh,NULL);
int i=0;
for(i=0;i<5;i++)
{
printf("Value of the main thread: %d\n",i);
sleep(1);
}
pthread_join(mainTh,NULL);
return 0;
}
```

## Демонстрація роботи




## Висновок

Я вивчив та застосував програмні інтерфейси ОС для керування процесами та потоками, виконав усі три частини роботи, продемонструвавши завершення процесів-зомбі, створення власного командного інтерпретатора та реалізацію паралельного виконання коду двома потоками.