

南昌大学实验报告4

姓名：谢志彬 学号：6103115112

邮箱地址：siliconx@163.com

专业班级：计算机科学与技术153

实验日期：2018/04/22

课程名称：Linux程序设计实验

实验项目名称

Multi-processing in Linux

实验目的

- 1.理解多进程机制
- 2.理解进程调度
- 3.理解多线程编程

实验基础

C语言、多进程、多线程

实验步骤

Q1: The Fork Question

I.编写Makefile

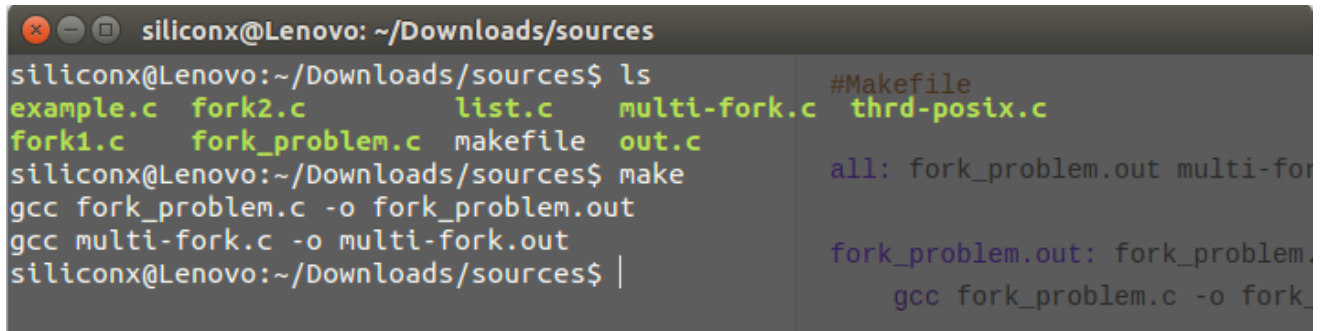
```
#Makefile

all: fork_problem.out multi-fork.out

fork_problem.out: fork_problem.c
    gcc fork_problem.c -o fork_problem.out

multi-fork.out: multi-fork.c
    gcc multi-fork.c -o multi-fork.out
```

II.编译



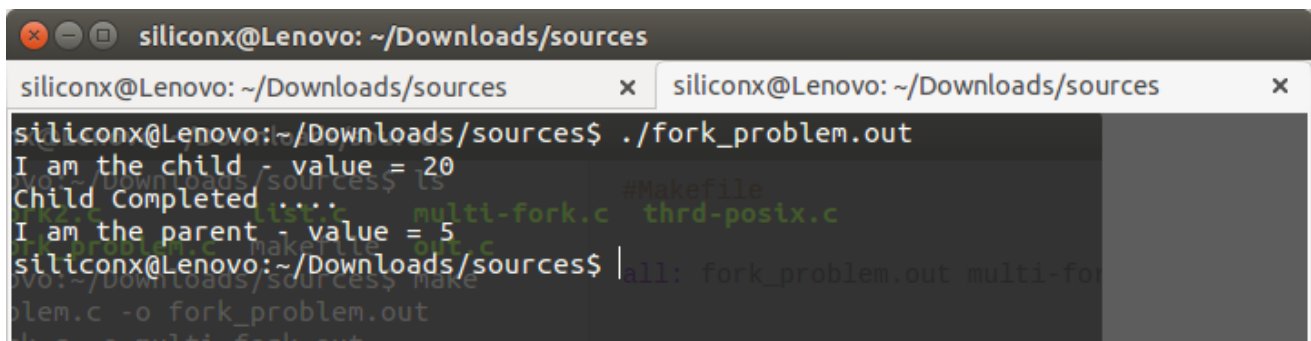
```
siliconx@Lenovo: ~/Downloads/sources
siliconx@Lenovo:~/Downloads/sources$ ls
example.c  fork2.c      list.c      multi-fork.c  thrd-posix.c
fork1.c    fork_problem.c  makefile  out.c
siliconx@Lenovo:~/Downloads/sources$ make
gcc fork_problem.c -o fork_problem.out
gcc multi-fork.c -o multi-fork.out
siliconx@Lenovo:~/Downloads/sources$ |
```

III.答

a -- fork_problem.c

LINE X将输出: I am the child - value = 20
 LINE Y将输出: I am the parent - value = 5
 因为pid == 0时表示子进程, pid > 0时表示父进程

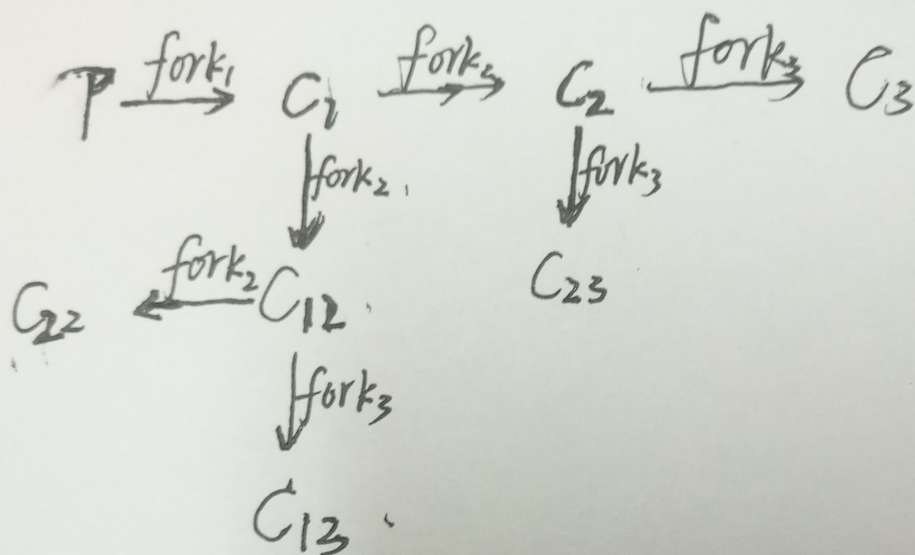
运行程序验证结论正确



```
siliconx@Lenovo: ~/Downloads/sources
siliconx@Lenovo:~/Downloads/sources$ ./fork_problem.out
I am the child - value = 20
Child Completed ....
I am the parent - value = 5
siliconx@Lenovo:~/Downloads/sources$ |
```

b -- multi-fork.c

将产生8个进程，推导如下



P: 父进程.

fork_i : 代码中的第 i 个 fork 调用

C_{ij} : P 进程执行第 i 个 fork 产生的子进程
执行第 j 个 fork 所产生的下一进程.

运行代码，验证结论
共有8个不同的pid



The image shows a terminal window with the title 'siliconx@Lenovo: ~/Downloads/sources'. The command `./multi-fork.out | sort | uniq` has been executed, resulting in a list of eight distinct PIDs: 12707, 12710, 12711, 12712, 12713, 12714, 12715, and 12716. In the background, a blurred image of a document with Chinese text is visible. The text includes '的子进程' (child process) and '下一进程.' (next process).

```
siliconx@Lenovo: ~/Downloads/sources
siliconx@Lenovo: ~/Downloads/sources$ ./multi-fork.out | sort | uniq
12707
12710
12711
12712
12713
12714
12715
12716
siliconx@Lenovo: ~/Downloads/sources$
```

Q2 Fork and fork more

step1.补全代码

fork1.c

```
#include <stdio.h>

void fork2();

int main() {
    fork2();
    return 0;
}

void fork2() {
    printf("L0");
    fork();
}
```

```
    printf("L1\n");  
    fork();  
    printf("Bye\n");  
}
```

fork2.c

```
#include <stdio.h>  
  
void fork2();  
  
int main() {  
    fork2();  
    return 0;  
}  
  
void fork2() {  
    printf("L0\n");  
    fork();  
    printf("L1\n");  
    fork();  
    printf("Bye\n");  
}
```

step2.运行

```
siliconx@Lenovo: ~/Downloads/sources
siliconx@Lenovo: ~/go/src/hello x siliconx@Lenovo: ~/Downloads/sources x
siliconx@Lenovo:~/Downloads/sources$ gcc fork1.c -o fork1.out && ./fork1.out
L0L1;
Bye f("Bye\n");
L0L1
Bye
Bye
Bye
Bye
siliconx@Lenovo:~/Downloads/sources$ gcc fork2.c -o fork2.out && ./fork2.out
L0
L1
L1
Bye <stdio.h>
Bye
Bye
Bye
fork2();
Bye
siliconx@Lenovo:~/Downloads/sources$ |
ain() {
ork2();
eturn 0;

fork2() {
```

step3.解释

fork1和fork2之间的差别仅仅在于fork()函数的第一行的printf的内容有没有`\n`
却导致了不一样的输出顺序。
主要原因是在Linux下，\n会刷新缓冲区

Q3.Processes

step1 -- collatz.c

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

int main(int argc, char const *argv[]) {
    if (argc != 2) { // arguments error
        printf("ARG ERROR\n");
        exit(0);
    }
}
```

```

long n = atoi(argv[1]);
if (n < 1) { // convert failed
    printf("ARG ERROR\n");
    exit(0);
}

while (n > 1) { // gen the collatz sequence
    printf("%ld ", n);
    if (n % 2 == 1) {
        n = 3 * n + 1;
    } else {
        n /= 2;
    }
}
printf("1\n");
return 0;
}

```

step2 -- run collatz.out

```

siliconx@Lenovo:~/code/linux/c/collatz$ gcc collatz.c -o collatz.out
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz.out
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz.out 0
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz.out -12
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz.out abcd
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz.out 13
13 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz.out 23
23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz.out 33
33 100 50 25 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ |

```

step2 -- run

step3 -- collatz2.c

```

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

```

```
#include <unistd.h>

int main(int argc, char const *argv[]) {
    if (argc != 2) { // arguments error
        printf("ARG ERROR\n");
        exit(0);
    }

    execlp("./collatz.out", "./collatz.out", argv[1], NULL); // call collatz.out

    return 0;
}
```

step4 -- run collatz2.out



```
siliconx@Lenovo: ~/code/linux/c/collatz
siliconx@Lenovo:~/code/linux/c/collatz$ gcc collatz2.c -o collatz2.out
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz2.out
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz2.out abc
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz2.out -23
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz2.out 12 12
ARG ERROR
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz2.out 13
13 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz2.out 99
99 298 149 448 224 112 56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz2.out 56
56 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ |
```

step3 -- collatz2.c

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

int main(int argc, ch
```

Q4.Threads

step1 -- collatz-thrd.c

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



```

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

// linkedlist
struct node {
    int e;
    struct node *next;
};

struct node *head = NULL; // header of list
struct node *r = NULL; // pointer of current positoin

void *gen_collatz(void *param); // thread to generate collatz

int main(int argc, char *argv[]) {
    pthread_t tid; // the thread identifier
    pthread_attr_t attr; // set of attributes for the thread

    if (argc != 2) {
        fprintf(stderr, "usage: %s <integer value>\n", argv[0]);
        return -1;
    }

    // get the default attributes
    pthread_attr_init(&attr);

    // create the thread
    pthread_create(&tid, &attr, gen_collatz, argv[1]);

    // now wait for the thread to exit
    pthread_join(tid, NULL);

    // traversing the list
    r = head;
    while (r != NULL) {
        printf("%d ", r->e);
        r = r->next;
    }
    printf("\n");
}

/**
 * The thread will begin control in this function
 */
void *gen_collatz(void *param) {
    int n = atoi(param); // convert string to int

    if (n > 1) {
        struct node *temp;

        while (n > 1) { // gen the collatz sequence

            // create a node by malloc

```

```

temp = (struct node *) malloc(sizeof(struct node));
temp->e = n;
temp->next = NULL;

if (head == NULL) {
    head = temp; // first node
    r = temp;
} else { // latter node
    r->next = temp;
    r = r->next;
}

if (n % 2 == 1) {
    n = 3 * n + 1;
} else {
    n /= 2;
}
}

// latest node -- value: 1
temp = (struct node *) malloc(sizeof(struct node));
temp->e = 1;
temp->next = NULL;

r->next = temp;
}

pthread_exit(0);
}

```

step2 -- compile && run

```
siliconx@Lenovo: ~/code/linux/c/collatz
siliconx@Lenovo:~/code/linux/c/collatz$ gcc collatz-thrd.c -pthread -o collatz-thrd.out
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz-thrd.out
usage: ./collatz-thrd.out <integer value>
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz-thrd.out abc
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz-thrd.out -234
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz-thrd.out 13
13 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ ./collatz-thrd.out 123
123 370 185 556 278 139 418 209 628 314 157 472 236 118 59 178 89 268 134 67 202
101 304 152 76 38 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
siliconx@Lenovo:~/code/linux/c/collatz$ |
read_create(&tid, &attr, gen_collatz, argv[1]);

now wait for the thread to exit
read_join(tid, NULL);

traversing the list
= head;
while (r != NULL) {
    printf("%d ", r->e);
    r = r->next;
}
```

实验思考

- 理解多进程机制
- 理解进程调度、进程间的关系
- 理解多线程编程、使用pthread库

参考资料

- 《Linux程序设计》
- [GNU Hurd](#)/ POSIX Threading Library