# Assignment1 实验报告

58121124 张博彦

2023 年 4 月 11 日

## 1 实验内容

1. Write a CPU bound C program and a I/O bound C program (e.g. use a number of printf statements within a while(1) loop). Compile and execute both of them. Observe the effect of their CPU share using the top display and comment.

2. Write a program in C that creates a child process, waits for the termination of the child and lists its PID.

3. Compile and run the program code for asgn1.c and record your observations. Perform the modification mentioned and answer the questions that follow. (a) Comment the inner loop in both the if and the else blocks, compile and run program code for asgn1.c again. Record your observations. (b) Do you find any difference in the output. If not, then what do you think is the role of the inner loop in both if and the else blocks ? (c) Modify code for asgn1.c in order to make the child process finish before the parent process starts

4. Create a file named my_file.txt that contains the following four lines:

```
Child 1 reads this line
Child 2 reads this line
Child 3 reads this line
Child 4 reads this line
```

Write a C program that forks four other processes. After forking the parent process goes into wait state and waits for the children to finish their execution. Each child process reads a line from the file my file.txt (Child 1 reads line 1, child 2 reads line 2, child 3 reads line 3 and child 4 reads line 4 ) and each prints the respective line. The lines can be printed in any order.

5. Write two programs file1.c and file2.c. Program file1.c uses these

   (a) fork() to launch another process

   (b) exec() to replace the program driving this process, while supplying arguments to file2.c to complete its execution

   (c) wait() to complete the execution of the child process

   (d) file1.c takes two arguments x (a number less than 1) and n (number of terms to be added, 1 or more). For example: file1 0.5 5

   (e) When the child proces finishes, the parent prints: Parent(PID=yyy) : Done

   Program file2.c requires two arguments to obtain the approximate value of ex by adding the first n terms in the relation: ex=1+x+x2/2!+.....

# 2 实验目的

This assignment is intended to learn how to create, work with and manipulate processes in Linux. You are expected to refer to the text book and references mentioned in the course website before you start the lab.

# 3 设计思路和流程图

## 3.1 实验内容 1

使用 while (1) 循环计算任意表达式 (如: int result = i) 以及调用 printf 函数，从而达到持续占用 CPU 和 I/O 设备，从而检查其所占 CPU 份额。

### 3.2 实验内容 2

使用 wait 函数得到子进程的 pid。

### 3.3 实验内容 3

使用 wait 函数使子进程在父进程启动之前完成。

### 3.4 实验内容 4

使用 flock 函数使得文件同时只能被一个子进程读取。

### 3.5 实验内容 5

file.c 中使用 exec() 命令将 file2.c 加载到新创建的进程中，同时使用 wait 函数使父进程等待子进程结束。file2.c 中创建 caculate 函数依据 ex 的计算公式计算预期值

## 4 主要数据结构及其说明

本次实验中暂无重要的数据结构

## 5 源程序

### 5.1 cpu.c

```
# include<stdio.h>
int main()
{
    while(1)
    {
        for(int i = 0;i<1000;i++)
        {
            int result = i;
        }
    }
```

```
}
```

### 5.2 io.c

```
# include<stdio.h>
int main()
{
    while(1)
    {
        printf("Hello world");
    }
}
```

### 5.3 PID.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h> /* contains prototype for wait */
int main(void)
{
    int pid;
    int status;
    printf("Hello World!\n");
    pid = fork( );
    if (pid == -1) /* check for error in fork */
    {
        perror("bad fork");
        return 1;
    }
    if (pid == 0)
        printf("I am the child process.\n");
    else
```

```
    {
        pid_t terminatedd_pid = wait(&status);
        printf("pid is %d \n", terminatedd_pid);
        printf("I am the parent process.\n");
    }
    return 0;
}
```

### 5.4 asgn1.c

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

#define ITR 100000

int main()
{
    int i,j,pid;

    pid = fork();

    if( pid == 0 ){

        for ( i=0;i<20;i++){
            for (j=0;j<ITR;j++);
            printf("Child:%d\n",i);
        }
    }
    else{
        wait(NULL);
        for ( i=0;i<20;i++){
             for (j=0;j<ITR;j++);
```

```
            printf("Parent:%d\n",i);
        }
    }
}
```

### 5.5 deal_my_file.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/file.h>
#include <sys/wait.h>

# define length 100
int main()
{
    FILE* file;
    char line[length];
    pid_t pid;
    int i;

    file = fopen("my_file.txt", "r");

    int file_lock = fileno(file);
    flock(file_lock, LOCK_EX);

    for(i = 1; i <= 4; i++)
    {

        pid = fork();
        if(pid == 0)
        {
            fseek(file, 0, SEEK_SET);
```

```
            for(int j = 1; j < i; j++)
            {
                if(fgets(line, length, file) == NULL){}
            }

            if(fgets(line, length, file) != NULL)
            {
                printf("Child %d read: %s \n", i, line);
            }
        }
        flock(file_lock, LOCK_UN);
    }
    for(i = 0; i < 4; i++)
    {
        wait(NULL);
    }
    fclose(file);

    return 0;
}
```

### 5.6 file1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[])
{
    double x = atof(argv[1]);
    int n = atoi(argv[2]);
```

```
    pid_t pid = fork();

    if(pid == 0)
    {
        char n_str[10];
        sprintf(n_str, "%d", n);
        execl("./file2", "file2", argv[1], n_str, NULL);
        fprintf(stderr, "Exec failed.\n");
        return 1;
    }
    else
    {
        wait(NULL);
        printf("Parent(PID=%d): finished\n", getpid());
    }

    return 0;
}
```

### 5.7 file2.c

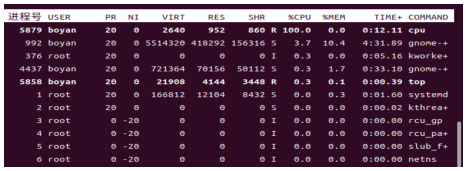
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <unistd.h>

double calculate(double x, int n)
{
    double result = 1.0;
    double term = 1.0;

    for(int i = 1; i <= n; i++)
    {
```

```
        term *= x / i;
        result +=term;
    }

    return result;
}

int main(int argc, char *argv[])
{
    double x = atof(argv[1]);
    int n = atoi(argv[2]);

    double ex = calculate(x, n);
    printf("Child(PID=%d): for x=%.2f, first %d term come out %.7f\n", (int)
    return 0;
}
```

# 6 程序运行结果及分析

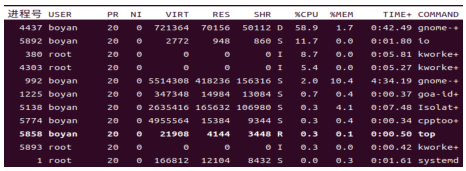
## 6.1 实验内容 1

运行结果如下



| 进程号 | USER | PR | NI | VIRT | RES | SHR | %CPU | %MEM | TIME+ | COMMAND |
|---|---|---|---|---|---|---|---|---|---|---|
| **5879** | **boyan** | **20** | **0** | **2640** | **952** | **860** | **R 100.0** | **0.0** | **0:12.11** | **cpu** |
| 992 | boyan | 20 | 0 | 5514320 | 418292 | 156316 | S 3.7 | 10.4 | 4:31.89 | gnome-+ |
| 376 | root | 20 | 0 | 0 | 0 | 0 | I 0.3 | 0.0 | 0:05.16 | kworke+ |
| 4437 | boyan | 20 | 0 | 721364 | 70156 | 50112 | S 0.3 | 1.7 | 0:33.10 | gnome-+ |
| **5858** | **boyan** | **20** | **0** | **21908** | **4144** | **3448** | **R 0.3** | **0.1** | **0:00.39** | **top** |
| 1 | root | 20 | 0 | 166812 | 12104 | 8432 | S 0.0 | 0.3 | 0:01.60 | systemd |
| 2 | root | 20 | 0 | 0 | 0 | 0 | S 0.0 | 0.0 | 0:00.02 | kthrea+ |
| 3 | root | 0 | -20 | 0 | 0 | 0 | I 0.0 | 0.0 | 0:00.00 | rcu_gp |
| 4 | root | 0 | -20 | 0 | 0 | 0 | I 0.0 | 0.0 | 0:00.00 | rcu_pa+ |
| 5 | root | 0 | -20 | 0 | 0 | 0 | I 0.0 | 0.0 | 0:00.00 | slub_f+ |
| 6 | root | 0 | -20 | 0 | 0 | 0 | I 0.0 | 0.0 | 0:00.00 | netns |

(a)



| 进程号 | USER | PR | NI | VIRT | RES | SHR | %CPU | %MEM | TIME+ | COMMAND |
|---|---|---|---|---|---|---|---|---|---|---|
| 4437 | boyan | 20 | 0 | 721364 | 70156 | 50112 | D 58.9 | 1.7 | 0:42.49 | gnome-+ |
| 5892 | boyan | 20 | 0 | 2772 | 948 | 860 | S 11.7 | 0.0 | 0:01.80 | lo |
| 380 | root | 20 | 0 | 0 | 0 | 0 | I 8.7 | 0.0 | 0:05.81 | kworke+ |
| 4303 | root | 20 | 0 | 0 | 0 | 0 | I 5.4 | 0.0 | 0:05.27 | kworke+ |
| 992 | boyan | 20 | 0 | 5514308 | 418236 | 156316 | S 2.0 | 10.4 | 4:34.19 | gnome-+ |
| 1225 | boyan | 20 | 0 | 347348 | 14984 | 13084 | S 0.7 | 0.4 | 0:00.37 | goa-id+ |
| 5138 | boyan | 20 | 0 | 2635416 | 105632 | 106980 | S 0.3 | 4.1 | 0:07.48 | Isolat+ |
| 5774 | boyan | 20 | 0 | 4955564 | 15384 | 9344 | S 0.3 | 0.4 | 0:00.34 | cpptoo+ |
| **5858** | **boyan** | **20** | **0** | **21908** | **4144** | **3448** | **R 0.3** | **0.1** | **0:00.50** | **top** |
| 5893 | root | 20 | 0 | 0 | 0 | 0 | I 0.3 | 0.0 | 0:00.42 | kworke+ |
| 1 | root | 20 | 0 | 166812 | 12104 | 8432 | S 0.0 | 0.3 | 0:01.61 | systemd |

(b)

根据输出结果发现与 CPU 绑定的程序 CPU 份额占比会远高于与 I/O 绑定的程序。

## 6.2 实验内容 2

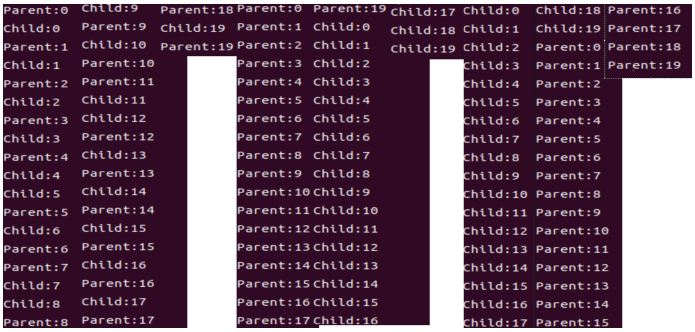
运行结果如下

```
boyan@boyan-VirtualBox:~/桌面/operating system$ ./PID
Hello World!
I am the child process.
pid is 5183
I am the parent process.
```

输出结果成功输出了子进程的 pid 且并未打印父进程的 pid，与实验要求一致

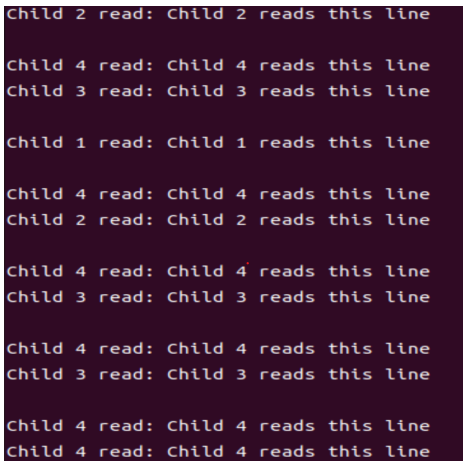
## 6.3 实验内容 3

运行结果依次如下




## 6.4 实验内容 4

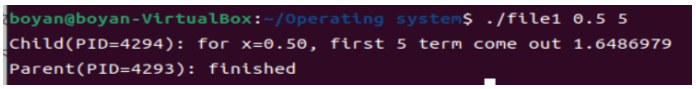
运行结果如下



成功实现了每个子进程读取该子进程所需读取的目标 txt 内容。

### 6.5 实验内容 5

运行结果如下




根据输入 0.5、5，成功输出了当 n=5 时 e 的 0.5 次方的值

## 7 实验体会

本次实验中，我初步了解了与 CPU、I/O 绑定的程序对于 CPU 份额占比，同时学会了使用 wait 函数的等待子进程结束以及使用 flock 函数使得 txt 文件同时只能被一个子进程读取。对于子进程和父进程的关系以及影响有了更加深入的了解。