

1 实验一 进程的建立

1.1 实验目的

学会通过基本的 Windows 或者 Linux 进程控制函数，由父进程创建子进程，并实现父子进程协同工作。

1.2 实验软硬件环境

Linux version 5.13.0-40-generic (buildd@ubuntu)
gcc (Ubuntu 9.4.0-1ubuntu1 20.04.1) 9.4.0,GNU ld (GNU Binutils for Ubuntu) 2.34

1.3 实验内容

创建两个进程，让子进程读取一个文件，父进程等待子进程读取完文件后继续执行，实现进程协同工作。进程协同工作就是协调好两个进程，使之安排好先后次序并以此执行，可以用等待函数来实现这一点。当需要等待子进程运行结束时，可在父进程中调用等待函数。

1.4 实验程序及分析

```
1  #include <unistd.h>
2  #include <sys/types.h>
3  #include <sys/wait.h>
4  #include <stdlib.h>
5  #include <stdio.h>
6
7  int main() {
8      pid_t pid = fork();
9      if (pid < 0) {
10         fprintf(stderr, "Fork error %d\n", pid);
11         exit(pid);
12     }
13     else if (pid == 0) {
14         printf("Child process started! PID = %d\n", getpid());
15         freopen("./data", "r", stdin);
16         puts("Child read file:");
17         char buf[205];
18         int n;
19         while ((n = read(0, buf, 200)) > 0)
20             write(1, buf, n);
21         puts("Child process will exit in 2 seconds!");
22         sleep(2);
23     }
24     else {
```

```

25     printf("Father process started! PID = %d\n", getpid());
26     puts("Wait for child process...");
27     int status;
28     waitpid(pid, &status, 0);
29     puts("Child process exited!");
30     puts("Father process will exit in 2 seconds!");
31     sleep(2);
32 }
33 return 0;
34 }

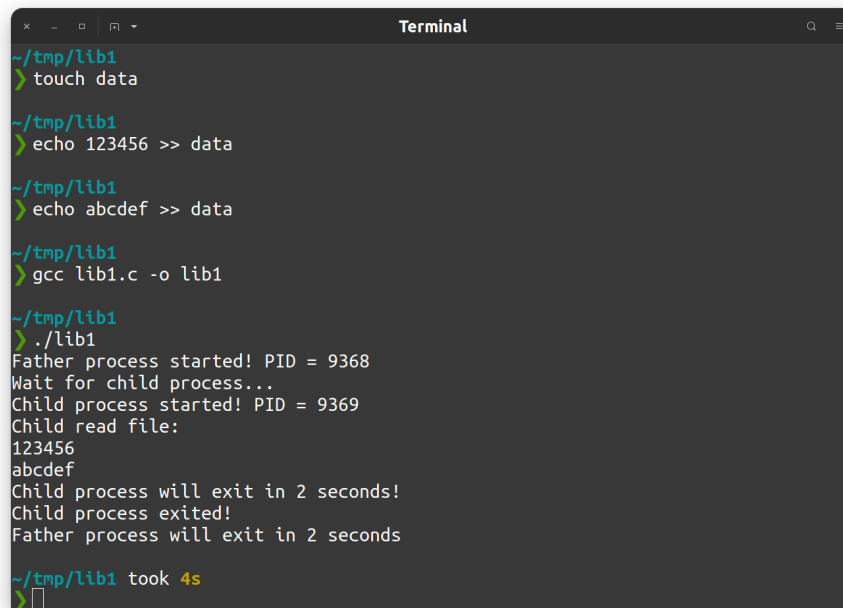
```

使用 `fork()` 函数创建子进程。

若 `fork()` 函数返回值等于 0，则说明此进程为子进程。子进程读取文件并输出到屏幕，然后退出。

若 `fork()` 函数返回值大于 0，则说明此进程为父进程，返回值为子进程 PID。父进程中使用 `waitpid()` 函数挂起，等待子进程运行结束。

1.5 实验截图



```

~/tmp/lib1
> touch data

~/tmp/lib1
> echo 123456 >> data

~/tmp/lib1
> echo abcdef >> data

~/tmp/lib1
> gcc lib1.c -o lib1

~/tmp/lib1
> ./lib1
Father process started! PID = 9368
Wait for child process...
Child process started! PID = 9369
Child read file:
123456
abcdef
Child process will exit in 2 seconds!
Child process exited!
Father process will exit in 2 seconds

~/tmp/lib1 took 4s
>

```

Figure 1: 实验一 进程的建立

1.6 实验心得体会

本次实验使用了 `fork()` 函数创建子进程，使用了 `waitpid()` 函数挂起父进程，达到父子进程协同工作的目的。通过实验，理解了进程的创建过程。

2 实验二 线程共享进程数据

2.1 实验目的

了解线程与进程之间的数据共享关系。创建一个线程，在线程中更改进程中的数据。

2.2 实验软硬件环境

Linux version 5.13.0-40-generic (bulld@ubuntu)
gcc (Ubuntu 9.4.0-1ubuntu1 20.04.1) 9.4.0, GNU ld (GNU Binutils for Ubuntu) 2.34

2.3 实验内容

在进程中定义全局共享数据，在线程中直接引用该数据进行更改并输出该数据。

2.4 实验程序及分析

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <pthread.h>
4  #include <string.h>
5  #include <stdlib.h>
6
7  char shared_data[100];
8
9  void *thread_fun(void *arg) {
10     printf("Thread started! Thread ID = %lu\n", pthread_self());
11     strcpy(shared_data, "thread");
12     puts("Shared data changed in thread!");
13     puts("Thread ended!");
14 }
15
16 int main() {
17     printf("Main process started! Process PID = %u\n", getpid());
18     strcpy(shared_data, "main");
19     printf("Shared data is '%s' now\n", shared_data);
20     pthread_t tid;
21     unsigned int err_code = pthread_create(&tid, NULL, thread_fun,
    ↪     NULL);
```

```

22     if (err_code) {
23         fprintf(stderr, "Thread create error: %u\n", err_code);
24         exit(err_code);
25     }
26     sleep(1);
27     printf("Shared data is \"%s\" now\n", shared_data);
28     puts("Main process ended!");
29     return 0;
30 }

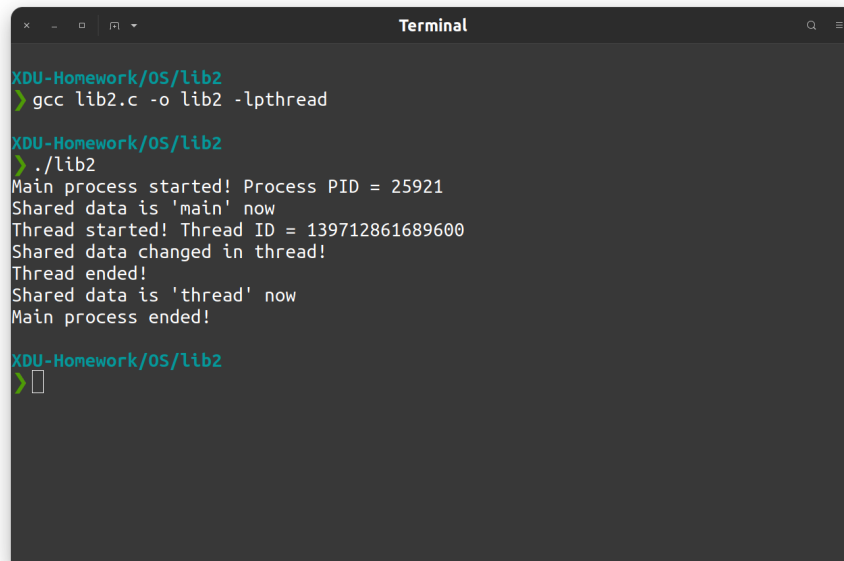
```

全局变量被进程和其子线程共享。故全局变量可被线程直接修改。

使用 `pthread_create()` 函数创建线程。其中第三个参数是函数指针，表示线程执行函数。

在进程中初始化全局共享变量并输出，然后在线程中修改全局共享变量并输出，最后等待线程函数执行完毕后，在进程输出全局共享变量。可以观察到线程成功修改了全局共享变量。

2.5 实验截图



```

XDU-Homework/OS/lib2
> gcc lib2.c -o lib2 -lpthread

XDU-Homework/OS/lib2
> ./lib2
Main process started! Process PID = 25921
Shared data is 'main' now
Thread started! Thread ID = 139712861689600
Shared data changed in thread!
Thread ended!
Shared data is 'thread' now
Main process ended!

XDU-Homework/OS/lib2
> 

```

Figure 2: 实验二 线程共享进程数据

2.6 实验心得体会

本次实验使用了 `pthread_create()` 函数创建线程。并通过定义全局变量在进程和各个线程中共享，以达到线程通信的目的。通过本次实验，理解了如何定义全局共享变量，如何实现进程与线程之间的简单通信。

3 实验三 信号通信

3.1 实验目的

利用信号通信机制在父子进程及兄弟进程间进行通信。

3.2 实验软硬件环境

Linux version 5.13.0-40-generic (buildd@ubuntu)
gcc (Ubuntu 9.4.0-1ubuntu1 20.04.1) 9.4.0,GNU ld (GNU Binutils for Ubuntu)
2.34

3.3 实验内容

父进程创建一个有名事件，由子进程发送事件信号，父进程获取事件信号后进行相应的处理。

3.4 实验程序及分析

阻塞型通信

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <signal.h>
4  #include <stdlib.h>
5  #include <sys/wait.h>
6
7  void handler(int sig) {
8      pid_t pid;
9      int status;
10     while (pid = waitpid(-1, &status, WNOHANG) > 0) {
11         printf("Child process %d died: %d\n", pid,
12             ↪ WEXITSTATUS(status));
13         printf("Parent process received SIGCHLD signal!");
14     }
15 }
16
17 int main() {
18     pid_t pid = fork();
19     if (pid == 0) {
20         printf("Child process started! PID = %u\n", getpid());
```

```

20     sleep(1);
21     printf("After sleep one second, child process PID = %u\n",
        ↪ getpid());
22     exit(0);
23 }
24 else if (pid > 0) {
25     printf("Parent process started! PID = %u\n", getpid());
26     signal(SIGCHLD, handler);
27     sleep(2);
28 }
29 else {
30     fprintf(stderr, "Fork error: %d\n", pid);
31     exit(pid);
32 }
33 return 0;
34 }

```

一个进程终止或者停止时，SIGCHLD 信号将被发送给其父进程。

创建一个子进程，打印子进程 PID。子进程运行一段时间后退出。

父进程调用信号处理函数 `signal()`，捕获信号 SIGCHLD 并在 `handler()` 函数中处理。在 `handler()` 函数中输出退出的子进程的 PID。可以观察到，父进程成功捕获子进程的终止信号。

非阻塞型通信

```

1  #include <stdio.h>
2  #include <unistd.h>
3  #include <signal.h>
4  #include <stdlib.h>
5
6  void handler(int sig) {
7      puts("Received SIGINT signal!");
8  }
9
10 int main() {
11     pid_t pid = fork();
12     if (pid == 0) {
13         printf("Child process started! PID = %u\n", getpid());
14         sleep(1);
15         printf("After sleep one second, child process PID = %u\n",
            ↪ getpid());
16         sleep(1);
17         printf("After sleep two seconds, child process PID = %u\n",
            ↪ getpid());
18         exit(0);
19     }

```

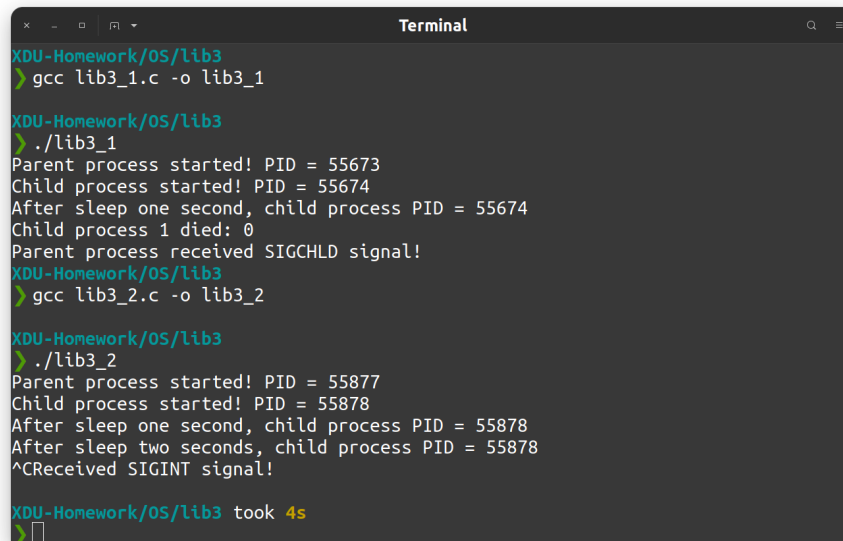
```

20     else if (pid > 0) {
21         printf("Parent process started! PID = %u\n", getpid());
22         signal(SIGINT, handler);
23         pause();
24     }
25     else {
26         fprintf(stderr, "Fork error: %d\n", pid);
27         exit(pid);
28     }
29     return 0;
30 }

```

在 Terminal 中按 Ctrl + c 可发送中断信号 SIGINT。
父进程调用信号处理函数 signal(), 捕获信号 SIGINT 并在 handler() 函数中打印信息。

3.5 实验截图



```

XDU-Homework/OS/lib3
> gcc lib3_1.c -o lib3_1

XDU-Homework/OS/lib3
> ./lib3_1
Parent process started! PID = 55673
Child process started! PID = 55674
After sleep one second, child process PID = 55674
Child process 1 died: 0
Parent process received SIGCHLD signal!
XDU-Homework/OS/lib3
> gcc lib3_2.c -o lib3_2

XDU-Homework/OS/lib3
> ./lib3_2
Parent process started! PID = 55877
Child process started! PID = 55878
After sleep one second, child process PID = 55878
After sleep two seconds, child process PID = 55878
^CReceived SIGINT signal!

XDU-Homework/OS/lib3 took 4s
>

```

Figure 3: 实验三 信号通信

3.6 实验心得体会

本次实验使用了 signal() 函数捕获信号并做相应的处理。通过本次实验，理解了如何在进程之间进行信号通信。