浙江工业大学

数据库原理及应用实验报告

(2021 级)

实验题目 实验 1　进程控制与描述

学生姓名　温家伟

学生学号　202103151422

学科(专业)　大数据分析 2101 班

所在学院　理学院

提交日期　2023 年 10 月 18 日

一、实验目的

利用`API`函数，编写程序，实现进程的创建和终止，加深对操作系统进程的理解，观察操作系统进程运行的动态性能，获得包含多进程的应用程序经验。

二、实验内容

1.进程的创建和终止

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

int main()
{
    pid_t id=fork();
    if(id==0)
    {
      // child
        printf("我是子进程，pid:%d\n", getpid());
        sleep(1);
    }
    else
    {
      // parent
      printf("我是父进程，pid:%d,\n",getpid());
    }
    return 0;
}
```

```
(base) [bbjsxl@VM-20-8-centos code1]$ make
gcc -o process process.c
(base) [bbjsxl@VM-20-8-centos code1]$ ./process
我是父进程，pid:14947,
我是子进程，pid:14948
(base) [bbjsxl@VM-20-8-centos code1]$
```

`fork()`函数可以创建一个子进程，他有两个返回值：子进程中返回0，父进程返回子进程`id`，出错返回-1

`return`是进程正常终止，可以在命令行用`echo $?`打印出退出码：

```
(base) [bbjsxl@VM-20-8-centos code1]$ ./process
我是父进程，pid:14947,
我是子进程，pid:14948
(base) [bbjsxl@VM-20-8-centos code1]$ echo $?
0
(base) [bbjsxl@VM-20-8-centos code1]$
```

因为`return`了`0`，表示一切正常，所以，退出码打印出也是`0`；

如果把代码换成`return 6`,表示程序非正常退出，打印出也是`6`：

```
(base) [bbjsxl@VM-20-8-centos code1]$ ./process
我是父进程，pid:16155,
我是子进程，pid:16156
(base) [bbjsxl@VM-20-8-centos code1]$ echo $?
6
(base) [bbjsxl@VM-20-8-centos code1]$
```

2.线程的创建和终止

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

void *thread_func(void *arg) {
    pthread_t tid = pthread_self();
    printf("Thread ID 是 %ld\n", tid);
    pthread_exit(NULL);
}

int main() {
    pthread_t tid;
    int ret;

    ret = pthread_create(&tid, NULL, thread_func, NULL);
    if (ret != 0) {
        printf("线程创建失败\n");
        exit(EXIT_FAILURE);
    }

    // 等待线程执行完毕
    pthread_join(tid, NULL);

    return 0;
}
```

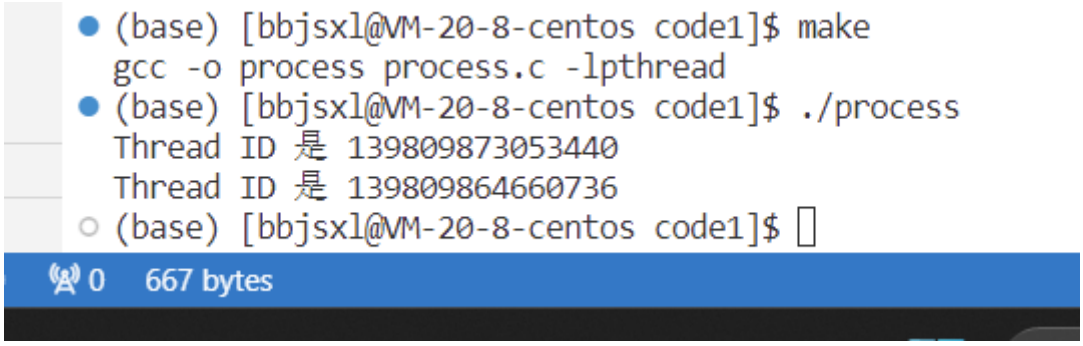
```
(base) [bbjsxl@VM-20-8-centos code1]$ make
make: “process”是最新的。
(base) [bbjsxl@VM-20-8-centos code1]$ ./process
Thread ID 是 140384201103104
(base) [bbjsxl@VM-20-8-centos code1]$
```

在上面的代码中，我们首先定义了一个名为 `thread_func` 的线程函数，在该函数中调用 `pthread_self()` 函数获取线程ID，并使用 `printf()` 函数将其输出到控制台。然后在 `main()` 函数中，我们使用 `pthread_create()` 函数创建一个新线程，并在其中传递 `thread_func` 作为线程函数。`pthread_join()` 函数用于等待新线程执行完毕，并释放其资源。

多线程：

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

```c
void *thread_func(void *arg)
{
    pthread_t tid = pthread_self();
    printf("Thread ID 是 %ld\n", tid);
    pthread_exit(NULL);
}

int main()
{
    pthread_t tid1, tid2;
    int ret;

    ret = pthread_create(&tid1, NULL, thread_func, NULL);
    if (ret != 0) {
        printf("线程1创建失败\n");
        exit(EXIT_FAILURE);
    }

    ret = pthread_create(&tid2, NULL, thread_func, NULL);
    if (ret != 0) {
        printf("线程2创建失败\n");
        exit(EXIT_FAILURE);
    }

    // 等待线程执行完毕
    pthread_join(tid1, NULL);
    pthread_join(tid2, NULL);

    return 0;
}
```



这段代码创建了两个线程，每个线程执行`thread_func`函数，并在该函数中获取自己的线程ID并输出到控制台上。`pthread_create()`函数用于创建新线程，`pthread_join()`函数用于等待线程执行完毕。在每个线程内部可以执行自己的逻辑，这样可以提高编程的并发性。

3.选择进程或线程`API`并使用

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

int main()
{
```

```
  pid_t id=fork();
  if(id==0)
  {
    // child
    while(1)
    {
      printf("我是子进程，我正在运行...pid:%d\n", getpid());
      sleep(1);
    }
  }
  else
  {
    // parent
    printf("我是父进程，pid:%d,我准备等待子进程了\n",getpid());
    sleep(40);
    pid_t ret=wait(NULL);
    if(ret<0)
    {
      printf("等待失败\n");
    }
    else
    {
      printf("等待成功:result:%d\n",ret);
    }
    sleep(20);
  }
  return 0;
}
```

这是一个父进程等待子进程结束的代码，在父进程中，调用`wait(NULL)`函数等待任一子进程结束，并获取子进程的终止状态。如果返回值小于`0`，表示等待失败；如果返回值大于`0`，表示成功等待到子进程结束，并返回子进程的`ID`。

我的理解：首先，为什么要进程等待，因为如果子进程退出，父进程不管不顾，就可能陷入僵尸进程，造成内存泄漏，而且`kill -9`也没办法杀死他，其次，子进程推出后，父进程也应该关心他的退出状态，所以父进程通过进程等待的方式，回收子进程资源，获取子进程退出信息。

`wait`的用法：

```
pid_t wait(int*status);
// 返回值：
// 成功返回被等待进程pid，失败返回-1。
// 参数：
// 输出型参数，获取子进程退出状态,不关心则可以设置成为NULL
```

三、实验技术

我是在`centos`下做的实验，借助了系统调用和线程库，完成了本次实验。

四、实验运行结果

见“二、实验内容”。