HW4

赵洁3160102525

1.Operating System Concept Chapter 4 Exercise:

4.8 Provide two programming examples in which multithreading does not provide better performance than a single-threaded solution.

- Any kind of sequential program is not a good candidate to be threaded.
- An example of this is a program that calculates an individual tax return.
- Another example is a "shell" program such as the C-shell or Korn shell. Such a program must closely monitor its own working space such as open files, environment variables, and current working directory.

4.9 Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

- When a kernel thread suffers a page fault, another kernel thread can be switched in to use the interleaving time in a useful manner. A single-threaded process, on the other hand, will not be capable of performing useful work when a page fault takes place. Therefore, in scenarios where a program might suffer from frequent page faults or has to wait for other system events, a multithreaded solution would perform better.

4.10 Which of the following components of program state are shared across threads in a multithreaded process?

- a. Register value
- b. Heap memory
- c. Global variables
- d. Stack memory

> *Heap memory(b.) and global variables(c.).*

4.17 Consider the following code segment:

```
pid_t pid;

pid=fork();
if(pid==0){
    fork();
    thread_create(. . .);
}
fork();
```

a. How many unique processes are created?

6

b. How many unique threads are created?

8

Explain:

- Every time we call fork(), the number of processes will double.
- After first fork(), `1*2=2` processes.
- The second fork() is only called by child process. So after the second fork(), `1+1*2=3` processes.
- After the third fork(), `3*2=6` processes
- And in each processes, every time we call thread_create(), the number of threads will plus one. We call thread_create in two processes, so `6+1+1=8` unique threads.

4.19 The program shown in Figure 4.23 uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

- LINE C:5

Explain:

The parent thread wait for the child thread to modifies the global variable.

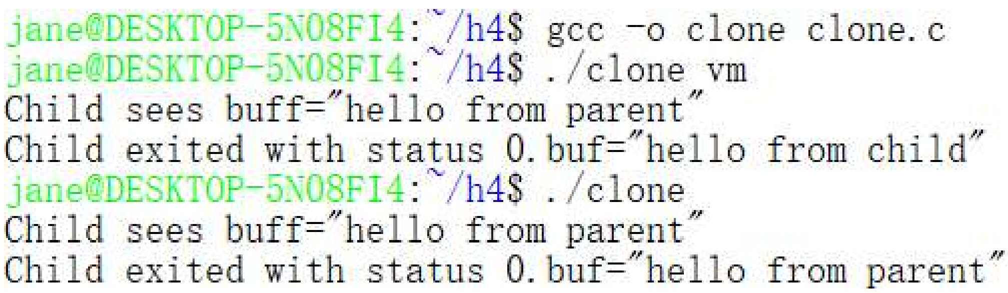
- LINE P:0

Explain:

The child duplicates the global variables from parent process and modifies the duplication. So the value in parent process remains to be 0.

2.Compile and run the following program twice.

- Screenshots of the running results:



```
jane@DESKTOP-5N08FI4:~/h4$ gcc -o clone clone.c
jane@DESKTOP-5N08FI4:~/h4$ ./clone vm
Child sees buff="hello from parent"
Child exited with status 0.buf="hello from child"
jane@DESKTOP-5N08FI4:~/h4$ ./clone
Child sees buff="hello from parent"
Child exited with status 0.buf="hello from parent"
```

- Explain： If CLONE_VM is set, the calling process and the child process run in the same memory space. Otherwise, the child process run in a memory space copyed from calling process.

  So, without 'vm', buffer is shared and child can directly modify buffer. While with 'vm', child can only modifiy a memory copy of the parent process and does nothing to the buffer of parent process.