HW 13

Joshua Thompson

April 29, 2018

Questions

1. (15 points) For each of the statements, indicate if the statment is true or false. Add a brief statement to support the claim

   (a) Threads are created just like processes by calling fork() except instead of checking the return value of fork() a specific funtion is used.

   False, threads created using clone()

   (b) Threads are scheduled just like other processes because POSIX threads are treated like individual processes by the OS.

   False, they are treated as a single process

   (c) Like multiple processes, threads provide resource isolation. Two threads from the same program do not share memory or other resources

   False, threads share the same memory

   (d) It's not possible for two threads of the same process to run simultaneously

   False

   (e) When any of the threads terminates, such as a call to exit(), all threads eliminate

   True, threads all run on the same process

2. (5 points) What are the equivalent thread commands for the following system calls:

   (a) fork()

   clone()

   (b) wait()

   pthread_join()

3. (15 points) Match the following terms, identifiers, funcitons to the descriptions below. (Online)

   (a) Retrieve the POSIX thread identifier for the calling thread pthread_self()

(b) The process identifier, shared by all threads of a multi-threaded program

pid

(c) Retrieve the UNIX OS thread identifier of the calling thread

syscall(SYS_gettid)

(d) Retrieve the UNIX OS process identifier of the calling process

getpid()

(e) the type of a POSIX thread identifier

pthread_t

(f) The type of the UNIX OS thread identifier

pid_t

(g) The thread identifier, unique to each thread and equal to the pid of the main thread

tid

4. (5 points) Complete the program given online. The thread should print the command line argument passed to it.

```
void * startup(void * args){
  char * str = (char*)args;
  printf("%s", str);
  return NULL;
}

int main(int argc, char * argv[]){
  pthread_t thread;

  pthread_create(&thread, NULL, startup, argv[1]);

  pthread_join(thread, NULL);
  return 0;
}
```

5. Answer the following Questions about the given program online. You could assume this would be run on a lab machine; if you wanted to run it to answer the questions.

   (a) (5 points) Based on the code, what are the two possible values for the argument to foo()?

   1 or NULL

   (b) (5 points) When you run this program, how many threads are running. You could use ps -l to count.

9

(c) (5 points) According to top what percent CPU does the program consume? Explain

900

6. (5 points) Explain why the following code snippet is not Atomic?

```
balance = balance+1;
```

During the addition the added part of the is stored in a temp variable before it is saved to the l value of balance. In between these actions the thread could try to access the variable and change it before the value can be saved to the l value. The action can be interrupted.

7. (5 points) For the code online, what is the expected output? Would you always get what you expect? Explain

”shared: 200” is expected but that will not always be the output because of the use of non atomic code. If scheduling changes, the output can become different.

8. (5 points) For the code in the previous questions, identify the critical section. What makes this section critical?

shared++ because it is non atomic

9. (5 points) Consider the naive locking solution shown online. Does this prove proper locking? Explain why or why not.

No because the the operations are not atomic. If the threads switch between the while command and the lock statement, there can be issues.

10. (5 points) Explain why using mutex avoids issues of a lack of atomicity in lock acquisition?

Mutex avoids these issues becaue it is always guaranteed to be atomic during operation. The mutex itself is a lock and has functions for locking and unlocking.

11. (10 points) The code shown online uses a coarse locking strategy, rewrite it to use fine locking.

```
pthread_mutext_t b_lock, c_lock;

int avail = MAX_FUNDS;
int local_1 = 0;
int local_2 = 0;

void * fun(void * args){
  int v,i;

  for(i=0; i < 100; i++){
    v = random() % 100;


    if(avail - v > 0){
      avail -= v;
    }
    if(random() % 2){
      pthread_mutext_lock(&b_lock);
      local_1 += v;
      pthread_mutext_unlock(&b_lock);
    }else{
      pthread_mutext_lock(&c_lock);
      local_2 += v;
      pthread_mutext_unlock(&c_lock);
    }

  }

  return NULL;
}
```

12. (5 points) What is deadlock? Provide a small code example of how deadlock can arise.

A deadlock is when two threads each hold a resource the other is waiting on. Example is the dining philosophers.

13. (5 points) Explain a strategy to avoid deadlock.

A way to avoid deadlock is to order the way we give resources out to the threads so that they always have enough to continue.