IC221 Homework: Threads Name Ian Coffey

AY23S 72 points total

(20) For each of the statements, indicate if the statement is true or false. You must add a brief statement to support your claim.

T / F: 1. Threads are created just like processes by calling fork() except instead of checking the return value of fork() a specific function is used.

|  |
| --- |
| False. You do not call fork when creating a new thread, you call pthread\_create. |

T / F: 2. POSIX threads (pthreads) are scheduled individually, similar to how processes are scheduled by the OS.

|  |
| --- |
| False, threads are not scheduled invidiually. They are scheduled in the order they arrive to the processor in, which is not always in order. |

T / F: 3. Like multiple processes, threads provide resource isolation. Two threads from the same program do not share memory or other resources.

|  |
| --- |
| False, threads are reliant on the resources of the parent process that created the threads, and share the resources between the child threads |

T / F: 4. It's possible for two threads of the same process to run simultaneously, on different CPU cores.

|  |
| --- |
| True, it is possible for two threads to run simultaneously on different cores. |

T / F: 5. When a process terminates, such as with a call to exit(), all threads of that terminate.

|  |
| --- |
| True, since the threads are reliant on the resources provided by the parent process, when the parent process is terminated, the threads have no resources to continue running, and are henceforth killed |

(8) 6. What are the equivalent thread calls, for each of the following calls for processes:

|  |  |
| --- | --- |
| Process Command | Equivalent Thread Command |
| fork() | pthread\_create |
| wait() | pthread\_join |

(21) 7. Match the following terms, identifiers, functions to the descriptions below. Choose from: tid, pid, pid\_t, pthread\_t, gettid(), getpid(), pthread\_self()

|  |  |
| --- | --- |
| Description | Item |
| Call to retrieve the POSIX thread identifier for the calling thread | pthread\_self() |
| The process identifier, shared by all threads of a multi-threaded process. | pid |
| Call to retrieve the OS thread identifier of the calling thread | gettid() |
| Call to retrieve the OS process identifier of the calling process | getpid() |
| The *type* of a POSIX thread identifier (ret. by pthread\_self()) | pthread\_t |
| The *type* of the OS thread identifier (returned by gettid()) | pid\_t |
| The thread identifier, unique to each thread and also equal to the pid of the main thread of the process | tid |

(8) 8. Complete the following program below. The thread should print the command line argument passed to it:

|  |
| --- |
| void \* startup( void \* args){  char \* str; //variable to reference string to print  str = (char \*) args;  printf("%s\n", str);  return NULL;  }  int main(int argc, char \* argv[]){  pthread\_t thread; //POSIX thread identifier  // TODO: Create a thread to run startup with argument argv[1]  pthread\_create(&thread, null, startup, argv[1]);  // TODO: Block until the thread completes  pthread\_join(thread, null);  return 0;  } |
|  |

Answer the following questions about the program below. You can assume this would be run on a lab machine.

/\* foo.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

void \* foo(void \* args){

pthread\_t thread;

if(args == NULL){

pthread\_create(&thread, NULL, foo, (void \*) 1);

}

while(1);

}

int main(int argc, char \* argv[]){

pthread\_t threads[4];

int i;

for(i=0;i<4;i++){

pthread\_create(&threads[i], NULL, foo, NULL);

}

while(1);

}

(5) 9. When you run this program, how many threads are running. (Hint: you can run this program then use ps -L to count).

|  |
| --- |
| 9 |

(5) 10. What is deadlock? Provide a small (pseudo-code okay) example of how deadlock can arise.

|  |
| --- |
| When we talk about deadlock, we need to understand that is arises from when multiple threads require variables from one another. Given that these threads will pause while waiting for these variables, it can cause some issues when mutliple threads are waiting on eachother causing deadlock, when threads are mutually pasued waiting on one another.  pthread varX  pthread varY  func1:  mutexloc(varX)  mutexloc(varY)  mutexunloc(varX)  mutexunloc(varY)  func2:  mutexloc(varX)  mutexloc(varY)  mutexunloc(varY)  mutexunloc(varX)  main:  pthread 1, 2  init(varX)  inti(varY);  pthread\_create(&1, null, func1, null)  pthread\_create(&2, null, func2, null)  pthread\_join(1, null)  pthread\_join(2, null) |

(5) 11. For the code below, what is the expected output? Would you always get what you expect? Explain. You may want to run it a bunch of times.

/\* fun.c \*/

int shared = 0;

void \* fun(void \* args){

int i;

for(i=0;i<1000000;i++){ shared++; }

return NULL;

}

int main(){

pthread\_t t1,t2;

pthread\_create(&t1, NULL, fun, NULL);

pthread\_create(&t2, NULL, fun, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

printf("shared: %d\n", shared);

}

|  |
| --- |
| This function creates 2 threads that both increment share in a for loop that runs 1 million times. Given that there are 2 threads doing 1 million iterations, the expected output would be that shared Is 2 million, but we see that this is not the case, we get a different number each time. This is because threads are not always sheculed to run at the same time. So the output is far below 2 million, it only goes to a few thousand above 1 million. This is because once one thread has completed its for loop, the program will stop and print share, so it will be 1 million plus whatever integer the second thread got to before the first thread finished. |