Matt Ritchie

CS420

HW3

1) Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single-processor system? Yes or No or “it depends”? Explain.

Yes, because running the threads back to back will always take longer than spreading the threads across cores, like filling a bucket with multiple hoses instead of one.

2) The program shown below uses the Pthreads API. What would be the output from the program at LINE C and LINE P? Briefly explain how you arrived at your answer.

#include <pthread.h>

#include <stdio.h>

int value = 0;

void \*runner(void \*param); /\* the thread \*/

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

{

int pid;

pthread\_t tid;

pthread\_attr\_t attr;

pid = fork();

if (pid == 0) { /\* child process \*/

pthread\_attr\_init(&attr);

pthread\_create(&tid, &attr, runner, NULL);

pthread\_join(tid, NULL);

printf("CHILD: value = %d\n", value); /\* LINE C \*/

}

else if (pid > 0) { /\* parent process \*/

wait(NULL);

printf("PARENT: value = %d\n", value); /\* LINE P \*/

}

}

void \*runner(void \*param) {

value = 5;

pthread\_exit(0);

}

The value at C is 5 because after value is set to 5 in the last chunk of the code, which executes first for the child, but the parent doesn’t execute that so at P, value = 5.

3) Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be more than the number of processors in the system. Discuss the performance implications of the following scenarios.

a) The number of kernel threads allocated to the program is less than the number of processors.

Performance will be throttled since this configuration doesn’t take full advantage of the hardware by not using all of the processors.

b) The number of kernel threads allocated to the program is equal to the number of processors.

This would be the optimal configuration since each kernel thread can have its own dedicated processor.

c) The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.

This setup may or may not run as efficiently as setup B since some of the kernel threads will have to share processors or wait for an open one, but it also incurs overhead since you can take the user threads from the overflowed kernel threads and assign them to the other kernel threads.