COM S 352 Homework 3

Alec Meyer

February 18, 2021

Question 1

Heap memory
Global variables

Question 2

No, the operating system only sees a single process at a time so it will not schedule other threads. Being able to have multiple user-level threads does not result in parallelism. That means the kernel can only schedule one kernel thread at a time.

Question 3

yes, generally concurrency occurs when you are working with a single core system. Single core systems cannot have parallelism so the threads must be scheduled over time intervals, while parallelism allows multiple threads to run simultaneously.

Question 4

$\frac{1}{S+\frac{1-S}{N}}$ or $\frac{1}{1-P+\frac{P}{N}}$

A.
$\frac{1}{0.05+\frac{1-0.05}{8}} = 5.926$
$\frac{1}{0.05+\frac{1-0.05}{16}} = 9.143$
$\frac{9.143}{5.926} = 1.543 times$

B.
$\frac{1}{0.50+\frac{1-0.50}{8}} = 1.778$
$\frac{1}{0.50+\frac{1-0.50}{16}} = 1.882$
$\frac{1.882}{1.778} = 1.058 times$

A system depends on a lot more than just number of processing cores when measuring speeds. Algorithm efficiency, hardware efficiency also play a factor in speed.

Question 5

A. You would need two threads total, one for input and one for output. There is no reason to add anymore threads, efficiency will not be affected.
B. Four threads, because we want the same amount of threads as cores for the system since each core can use one thread at a time. Any less threads and we are wasting computing, and more threads than cores wouldn't benefit the system.

Question 6

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int id[4];
/* the thread */
void *runner(void *param)
{
    int *id = (int *)param;
    printf("THREAD %d FINISHED\n", *id);
    pthread_exit(0);
}
int main(int argc, char *argv[])
{
    pid_t pid;
    pthread_t tid[4];
    pthread_attr_t attr;
    pid = fork();
    pthread_kill(tid[1], 3);

    if (pid == 0)
    {
        /* child process */
        pthread_attr_init(&attr);
        for (int i=0; i<4; i++)
        {
            id[i] = i;
            pthread_create(&tid[i], &attr, runner, &id[i]);
        }
        for (int i = 0; i < 4; i++)
            pthread_join(tid[i], NULL); //waits for each thread

        printf("CHILD PROCESS FINISHED\n");
    }
    else if (pid > 0)
    {
        /* parent process*/
        wait(NULL); //waits for children
```

```
        printf("PARENT PROCESS FINISHED\n");
    }
    return 0;
}


OUTPUT:
    THREAD 0 FINISHED
    THREAD 1 FINISHED
    THREAD 2 FINISHED
    THREAD 3 FINISHED
    CHILD PROCESS FINISHED
    PARENT PROCESS FINISHED
```