Assignment 2: Multithreaded Programming Concepts

Ekkapot Charoenwanit
Course: Parallel Computing

March 11, 2021

Problem 2.1. Threads and Processes

1) Consider the following code and identify the segments in which the following variables/data live, assuming that they are not cached in a register.

```
static int n = 10240;

double dotProduct(double * a , double * b , int n)
{
    double res = 0.0;

    for(int i=0;i<n;i++)
        res += a[i]*b[i];

    return res;
}

int main(int argc,char** argv){

    double * a = new double[n];
    double * b = new double[n];

    for(int i=0;i<n;i++){
        a[i] = (i+1.2)/2.5;
    }

    for(int i=0;i<n;i++){
        b[i] = 1.2/(3.2*i+1.0);
    }

    std::cout << dotProduct(a,b,n) << std::endl;

    delete [] a;
    delete [] b;

    return 0;
}
```

- a and b inside *main*
- the data element $a[i]$ for some $0 \leq i \leq n-1$
- the *main* and the *dotProduct* function

2) What is wrong with the following code?

```
double * add(double a , double b)
{
```

```
    double res = a+b;
    return &res;
}

int main(int argc,char** argv)
{

  double * c_ptr = add(3.0,4.1);

  std::cout << *c_ptr << std::endl;

  return 0;
}
```

3) Suppose that you need to allocate an array of a sufficiently large size. Why is the following code bound to fail?

```
void someFunction(...)
{
//some other code
...
...
double myArray[ARRARY_SIZE];
//some other code
...
...
}
```

4) Are the following statements true or false?

- Context switches across threads within the same process are more lightweight than ones across processes.
- Threads of the same process share the same stack pointer.
- Threads of the same process share the virtual address space.
- Threads of the same process have their own instruction counters.
- Threads of the same process share the same general-purpose registers.

5) What are the advantage(s) and disadvantage(s) of multithreaded programming over multiprocess programming?

Problem 2.2. Multithreading

1) Define what a race condition is.

2) Given the following code snippet, demonstrate how a race condition can occur from running the code. Assume that the *runnerA* and the *runnerB* function are running in two threads, Thread A and Thread B, respectively.

```
int counter = 0;

void runnerA()//run by Thread A
{
    int i = 100;
    while(i>0){
      counter++;
      i--;
    }

}
```

```
void runnerB()//run by Thread B
{
    int i = 100;
    while(i>0){
      counter--;
      i--;
    }
}
```

3) Explain the difference(s) between a mutex implemented using a sleep-and-wake-up mechanism and one implemented using a busy-waiting mechanism.

4) Explain why spinlocks are not useful in uniprocessor systems.

5) In an I/O-intensive application, where threads periodically perform blocking I/O operations, how can oversubscription of threads improve the performance?