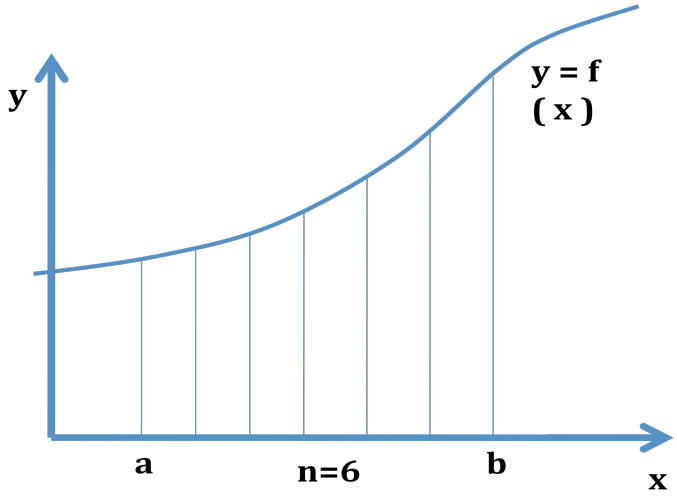
Eng 67: Lab 2

Lab 2 Learning Objectives:

1. To demonstrate the concept of functional decomposition.
2. Explore the use of multi-threading and termination control with POSIX threads.
3. Explore the use of mutual exclusion to control updates to shared data.

Background: Functional decomposition relates to decomposing a function into pieces that can be solved concurrently. Typically, if the function operates over some range of input values, this method is employed by dividing the range into partitions, concurrently solving each partition independently, and combining the individual results.

To see how this method is used, consider the general problem of performing numerical integration of a function **y=f(x)** over a closed interval **[a,b].** This can be achieved by computing an *approximation* to the area under a curve described by the function **y** between two points ***x=a*** and ***x=b*** as shown in Figure 1.



The area can be *approximated* using a simple idea:

1. Divide the interval between ***a*** and ***b*** into ***n strips*** as shown in Figure 1, each strip has width ***w=(b-a)/n***.

2. Number the strips from **1** to **n**. For each strip, approximate the area of the ***i*** th strip using a parallelogram: The length of the two sides of strip ***i*** are given by **f(a+((i-1)*w))** and **f(a+(i*w))**, and the perpendicular distance between the sides is ***w***.

3. Add up the area of all the strips to produce the area under the curve.

TASKS

1. Write a *sequential, re-usable* program module called ***integrate***. Your module should export a single function that approximates an integral to within a given level of *precision*. The function should take the following arguments as input:

    - A function ***f*** to be integrated.
    - An interval ***[a,b]*** over which to conduct the integral.
    - A precision ***p***, within which to calculate the integral.

    Your function should *repeatedly* calculate the integral with increasingly large numbers of strips ***s*** until the difference between successive approximations is less than the precision. Your function should return:

    a. The approximate value of the integral, *and*
    b. The total number of strip calculations used to construct the approximation.

2. Write a *mutli-threaded concurrent program* that computes approximations to several predefined functions. Your program should accept, as command line arguments, the following inputs:

    a. A number designating the built in function to be approximated.
    b. The interval over which to conduct the integration.
    c. The level of desired precision; a default number 0.01 should be used if no prevision is provided.
    d. The number of *theads* ***m*** to use for the calculation; a default number 10 should be used if no value ***m*** is provided. Each thread should utilize your *integrate* module (see Task 1) to compute a **1/mth** *portion* of the interval.

    Your program should print:

    1. The accumulated value of the integral.
    2. The total number of strip calculations used to calculate the approximation.

    Make sure you mutually exclude threads when updating the accumulated integral and number of strips.

3. Check your program operates correctly using the function **y=x+5** over the interval **a=5** to **b=10**. Make sure you calculate the area by hand first, so that you can check you get the right answer. Notice what happens if you use different levels of precision – *why*?

4. Check your program on the function **y=2x^2+9x+4** over the interval **3** and **12** and again notice what happens with different levels of precision – *why?*

5. **[ENG 115: GRADUATE STUDENTS ONLY]** Modify your program to reuse the *queue* module you designed in Lab 1. Your program should:

    a. Use ***(m + 1)*** threads.
    b. One thread, the *generator*, should generate a queue of partitions to be solved -- assume the number of partitions is 15 times the number of threads.
    c. The remaining ***m*** threads should repeatedly remove a partition from the queue, solve it, and update the accumulated results.
    d. When no partitions remain to be solved, the final results are printed.

    *Ensure that you mutually exclude threads from the queue when updating its content. Ensure that you consider the possibility that the **m** threads may all terminate before the generator has completed its work !*

TOPICS

Research the following concepts:

- TCP and UDP are network *protocols – what is a protocol?*
- What is the difference between TCP and UDP?
- What is a Packet and how big is the largest UDP packet ?
- What is the difference between a MAC address and an IP address ?
- What are the concepts of PORT and SOCKET in the context of TCP and UDP?
- What are the *sockaddr **and** sockaddr_in* structures?
- How do you initialize a *sockaddr_in* structure?
- What is the constant AF_INET used for?
- What do the following functions do in UDP:
    - socket(…)
    - memset(…)
    - inet_addr(…)
    - htons(…)
    - sendto(…)
    - recvfrom(…)
    - close(…)

- **ENGS 115 ONLY:**
    - What is the constant INADDR_ANY used for?
    - What do the bind(…) and htonl(…) functions do in a UDP server?