Andrey Varakin
Jeremy Goldberg
Miliano Mikol

HW #1

1. What is the difference between an operating system and middleware?

An operating system uses hardware resources of a computer system to provide support for the execution of other software while middleware occupies a middle position between application programs and operating systems.

- They rely upon different underlying providers of lower-level services.
- An operating system provides the services by using features of hardware.

Middleware provides services by using features supported by an operating system. Thus, middleware is layered on top of an operating system.

2. What is the relationship between threads and processes?

A thread is the unit of concurrency. Any one sequence of programmed actions is a thread and a program may create multiple threads. That is, if the program calls for several independent sequences to run concurrently. Running a program also entails creating a process. A process is a container that holds the thread(s) and protects them from unwanted interactions with other unrelated threads running on the same computer.

3. Of all the topics previewed in chapter one of the text book, which one are you most looking forward to learning more about? Why?

Transaction atomicity and how the integrity of metadata, directories, and indexes are protected. These topics were briefly discussed in the databases class. I am interested in seeing what additional mechanisms are used in systems to ensure a steady level of security for data. The reason why is both to satisfy mere curiosity along with general interest in the field of cybersecurity.

Andrey Varakin
Jeremy Goldberg
Miliano Mikol

4. Suppose thread A goes through a loop 100 times, each time performing one disk I/O operation, taking 10 milliseconds, and then some computation, taking 1 millisecond. While each 10-millisecond disk operation is in progress, thread A cannot make any use of the processor. Thread B runs for 1 second, purely in the processor, with no I/O. One millisecond of processor time is spent each time the processor switches threads; other than this switching cost, there is no problem with the processor working on thread B during one of thread A's I/O operations. (The processor and disk drive do not contend for memory access bandwidth, for example.)

a) **Suppose the processor and disk work purely on thread A until its completion, and then the processor switches to thread B and runs all of that thread. What will the total elapsed time be?**

2.101 seconds (math below).

b) **Suppose the processor starts out working on thread A, but every time thread A performs a disk operation, the processor switches to B during the operation and then back to A upon the disk operation's completion. What will the total elapsed time be?**

1.3 seconds (math below).

c) **In your opinion, which do you think is more efficient, and why?**

The latter method because it makes use of the processor during thread A's I/O disk operation. It supports greater resource utilization and reduces overall runtime.

#4

A: Loop = 100; I/O = 0.010_s; Compute = 0.001_s

B: Process = 1_s

Switch = 0.001_s

a) A Then B

$[A] + [\text{switch}] + [B] = [100(0.010 + 0.001)] + [0.001] + [1] = \boxed{2.101_s}$

b) A and B

$[A] + [2 \cdot \text{switch} \cdot A_{loop}] + [A_{loop}(-A_{IO}) + B)] = [100(0.010 + 0.001)] + [2 \cdot 100(0.001)] + [100(-0.001) + 1] = \boxed{1.3_s}$

Andrey Varakin
Jeremy Goldberg
Miliano Mikol

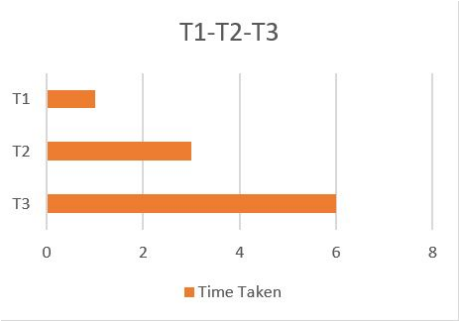
5. Find and read the documentation for pthread_cancel(). Then, using your C programming environment, use the information and the model provided in Figure 2.4 on page 26 of the text book to write a program in which the initial (main) thread creates a second thread. The main thread should read input from the keyboard, waiting until the user presses the Enter key. At that point, it should kill off the second thread and print out a message reporting that it has done so. Meanwhile, the second thread should be in an infinite loop, each time around sleeping five seconds and then printing out a message. Try running your program. Can the sleeping thread print its periodic messages while the main thread is waiting for keyboard input? Can the main thread read input, kill the sleeping thread, and print a message while the sleeping thread is in the early part of one of its five-second sleeps?

```c
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void *childThread(void *ignore) {
  while(1) {
    sleep(5);
    printf("The child thread continues to sleep\n");
  }
}

int main( int argc, char * argv[] ) {
  pthread_t childThreadID;

  //initializing child thread
  pthread_create(&childThreadID, NULL, childThread, NULL);

  //Waiting for the ENTER key to be pressed in order to cancel the child thread
  char ch = fgetc(stdin);
  if(ch == 10) {
    pthread_cancel(childThreadID);
  }
  printf("Child thread has been cancelled\n");
  return 0;
}
```

Yes, the sleeping thread will continue to output its message as long as the enter key has not been pressed. The main thread can read input, kill the sleeping thread, and print its message at any point, even if the sleeping thread is in one of its cycles.

Andrey Varakin
Jeremy Goldberg
Miliano Mikol

6. Suppose a system has three threads (T1, T2, and T3) that are all available to run at time 0 and need one, two, and three seconds of processing, respectively. Suppose each thread is run to completion before starting another. Draw six different Gantt charts, one for each possible order the threads can be run in. or each chart, compute the turnaround time of each thread; that is, the time elapsed from when it was ready (time 0) until it is complete. Also, compute the average turnaround time for each order. Which order has the shortest average turnaround time? What is the name of the scheduling policy that produces this order?



Avg. Turnaround time: 3.33s

Avg. Turnaround time: 3.66s

Avg. Turnaround time: 3.66s

Avg. Turnaround time: 4.33s

Avg. Turnaround time: 4.33s

Avg. Turnaround time: 4.66s

The T1-T2-T3 order has the shortest turnaround time at 3.33 seconds. It follows the Shortest Job First (SJF) scheduling policy.

Andrey Varakin
Jeremy Goldberg
Miliano Mikol

7. Google the C standard library API and find out how to get information from the command line by using a printf() call to display a prompt, then another call [which you will look up] to get the user input. Write a program in C to prompt the user demographic information including name, age, class year, and any three other data times you wish. Structure the program as a call-and-response program such that each data item is a single question with a single answer. When all data has been obtained, display the data on the console. Each data item must be on a separate line, and it must be appropriately labeled. The output must be done using a single printf() statement.

```c
#include <stdio.h>

int main(int argc, char * argv[]) {
   char name[100], age[3], class_yr[4], major[100], ethnicity[100], gender[100];

   printf("What is your name?\n");
   scanf("%s", name);

   printf("What is your age?\n");
   scanf("%s", age);

   printf("What is your class year?\n");
   scanf("%s", class_yr);

   printf("What is your major?\n");
   scanf("%s", major);

   printf("What is your ethnicity?\n");
   scanf("%s", ethnicity);

   printf("What is your gender?\n");
   scanf("%s", gender);

   printf("Name: %s\nAge: %s\nClass Year: %s\nMajor: %s\nEthnicity: %s\nGender: %s\n",
name, age, class_yr, major, ethnicity, gender);

   return 0;
}
```