Aaron Floreani
Ryan Toan Nguyen
Coby Schumitzky
Shaun Seah

Operating Systems Homework #1

1. The operating system is the foundation upon which all other software is built and serves as the interface between applications and the computer's hardware while the middleware acts as a bridge between an operating system and its applications where it enables the communication of data between them. In addition, unlike an operating system, middleware is not a standalone software and requires an operating system to run.

2. The relationship between threads and processes can be visualized as multiple threads within a single process working together to accomplish a common goal, while each thread has its own specific task to perform. To clarify a process is an instance of a running program, with its own memory space and system resources. A thread, on the other hand, is an independent unit of execution within a process that shares the same memory space and system resources.

3. Out of all the future topics in this class, we are the most excited about shell scripting. We want to be able to automate routine tasks and see how we can implement custom tools and scripts. This is also a great first step in learning more advanced programming concepts and could speed up our coding/work in the future.

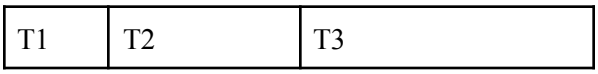
4. A. Thread A = 100 operations * 10ms for disk operations + 100 operations * 1ms for computation
   = 1100ms
   +1ms switch time
   Thread B = 1000ms in processor

   Total = 2101ms
   B. Thread A has 100 operations * 1ms for switching time 100 times to thread B = 100ms
   + Thread B runs concurrent with Thread A therefore 10ms * 100 = 1000ms
   + Switch from Thread B to Thread A = 100* 1ms = 100ms
   + Thread A computation 100 operations * 1ms = 1000ms

   Total = 1300ms
   C. Due to the computations above, part B would be more efficient because it takes advantage of the fact that Thread A and Thread B could run concurrently

5. Note: For the code for 5, refer to the program titled “threadCancel.c” in the Github Repository. The sleeping thread does, indeed, print periodic messages as the main, or primary, thread awaits input. Furthermore, when running the program, we can see that the sleep does not put the child thread into a state that does not respond to the `pthread_cancel()`. When running, even when in an early stage of the sleeping, the thread still responds to the cancellation request which implies that the `sleep()` function does not put the child thread into a disabled cancelability state and the queued cancellation rTequest is handled immediately.

6.

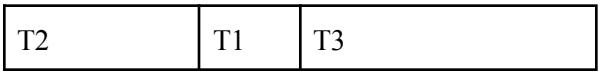


0 1 3 6

Turnaround times (seconds): T1 = 1, T2 = 3, T3 = 6
Average turnaround time for this order: 3.33 seconds

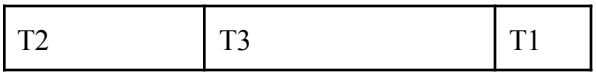
0 1 4 6

Turnaround times (seconds): T1 = 1, T2 = 6 , T3 = 4
Average turnaround time for this order: 3.67 seconds



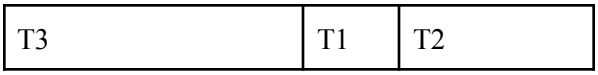
0 2 3 6

Turnaround times (seconds): T1 = 3, T2 = 2, T3 = 6
Average turnaround time for this order: 3.67 seconds



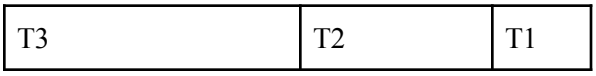
0 2 5 6

Turnaround times (seconds): T1 = 6, T2 = 2, T3 = 5
Average turnaround time for this order: 4.33 seconds



0 3 4 6

Turnaround times (seconds): T1 = 4, T2 = 6, T3 = 3
Average turnaround time for this order: 4.33 seconds



0 3 5 6

Turnaround times (seconds): T1 = 6, T2 = 5, T3 = 3
Average turnaround time for this order: 4.67 seconds

The shortest average turnaround time is 3.33 seconds for the first order, which is T1, T2, and T3. This scheduling policy is known as Earliest Deadline First Scheduling.

7.

To get information from the command line in C, you can use printf() to display a prompt and then scanf() to get the user input. printf() is a function from the standard library API in C that outputs formatted data to the standard output, usually the console. scanf() is another function from the standard library API in C that reads formatted input from the standard input, usually the keyboard.

Here's an example program that prompts the user for their name and age:

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
   char name[100];
   int age;

   printf("Please enter your name: ");
   fgets(name, 100, stdin);
   name[strlen(name) - 1] = '\0';
   printf("Please enter your age: ");
   scanf("%d", &age);

   printf("Your name is %s and you are %d years old.\n", name, age);

   return 0;
}
```

In this example, printf("Please enter your name: ") displays the prompt for the user's name, and scanf("%s", name) reads the user's input and stores it in the name variable. Similarly, printf("Please enter your age: ") displays the prompt for the user's age, and scanf("%d", &age) reads the user's input and stores it in the age variable. Finally, printf("Your name is %s and you are %d years old.\n", name, age) outputs the user's name and age to the console.

Here's a program in C that prompts the user for their demographic information including name, age, class year, major, GPA, and club:

```c
#include <stdio.h>

int main(void)
{
   char name[100];
   int age;
   int class_year;
   char major[100];
   float GPA;
   char club[100];

   printf("Please enter your name: ");
   fgets(name, 100, stdin);
   printf("Please enter your age: ");
   scanf("%d", &age);
   getchar();
   printf("Please enter your class year: ");
   scanf("%d", &class_year);
   getchar();
   printf("Please enter your major: ");
   fgets(major, 100, stdin);
   printf("Please enter your GPA: ");
   scanf("%f", &GPA);
   getchar();
   printf("Please enter your club: ");
   fgets(club, 100, stdin);

   printf("\nYour demographic information: \nName: %s\nAge: %d\nClass Year:
%d\nMajor: %s\nGPA: %.2f\nClub: %s\n", name, age, class_year, major, GPA, club);

   return 0;
}
```

The program uses printf() to display a prompt for each data item and scanf() along with fgets to get the user input. Fgets() terminates at the size of the buffer or at a new line. Furthermore, the variable used with fgets has a new line at the end of the buffer so we replace that with a null character. Lastly, the user input is then displayed on the console using a single printf() statement.