

Homework01

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

The operating system is a software that uses the hardware and software resources of a computer system to provide support for the execution of other software. Middleware is the software between the application programs and the operating system which provides common services and capabilities to applications other than what is offered by the operating system.

2. What is the relationship between threads and processes?

Threads are the “unit of concurrency,” or a sequence of computational steps that follow from the instructions laid out in a program. A process is a “container” that holds and protects threads to ensure that they do not overlap each other. Therefore, a process can hold one or multiple threads, in which those threads do not interact with threads within other processes.

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

We are most excited about learning more on network security and data gathering. For the common person cyber security and internet or data privacy is non-existent. With the amount of thriving technology, our society is only beginning to recognize laws and policies that protect users and their information. So knowing the ins and outs of your network is essential.

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?**

Patrick Jayoma
Parker Bath
Kyle Traverse
CMSI - 387 - 01

Thread A: 10 milliseconds + 1 millisecond = 11 milliseconds * 100 TIMES = 1100 milliseconds = 1.1 seconds
SwitchThreads (A,B) = 1 millisecond

Thread B: 1 second

Total: 1100 milliseconds + 1 millisecond + 1 millisecond = 2101 milliseconds.

- 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?**

Total Time = 100 * (10ms + 1ms + 1ms + 1ms) = 1300ms

The total elapsed time is 1300 milliseconds.

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

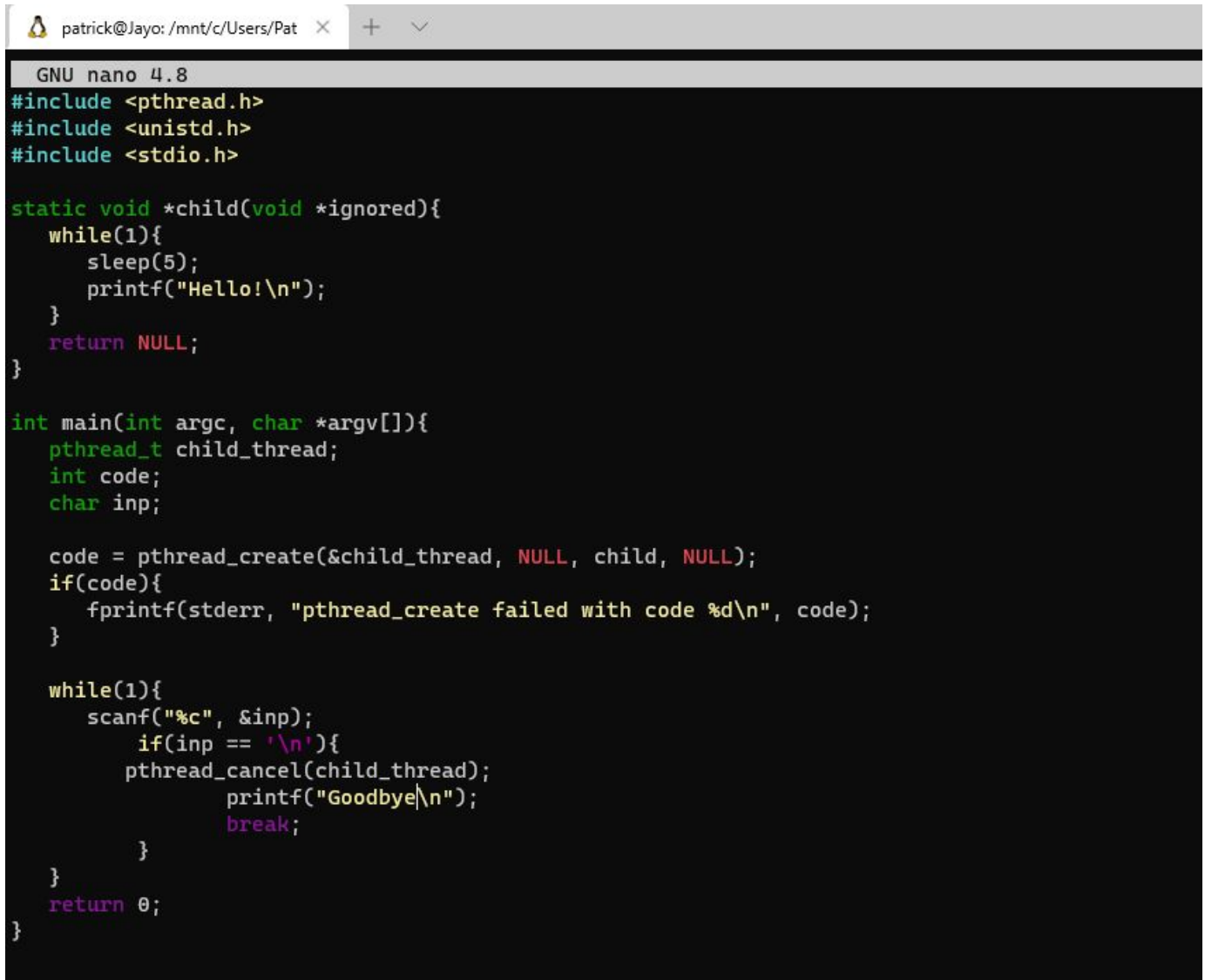
B is more efficient because of its resource utilization: keeping the hardware occupied if a thread is idle by allowing other threads to make use of that hardware space. Such resourcefulness may reduce run time significantly

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 textbook to write a program in which the initial (main) thread creates a second thread. The main thread should sit on a read call of some kind, waiting to 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?

The sleeping thread can print its concurring messages while the main thread is waiting for keyboard input.

The main thread can 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

Patrick Jayoma
Parker Bath
Kyle Traverse
CMSI - 387 - 01



```
patrick@Jayo: /mnt/c/Users/Pat x + -
GNU nano 4.8
#include <pthread.h>
#include <unistd.h>
#include <stdio.h>

static void *child(void *ignored){
    while(1){
        sleep(5);
        printf("Hello!\n");
    }
    return NULL;
}

int main(int argc, char *argv[]){
    pthread_t child_thread;
    int code;
    char inp;

    code = pthread_create(&child_thread, NULL, child, NULL);
    if(code){
        fprintf(stderr, "pthread_create failed with code %d\n", code);
    }

    while(1){
        scanf("%c", &inp);
        if(inp == '\n'){
            pthread_cancel(child_thread);
            printf("Goodbye!\n");
            break;
        }
    }
    return 0;
}
```

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. For 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?

Patrick Jayoma
Parker Bath
Kyle Traverse
CMSI - 387 - 01

T1		T2			T3			
0		1			3			6
T1		T3				T2		
0		1				4		6
T2		T1		T3				
0		2		3				6
T2				T3			T1	
0				2			5	6
T3				T1		T2		
0				3		4		6
T3				T2			T1	
0				3			5	6

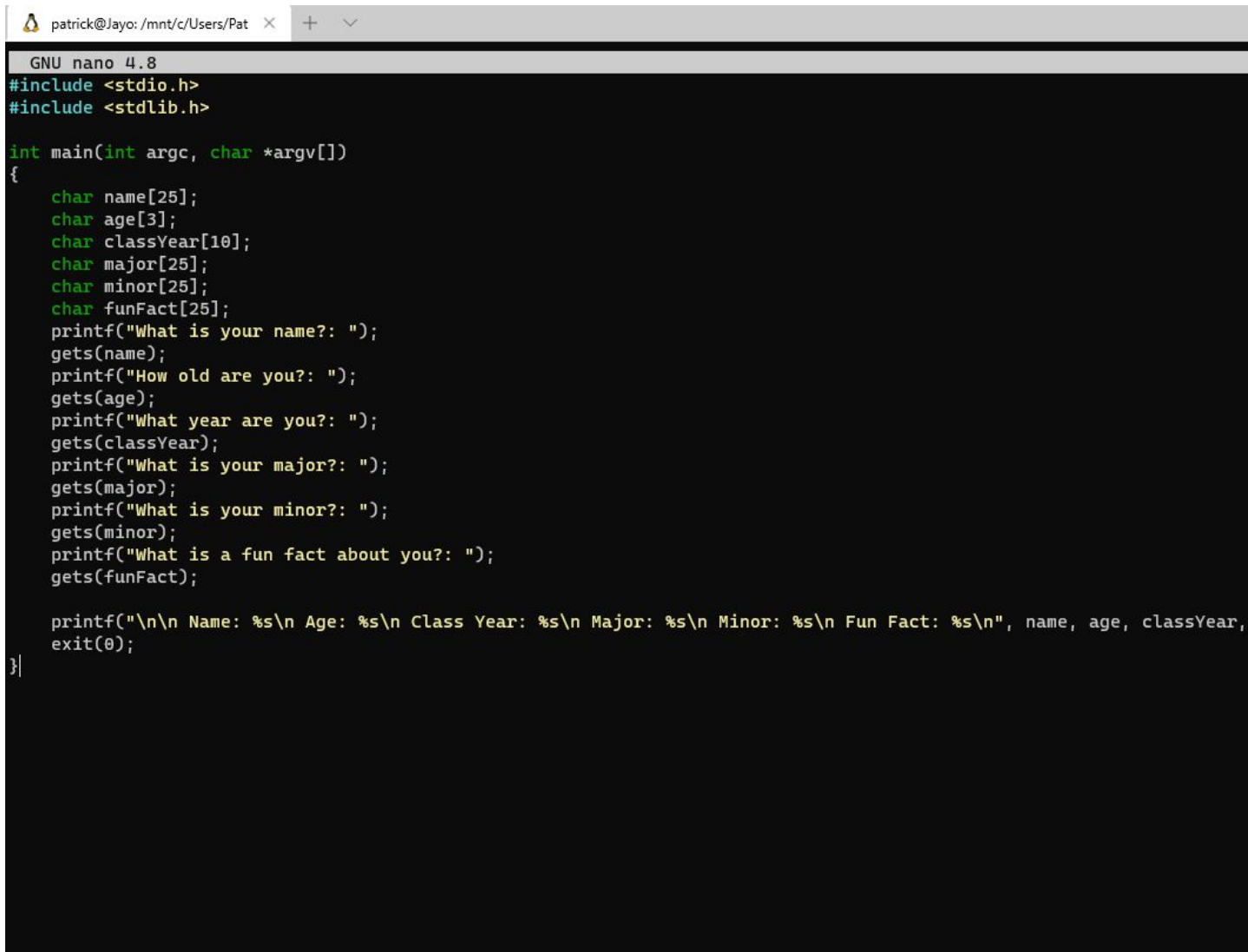
T1	1	
T2	3	
T3	6	Average: $10/3 = 3.3333333$ seconds
T1	1	
T2	6	
T3	4	Average: $11/3 = 3.6666667$ seconds
T1	3	
T2	2	
T3	6	Average: $11/3 = 3.6666667$ seconds

Patrick Jayoma
Parker Bath
Kyle Traverse
CMSI - 387 - 01

T1	6	
T2	2	
T3	5	Average: $13/3 = 4.333333333$ seconds
T1	4	
T2	6	
T3	3	Average: $13/3 = 4.333333333$ seconds
T1	6	
T2	5	
T3	3	Average: $14/3 = 4.66666667$ seconds

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.

Patrick Jayoma
Parker Bath
Kyle Traverse
CMSI - 387 - 01

A screenshot of a terminal window with a dark background. The window title bar shows 'patrick@Jayo: /mnt/c/Users/Pat' and standard window controls. The terminal content shows the GNU nano 4.8 editor with a C program. The code includes `<stdio.h>` and `<stdlib.h>`. The `main` function takes `argc` and `argv` as arguments. It declares several character arrays: `name[25]`, `age[3]`, `classYear[10]`, `major[25]`, `minor[25]`, and `funFact[25]`. It then uses `printf` to prompt the user for their name, age, class year, major, minor, and a fun fact, followed by `gets` to read the input. Finally, it prints all the collected information in a single line and calls `exit(0)`.

```
GNU nano 4.8
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char name[25];
    char age[3];
    char classYear[10];
    char major[25];
    char minor[25];
    char funFact[25];
    printf("What is your name?: ");
    gets(name);
    printf("How old are you?: ");
    gets(age);
    printf("What year are you?: ");
    gets(classYear);
    printf("What is your major?: ");
    gets(major);
    printf("What is your minor?: ");
    gets(minor);
    printf("What is a fun fact about you?: ");
    gets(funFact);

    printf("\n\n Name: %s\n Age: %s\n Class Year: %s\n Major: %s\n Minor: %s\n Fun Fact: %s\n", name, age, classYear,
    exit(0);
}
```