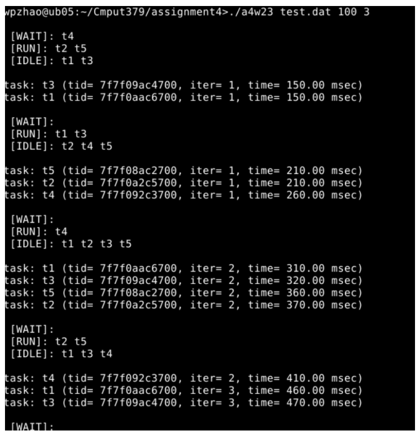
Cmput 379 Project Report

- **Objectives**: state the project objectives and value from your point of view (which may be different from the one mentioned above)
    - The objective of this project is to help us learn how to use and create threads with synchronization and deadlock in mind, and build multiple threads that help us solve this issue. Through solving the different issues occurring we can observe these concepts in much greater detail.
- **Design Overview:** highlight in point-form the important features of your design
    - I created the main TASK structure for each task to hold its values, and the RESOURCE structure to hold each resource, and I created a list of mutexes to guard each of the resource types in the resource array. In the thread function, it will first try to get all of the mutex, once it gets all of the mutex then it will take away the resource and release the locks, or it will not take and release all the previously held locks. If it has gotten the lock and there is no resource then it will give all the held resources back and release the locks. There is also a sleep time for the function after each attempt.
    - The monitor thread will control one single value which is the printing boolean, once it starts printing all other task threads will be in a busy loop and not updating the status until the printing has stopped.
        - It extracts the monitor time from the arg argument, which is passed as a pointer to an integer, by dereferencing it with *((int *)arg).
        - Inside the infinite loop, it sleeps for the monitor time using usleep(monitor_time*1000), where usleep is a function that sleeps for a specified number of microseconds.
        - It then sets a global flag printing to true to indicate that printing is in progress.
        - It prints the status of tasks and resources using a series of nested loops and conditional statements.
        - It sets the printing flag back to false after printing is complete.
    - The main function in the program reads input from a file specified as a command-line argument, tokenizes the input to extract resource and task information, and initializes data structures accordingly. It creates a monitoring thread using pthreads and starts the monitoring process. The program appears to be designed to monitor the status of tasks and resources in a concurrent system.
    - The end result will be printed in the end of the main thread, all of the results will be getting from tasks that are stored as a global variable so we can access it at the end.

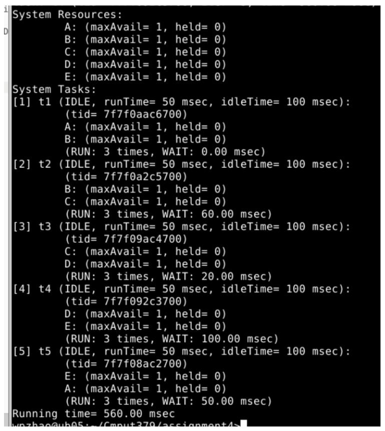
- **Project Status:** describe the status of your project (to what degree the programs work as specified, e.g., tested and working, working with some known issues, etc.) mention difficulties encountered in the implementation

    - The assignment was completed successfully and this is shown through the testing. During the test, the most difficult issue occurred would be setting up each thread to avoid deadlock with mutex. There were many errors occurring on how to find out where I should check if each mutex is held by the thread or not. In addition, there were some edge cases that were very challenging.
    - Another challenging element is the fact of storing the correct data in the correct place along with the mutex, but it was able to be solved in the end and the program works as intended

- **Testing and Results:** comment on how you tested your implementation, and discuss the obtained results.
    - The testing was done on the different lab machine with the sample data provided as well as mine own tests which consist of the same commands.
    - 

```
wpzhao@ub05:~/Cmput379/assignment4>./a4w23 test.dat 100 3

 [WAIT]: t4
 [RUN]: t2 t5
 [IDLE]: t1 t3

task: t3 (tid= 7f7f09ac4700, iter= 1, time= 150.00 msec)
task: t1 (tid= 7f7f0aac6700, iter= 1, time= 150.00 msec)

 [WAIT]:
 [RUN]: t1 t3
 [IDLE]: t2 t4 t5

task: t5 (tid= 7f7f08ac2700, iter= 1, time= 210.00 msec)
task: t2 (tid= 7f7f0a2c5700, iter= 1, time= 210.00 msec)
task: t4 (tid= 7f7f092c3700, iter= 1, time= 260.00 msec)

 [WAIT]:
 [RUN]: t4
 [IDLE]: t1 t2 t3 t5

task: t1 (tid= 7f7f0aac6700, iter= 2, time= 310.00 msec)
task: t3 (tid= 7f7f09ac4700, iter= 2, time= 320.00 msec)
task: t5 (tid= 7f7f08ac2700, iter= 2, time= 360.00 msec)
task: t2 (tid= 7f7f0a2c5700, iter= 2, time= 370.00 msec)

 [WAIT]:
 [RUN]: t2 t5
 [IDLE]: t1 t3 t4

task: t4 (tid= 7f7f092c3700, iter= 2, time= 410.00 msec)
task: t1 (tid= 7f7f0aac6700, iter= 3, time= 460.00 msec)
task: t3 (tid= 7f7f09ac4700, iter= 3, time= 470.00 msec)

 [WAIT]:
```

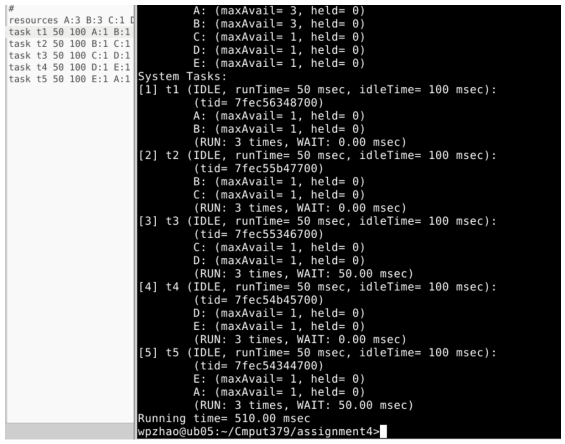
make c

    - When running the assignment we can see that all of the result printed works with what you expect, since t1 has all of A and B resource t2 is gonna have to start later when t1 releases all its held resources after its runs, the rest are the same
    - 

```
System Resources:
        A: (maxAvail= 1, held= 0)
        B: (maxAvail= 1, held= 0)
        C: (maxAvail= 1, held= 0)
        D: (maxAvail= 1, held= 0)
        E: (maxAvail= 1, held= 0)
System Tasks:
[1] t1 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7f7f0aac6700)
        A: (maxAvail= 1, held= 0)
        B: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 0.00 msec)
[2] t2 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7f7f0a2c5700)
        B: (maxAvail= 1, held= 0)
        C: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 60.00 msec)
[3] t3 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7f7f09ac4700)
        C: (maxAvail= 1, held= 0)
        D: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 20.00 msec)
[4] t4 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7f7f092c3700)
        D: (maxAvail= 1, held= 0)
        E: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 100.00 msec)
[5] t5 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7f7f08ac2700)
        E: (maxAvail= 1, held= 0)
        A: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 50.00 msec)
Running time= 560.00 msec
wpzhao@ub05:~/Cmput379/assignment4>
```

    - We can also see from the example run, we can see that t1 did not wait at all, as this is expected as it ran first and the only same resources that can be taken by is A and B, but since we already taken A first, t5 can never taken it first as we start t1 first, and B had by t2, but because t1 ran first it will no wait for resource B, but sometimes with rae case it have to wait for B as t2 got the mutex lock for B first.

There were also other tests created similar to the example test to test if there is more than one resource available what would happen.

- 

```
#
resources A:3 B:3 C:1 D
task t1 50 100 A:1 B:1
task t2 50 100 B:1 C:1
task t3 50 100 C:1 D:1
task t4 50 100 D:1 E:1
task t5 50 100 E:1 A:1
```

```
        A: (maxAvail= 3, held= 0)
        B: (maxAvail= 3, held= 0)
        C: (maxAvail= 1, held= 0)
        D: (maxAvail= 1, held= 0)
        E: (maxAvail= 1, held= 0)
System Tasks:
[1] t1 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7fec56348700)
        A: (maxAvail= 1, held= 0)
        B: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 0.00 msec)
[2] t2 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7fec55b47700)
        B: (maxAvail= 1, held= 0)
        C: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 0.00 msec)
[3] t3 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7fec55346700)
        C: (maxAvail= 1, held= 0)
        D: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 50.00 msec)
[4] t4 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7fec54b45700)
        D: (maxAvail= 1, held= 0)
        E: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 0.00 msec)
[5] t5 (IDLE, runTime= 50 msec, idleTime= 100 msec):
        (tid= 7fec54344700)
        E: (maxAvail= 1, held= 0)
        A: (maxAvail= 1, held= 0)
        (RUN: 3 times, WAIT: 50.00 msec)
Running time= 510.00 msec
wpzhao@ub05:~/Cmput379/assignment4>
```

- As expected we do not have to wait at all for both t1, t2 as there is more than 1 B resource.

- **Acknowledgments:** acknowledge sources of assistance
    - Cmput 379 lab notes posted on eclass
    - Advanced Programming in the UNIX® Environment, Third Edition
    - Operating System Concepts, 10th Edition