CSCI-3753: Operating Systems
Fall 2018

Anh Nguyen

Department of Computer Science

University of Colorado Boulder

Announcements

- **Mid-term feedback survey**

https://cuboulder.qualtrics.com/jfe/form/SV_cBa3b6MJQPSg3vD

- **PS2** correctness
  - Question 1: What are the different ways of message passing for interprocess communication?
  - Question 3: What are the different ways to do interprocess communication?

- **PA3** grading rule
  - 40% on code submitted
  - 60% on answering questions in the interview

Programming Assignment Three

Overview

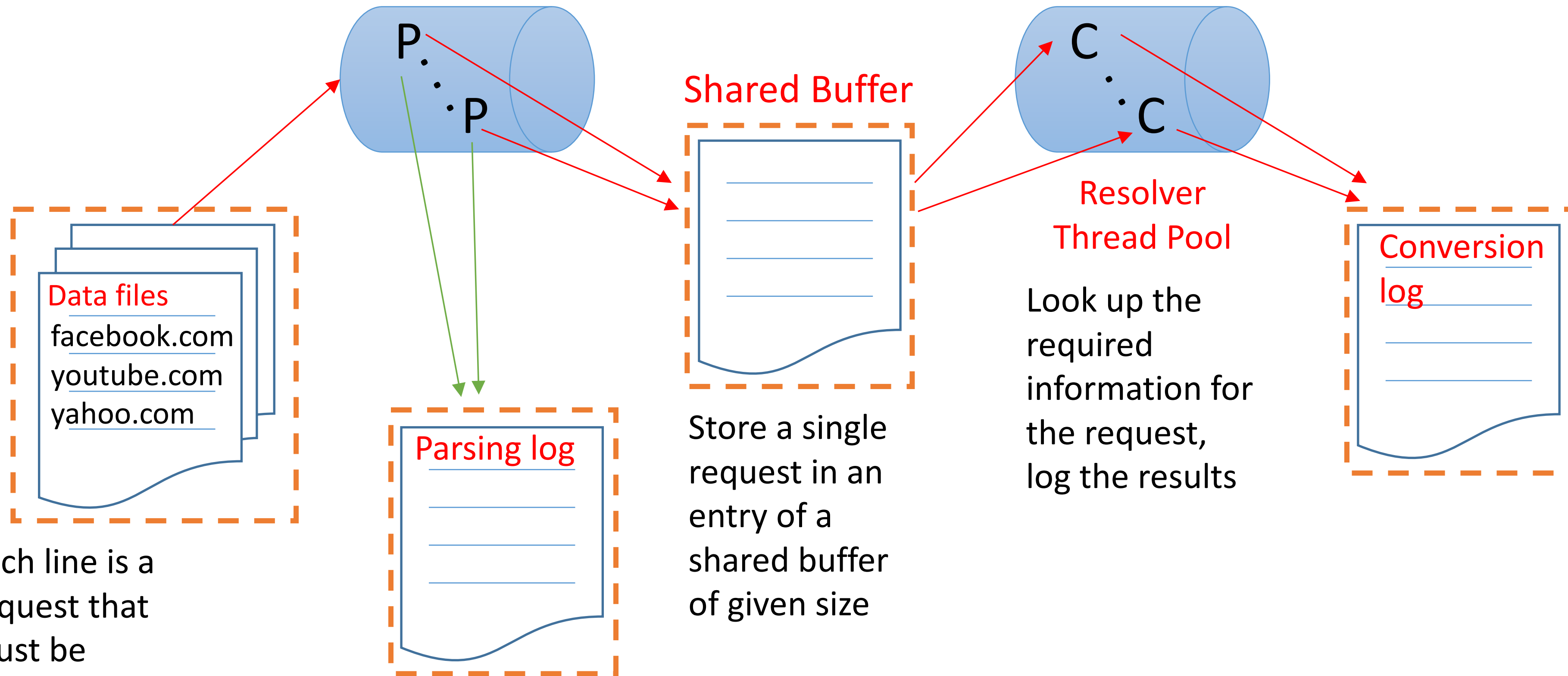
- GOAL: Using pthreads to code a DNS name resolution engine

```
For each file
  For each line of file
    Parse the line for domain names
      For each domain name
        Find IP for domain name
        Write information to output file
```

- There are multiple files to process.
- Get the next line from a file and parse the line into multiple requests.
- The producer will place each request into the shared buffer.

- The consumer will get the next request from the shared buffer.
- Lookup the IP address from the name.
- Log the information into the output file.

Overview



Each line is a request that must be processed individually.

Record which process handled which request

Store a single request in an entry of a shared buffer of given size

Look up the required information for the request, log the results

Each dotted line is around a shared resource that must be protected from race conditions.

Program Inputs

- Number of parser threads to place into the thread pool.
- Number of converter threads to place into the thread pool.
- Parsing log: the file into which all the parser status information is written.
  - Per line format: Thread <thread id> serviced ### files.
  - pthread_self()
- Conversion log: The file into which all the converter status information is written.
  - Per line format: www.google.com, 74.125.224.81
- Data files: List of filenames that are to be processed. Each file contains a list of domain names, one per line, that are to be resolved.

Program Focus

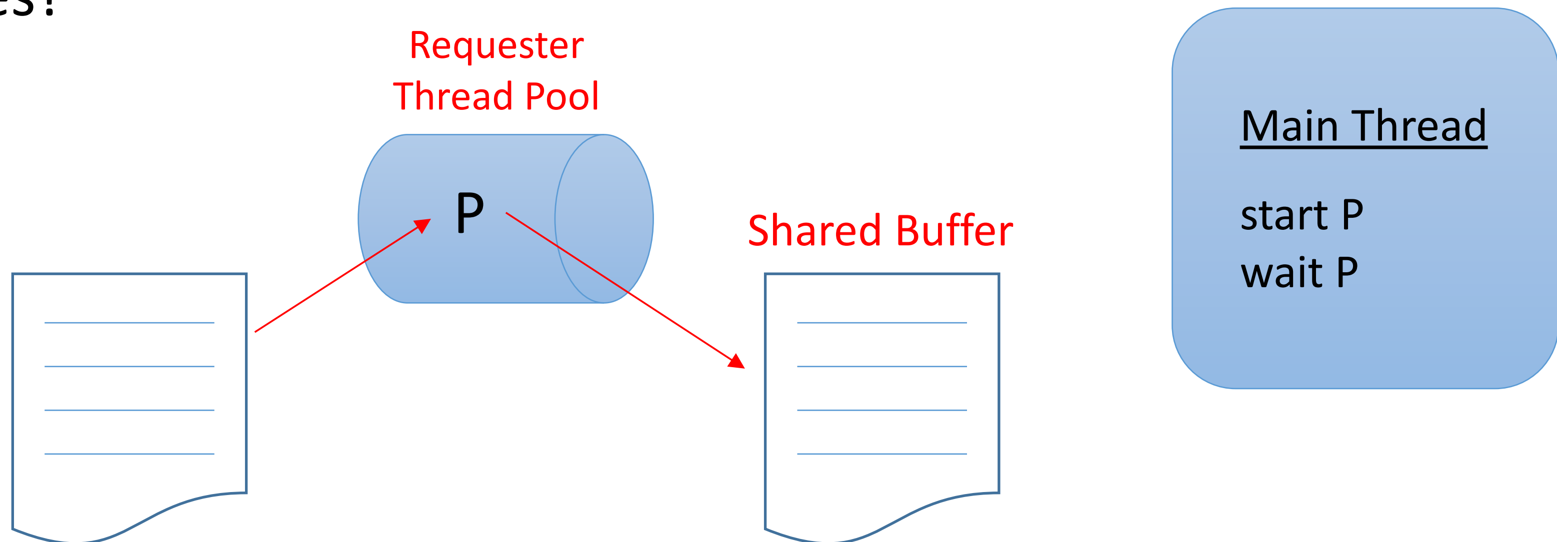
- Synchronization
- Deadlock

→ Suggested solutions

- Mutex
- Semaphore
- Conditional variables

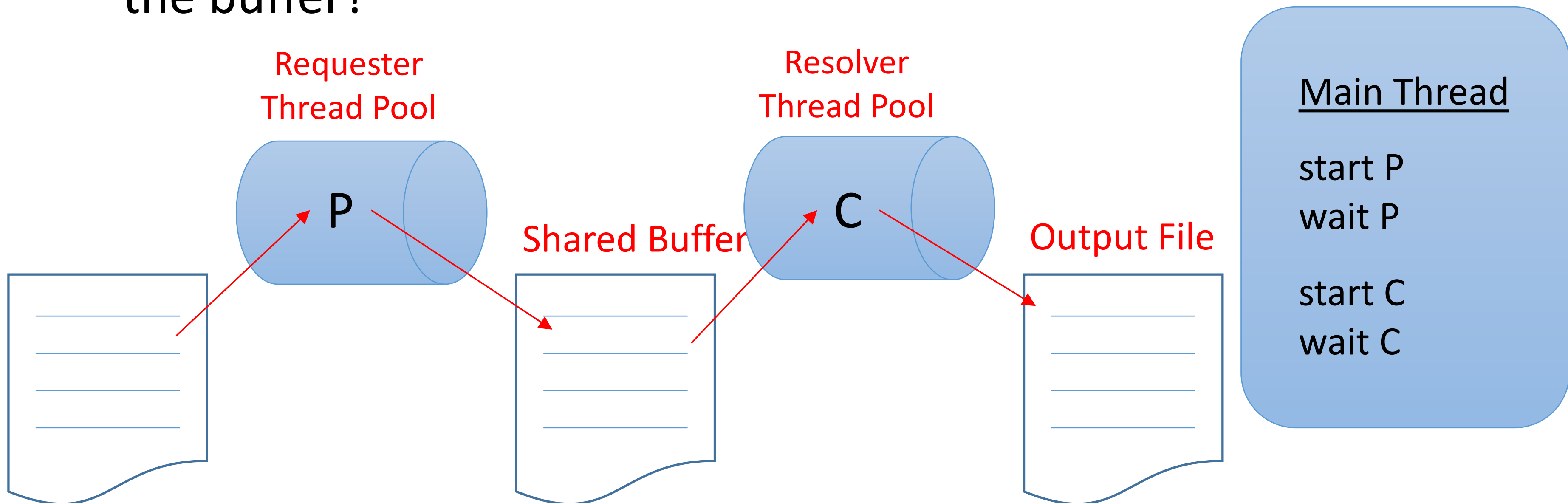
Implementation – Step 1

- Action: Create a simple program to create a parsing thread that will repeatedly read a line from a given file and add an entry into the shared buffer
- Validation: Does the buffer have the correct number of entries?



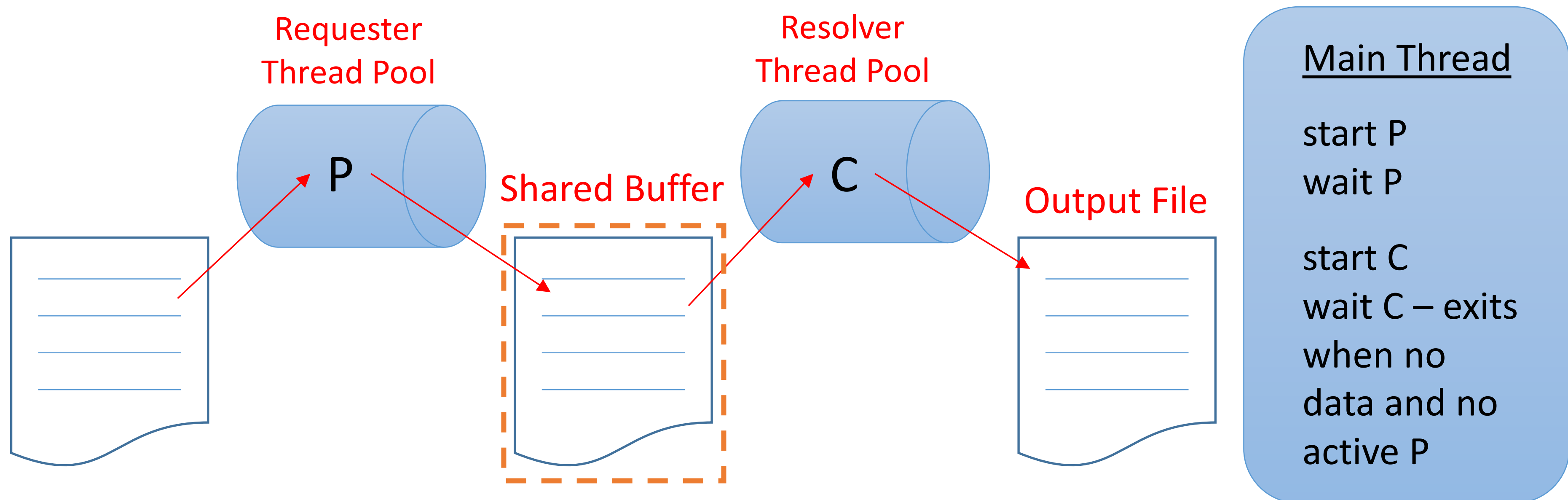
Implementation – Step 2

- **Action:** Use the result in step 1, once that process is complete, start a conversion thread to take items out of the buffer. Then, write the results to an output file.
- **Validation:** Does the output file contain the entries stored in the buffer?



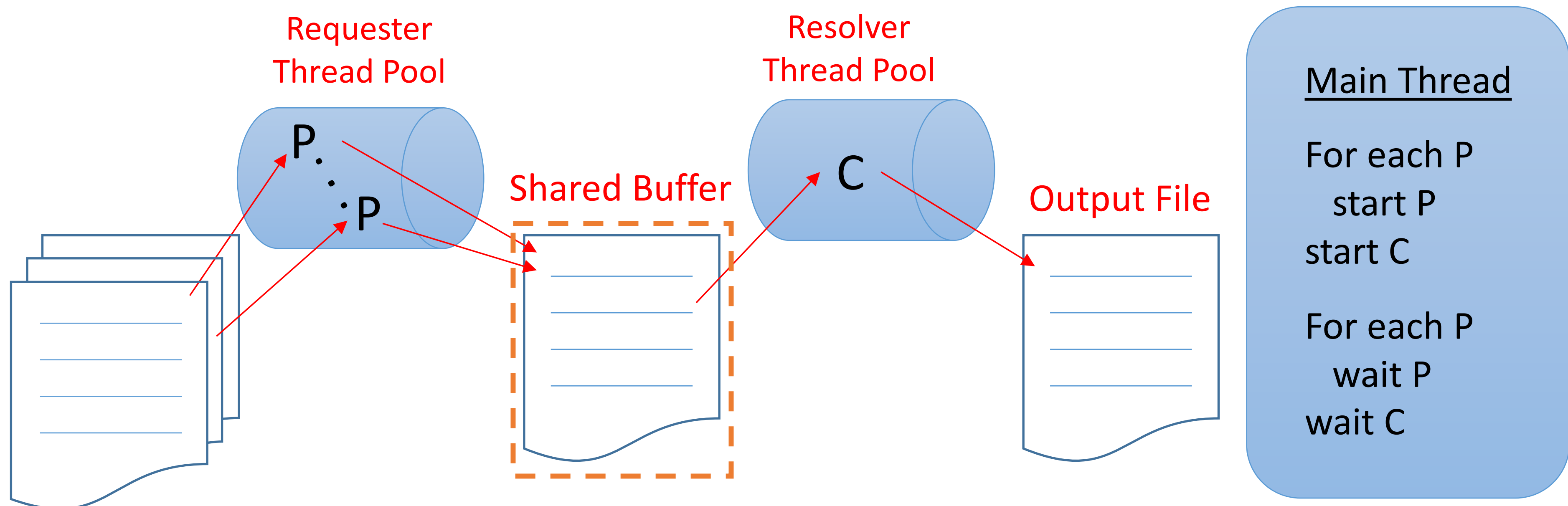
Implementation – Step 3

- **Action:** Once you are assured that your application can write and read to the buffer correctly (although serially), then try to make them run concurrently. Multiple processes accessing and modifying the same data can cause race conditions. You must protect the critical sections of each thread with a mutex.



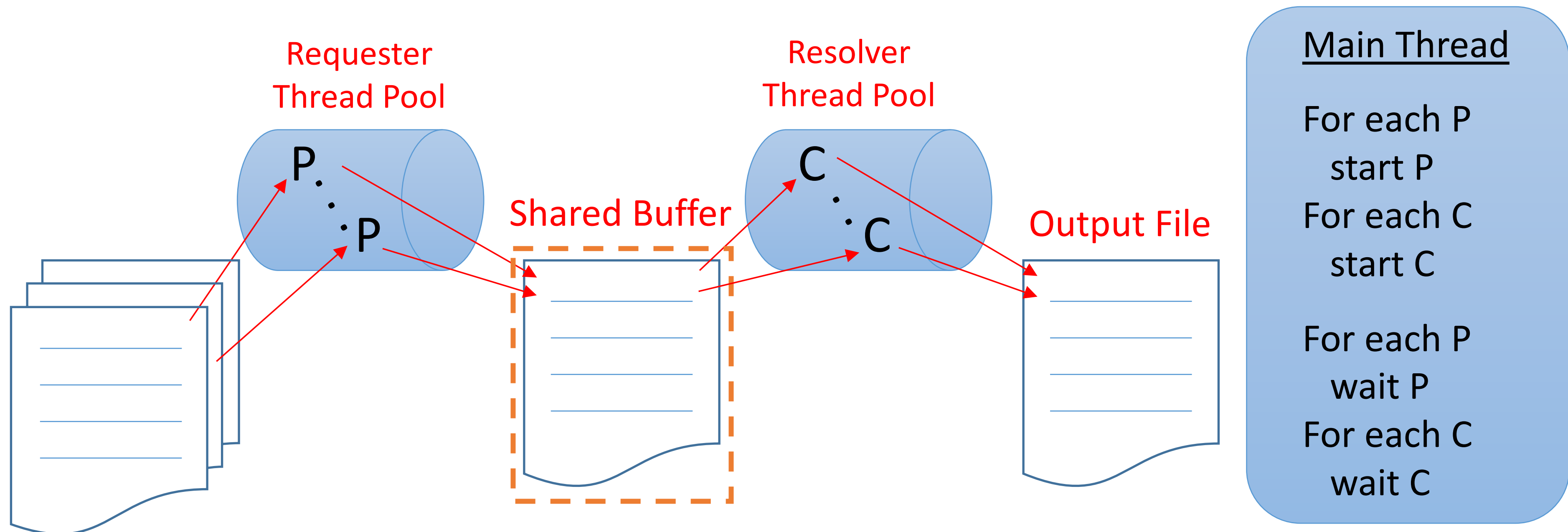
Implementation – Step 4

- Action: create multiple parsor threads to read from multiple different files. Each parsor can read single lines from a different file. The parsor will terminate when all lines from the file have been processed.



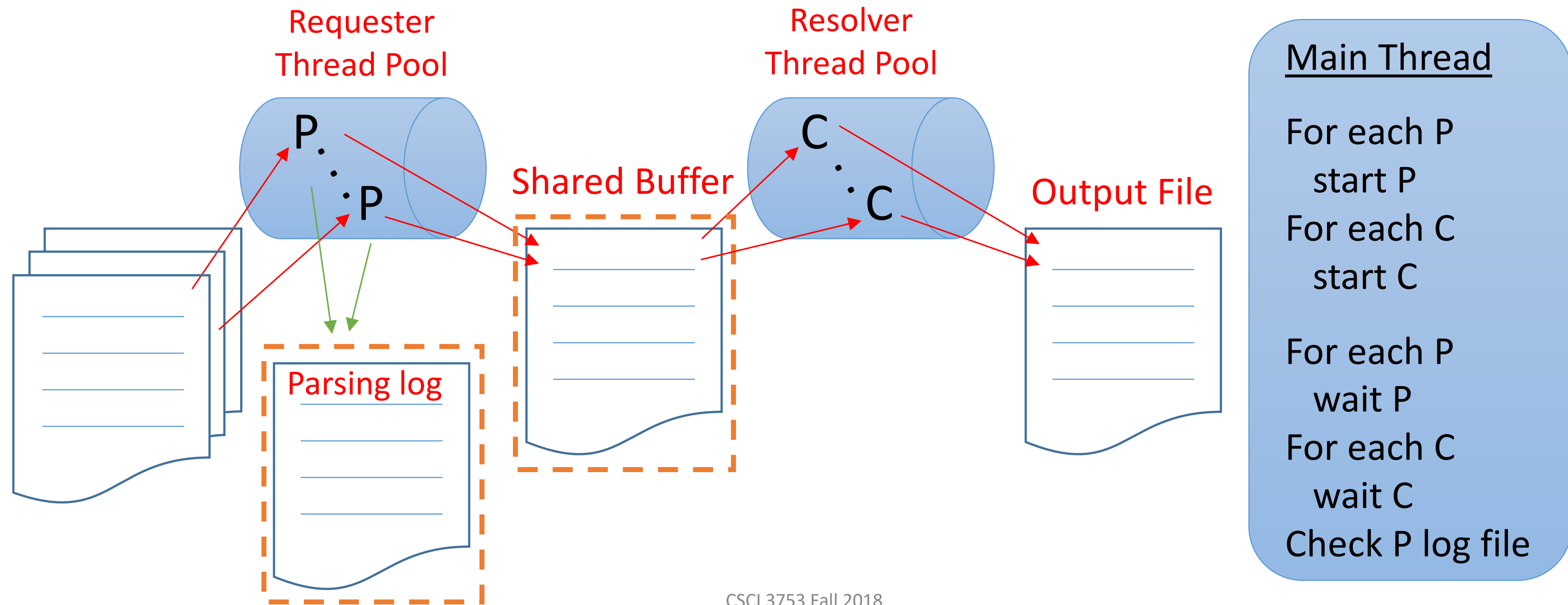
Implementation – Step 5

- Action: create multiple converter threads to read from multiple parsor threads via a single shared buffer.
  The converter will wait for data (spin wait is acceptable) but will terminate if there are no active parsor and the buffer is empty.



Implementation – Step 6

- **Action:** move back to the parsor threads, each thread must record the data it has processed. Files are a shared resource and therefore must be protected from multiple processes accessing it.



Program Check – Memory Leakage

- To verify that you do not leak memory, use `valgrind()` to test your program
- To install `valgrind()`, use the following command:

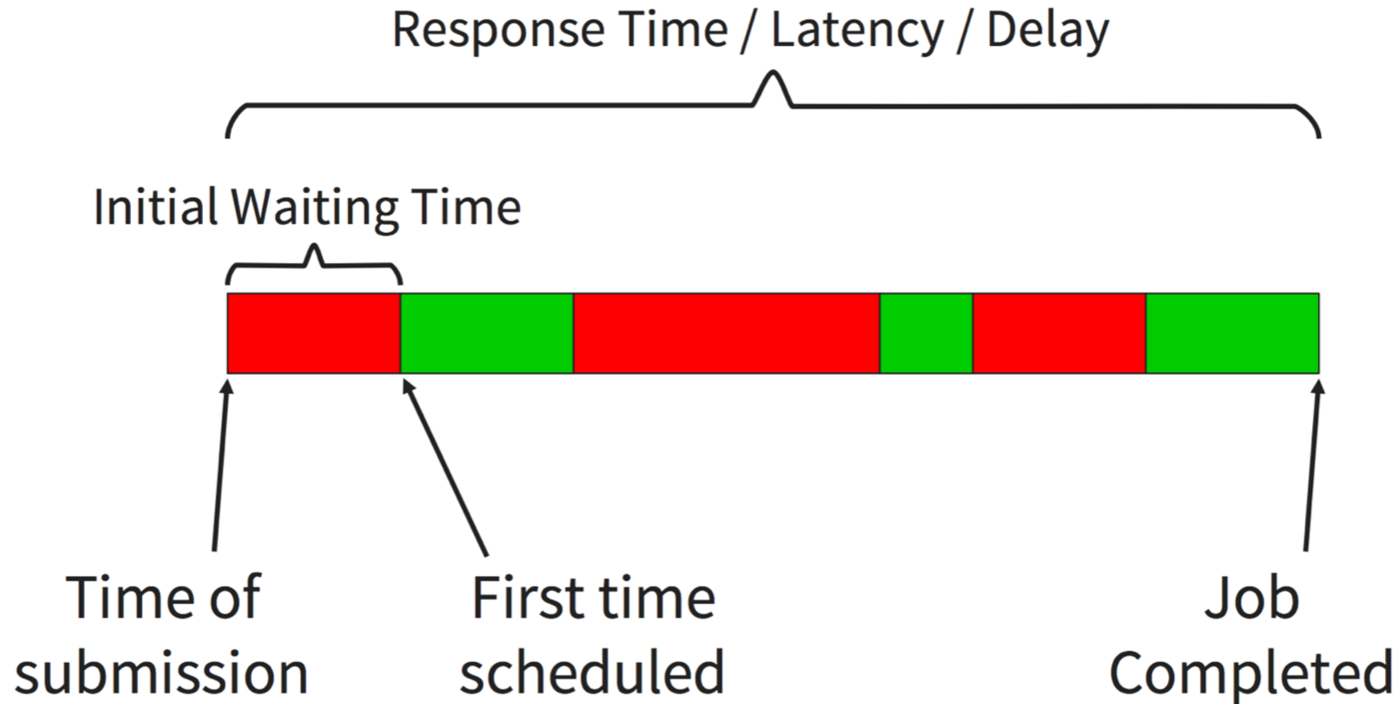
```
sudo apt-get install valgrind
```

- To use `valgrind()` to monitor your program, use this command:

```
valgrind ./pa3main
  text1.txt ...... textN.txt results.txt
```

Week 7: Process Scheduling

Per Job or Task Metrics



Total Waiting Time: sum of “red” periods

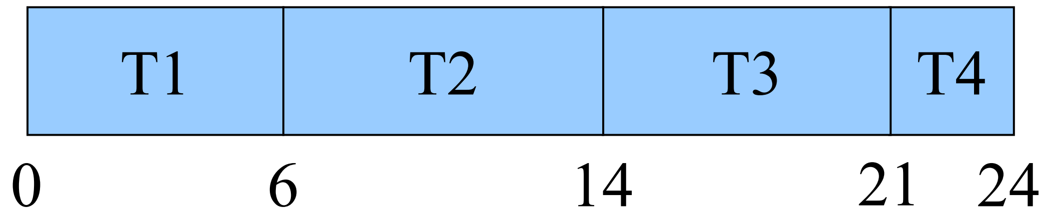
Scheduling Policies

- First Come First Serve (FCFS) Scheduling
- Shortest Job First (SJF) Scheduling
- Round Robin Scheduling

Task	CPU Execution Time (ms)
T1	6
T2	8
T3	7
T4	3

Scheduling Policies

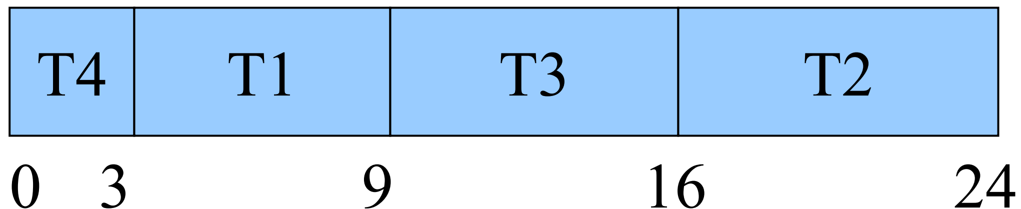
- First Come First Serve (FCFS) Scheduling
  - Average waiting time?
  - Average turnaround time?



Task	CPU Execution Time (ms)
T1	6
T2	8
T3	7
T4	3

Scheduling Policies

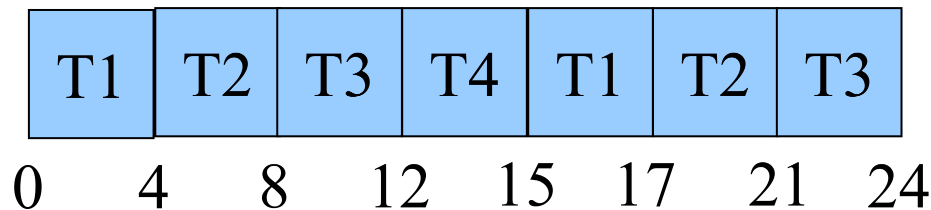
- Shortest Job First (SJF) Scheduling
  - Average waiting time?
  - Average turnaround time?



Task	CPU Execution Time (ms)
T1	6
T2	8
T3	7
T4	3

Scheduling Policies

- Round Robin Scheduling
  - Let time slice = 4 ms
  - Assuming a 1ms switching time
  - Average waiting time?
  - Average turnaround time?
  - Average response time?



Task	CPU Execution Time (ms)
T1	6
T2	8
T3	7
T4	3

Week 7 – Checklist

- ❑ Announcements
- ❑ PA3 instructions
- ❑ Read the PA3_Addendum.pdf
- ❑ Discuss process scheduling polycies
- ❑ Read more about scheduling