Name: Zhenjiang Tian
BlazerID: ztian
Course: CS 532
Assignment: HW4

1. Design and Implementation

1.1 Overall Architecture

The program uses one parent process and one child process created by fork(). A unidirectional pipe is used for communication: the parent writes integers to the pipe, while the child reads them.

- Parent process:
  - Creates the pipe.
  - Forks the child process.
  - Closes the read-end of the pipe.
  - Creates 10 producer threads.
  - Waits for all producer threads to finish.
  - Closes the write-end of the pipe.
  - Sends SIGUSR1 to the child to indicate that all numbers have been generated.

- Child process:
  - Installs a SIGUSR1 signal handler.
  - Closes the write-end of the pipe.
  - Waits using pause() until the SIGUSR1 signal is received.
  - Creates 20 consumer threads to read from the pipe.
  - Waits for all consumer threads to finish.
  - Computes the average of the 20 partial sums and prints the result.

1.2 Producer Threads

The parent process creates NUM_PRODUCERS = 10 threads. Each producer thread:
- Generates NUM_PER_PRODUCER = 500 unique random integers in the range [0, 1000]. A local array (used[1001]) is maintained per thread to guarantee uniqueness within that thread.
- Uses rand_r() with a per-thread seed to avoid data races in random generation.
- Before calling write() on the pipe, the thread acquires a mutex (pipe_mutex). This ensures that at most one producer writes to the pipe at a time and prevents interleaving of partial writes.
- After write() completes, the thread releases the mutex.
- Periodically prints progress (every 50 numbers) and a final completion message. A separate mutex (print_mutex) is used to keep console output readable.

This satisfies the requirements of using threads, generating random numbers, and applying synchronization to avoid race conditions and data corruption.

1.3 Consumer Threads

The child process creates NUM_CONSUMERS = 20 threads after receiving SIGUSR1.
Each consumer thread:

- Reads NUM_PER_CONSUMER = 250 integers from the pipe using read().
- Accumulates these values into a local long long sum.
- Stores its result in a global array consumer_sums[tid].
- Prints its own completion message and sum (also protected by print_mutex).

When all consumer threads have completed, the child process:

- Adds up all 20 entries of consumer_sums.
- Computes the average = total / NUM_CONSUMERS.
- Prints the final average to standard output, which is redirected to result.txt.

1.4 Signal Mechanism (Graduate Requirement)

To meet the graduate requirement of signaling:

- The child process installs a signal handler for SIGUSR1:
    - The handler sets a global flag start_reading = 1.
- Before creating any consumer threads, the child calls pause() in a loop:
    - while (!start_reading) pause();
- The parent process sends SIGUSR1 to the child using:
    - kill(child_pid, SIGUSR1);

  but only after:
    - All producer threads have finished.
    - The write-end of the pipe has been closed.

This design guarantees that:

- The child does not start reading from the pipe until all numbers have been generated.
- The signal serves exactly as the notification mechanism required by the assignment.

1.5 Synchronization and Data Integrity

- pipe_mutex: protects write() operations from concurrent producers.
- print_mutex: protects printf() output from multiple threads.
- The pipe itself guarantees ordered delivery of the written integers.
- On macOS, POSIX unnamed semaphores are not available, so mutexes are used instead to provide mutual exclusion around critical sections.

2. AI Tool Usage

2.1 Tool Used

- ChatGPT

2.2 How the AI Tool Helped

The AI tool was used for debugging.

2.3 Example Prompts Used

I wrote a C code, but the system prompts the following problem at runtime. I really don't know why. Can you tell me the reason and how I should modify it

```
(base) ztian@Zhenjiangs-MacBook-
Pro hw4 % gcc -Wall -Wextra -O2 -std=c11 hw4_ztian.c -lpthread
-o hw4 hw4_ztian.c:134:13: warning: 'sem_init' is deprecated [-
Wdeprecated-declarations] 134 | if (sem_init(&pipe_sem, 0, 1) !=
0) { | ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/in
clude/sys/semaphore.h:55:42: note: 'sem_init' has been explicitly
marked deprecated here 55 | int sem_init(sem_t *, int, unsigned
int) __deprecated; | ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/in
clude/sys/cdefs.h:214:40: note: expanded from macro
'__deprecated' 214 | #define __deprecated
__attribute__((__deprecated__)) | ^ hw4_ztian.c:159:9: warning:
'sem_destroy' is deprecated [-Wdeprecated-declarations] 159 |
sem_destroy(&pipe_sem); | ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/in
clude/sys/semaphore.h:53:26: note: 'sem_destroy' has been
explicitly marked deprecated here 53 | int sem_destroy(sem_t *)
__deprecated; | ^
/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/in
clude/sys/cdefs.h:214:40: note: expanded from macro
'__deprecated' 214 | #define __deprecated
__attribute__((__deprecated__)) | ^ 2 warnings generated. (base)
ztian@Zhenjiangs-MacBook-Pro hw4 % ./hw4 > result.txt
sem_init: Function not implemented
```

Thought for 24s ›

This isn’t your fault — it’s macOS being weird. 😅

This isn’t your fault — it’s macOS being weird. 😅
Your code is basically fine for Linux, but **macOS does not support `sem_init` / `sem_destroy` for unnamed semaphores**, so at runtime it fails with:

```text
sem_init: Function not implemented
```

Let’s break it down and then I’ll show you how to fix it.

---

1. What the warning means

When you compile:

```bash
gcc -Wall -Wextra -O2 -std=c11 hw4_ztian.c -lpthread -o hw4
```

you see:

```text
warning: 'sem_init' is deprecated
warning: 'sem_destroy' is deprecated
```

On macOS, the header `/usr/include/sys/semaphore.h` marks `sem_init` and `sem_destroy` as `__deprecated`. That’s the compiler telling you: “these functions are old and not really supported anymore”.

Then at runtime, your program calls:

c

Copy code

```c
if (sem_init(&pipe_sem, 0, 1) != 0) {
    perror("sem_init");
    exit(1);
}
```

and `sem_init` returns `-1` with `errno = ENOSYS`, so `perror` prints:

text

Copy code

```text
sem_init: Function not implemented
```

So: **on macOS, unnamed POSIX semaphores (`sem_t`, `sem_init`, `sem_destroy`) simply don't work.** They're stubs.

On Linux they work fine, which is why your professor's code or slides probably use them.

---

2. What you can do instead

You have three realistic options:

Option A: Run the same code on Linux (no code change)

If you want to keep using `sem_t` exactly as in your homework:

- Compile and run on:
  - Your course's Linux server, **or**
  - Ubuntu in a VM, **or**
  - WSL on Windows (if you ever use Windows),

On Linux, the same code:

Copy code

```
sem_t pipe_sem;

if (sem_init(&pipe_sem, 0, 1) != 0) {
    perror("sem_init");
    exit(1);
}
...
sem_destroy(&pipe_sem);
```

will work and not print “Function not implemented”.

If your professor will grade on a Linux machine anyway, this is the *safest* option for homework correctness.

Option B: Replace the semaphore with a `pthread_mutex_t` (for macOS)

If you’re just using the semaphore as a **lock** (to protect some shared data or a pipe), a mutex does the same job for threads.

1. **At the top of your file**, replace the semaphore with a mutex:

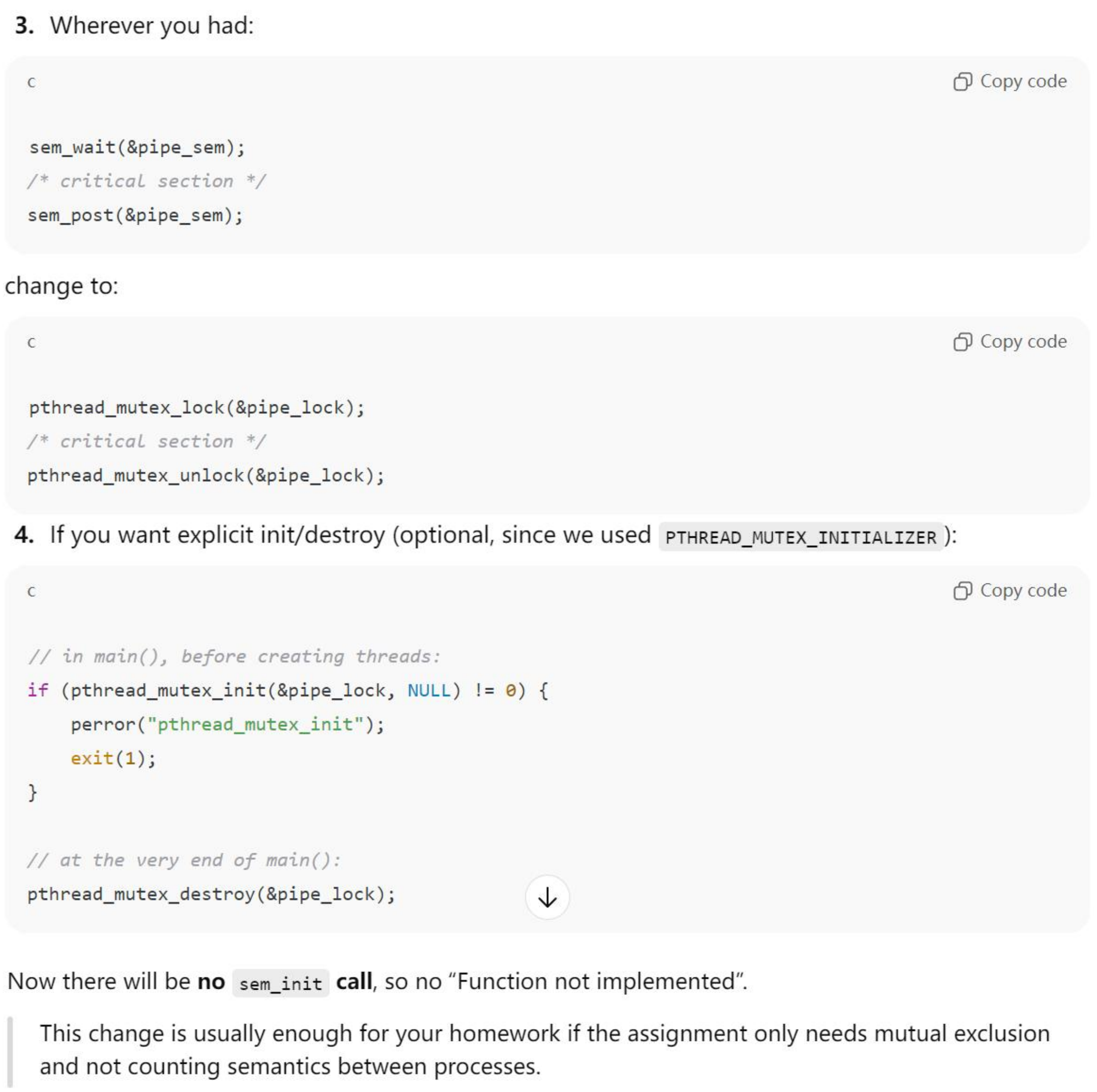
c

Copy code

```c
#include <pthread.h>
// #include <semaphore.h>   // you can remove this if you no longer use sem_t

pthread_mutex_t pipe_lock = PTHREAD_MUTEX_INITIALIZER;
```

2. **Remove** your `sem_init` and `sem_destroy` calls (e.g. at lines 134 and 159 in your code).
3. Wherever you had:



3. Wherever you had:

```c
sem_wait(&pipe_sem);
/* critical section */
sem_post(&pipe_sem);
```

change to:

```c
pthread_mutex_lock(&pipe_lock);
/* critical section */
pthread_mutex_unlock(&pipe_lock);
```

4. If you want explicit init/destroy (optional, since we used `PTHREAD_MUTEX_INITIALIZER`):

```c
// in main(), before creating threads:
if (pthread_mutex_init(&pipe_lock, NULL) != 0) {
    perror("pthread_mutex_init");
    exit(1);
}

// at the very end of main():
pthread_mutex_destroy(&pipe_lock);
```

Now there will be **no** `sem_init` **call**, so no “Function not implemented”.

> This change is usually enough for your homework if the assignment only needs mutual exclusion and not counting semantics between processes.

So I just followed the steps of option B and modified my code, and then it worked well.