W23055814

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

#define FILENAME "shared_file.txt"

// Task 3 - Mutex for synchronization
pthread_mutex_t lock;
sem_t sem;  // Semaphore for Task 4

// Producer function
void *producer(void *arg) {
    FILE *file;
    pthread_mutex_lock(&lock);



file = fopen("shared_file.txt", "w");
if (file == NULL) {
printf("could not open file.\n");
pthread_mutex_unlock(&lock);
return NULL;
}
// writing to file using even odd conditions with delay
for (int i = 1; i <= 10; i++) {
if (i % 2 == 0) {
fprintf(file, "even number: %d\n", i);
} else {
fprintf(file, "odd number: %d\n", i);
}
sleep(1);
}
fclose(file);
printf("finished writing to file.\n");

pthread_mutex_unlock(&lock);
sem_post(&sem);
return NULL;
}
```

```
  // TODO: Write multiple data entries to a file with conditions on when/how to write (e.g., even/odd).
  //      You can choose to implement any logic (loop types, conditions) to control the write behavior.
  //      Add delays between entries.

  /////////////////////
```

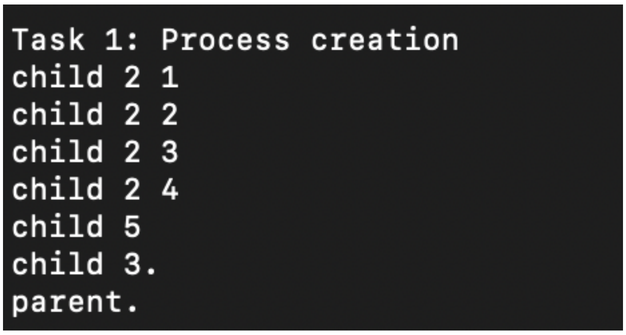
```
// consumer function
void *consumer(void *arg) {
FILE *file;
sem_wait(&sem);
pthread_mutex_lock(&lock);
char line[256];

file = fopen(FILENAME, "r");
if (!file) {
perror("error opening file");
pthread_mutex_unlock(&lock);
return NULL;
}
int line_num = 1;
while (fgets(line, sizeof(line), file)) {
if (line_num % 2 == 0) {
printf("consumer: even line %d - %s", line_num, line);
} else {
printf("consumer: odd line %d - %s", line_num, line);
}
line_num++;
}
fclose(file);
pthread_mutex_unlock(&lock);
return NULL;
}
```

```
   // TODO: Read multiple data entries from the file.
   //      Consider how you want to handle reading lines and displaying the output (e.g., even/odd).
   //      You have the flexibility to use different methods (loops, conditions) for reading and displaying.

   pthread_mutex_unlock(&lock);
   return NULL;
}

// Task 1 - Process creation
void create_process() {
   pid_t pid;
   pid = fork();

if (pid < 0) {
printf("error.\n");
return;
} else if (pid == 0)
{
int i = 1;
while (i <= 5) {
if (i == 5) {
printf("child %d\n", i);
} else {
printf("child 2 %d\n", i);
}
sleep(1);
i++;
}
printf("child 3.\n");
exit(0);
} else {
wait(NULL);
printf("parent.\n");
}

   // TODO: Create a new process and determine how it should behave.
   //      Design a loop where the child process performs some repetitive task.
   //      Use conditional logic (e.g., print something special) for the final iteration.

}

//////////////////////////
// Task 2 - Thread scheduling
void *thread_function(void *arg) {
int thread_num = *((int *)arg);
printf("Thread %d: Starting task.\n", thread_num);
for (int i = 1; i <= 5; i++) {
if (thread_num == 1) {
printf("thread %d: processed iteration %d with high priority.\n", thread_num, i);
sleep(2);
} else {
printf("thread %d: processed iteration %d with low priority.\n", thread_num, i);
usleep(1000000);
}
```

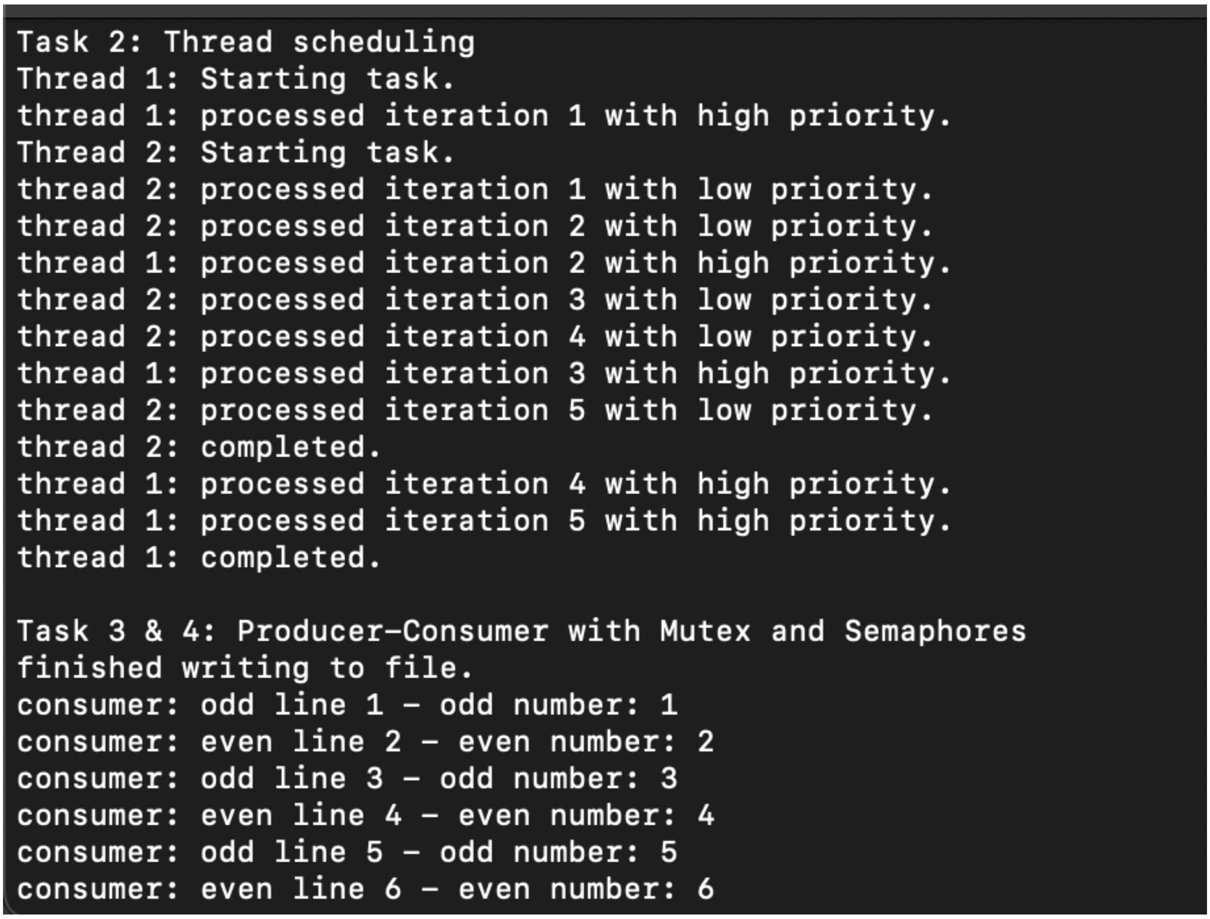
```
 }

printf("thread %d: completed.\n", thread_num);
return NULL;
}

   // TODO: Implement a loop where threads perform a task with priority or conditionally based on the thread number.
   //      You can introduce conditional checks or delays based on thread characteristics.
   //      The goal is to showcase some control over thread behavior.

   return NULL;
}

// Main function
int main() {
   pthread_t thread1, thread2, prod, cons;
   int thread_num1 = 1, thread_num2 = 2;

   // Initialize mutex and semaphore
   pthread_mutex_init(&lock, NULL);
   sem_init(&sem, 0, 0);  // Binary semaphore

   // Task 1 - Process creation
   printf("Task 1: Process creation\n");
   create_process();

   // Task 2 - Thread creation and scheduling
```

```
  printf("\nTask 2: Thread scheduling\n");
  pthread_create(&thread1, NULL, thread_function, &thread_num1);
  pthread_create(&thread2, NULL, thread_function, &thread_num2);

  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);

  // Task 3 & 4 - Producer and Consumer problem with synchronization
  printf("\nTask 3 & 4: Producer-Consumer with Mutex and Semaphores\n");
  pthread_create(&prod, NULL, producer, NULL);
  pthread_create(&cons, NULL, consumer, NULL);

  pthread_join(prod, NULL);
  pthread_join(cons, NULL);

  // Cleanup
  pthread_mutex_destroy(&lock);
  sem_destroy(&sem);

  return 0;
}
```

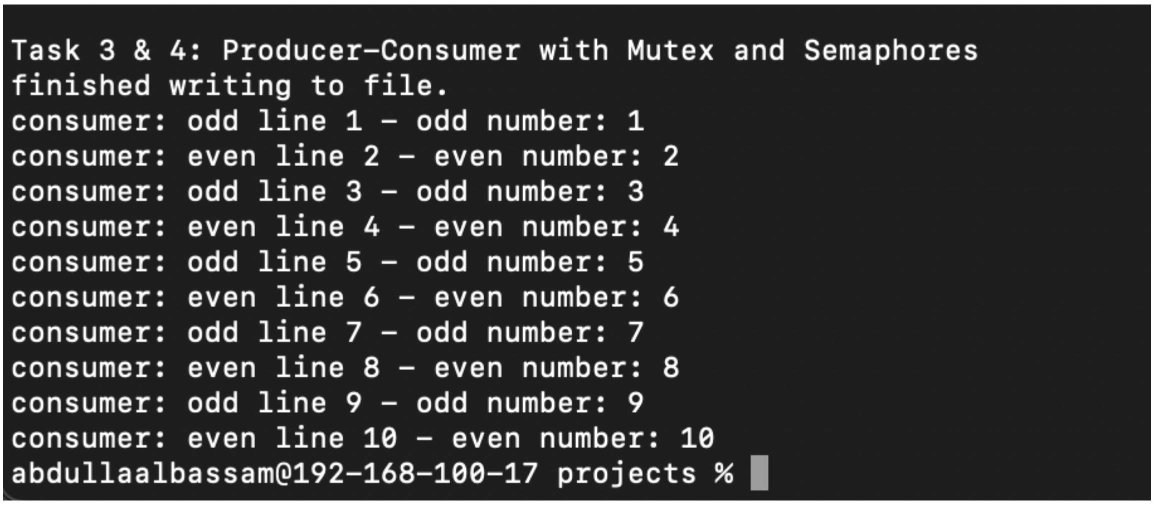
Ran on “terminal” on my MacBook, screenshots of the code running successfully:



```
Task 1: Process creation
child 2 1
child 2 2
child 2 3
child 2 4
child 5
child 3.
parent.
```



```
Task 2: Thread scheduling
Thread 1: Starting task.
thread 1: processed iteration 1 with high priority.
Thread 2: Starting task.
thread 2: processed iteration 1 with low priority.
thread 2: processed iteration 2 with low priority.
thread 1: processed iteration 2 with high priority.
thread 2: processed iteration 3 with low priority.
thread 2: processed iteration 4 with low priority.
thread 1: processed iteration 3 with high priority.
thread 2: processed iteration 5 with low priority.
thread 2: completed.
thread 1: processed iteration 4 with high priority.
thread 1: processed iteration 5 with high priority.
thread 1: completed.

Task 3 & 4: Producer-Consumer with Mutex and Semaphores
finished writing to file.
consumer: odd line 1 - odd number: 1
consumer: even line 2 - even number: 2
consumer: odd line 3 - odd number: 3
consumer: even line 4 - even number: 4
consumer: odd line 5 - odd number: 5
consumer: even line 6 - even number: 6
```



```
Task 3 & 4: Producer-Consumer with Mutex and Semaphores
finished writing to file.
consumer: odd line 1 - odd number: 1
consumer: even line 2 - even number: 2
consumer: odd line 3 - odd number: 3
consumer: even line 4 - even number: 4
consumer: odd line 5 - odd number: 5
consumer: even line 6 - even number: 6
consumer: odd line 7 - odd number: 7
consumer: even line 8 - even number: 8
consumer: odd line 9 - odd number: 9
consumer: even line 10 - even number: 10
abdullaalbassam@192-168-100-17 projects %
```