Operating Systems Lab Assignment: Synchronization and Scheduling

Bryce Coleman

October 24th, 2025

1 Introduction

This report documents the implementations and analyses for the synchronization and scheduling lab assignment, covering five provided problems and four additional exercises using mutexes and condition variables.

2 Exercise 1: Hello World

```c
#include <pthread.h>
#include <stdio.h>

pthread_mutex_t lock;
pthread_cond_t cv;
int hello = 0;

void* print_hello(void* arg) {
    pthread_mutex_lock(&lock);
    hello += 1;
    printf("First␣line␣(hello=%d)\n", hello);
    pthread_cond_signal(&cv);
    pthread_mutex_unlock(&lock);
    pthread_exit(0);
}

int main() {
    pthread_t thread;
    pthread_mutex_init(&lock, NULL);
    pthread_cond_init(&cv, NULL);

    pthread_create(&thread, NULL, print_hello, NULL);
    pthread_mutex_lock(&lock);
    while (hello < 1) {
        pthread_cond_wait(&cv, &lock);
    }
    printf("Second␣line␣(hello=%d)\n", hello);
    pthread_mutex_unlock(&lock);

    pthread_join(thread, NULL);
    pthread_mutex_destroy(&lock);
    pthread_cond_destroy(&cv);
    return 0;
}
```

Explanation: The original code fails because the main thread prints before the helper thread updates the shared variable. Using a mutex and condition variable ensures the main thread waits until the helper signals that the update is complete, fixing the synchronization issue.

Analysis: This program demonstrates basic synchronization using a mutex and condition variable. The threads communicate safely without race conditions, and the output order remains consistent regardless of execution timing.

Screenshot: Include a screenshot of compiling and running `hello_world.c`.

```
● @brryc3 ➜ /workspaces/sync (main) $ gcc -o hello_world hello_world.c
● @brryc3 ➜ /workspaces/sync (main) $ ./hello_world
  First line (hello=1)
  Second line (hello=1)
```

Figure 1: Compilation and execution of hello_world.c

3 Exercise 2: SpaceX Problems

```c
#include <pthread.h>
#include <stdio.h>

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
int n = 3;

void* counter(void* arg) {
    pthread_mutex_lock(&lock);
    while (n > 0) {
        printf("%d\n", n);
        n--;
        pthread_cond_signal(&cv);
        pthread_mutex_unlock(&lock);
        pthread_mutex_lock(&lock);
    }
    pthread_mutex_unlock(&lock);
    return NULL;
}

void* announcer(void* arg) {
    pthread_mutex_lock(&lock);
    while (n != 0) {
        pthread_cond_wait(&cv, &lock);
    }
    printf("FALCON␣HEAVY␣TOUCH␣DOWN!\n");
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, counter, NULL);
    pthread_create(&t2, NULL, announcer, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    return 0;
}
```

Explanation: The announcer thread printed too early because threads weren't synchronized. The fix uses a condition variable so the announcer waits until all countdown threads finish before printing the final message.

Analysis: All countdown threads complete before the announcer prints, showing effective coordination between multiple threads. The use of condition variables ensures proper sequencing and prevents premature execution.

Screenshot: Include a screenshot of compiling and running `spacex.c`.

```
● @brryc3 → /workspaces/sync (main) $ gcc -o spacex spacex.c
● @brryc3 → /workspaces/sync (main) $ ./spacex
  3
  2
  1
  FALCON HEAVY TOUCH DOWN!
```

Figure 2: Compilation and execution of spacex.c

4 Exercise 3: Subaru Synchronization

```
\documentclass{article}
\usepackage{geometry}
\geometry{a4paper, margin=1in}
\usepackage{graphicx}
\usepackage{listings}
\usepackage{xcolor}
\usepackage[utf8]{inputenc}

\lstset{
  basicstyle=\ttfamily\small,
  breaklines=true,
  frame=single,
  language=C,
  keywordstyle=\color{blue},
  commentstyle=\color{green!50!black},
  stringstyle=\color{red}
}

\begin{document}

\title{Operating Systems Lab Assignment: Synchronization and Scheduling}
\author{Bryce Coleman}
\date{October 24th, 2025}
\maketitle

\section{Introduction}
This report documents the implementations and analyses for the synchronization
    and scheduling lab assignment, covering five provided problems and four
    additional exercises using mutexes and condition variables.

\section{Exercise 1: Hello World}
\lstinputlisting{hello_world.c}

\textbf{Explanation:} The original code fails because the main thread prints
    before the helper thread updates the shared variable. Using a mutex and
    condition variable ensures the main thread waits until the helper signals
    that the update is complete, fixing the synchronization issue.

\textbf{Analysis:} This program demonstrates basic synchronization using a
    mutex and condition variable. The threads communicate safely without race
    conditions, and the output order remains consistent regardless of execution
    timing.

\textbf{Screenshot:} Include a screenshot of compiling and running \texttt{
    hello\_world.c}.
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{exercise1_screenshot.png}
\caption{Compilation and execution of hello\_world.c}
\end{figure}

\section{Exercise 2: SpaceX Problems}
```